μITRON-LP: Power-Conscious Real-Time OS Based on Cooperative Voltage Scaling for Multimedia Applications

Hiroshi Kawaguchi, *Member, IEEE*, Youngsoo Shin, *Member, IEEE*, and Takayasu Sakurai, *Fellow, IEEE*

***Abstract*—This paper presents a cooperative dynamic power management method and its implementation. The implementation consists of design of a real-time OS, applications including MPEG-4, and development of a supporting hardware platform with an off-the-shelf processor. We describe several factors that are important in the implementation and discuss its efficiency through experiment. The experimental results with the prototype system shows that 74% power saving is possible in multi-task multimedia environment.**

***Index Terms*—Application slicing, dynamic voltage scaling, embedded system, low power, MPEG-4, multimedia application, real-time OS.**

I. Introduction

FOR MULTIMEDIA mobile systems powered by a battery such as a 3G cellphone, power-efficient design managing both low power and high speed is required. As processor performance improves, the power management of the systems is increasingly realized through software [1]–[6]. Consequently, the design of the software components including an operating system and applications is becoming important for the high power efficiency.

Crusoe adopts software power management called LongRun [7], [8], which basically relies on its workload history. Crusoe, however, cannot reduce power by making use of data-dependent nature of multimedia applications nor guarantee real-time feature. Thus, LongRun works fine in PC environment, but is not suitable for embedded systems.

On the other hand, cooperative voltage scaling (CVS) [9] is a dynamic power management method, which encompasses interaction among a real-time operation system (RTOS), applications, and hardware to reduce power consumed by a processor. The RTOS is modified so that it maintains and provides timing information to the applications. The application is also modified so that it consists of a sequence of slices, and additional code fragments are inserted at the head of each slice. The code fragments of the application determine the operation frequency (f) and supply voltage ($\mathrm{V_{DD}}$) of the slices based on both the timing information provided by the RTOS and its own worst-case execution time (WCET). The rationale of the CVS is that the RTOS knows only global timing information among tasks while each application has better knowledge about its own structure and behavior.

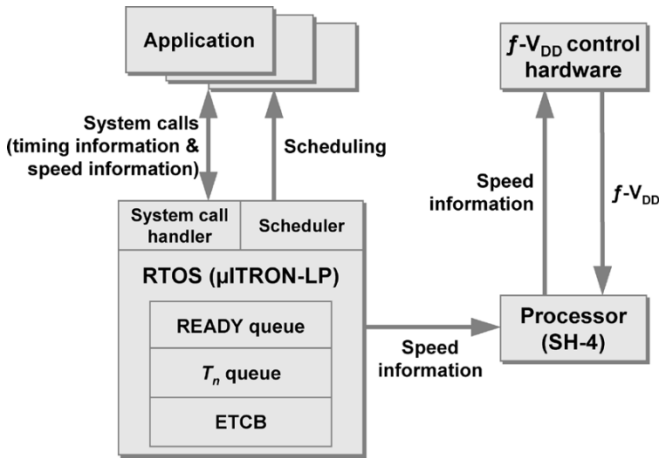


Fig. 1. Structural model of CVS. A task gets timing information and sends speed information to external f-$\mathrm{V_{DD}}$ control hardware via processor. By using this speed information, a combination of f and $\mathrm{V_{DD}}$ is supplied to the processor.

In this paper, we address experimental implementation of the CVS to evaluate feasibility and efficiency of its model. The implementation consists of three components: *design of a power-conscious RTOS*, *design of applications with the concept of application slicing*, and *design of a hardware platform*.

The remainder of this paper is organized as follows. In the Section II, we explain the CVS from the software point of view. In Section III, the hardware implementation details are presented. In Section IV, we discuss the experimental results to evaluate the CVS. Finally, a summary follows in Section V.

II. Cooperative Voltage Scaling

A. Model

Fig. 1 shows the structural model of the CVS. The software architecture of the CVS consists of a power-conscious RTOS and applications. In order to realize the RTOS, Hitachi HI7750 [10] that is based on the μITRON specification [11] is redesigned, which we call μITRON-LP in this paper. In μITRON-LP, real-time tasks are scheduled according to

Manuscript received November 20, 2002; revised May 17, 2003. This work was supported by grants from Hitachi and the Japan Society for the Promotion of Science. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ryoichi Komiya.

H. Kawaguchi is with the Institute of Industrial Science, University of Tokyo, Tokyo 153-8505, Japan (e-mail: kawapy@iis.u-tokyo.ac.jp).

Y. Shin was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA. He is now with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Korea (e-mail: youngsoo@ee.kaist.ac.kr).

T. Sakurai is with the Center for Collaborative Research, University of Tokyo, Tokyo 153-8505, Japan (e-mail: tsakurai@iis.u-tokyo.ac.jp).

Digital Object Identifier 10.1109/TMM.2004.840592

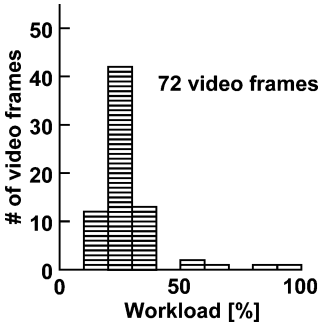


Fig. 2. Example of workload histogram of MPEG-4 codec. This shows the case where H.263 standard image sequence "carphone" is used as input data. The total number of video frames is 72. This sequence is also used in the experiment described in Section IV.

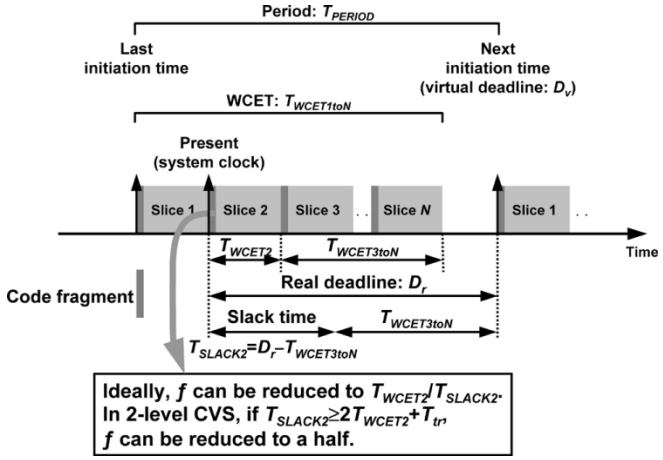


Fig. 3. Two-level application slicing. At the head of each slice, a code fragment is inserted to determine speed of a processor. T_{tr} indicates transition time of f and $\mathrm{V_{DD}}$.

fixed-priority preemptive scheduling algorithm even though other scheduling algorithm can be used.

In μITRON-LP, an absolute time called system clock is maintained by cyclic interrupt from a hardware timer, whose interval is set to 1 ms, meaning 1 ms is the time resolution of the system. Since the timer interrupt involves interrupt service routines that consume certain processor cycles, we cannot arbitrarily decrease the time resolution.

A RTOS kernel is frequently realized with task control blocks (TCBs) and a set of priority queues. The TCB holds task-specific information such as priority and start address. Each queue maintains a list of tasks under the same scheduling status. We add the READY queue and T_n (next initiation time) queue to μITRON-LP. The READY queue holds a currently running task as well as tasks waiting to run in order of priority. If a task currently occupies a processor, it is called a RUN task, which is at the head of the READY queue. It should be noted that the RUN task is still in the READY queue even though it is running. The T_n queue holds all tasks in ascending numerical order of the time, at which their next initiation is due.

We also extend the traditional TCB, which we call the extended task control block (ETCB) to contain specific timing information.

In addition, the scheduler in μITRON-LP is customized to perform necessary actions during task state transition. These include managing the READY queue and T_n queue, computing timing information in the ETCB, and putting the processor into a sleep mode if there is no task in the READY queue. The processor, however, wakes up in every system clock to keep the system clock counting and after that, return to the sleep mode. The details will be explained later in Section II-C.

B. Application Slicing

Multimedia applications usually synchronize with their own regular periods, for instance, 60 Hz for MPEG2 and 44.1 kHz for CD audio. The period itself is always larger than the WCET of the application. The execution time of the application, however, is frequently less than the WCET, sometimes by a large amount since workload strongly depends on data imposed on a processor [2]. As an example of MPEG-4 codec, the workload becomes higher as objects in an image move fast. However, as shown in Fig. 2, the worst-case data seldom occur in MPEG-4 codec and in most cases, the task finishes well before the WCET. In addition, even if the worst-case data occurs, we still have a time margin because the WCET is less than the period.

This is one of motivations for the CVS; execution time is not constant, that is, it does not always take the WCET to execute a task. At the start of each application, however, we do not have any information about its future execution time and hence, it is impossible to predict future workload without an error. We solve this problem by introducing application slicing in the manner of the $\mathrm{V_{DD}}$-hopping [12]. If a task is sliced, unused time from the previous slices can be exploited by the following slices. By checking the current time and slack time to execute the next slice, the application slicing adaptively selects optimum f and $\mathrm{V_{DD}}$ at run time to minimize power.

Although application slicing incurs much engineering effort, this can be done by application designers or middleware providers once and for all. Furthermore, recognizing the fact that a processor is occupied mostly by highly demanding applications such as MPEG-4, and the number of such applications is small in nature, system designers can compose systems with the sliced applications provided by the middleware providers and their own custom applications, which can be designed either by application slicing or not.

With the help of Fig. 3, we explain the concept of the application slicing under the assumption that only one task is running on a processor. In the figure, an application is periodic and its period is T_{PERIOD}. If applications are not periodic, application slicing is not applicable, however, fortunately multimedia applications are periodic as described at the beginning of this subsection. In other words, the CVS is suitable for synchronous tasks like multimedia applications, and not suitable for asynchronous tasks like communication processing. In the MPEG-4, although communication between terminals may be needed, communication rate is low, say 64 kbps and the overhead of the communication is estimated at less than 1%, which is negligible compared to the MPEG-4 itself.

The WCET of the application, $T_{WCET1toN}$ is chopped into N slices with potentially different length each other. The WCET of the ith slice, T_{WCETi} $(i = 1, \ldots, N)$ and the WCET from the ith to the Nth slices, $T_{WCETitoN}$ can be obtained through

```
structure ETCB {
    T_PERIOD;  // Task initiation period
    T_n;               // Next initiation time
    T_sta;             // Time when dispatched
    T_exe;     // Time executed already
    D_v;               // Virtual deadline
};
```

Fig. 4. Pseudo code of ETCB structure.

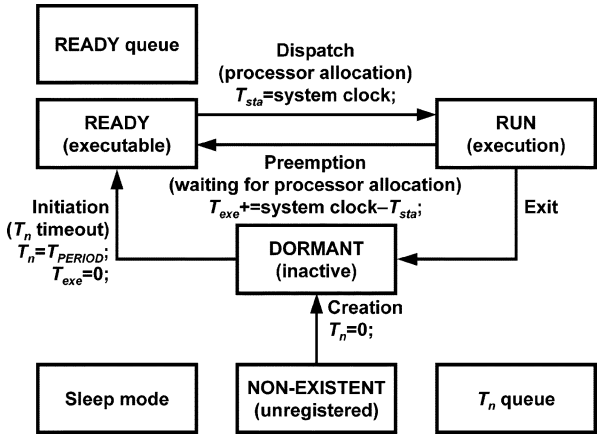


Fig. 5. Task state transition in μITRON-LP. The READY queue and T_n queues are renewed when a task is initiated or exits.

static analysis or direct measurement in design stage [13]. In the code fragment at the head of the ith slice, we now compute the interval of time that is allowed to execute the slice. μITRON-LP knows the next initiation time of this task since it is stored in the ETCB. The task itself obtains the time interval to the next initiation time as the virtual deadline, D_v from μITRON-LP with a system call.

Next, the task compares D_v to its own WCET and chooses larger one as the real deadline, D_r. In Fig. 3, D_v is essentially larger than the WCET and thus, D_v becomes D_r.

Then, by using D_r, the slack time, T_{SLACKi} is checked. T_{SLACKi} is obtained by subtracting $T_{WCETi+1toN}$ from D_r. Ideally, f can be reduced to T_{WCETi}/T_{SLACKi}. In reality, however, the arbitrary choice of f causes a serious problem at interfaces with peripheral devices. To solve this issue, in the CVS, the candidate f is limited only to $f_{\max}$ or $f_{\max}/2$ [14], [15], where $f_{\max}$ is the maximum frequency of the processor. In this two-level application slicing, the ith slice is carried out at $f_{\max}/2$ if $T_{SLACKi} \geq 2T_{WCETi} + T_{tr}$.

According to the above-mentioned process, the optimum f is adaptively selected by the software on a slice-by-slice basis. After finishing the Nth slice, the processor goes into a sleep mode until the next initiation of the task. In the CVS, the timing information including D_v is provided by μITRON-LP through the ETCB.

C. ETCB

Each task is associated with the ETCB. Fig. 4 shows a pseudo code of the ETCB structure, where each element is managed based on the task state transition as shown in Fig. 5.

- T_{PERIOD} refers to a regular period of task initiation. This is fixed and thus, does not change at run time.
- T_n refers to relative time at which next initiation is supposed to arrive. Every system clock, T_n of any task in any state is always decremented by one except for the case when T_n is 0 (T_n time-out). In μITRON-LP, the T_n queue is adopted to monitor the T_n time-out. All tasks are sorted in ascending numerical order of T_n to easily find the T_n time-out. If the T_n time-out happens, the associated task is automatically initiated and then, T_{PERIOD} is set to T_n. A newly created task is also immediately initiated because its T_n is reset.
- T_{sta} refers to system clock at which a RUN task is dispatched. T_{sta} is valid only when the task is in the RUN state.
- T_{exe} refers to accumulated time that had been already executed before the last preemption. It should be noted that T_{exe} is incremented by the remainder between the system clock and T_{sta} only when the task is preempted. T_{exe} is reset when the task is initiated.
- D_v refers to relative time to a virtual deadline of a RUN task and is provided to the RUN task by a system call for calculation of a real deadline. D_v is valid only when the task is in the RUN state. D_v becomes 0 if there are two tasks or more in the READY queue. In this event, the RUN task should be executed in its own WCET. On the other hand, if the RUN task is the only one in the READY queue, μITRON-LP chooses the smallest T_n in the T_n queue as D_v of the RUN task. In this case, the RUN task can occupy the processor at least until D_v because there is no task waiting for its execution. Fig. 6 shows how to determine D_v. The smallest T_n of the tasks in the T_n queue can be easily obtained because the tasks are sorted in ascending numerical order of T_n. Incidentally, D_v of the RUN task is also renewed every system clock.

D. Real Deadline

In each slice, the code fragment has to compute its own WCET to obtain the real deadline. Then, the code fragment compares it to D_v unless D_v is 0. The longer one is the real deadline, D_r as mentioned above.

As shown in Fig. 7, since μITRON-LP adopts the preemptive scheduling algorithm, the WCET should be acquired by subtracting accumulated execution time up to the present from $T_{WCET1toN}$, that is, the WCET becomes $T_{WCET1toN}$–T_{exe}–(system clock–T_{sta}).

E. Example

Now, we explain how the CVS works using an example of a task set as shown in Fig. 8. Suppose that there are three periodic Tasks A, B, and C, and T_{tr} is 0. Task A consists of three slices with each slice taking two time units in the worst case. Task B consists of six slices with total 12 time units in the worst case. Task C has only one slice whose WCET is two time units.

As for the workloads of the tasks in Fig. 8, we assume 50% of the worst case for Task A, that is, it takes one time unit to execute one slice. For Tasks B and C, 100% of workload is assumed meaning that they run in their WCETs.

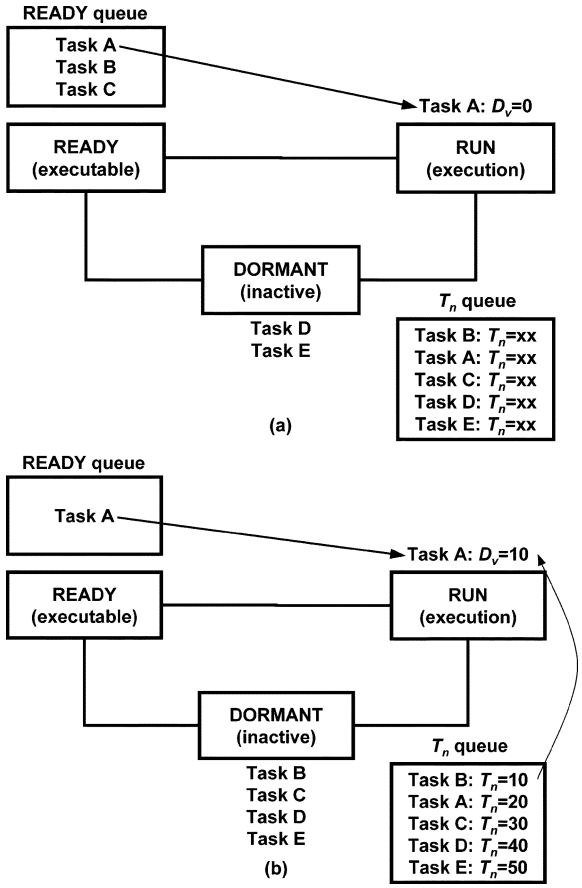


Fig. 6. How to determine D_v. (a) If there are two tasks or more in the READY queue, D_v of the RUN task becomes 0 regardless of T_n of tasks in the T_n queue. In this event, the RUN task should finish within its WCET. It should be noted that there is still possibility to decrease f and $\mathrm{V_{DD}}$ because some slices might complete their execution earlier than their WCETs. (b) If the RUN task is the only one in the READY queue, D_v of the RUN task becomes T_n of the task at the head of the T_n queue, which is the smallest T_n of all tasks. In this case, D_v can be exploited to lower f and $\mathrm{V_{DD}}$ if D_v is longer than the WCET of the RUN task.

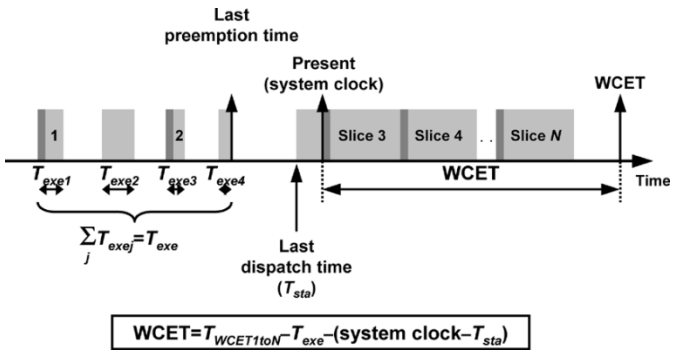


Fig. 7. Method to obtain WCET. A RUN task was preempted four times. The accumulated execution time before the last preemption is T_{exe}, and the execution time from the last dispatch time up to the present is (system clock–T_{sta}). The RUN task can get its own T_{exe} and T_{sta} with system calls.

In the original μITRON, the scheduling looks like Fig. 8(a), while the scheduling in μITRON-LP is shown in Fig. 8(b) when $f_{\max}$ and $f_{\max}/2$ are provided as available frequencies.

In μITRON-LP, at time 0, Tasks A, B, and C are initiated at the same time. Task A starts first since it has the highest priority. At the first slice of Task A, D_v is 0 because there are three

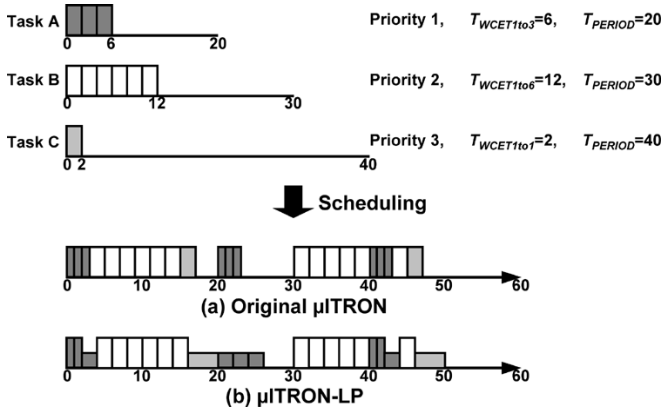


Fig. 8. Scheduling example of Tasks A, B, and C. Horizontal axis indicates time scale and height of the slice shows the magnitude of f. (a) Original μITRON. (b) μITRON-LP when f is limited to two levels.

tasks in the READY queue. In this case, the real deadline, D_r is 6, which is $T_{WCET1to3}$ of Task A. Then, as $T_{WCET2to3}$ is 4, T_{SLACK1} is 2. f remains $f_{\max}$ since T_{WCET1} is 2.

At time 1, the first slice finishes its execution because the workload of Task A is 50%. At the second slice, the WCET is 5 since T_{exe} is 0 and (system clock–T_{sta}) is now 1. $T_{WECT3to3}$ is 2 and then, T_{SLACK2} is 3. This is not enough to reduce f to a half. Thus, the second slice is also executed at $f_{\max}$. At the last slice of Task A, the situation is different from the previous slices. The WCET is 4 and T_{SLACK3} is also 4. Therefore, the third slice is carried out at a half speed, $f_{\max}/2$ and the power saving is possible.

Task A completes at time 4. Then, Task B takes over and is executed between time 4 and 16.

At time 16, Task C is allocated to the processor. At this time, only Task C is in the READY queue. The real deadline is the longer interval between the WCET of Task C and D_v. In this case, the real deadline is D_r of 4, which is T_n of Task A. Even though this slice is the first slice, it can be executed by $f_{\max}/2$ unlike the other tasks. Task C finishes at time 20.

Then, Task A starts again likewise. In the case where there is no task to execute, μITRON-LP brings the processor into the sleep mode until the next initiation.

III. Hardware Implementation

Fig. 9 shows a snapshot of the CVS experimental system. An embedded system board with Hitachi SH-4 is used as a target platform. The block diagram of the target platform is shown in Fig. 10. SH-4 has a frequency control register called FRQCR. The internal operation frequency is synchronized with the external clock frequency of 33 MHz and can be changed instantaneously by accessing the FRQCR. Since the operation frequencies of 200 MHz and 100 MHz are used and they are divisible by the external clock frequency, there is no synchronization problem at interfaces with peripheral devices. For processors that do not have a clock frequency control register, a clock frequency should be externally changed to provide $f_{\max}$ and $f_{\max}/2$. In this case, the processors must be halted during settling time of a clock distribution network include a PLL/DLL to eliminate malfunction.

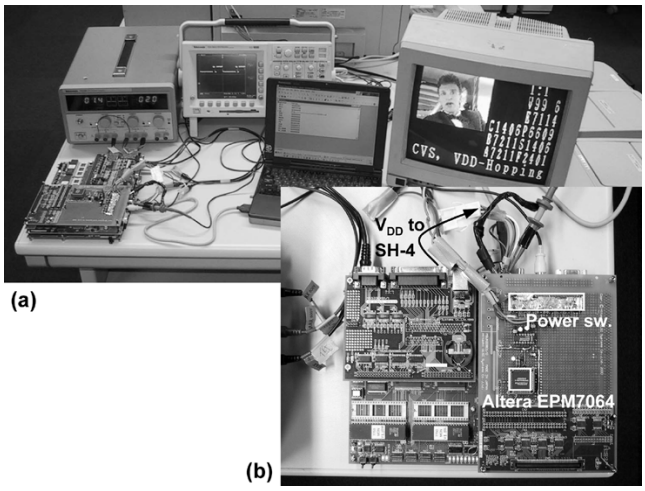


Fig. 9. (a) Snapshot of CVS experimental system. An output image of MPEG-4 codec is displayed on a monitor. (b) $\mathrm{V_{DD}}$ supply board on SH-4 embedded system board.

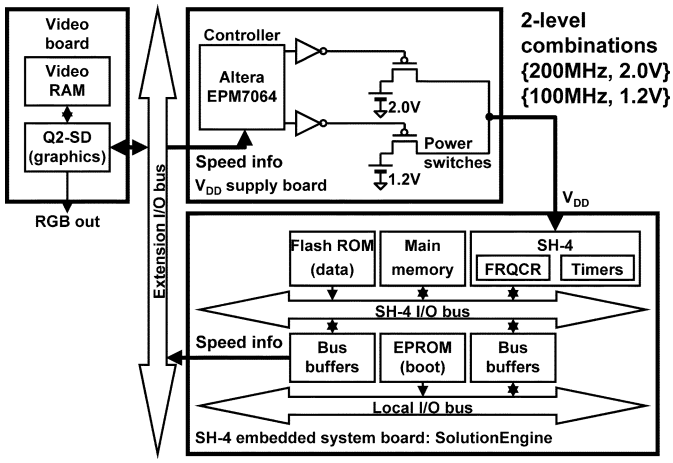


Fig. 10. Block diagram of target platform.

In the CVS, $\mathrm{V_{DD}}$ must be changed according to f. The speed information is sent to the $\mathrm{V_{DD}}$ supply board through an extension I/O bus by a system call. By using this speed information, $\mathrm{V_{DD}}$ is selected out of 2.0 V for 200 MHz or 1.2 V for 100 MHz by power switches on the $\mathrm{V_{DD}}$ supply board. The relationship between f and $\mathrm{V_{DD}}$ is obtained by measuring physical characteristics of the processor.

The measured falling and rising times for the $\mathrm{V_{DD}}$ transition are less than 200 μs and 100 μs respectively with a decoupling capacitor of 30 μF as shown in Fig. 11. In order to avoid malfunction, the processor stays in the sleep mode during the $\mathrm{V_{DD}}$ transition. This is realized by using a timer in the processor, which is different from the system clock timer. Before the $\mathrm{V_{DD}}$ transition, 200 μs is set to expire at the end of the $\mathrm{V_{DD}}$ transition for both the falling and rising cases and then, the processor moves to the sleep mode. The $\mathrm{V_{DD}}$ transition timer wakes up the processor with an interrupt when the preset time expires. All interrupts must be masked to eliminate malfunction during the $\mathrm{V_{DD}}$ transition except for the $\mathrm{V_{DD}}$ transition timer. This means that the interrupt level of the $\mathrm{V_{DD}}$ transition timer should be highest. Since the $\mathrm{V_{DD}}$ transition time is relatively long, the CVS is not suitable for fast-response systems like servo systems.

In the calculation of the timing information, the $\mathrm{V_{DD}}$ transition time, T_{tr} is set to 1 ms instead of 200 μs since the resolution of the system clock recognized in μITRON-LP is as coarse as 1 ms. It should be noted that T_{tr} must be smaller than the system clock resolution to preserve accuracy of the system clock. Alternatively, interrupts from the system clock timer are not properly acknowledged because the interrupt level of the system clock timer is lower than that of the $\mathrm{V_{DD}}$ transition timer.

The power characteristics of SH-4 are shown in Fig. 12. The power at 200 MHz is 0.8 W while the power at 100 MHz is 0.16 W. This means that the energy at 100 MHz is 2.5 times as efficient as the energy at 200 MHz. The sleep mode is operated at 100 MHz and 1.2 V in order to suppress standby power. The power in the sleep mode is 0.07 W. In the original μITRON, a NOP loop is carried out in place of the sleep mode when there is nothing to do. In the NOP loop, the processor consumes 0.58 W.

By using Fig. 12, we can obtain ideal CVS behavior and power characteristics as shown in Fig. 13. The power consumption of the original μITRON falls on Line A in the right graph. Line B shows the case where the processor can enter the sleep mode if there is no task to execute. In the sleep mode, the processor is clock-gated and completely cut off dynamic power. Unfortunately, the original μITRON does not support the sleep mode. This is because next initiation time of a real-time application cannot be generally predicted and a sleep mode is dependent on hardware. If the CVS works ideally as shown in the left graph, the power dependence on workload becomes Line C. The power of the CVS theoretically lies somewhere in Region S between Lines B and C.

IV. Experimental Results

In order to demonstrate the feasibility of the CVS, we construct a task set that consists of KEYBOARD routine, MPEG-4 codec, and 4096-points fast Fourier transform (FFT). H.263 standard image sequence "carphone" is used as MPEG-4 input data. Table I shows characteristics of each slice in the applications. The applications are sliced into the number of the functional blocks to be able to add the code fragments.

Fig. 14 shows the measured waveforms of $\mathrm{V_{DD}}$ and a sleep signal of the processor. There are five falling and five rising $\mathrm{V_{DD}}$ transitions. Thus, the overhead of the transition is just 2 ms during 360 ms.

It should be noted that 2.0 V is used only 14% of the total time while the sleep takes 38% of the time. This means that the remaining 48% of the time is used for the low-power operation at 1.2 V. This gives us the average workload of 38% ($14\% \times 1 + 48\% \times 0.5 + 38\% \times 0$).

The behavior of the measured waveform can be explained as follows with the help of Fig. 15. The absolute time is used for simplicity.

1) At the beginning, the KEYBOARD routine is dispatched. The virtual deadline, D_v is set to 0 because MPEG-4 and FFT are also in the READY queue waiting for running. Therefore, KEYBOARD should complete its execution in

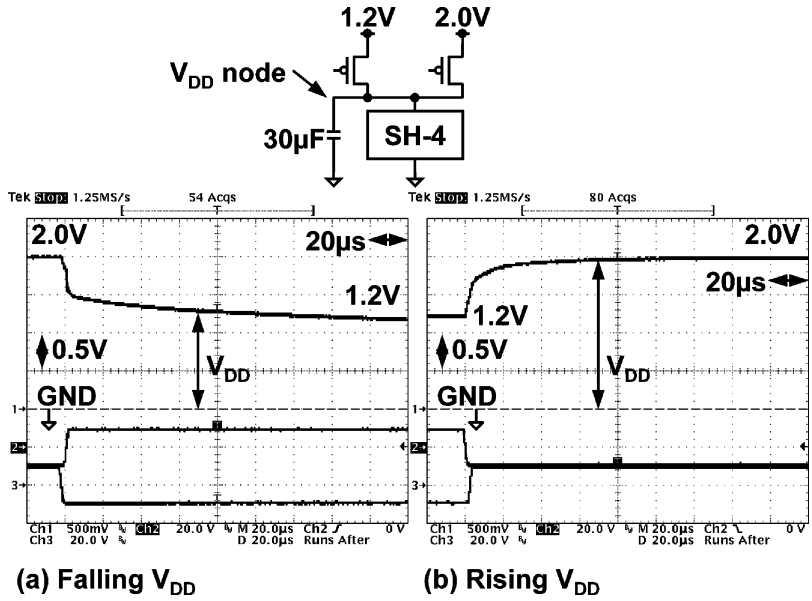


Fig. 11. Measured waveforms of (a) falling $\mathrm{V_{DD}}$ and (b) rising $\mathrm{V_{DD}}$. In the case of the falling $\mathrm{V_{DD}}$, we should decrease f first and then, decrease $\mathrm{V_{DD}}$. On the other hand, in the case of the rising $\mathrm{V_{DD}}$, we increase $\mathrm{V_{DD}}$ first and then, increase f.

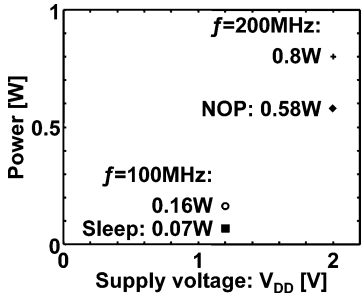


Fig. 12. Power characteristics of SH-4.

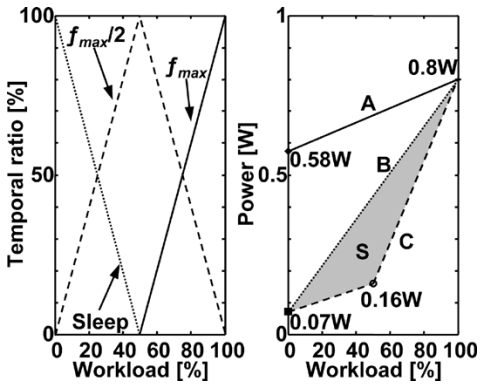


Fig. 13. Ideal CVS behavior and power characteristics. The left graph shows temporal ratio in the ideal case when T_{tr} is 0 and N is infinite. In the ideal case, at 0% workload, 100% sleep. At 50% workload, 100% $f_{\max}/2$ operation. At 100% workload, 100% $f_{\max}$ operation.

TABLE I
CHARACTERISTICS OF SLICES

Applications	# slices	WCET	Function
KEYBOARD	1	2ms	Polling
MPEG4	1	1ms	Initialization
	20	64ms	Macroblock calculation
	1	14ms	Display
FFT	1	2ms	Bit-reversal
	1	33ms	Danielson-Lanczos

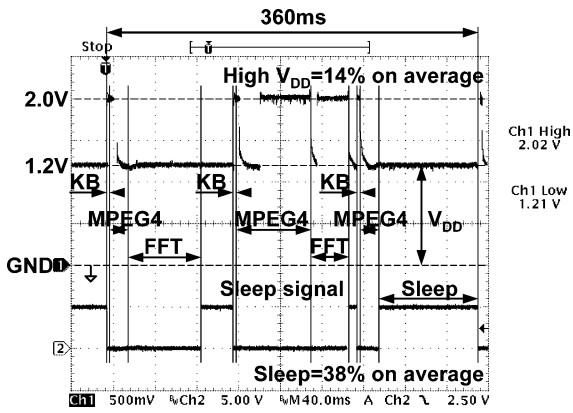
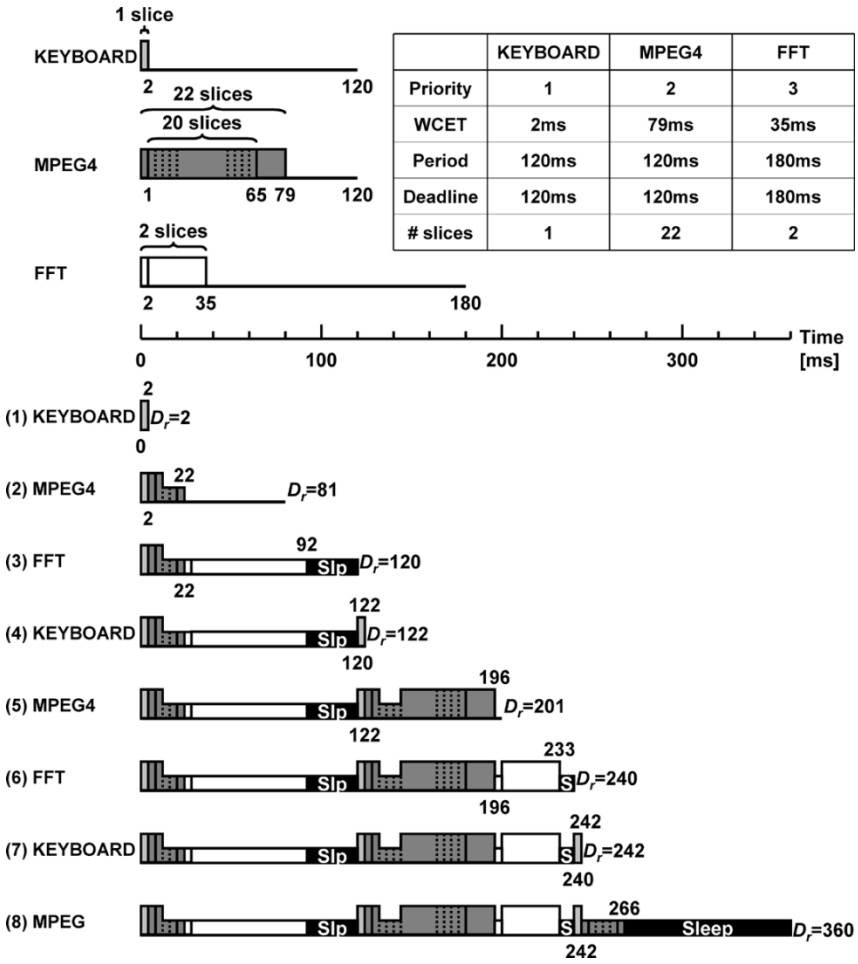


Fig. 14. Measured waveforms of $\mathrm{V_{DD}}$ and sleep signal. KB indicates the KEYBOARD routine. When the sleep signal is high, the processor is in the sleep mode.

its WCET of 2 ms, which means the real deadline, D_r. KEYBOARD finishes at 2 ms since KEYBOARD does not have data dependency and its execution time is always fixed.

2) At 2 ms, MPEG-4 is executed. D_v is also set to 0 because FFT is still in the READY queue. Then, D_r becomes 81 ms because the WCET of MPEG-4 is 79 ms. In this task, since the workload is much lighter than the worst case, some slices are executed at 200 MHz and the remaining slices are done at 100 MHz. Eventually, MPEG-4 ends at 22 ms.

3) At 22 ms, FFT occupies the processor. Because only FFT requires the processor, D_v and D_r are set to 120 ms that is equal to T_n of both KEYBOARD and MPEG-4. Thus, 98 ms is allowed to execute FFT whose WCET is 35 ms.



	KEYBOARD	MPEG4	FFT
Priority	1	2	3
WCET	2ms	79ms	35ms
Period	120ms	120ms	180ms
Deadline	120ms	120ms	180ms
# slices	1	22	2

Fig. 15. Explanation of Fig. 14. Height of slices shows the magnitude of f and $\mathrm{V_{DD}}$. Contrast with the $\mathrm{V_{DD}}$ waveform in Fig. 14.

This means that both slices of FFT can be executed at half speed. At 92 ms, upon the completion, the processor goes to the sleep mode and then, sleeps until 120 ms because there is nothing to execute. The sleep mode is carried out at 100 MHz and 1.2 V to save power as described in Section III.

4) At 120 ms, the second instance of KEYBOARD is dispatched.
5) At 122 ms, MPEG-4 is executed again with D_v of 180 ms, which is T_n of FFT. Since the time interval to D_v (58 ms $=$ 180 ms $-$ 122 ms) is less than the WCET of MPEG-4, the advantage of the virtual deadline cannot be exploited. In this case, D_r is set to the WCET, which is 201 ms. Here, unlike the first instance, data is close to the worst case and most slices are executed at the high speed of 200 MHz. Then, the last slice completes its execution at 196 ms.
6) Next, the second FFT waiting for execution takes over. The remaining instances can be understood similarly.

Fig. 16 shows the comparison of the average power among the CVS and other cases including the original μITRON. In the original μITRON, the processor executes NOPs for the idle time and consumes 0.66 W while the CVS is measured to consume 0.22 W when the workload is 38%. If the original μITRON supported the sleep mode, the power consumption would be estimated at 0.35 W. Unfortunately, I/O buffers of SH-4 do not work below 1.2 V. If the I/O buffers were designed carefully, operation below 0.9 V could be achieved instead of 1.2 V. In that case, the power of the CVS would become 0.17 W and could be reduced to about a quarter of the original μITRON case. Line C' in the right graph corresponds to such

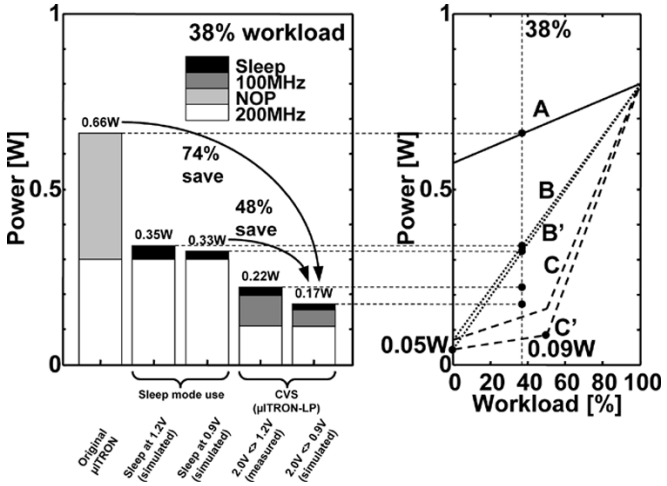


Fig. 16. Power comparison. Lines A, B, and C in the right graph are the same ones in Fig. 13.

case where the power at 100 MHz is 0.09 W and 0.05 W in the sleep mode. The power characteristic is improved compared to 1.2 V case particularly in a low workload region. Likewise, even compared to the case where the original μITRON uses the sleep mode at 0.9 V that corresponds to Line B', the CVS still saves about a half power.

In reality, power saving with the CVS depends on combination of task periods, which in turn determines how much we can benefit from virtual deadline. It is also dependent on distribution of execution time. Nevertheless, the CVS efficiently exploits slack time between tasks and data-dependent variations of multimedia applications and for this reason, we can expect power saving with the CVS.

V. Summary

In this paper, we have introduced the CVS involving design of a power-conscious RTOS, a run-time mechanism of applications with the concept of application slicing, and development of supporting hardware including an off-the-shelf processor. The CVS achieves power saving by exploiting slack time arising from variation of execution time of tasks and interaction among the tasks.

The experimental results have verified that μITRON-LP which is a prototype realizing the concept of the CVS achieves 74% power saving across multi-task environment compared to the original μITRON when workload is 38%.

Acknowledgment

The authors would like to thank K. Aisaka, K. Toyama, Dr. K. Ishibashi, and Dr. K. Uchiyama of Hitachi for fruitful discussion and H. Yamaki of Hitachi Yonezawa Electronics for tests and helpful advice.

References

[1] T. Okuma, H. Yasuura, and T. Ishihara, "Software energy reduction techniques for variable-voltage processors," *IEEE Design Test Comput.*, vol. 18, no. 2, pp. 31–41, Mar.-Apr. 2001.

[2] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automation Conf.*, 1999, pp. 134–139.

[3] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. USENIX Symp. on Operating Systems Design and Implementation*, 1994, pp. 13–23.

[4] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Foundations of Computer Science*, 1995, pp. 374–382.

[5] C. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, 1997, pp. 28–32.

[6] Y. Lee and C. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems," in *Proc. Int. Conf. on Real-Time Computing Systems and Applications*, 1999, pp. 272–279.

[7] D. R. Ditzel, "Transmeta's Crusoe: A low-power x86-compatible microprocessor built with software," in *Proc. Int. Symp. on Low-Power and High-Speed Chips (Cool Chips)*, 2000, pp. 1–30.

[8] Transmeta's Crusoe Web Site [Online]. Available: http://www.transmeta.com/technology/

[9] Y. Shin, H. Kawaguchi, and T. Sakurai, "Cooperative voltage scaling (CVS) between OS and applications for low-power real-time systems," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2001, pp. 553–556.

[10] Hitachi HI Series OS Web Site [Online]. Available: http://www.renesas.com/eng/products/mpumcu/tool/realtime_os/itron/

[11] TRON Project Web Site [Online]. Available: http://www.tron.org/index-e.html

[12] H. Kawaguchi, G. Zhang, S. Lee, Y. Shin, and T. Sakurai, "A controller LSI for realizing V_{DD}-hopping scheme with off-the-shelf processor and its application to MPEG4 system," *IEICE Trans. Electron.*, vol. E85-C, no. 2, pp. 263–271, 2002.

[13] S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Kim, "An accurate worst case timing analysis for RISC processors," in *Proc. IEEE Real-Time Systems Symp.*, 1994, pp. 97–108.

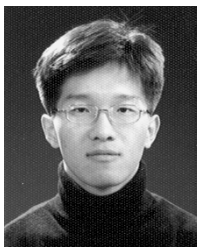
[14] S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," in *Proc. Asia and South Pacific Design Automation Conf.*, 2000, pp. 381–386.

[15] ——, "Run-time voltage hopping for low-power real-time systems," in *Proc. Design Automation Conf.*, 2000, pp. 806–809.

Hiroshi Kawaguchi (M'98) was born in Kobe, Japan, in 1968. He received the B.S. and M.S. degrees in electronic engineering from Chiba University, Japan, in 1991 and 1993, respectively.

He joined Konami Corporation, Japan, in 1993, where he developed arcade entertainment systems. He moved to the Institute of Industrial Science, University of Tokyo, Japan, in 1996 as a Technical Associate, and is currently a Research Associate. His research interests include low-voltage VLSI designs, low-power hardware systems, and wireless circuits.

Mr. Kawaguchi is a member of the ACM.



Youngsoo Shin (M'00) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Korea, in 1994, 1996, and 2000, respectively.

He is currently an Assistant Professor in the Department of Electrical Engineering at Korea Advanced Institute of Science and Technology (KAIST), Daejeon. Before joining KAIST in July 2004, he was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, from August 2001, as a Research Staff Member. Prior to joining IBM, he worked at the University of Tokyo, Japan, as a Research Associate. His research interests are VLSI design methodology and CAD, especially in the field of low-power and system-level design. He has published more than 30 papers in international journals and conferences.

Dr. Shin has served as a member of Technical Program Committees of ISLPED, ICCAD, and ASPDAC.

Takayasu Sakurai (S'77–M'78–SM'01–F'03) received the Ph.D. degree in electronics engineering from the University of Tokyo, Japan, in 1981.

In 1981, he joined Toshiba Corporation, where he designed CMOS DRAM, SRAM, RISC processors, DSPs, and SoC Solutions. He has worked extensively on interconnect delay and capacitance modeling known as Sakurai model and alpha power-law MOS model. From 1988 through 1990, he was a Visiting Researcher at the University of California, Berkeley, where he conducted research in the field of VLSI CAD. Since 1996, he has been a Professor at the University of Tokyo, working on low-power high-speed VLSI, memory design, interconnects, and wireless systems. He has published more than 250 technical papers including more than 50 invited papers and several books and holds 50 patents.

Dr. Sakurai was a conference chair and/or a technical program committee chair for the IEEE Symposium on VLSI Circuits, IEEE ICICDT, IEEE A-SSCC and a technical program committee member for ISSCC, CICC, DAC, ICCAD, FPGA workshop, ISLPED, ASPDAC, TAU, and other international conferences. He is a keynote speaker for the 2003 ISSCC. He is an an elected Administration Committee member for the IEEE Solid-State Circuits Society and an IEEE CAS Distinguished Lecturer.