Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers

Chaeseok Im
School of Electrical Engineering and Computer Science,
Seoul National University, Seoul Korea
82-2-880-7292
csim@iris.snu.ac.kr

Huiseok Kim
School of Electrical Engineering and Computer Science,
Seoul National University, Seoul Korea
82-2-880-7292
netti@iris.snu.ac.kr

Soonhoi Ha
Scholl of Electrical Engineering and Computer Science,
Seoul National University, Seoul Korea
82-2-880-7292
sha@iris.snu.ac.kr

ABSTRACT

As multimedia applications are used increasingly in many embedded systems, power efficient design for the applications becomes more important than ever. This paper proposes a simple dynamic voltage scheduling technique, which suits the multimedia applications well. The proposed technique fully utilizes the idle intervals with buffers in a variable speed processor. The main theme of this paper is to determine the minimum buffer size to achieve the maximum energy saving in three cases: single-task, multiple subtasks, and multi-task. Experimental results show that the proposed technique is expected to obtain significant power reduction for several real-world multimedia applications.

1. INTRODUCTION

Recently, multimedia applications are being used increasingly in many embedded and portable systems, such as mobile phones and PDAs, which require low power consumption within throughput constraints. Therefore, power reduction methods optimized for the applications are becoming more important than ever. In general, multimedia applications put less emphasis on latency than on the constant rate of output production or input consumption. They stabilize fluctuating inter-arrival time of input frames by buffering several frames beforehand for constant rate of input consumption. Similarly, output buffering is also used to cope with fluctuating processing time for constant rate of output production. In short, latency can be tolerated to some extent in multimedia applications.

One of the most promising methods to reduce power consumption is to use a *variable speed processor* (VSP), which can change its speed by varying the clock frequency along with the supply voltage while not degrading the required performance [1]. Reducing power consumption of a VSP is achieved by exploiting idle intervals of the processor [1]. We identify two kinds of idle intervals, and denote them as *worst-case slack time* (WST) and *workload-variation slack time* (VST) as in [2]. WST occurs because the utilization of a processor is usually less than 100% even if all tasks run at their worst-case execution time, and is extracted from scheduling results before task execution. On the other hand, VST comes from run-time variation of execution time due to data-dependent computation, over-estimation of worst-case execution time, and so on. It can be known only after being executed actually. Our focus is on the fact that the more we utilize the slack time, the more energy saving can be achieved. In this paper, we denote *voltage scheduling* (VS) as a method that assigns a supply voltage for each task of a processor to utilize the slack time.

VS techniques for real-time applications, in either static or dynamic manners, were proposed in [3, 4, 5]. The limitation of their approaches is that they do not consider the VST. Thus, the full potential of energy saving can not be obtained when variation of execution time exists. This is overcome in [1, 2, 6, 7]. Besides WST, VST is also exploited in their approaches. However, the slack time still could not be fully achieved in [1, 2, 6], because they can not avoid idle intervals when no task is ready to run. Or, they need intra-task analysis to be performed by a programmer [2], or by a compiler [7]. In addition, intra-task VS techniques usually incur much more frequent voltage switches compared with inter-task, or task-by-task, VS techniques.

In this paper, we propose a simple but novel task-by-task VS technique suitable for multimedia applications. We exploit the slack time fully by buffering multiple input data or output results to let there be always at least one runnable task on the processor. The idea to use buffers in power reduction was already proposed in [8, 9]. They use buffers to average workload among input samples. However, unlike ours, one of their approaches may result in buffer overflow [8], and the other adopts a critical assumption that exact workloads of all input samples can be known a priori [9].

The main theme of the proposed technique is to estimate the minimum buffer size to achieve the maximum energy saving. As mentioned above, the target application of the proposed VS technique is soft real-time ones (including multimedia

This work is sponsored by Brain Korea 21 project.

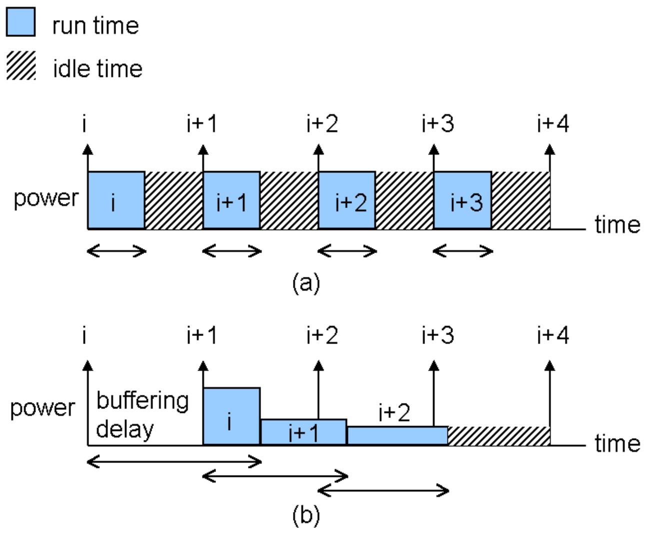

Figure 1. A motivational example. (a) VS technique without input buffering (b) VS technique with input buffering

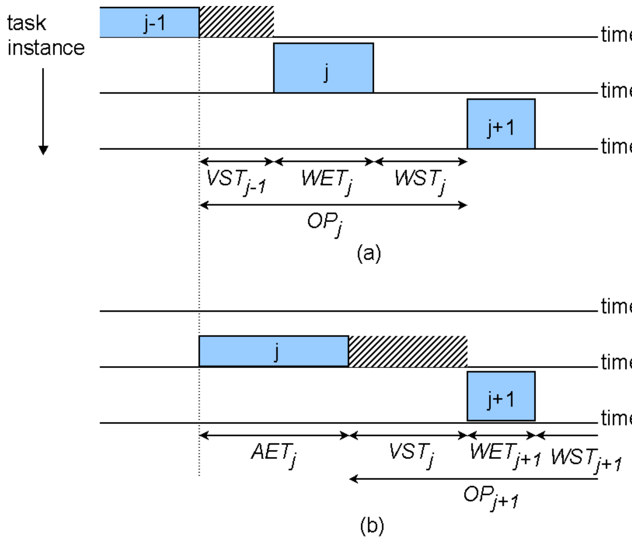

Figure 2. Proposed VS technique.

applications), in which latency can be tolerated to some extent. As a result, significant energy saving can be achieved. Even if there are some latency constraints, the proposed technique is still applicable, but the amount of energy saving may be reduced because the slack time may not be fully used up due to the constraints.

The remainder of the paper is organized as follows: In the next section, we discuss the motivation of our VS technique, and introduce the algorithm with a simple example. In section 3, we estimate the minimum buffer size to achieve the maximum energy saving in three cases: single-task, multiple subtasks and multi-task. In section 4, we prove the effectiveness of the proposed technique with several real-world multimedia applications, and draw conclusions in section 5.

2. PROPOSED METHOD

2.1 Motivation

Consider a periodic task of which the deadline is equal to the period and the actual execution time is half of the worst-case. If we assume that the worst-case execution time is equal to the period, there is no worst-case slack time in this example. A typical inter-task VS technique without input buffering can not exploit the VST fully, because a new instance of the task can not be scheduled before the release point of the task. Thus, the processor is in idle state during half of the time, as shown in Figure 1(a). On the other hand, we can schedule the next instance of the task to exploit the slack time fully by buffering one input packet beforehand, as shown in Figure 1(b). Even though the processor in idle state may go into the power-down mode in the former case, there is more energy saving in the latter case. As depicted by arrows in Figure 1, buffering increases latency. So, the buffering scheme can be used as long as buffering delay is acceptable.

2.2 Algorithm

The proposed technique is not restricted to the single-task environment, but also extensible to the multi-task environment. Even though the proposed technique is applicable to both input buffering and output buffering case, our presentation is based on input buffering case for simplicity. To sketch the algorithm, we first define the following notations (Figure 2).

- s_j : the j^{th} scheduled instance of a given task set ($j \geq 1$),
- WST_j : *worst-case slack time* of s_j,
- VST_j : *workload-variation slack time* of s_j,
- WET_j : *worst-case execution time* of s_j at full supply voltage,
- BET_j : *best-case execution time* of s_j at full supply voltage,
- AET_j : *actual execution time* of s_j,
- OP_j : *occupation period* of s_j, which is the maximum duration that s_j can run without violating the timing constraints. OP_j is the sum of VST_{j-1}, WET_j, and WST_j as shown in Figure 2.

We assume that, for all j, WET_j and BET_j are known or approximately estimated. We also assume that the task set is already scheduled with a fixed-priority scheduling algorithm. From the schedule, we obtain s_j, WST_j and WET_j, as shown in Figure 2. Now the proposed technique runs as in Figure 3.

```
1.  Initially, buffer input packets up to the size as estimated in section 3, and set $VST_0 = 0$.
2.  For $j \geq 1$,
    2a.  $OP_j = VST_{j-1}+WET_j+WST_j$.
    2b.  Execute $s_j$ with the reduced voltage to utilize the $OP_j$ fully.
    2c.  $VST_j = OP_j - AET_j$.
3.  Repeat 2.
```

Figure 3. Algorithm of the proposed VS technique.

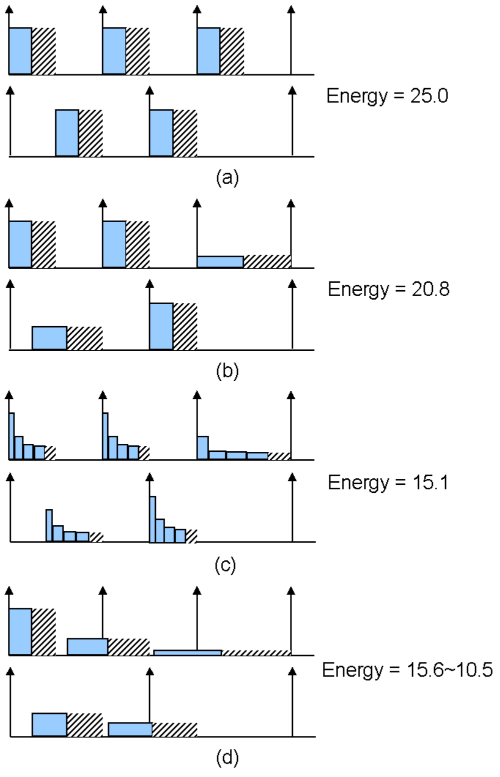

Figure 4. Comparison of VS techniques. (a) no VS technique with power-down (b) inter-task VS technique (c) intra-task VS technique (4 timeslots) (d) VS technique with input buffering

2.3 Comparison

Consider two periodic tasks $\tau_1(20,10,5)$ and $\tau_2(30,10,5)$ as shown in Figure 4, where $\tau_i(T_i,WET_i,AET_i)$ is characterized by period T_i, worst-case execution time WET_i, and actual execution time AET_i. We assume that the deadlines of tasks are equal to their periods. The tasks are to be scheduled on a VSP. Four types of voltage scheduling techniques are evaluated, and the corresponding energy consumption is calculated. Power is normalized to 1 at full speed of the processor, and power-delay model is assumed to be $Power \propto 1/Delay^2$ for simple comparison. Figure 4(a) shows the most energy dissipation when no VS technique is applied, that is 25.0. If we exploit the VST on a task basis as in Figure 4(b), and on a timeslot basis as in Figure 4(c), energy dissipation decreases to 20.8 and 15.1, respectively.

The proposed method, shown in Figure 4(d), consumes energy about 15.6, nearly equal to the result of the intra-task VS technique. However, when we proceed the task execution one more *hyper period* (least common multiple of tasks' periods) again, energy dissipation of the proposed method decreases to 10.5 within a period, while other approaches maintain the same amount of energy dissipation. The reason is that the proposed method can exploit the slack time over the period bound of tasks. In summary, the proposed technique achieves about 30 to 58% of energy saving in this simple example, compared with other approaches in the long run. Even though the amount of energy saving is dependent on workload variation, we expect noticeable energy reduction by adding input buffers within latency constraints.

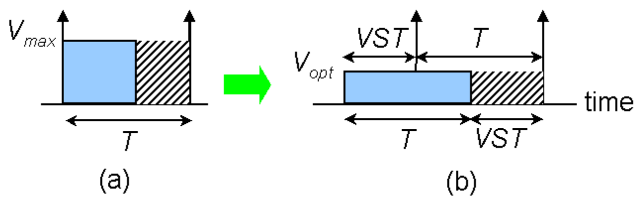

Figure 5. Worst case scenario for the maximum VST.

3. BUFFER SIZE ESTIMATION

In order to utilize the VST fully, the input buffers should be large enough to make tasks runnable at all times. In this section, we estimate the minimum buffer size to achieve the maximum energy saving ignoring the latency constraints. In section 4, we investigate the effect of latency constraints on energy saving.

To make the problem more tractable, we make some assumptions, which are not inherent to the proposed technique, thus can be removed later.

- All tasks are periodic.
- The deadline of a task is equal to the period of the task.

Suppose there are n periodic tasks $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled on a single VSP. Associated with each task τ_i, there are three parameters, *deadline* (T_i), *worst-case execution time* (WET_i), *best-case execution time* (BET_i), and which are computed at the full supply voltage. In this paper, we solve the problem in three cases separately: 1) *single-task*, 2) *single-task that consists of multiple subtasks with different parameters* and 3) *multi-task.*

3.1 Single-Task Case (n=1)

The basic observation is that the maximum *VST* is obtained when tasks are always executed at their best-case execution cycles and the supply voltage is maximally reduced to fully exploit the *VST*. Then, the length of the *VST* is dependent upon the ratio between *WET* and *BET*[1]. The steady state of the worst-case scenario is depicted in Figure 5(b). The current execution remains the same amount of the VST as the previous execution. Therefore, $\frac{BET}{WET} = \frac{T}{T+VST}$, which is reduced to

$$VST = T\frac{WET}{BET} - T = T\left(\frac{WET}{BET} - 1\right) \tag{1}$$

where T is the period of the task. For the task to exploit the *VST* fully, the input data should be ready earlier than the scheduled time by the VST. Therefore, the buffer size h should satisfies the following inequality:

$$h \ge \left\lceil \frac{VST}{T} \right\rceil = \left\lceil \frac{WET}{BET} - 1 \right\rceil . \tag{2}$$

[1] Since *execution time = execution cycles / clock frequency*, the ratio between the worst-case and the best-case execution cycles is equivalent to the ratio between *WET* and *BET.*

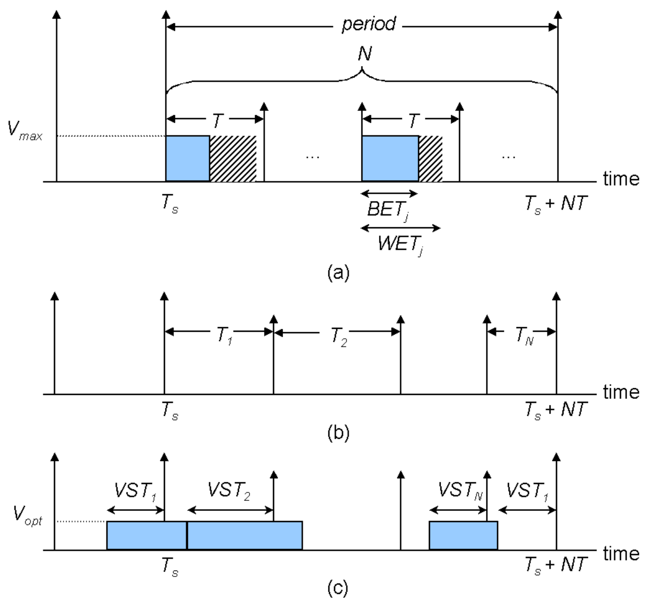

Figure 6. Multiple subtask case.

Thus, the minimum buffer size h_{min} becomes $\left\lceil \frac{WET}{BET} - 1 \right\rceil$, which is dependent only on the ratio r, $r = \frac{WET}{BET}$.

Example 1: Consider a task $\tau(20,10,3)$. Then we have $h_{\min} = \left\lceil \frac{10}{3} - 1 \right\rceil = 3$.

3.2 Multiple-Subtask Case

In case of a MPEG video application, the task shows different workload distribution depending on the frame type (I, B and P). Also, the periodic behavior is defined in the unit of the GOP (group of picture) that consists of a fixed sequence of input frames: for example four input frames {I, P, B, B} may define a GOP. Figure 6(a) shows a general task schedule whose period contains N subtasks $\{s_j\}$ that have the same deadline T, but different parameters WET_j, and BET_j. Note that there may be *WST*s after a subtask since the deadline is chosen as the maximum of *WET*s of the subtasks. If we ignore such workload variation between frame types, we can apply the similar formula of the previous section as follows:

$$VST = \max\{WET\}\left(\frac{\max\{WET\}}{\min\{BET\}} - 1\right) \tag{3}$$

where maximum is taken among frame types.

There is a better approach to tackle this problem. We regard a different workload distribution as a separate subtask. For example, the MPEG video decoder task consists of a fixed sequence of three different subtasks. The *VST* will increase as the actual execution time becomes shorter than the deadline. Then, we decrease the supply voltage to reduce the *VST*. Thus, the maximum energy saving is expected when all subtasks take their best-case execution times and the supply voltage is reduced maximally and remains uniform during the task period as shown in Figure 6(c). To achieve this behavior we have to reassign the deadline of each subtask, T_j, somehow considering the *BET*s and *WET*s of the subtasks within the interval *NT* (Figure 6(b)). Then the T_j and the VST_j for each subtask becomes as follows:

$$T_j = \frac{NT}{\sum BET}\left(WET_j - WET_{j-1} + BET_{j-1}\right) \tag{4}$$

$$VST_j = \frac{NT}{\sum BET}\left(WET_{j-1} - BET_{j-1}\right). \tag{5}$$

Proof of equations (4) and (5): Let s_j denote the j^{th} subtask in *the sequence of subtasks* (*SOS*) within a period. Due to the periodic behavior, Tj is reassigned within time interval NT.

$$\sum T_j = NT. \tag{6}$$

Also we have

$$\gamma\left(\sum BET_j\right) = NT \tag{7}$$

where γ is the ratio of execution times between when tasks are executed at the minimum supply voltage and at the maximum voltage. And for the individual s_j we have

$$\frac{VST_j + T_j}{WET_j} = \gamma, \text{ for } 1 \le j \le N, \tag{8}$$

because the maximum energy saving is achieved when the supply voltage is uniform for all s_j. Finally, we have

$$VST_j + T_j = VST_{j+1} + \gamma BET_j, \text{ for } 1 \le j \le N-1. \tag{9}$$

VST_{j+1} is the workload variation slack time for the next subtask after the current subtask s_j tasks γBET_j during the occupation period (VST_j+T_j). Now there are 2*N*+1 unknown variables and the same number of equations. From equation (8) and (9), $VST_{j+1} = \gamma\left(WET_j - BET_j\right)$. Since $\gamma = \frac{NT}{\sum BET_j}$ from (6), we obtain equation (5). From equation (8) and (5), we also obtain equation (4). QED.

Compared with equation (3), equation (5) gives a shorter *VST* value. From the VST_j value, we can obtain the minimum buffer size h_{min}. We count the number of buffers needed within each VST_j according to the given subtask sequence, which we denote as h_j. Then h_{min} becomes $\max\{h_j\}$.

Example 2: Consider a task $\tau(10, 10, 4)$ with two subtasks $s_A(10, 10, 9)$ and $s_B(10, 5, 4)$ and $SOS = \{s_A, s_B, s_B\}$. From equation (3), the minimum buffer size becomes 2, while it becomes 1 from equation (5).

Actually, it is difficult to know beforehand the exact deadline assignment that gives best energy saving for the actual workload. Therefore, there could be other deadline assignment methods which are more energy-efficient for some workload patterns. Another deadline assignment policy we used in the experiments is to use average workload. The deadlines can be obtained when *BET*s in equation (4) are replaced by average execution times. It gives better result in our experiments.

3.3 Multi-Task Case

Multi-task case is similar to the multiple-subtask case except that each task has a separate deadline and needs a separate buffer. Also the schedule period of this multi-task scenario becomes the hyper period of the given task set. Therefore, we rewrite the equations of the previous section as follows:

$$T_j = \frac{H}{\sum BET}\left(WET_j - WET_{j-1} + BET_{j-1}\right) \tag{10}$$

$$VST_j = \frac{H}{\sum BET}\left(WET_{j-1} - BET_{j-1}\right) \tag{11}$$

where H is the hyper period of the given task set. Also the minimum buffer size of τ_i, $h_{min(i)}$, can be obtained in the manner similar to the multiple-subtask case. That is,

$$h_{\min(i)} = \max_{s_j \in \tau_i} \{h_j\} \tag{12}$$

where h_j is the size of buffers needed within each VST_j according to the given multi-task schedule.

Example 3: Consider two tasks $\tau_1(20, 10, 7)$ and $\tau_2(30, 8, 3)$, and the schedule is $\{\tau_1, \tau_2, \tau_1, \tau_2, \tau_1\}$. From the equation (11) and (12), the minimum buffer sizes are calculated: $h_{min(1)} = 1$ and $h_{min(2)} = 1$.

Since the VST of a task is relatively shorter in multi-task case than in single-task, the buffer requirement for each task is usually small.

In this section, we made a pessimistic assumption that all tasks run at their *BET*s. However, if we use average-case execution time instead of the best-case, the average buffer size is estimated, and it has importance at a practical viewpoint.

4. EXPERIMENTAL RESULTS

Table 1 lists several real-world multimedia applications we use for experiments. For single-task experiments, all applications in Table 1 are used, and for multi-task experiments, a video phone application is used, which is composed of MPEG-4 video encoder/decoder and VSELP (Vector-Sum-Exited Leaner Prediction) voice encoder/decoder. All execution times are measured on an UltraSPARC-II CPU that operates at 450MHz. The *BET*s and the *WET*s are determined from the simulation results; supposing that the execution times of the applications follow a normal distribution, we take 3σ variations around the mean of the distribution as boundary values. In single-task experiments, we assign the period with the *WET*, and in multi-task experiments, we assign the periods as long just enough for the task set to be schedulable. As for the voltage scalable processor, we used the power-delay curve of ARM8 microprocessor core form the author [10, 11, 12]. The processor can operate from 80MHz with 3.4V down to 8MHz with 1.2V.

Figure 7 shows energy dissipation of various VS techniques: 1) no VS technique with power-down, 2) intra-task timeslot VS technique, 3) the proposed buffering technique, 4) the proposed technique with multiple subtasks (for both of best and average execution time assumptions). The results show the relative energy dissipation compared with no power management case, where the maximum voltage is supplied all the time. The proposed technique achieves the most energy saving up to 80% compared to the reference technique. Compared with the intra-task technique,

Table 1. Test application set.

	BET (ms)			*WET* (ms)		
	I	*P*	*B*	*I*	*P*	*B*
MPEG-2 video decoder	9.8	8.3	12.3	24.8	30.6	18.6
MPEG-4 video encoder	20.4	52.6		30.5	115.0	
MPEG-4 video decoder	4.6	1.0		9.7	9.1	
VSELP encoder	8.5			8.7		
VSELP decoder	2.1			2.2		

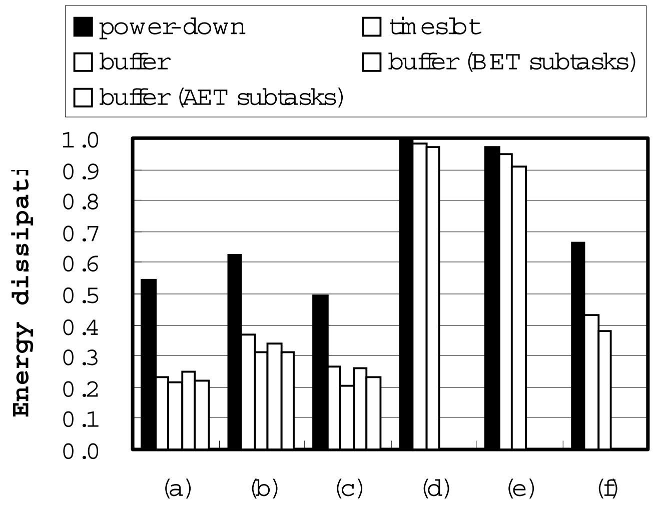

Figure 7. Comparison of relative energy dissipation among three VS technique. (a) MPEG-2 decoder (b) MPEG-4 encoder (c) MPEG-4 decoder (d) VSELP encoder (e) VSELP decoder (f) Video phone (supposing that b, c, d and e are executed on a VSP concurrently)

which is among the most efficient techniques in the previous researches, it achieves up to 14% performance improvement without intra-task analysis. However, the amount of energy saving varies according to the variation of the application execution time. In the intra-task technique, we make the timeslot size equal to 1 msec as assumed in [2]. In two multiple subtask approaches discussed in section 3.2, the case of average execution time assumption shows better performance than best execution time assumption.

Figure 8 shows the actual buffer sizes through simulation in addition to the estimated maximum buffer size. Up to 9 buffers are estimated depending on the applications, while actual buffer sizes are 1 to 4. Note that smaller buffer sizes, up to 5 buffers, are estimated considering the multiple subtasks separately in the case of MPEG video applications as discussed in section 3.2.

Figure 9 demonstrates the effect of latency constraints in energy saving. For the experiment, we use the MPEG-4 video decoder, which needs maximum 3 buffers as shown in Figure 8(c). The energy dissipation is compared to the power-down technique, which is Figure 9(d). As shown in Figure 9, the more latency

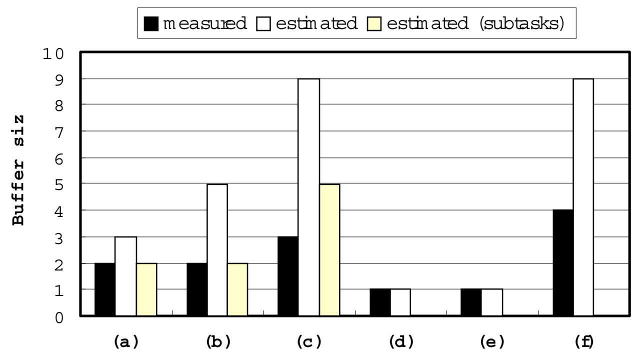

Figure 8. Buffer size for the application set of Figure 7.

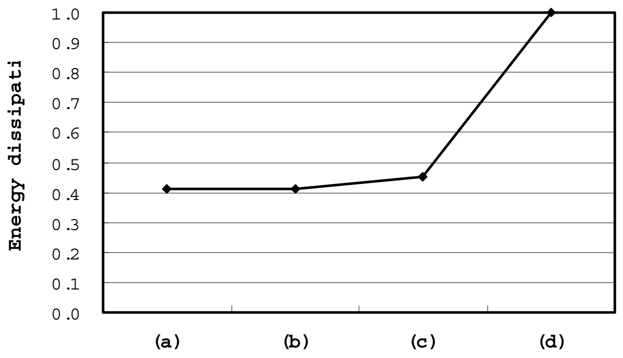

Figure 9. The effect of latency constraints in saving energy. (a) no constraints (extra buffer size = 3) (b) extra buffer size = 2 (c) extra buffer size = 1 (d) no input buffering (extra buffer size = 0)

constraints are forced, the less energy saving is achieved. Also it shows that most energy saving is achieved with the first extra buffer. It means that we can achieve much of energy saving with smaller number of buffers than pessimistically estimated although there are some latency constraints. It is noteworthy that real applications need extra buffers anyhow without any consideration of applying the proposed technique. These buffers are needed to stabilize the fluctuating input arrival times for networked decoding applications or the fluctuating output production times for real-time encoding applications. Considering these buffers, the extra buffers computed from our estimation is expected to be often negligible.

5. CONCLUSIONS

In this paper, we propose a novel VS technique suitable for multimedia applications. We exploit the slack time fully by buffering multiple input frames beforehand. The proposed technique is simple but effective as experimental results show. Even though buffering increases latency, if the latency can be tolerated as usual in multimedia applications, significant energy saving can be achieved. Also, we present the estimation methods for the minimum buffer size to achieve the maximum energy saving in several cases.

Further Research is needed to eliminate the assumption of knowing *BET* and *WET* beforehand. Also, the energy overhead of additional buffer memory needs to be investigated, and trade-off between saved memory due to buffering and consumed energy due to additional buffer itself needs to be explored. It is another future work to extend the proposed idea considering quality-of-service(*QoS*) constraints.

6. REFERENCES

[1] Y. Shin, K. Choi and T. Sakurai, “Power Optimization of Real-Time Embedded Systems on Variable Speed Processors,” *Proc. Int’l Conf. on Computer-Aided Design*, pp. 365-368, 2000.

[2] S. Lee and T. Sakurai, “Run-Time Voltage Hopping for Low-Power Real-Time Systems,” *Proc. Design Automation Conference*, pp. 806-809, 2000.

[3] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. B. Srivastava, “Power Optimization of Variable-Voltage Core-Based Systems,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no 12*, pp. 1702-1713, Dec. 1999.

[4] I. Hong, M. Potkonjak and M. B. Srivastava, “On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor,” *Proc. Int’l Conf. on Computer-Aided Design*, pp. 653-656, 1998.

[5] T. Okuma, T. Ishihara and H. Yasuura, “Real-Time Task Scheduling for a Variable Voltage Processor,” *Proc. 12th Int’l Symp. on Systems Synthesis*, pp. 24-29, 1999.

[6] Y. Shin and K. Choi, “Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems,” *Proc. Design Automation Conference*, pp. 134-139, 1999.

[7] D. Shin, J. Kim and S. Lee, “Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications,” *IEEE Design & Test of Computers, vol. 18, no. 2*, March-April 2001.

[8] V. Gutnik, “Variable Supply Voltage for Low Power DSP,” Master’s thesis, Massachusetts Institute of Technology, 1996.

[9] L. H. Chandrasena and M. J. Liebelt, “A Rate Selection Algorithm for Quantized Undithered Dynamic Supply Voltage Scaling,” *Proc. Int’l Symp. on Low Power Electronics and Design*, pp. 213-215, 2000.

[10] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen, “A Dynamic Voltage Scaled Microprocessor System”, *IEEE Int’l Solid-State Circuits Conference Digest of Technical Papers*, pp. 294-295, Feb. 2000.

[11] Thomas D. Burd and Robert W. Brodersen, “Design Issues for Dynamic Voltage Scaling”, *Proc. Int’l Symp. on Low Power Electronics and Design*, 2000.

[12] Thomas D. Burd and Robert W. Brodersen, “Processor Design for Portable Systems”, *Journal of VLSI Signal Processing*, Aug. 1996.