Run-Time Adaptive Workload Estimation for Dynamic Voltage Scaling

Sung-Yong Bang, Kwanhu Bang, *Student Member, IEEE*, Sungroh Yoon, *Member, IEEE*, and Eui-Young Chung, *Member, IEEE*

***Abstract*—Dynamic voltage scaling (DVS) is a popular energy-saving technique for real-time tasks. The effectiveness of DVS critically depends on the accuracy of workload estimation, since DVS exploits the slack or the difference between the deadline and execution time. Many existing DVS techniques are profile based and simply utilize the worst-case or average execution time without estimation. Several recent approaches recognize the importance of workload estimation and adopt statistical estimation techniques. However, these approaches still require extensive profiling to extract reliable workload statistics and furthermore cannot effectively handle time-varying workloads. Feedback-control-based adaptive algorithms have been proposed to handle such nonstationary workloads, but their results are often too sensitive to parameter selection. To overcome these limitations of existing approaches, we propose a novel workload estimation technique for DVS. This technique is based on the Kalman filter and can estimate the processing time of workloads in a robust and accurate manner by adaptively calibrating estimation error by feedback. We tested the proposed method with workloads of various characteristics extracted from eight MPEG video clips. To thoroughly evaluate the performance of our approach, we used both a cycle-accurate simulator and an XScale-based test board. Our simulation result demonstrates that the proposed technique outperforms the compared alternatives with respect to the ability to meet given timing and Quality of Service constraints. Furthermore, we found that the accuracy of our approach is almost comparable to the oracle accuracy achievable only by offline analysis. Experimental results indicate that using our approach can reduce energy consumption by 57.5% on average, only with negligible deadline miss ratio (DMR) around 6.1%. Moreover, the average of computational overheads for the proposed technique is just 0.3%, which is the minimum value compared to other methods. More importantly, the DMR of our method is bounded by 11.7% in the worst case, while those of other methods are twice or more than ours.**

***Index Terms*—Adaptive filter, dynamic voltage scaling (DVS), feedback control, workload estimation.**

Manuscript received November 19, 2008; revised February 6, 2009 and April 9, 2009. Current version published August 19, 2009. This work was supported in part by the IT R&D program of MKE/IITA 2009-S005-01 (Development of Configurable Devices and S/W environment), by the Korean Government (MEST) under Korea Research Foundation Grant KRF-2007-313-D00578 and Korea Science and Engineering Foundation (KOSEF) Grant 2009-0079888, by Yonsei University Institute of TMS Information Technology, which is a Brain 21 program, Korea, and by IDEC (IC Design Education Center). This paper was recommended by Associate Editor D. Atienza.

S.-Y. Bang was with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea. He is now with the China Global System For Mobile Communication R&D Group, Mobile Communication Division, Samsung Electronics, Suwon-City, Gyeonggi-do 443-742, Korea (e-mail: bsystar@dtl.yonsei.ac.kr).

K. Bang and E.-Y. Chung are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: khbang@dtl.yonsei.ac.kr; eychung@yonsei.ac.kr).

S. Yoon is with the School of Electrical Engineering, Korea University, Seoul 136-713, Korea (e-mail: sryoon@korea.ac.kr).

Digital Object Identifier 10.1109/TCAD.2009.2024706

I. Introduction

POWER consumption has become a critical design parameter as more transistors are integrated in a single chip, owing to the rapid advance of process technology. System-level power management techniques have received a large attention particularly from high-performance mobile devices to effectively reduce the power consumption while minimizing their performance degradation. Two representative system-level power management techniques are dynamic power management (DPM) and dynamic voltage scaling (DVS).

DPM is a design methodology aiming at reducing power consumption of electronic systems by performing selective shutdown of idle system resources [1]. Hence, the quality of DPM critically depends on the prediction accuracy of how long the system resources will stay in idle state. On the other hand, DVS aims at reducing energy consumption (EC) by scheduling voltage/frequency pairs for a task (or tasks) running on a computing unit to be completed in and as close as to a given deadline. One of the critical points in DVS is also the prediction accuracy of the task (or workload) completion time being executed.

Even though both methods seem to perform different predictions, their predictions are eventually in a dual relationship when the target application is periodic and has a real-time constraint, since the sum of the task execution time and idle time is the same as a period. In other words, both techniques need the knowledge of workloads *a priori* to maximally exploit the idle time. Hence, increasing the prediction accuracy is a very crucial task for both methods.

The prediction accuracy becomes more critical when the workload shows a large variation and/or nonstationary property. Many previous approaches dealt with such issues for DPM [2]–[7]. Surprisingly, only a few previous methods for DVS focused on the workload issue. Most previous DVS techniques adopt offline profiling to learn the average-case execution time (ACET) and/or worst case execution time (WCET). Such approaches cannot deal with the time-varying workloads. Recently, statistical DVS methods using cumulative distribution function (CDF) or probability density function (PDF) of workloads by offline profiling [8], [9] have been proposed. However, these methods are still limited to stationary workloads, since there is no mechanism to update the distribution functions corresponding to the workload variation. More details on workload estimation in DVS will be discussed in Section II.

In this paper, we propose a run-time workload estimation technique for DVS even when the workload is highly

nonstationary. More precisely, a DVS method equipped with the proposed estimator can learn the workload characteristics without requiring any offline profiling. For this purpose, we adopt the Kalman filter [10]–[12], which is one of the most popular adaptive filters. We choose the Kalman filter since it is known to be quickly adapted to the fast-varying observations with slow-varying parameters, thus providing a powerful and robust real-time tracking mechanism. To handle time-varying workloads more effectively, we further enhance the conventional Kalman filter by using time-varying process and measurement noise covariances. The effectiveness of our method is validated by applying it to a DVS method for MPEG-2 video decoder. Moreover, note that our method can be applicable to other periodic applications with soft real-time constraints in a similar manner without loss of generality.

The remainder of this paper consists of the following. In Section II, we summarize the previous work of DVS from the workload estimation perspective. In Section III, we present the introduction of the target multimedia application and adaptive filters as preliminaries. Section IV compares the conventional and proposed Kalman filters from a conceptual point of view. We then address the overall flows of our method and the details of the workload estimation technique in Section V. Finally, we show the experimental results in Section VI followed by a conclusion in Section VII.

II. Related Work

One of the earliest DVS methods was presented in [13], where they assumed that the arrival time and deadline of workloads are given as constants. The task execution time is also given as a constant in the unit of CPU cycle. They focused on the optimal voltage and frequency scheduling in a continuous domain under an assumption.

On the other hand, many of the task-level DVS techniques used offline profiling to obtain the ACET and/or WCET to characterize the workloads for the given application. In [14], they proposed a DVS technique using buffers to further reduce EC based on WCET workload model. The work in [15] addressed a DVS method considering task synchronization based on WCET workload model in multiprocessor environment. The work in [16] first proposed a DVS method for a task set which consists of both periodic and aperiodic tasks. They also used WCET workload model in their method. Another DVS method based on WCET workload model was presented in [17]. In [18], they proposed the best effort energy minimization and O^2ME (a hybrid offline/online minimization) which intentionally miss the decoding deadline of some frames in multimedia applications. Missing deadline causes the next frame to be dropped and the time allocated to the next frame can be utilized as idle time to save more EC, since a small number of dropped frames are tolerable to the human visual and auditory system. They also used WCET workload model for simplicity.

Similar to the task-level DVS techniques, most of the intratask level DVS techniques also used WCET and/or ACET workload models. In [19]–[22], they selected several control points inside the program code of a given task by profiling its control flow graph (CFG). In [19], the authors annotate the remaining worst-case execution cycle (RWEC) to each branching edge of the CFG by offline profiling. RWEC is defined as the number of cycles required to complete the task from the given branching point. Among these edges, they select several branching points called voltage scaling edge, which can maximally reduce EC when DVS technique is applied. In [21], they proposed a more aggressive DVS method based on ACET workload model by use of remaining average execution cycle (RAEC) instead of RWEC. Moreover, they proposed a method based on remaining optimal-case execution path, which resulted from the comparison of RWEC and RAEC in [20]. The method in [22] used WCET model to adaptively change the voltage scaling points.

Some other approaches aimed at tracking the variation of workload in heuristic manners, particularly for MPEG application which is a periodic soft real-time constrained application. In [23] and [24], the authors used a linear model of frame decoding time with respect to the frame size. The coefficients are obtained from offline profiling. They showed that the frame decoding time (task execution time) is linearly proportional to the frame size in streaming environment. Moreover, the authors in [25] used the similar concept for multiframes rather than a single frame to trade off the Quality of Service (QoS) and EC. The major drawback is that the estimation accuracy is largely affected by the workload characteristics used for training (profiling). To minimize such drawback of these approaches, the authors in [26] used a linear model whose coefficients are updated as time goes by based on a weighted mean (WM) algorithm. However, the method incurs large overhead to update the model coefficients. The linear model approach is further improved in [27] and [28] by partitioning a task into two parts—CPU- and memory-bounded parts. They used a linear model of the execution time of CPU-bounded part with respect to the frame size. A finer granularity version of the linear model can be found in [29], where they built a linear model in macro block level rather than frame level.

While the aforementioned techniques directly estimate the task execution time, some other methods proposed intratask level DVS techniques using probability distribution [8], [9] to estimate the workload in a statistical manner. In [8] and [9], they used CDF and PDF to represent the run-time distribution of a given task, respectively. Each program region has its own CDF or PDF and performs voltage/frequency scheduling in a statistical manner using the distribution functions. The distribution function can be obtained through either offline profiling or run-time monitoring. In the latter case, the monitoring overhead is not negligible. Moreover, both cases are appropriate for stationary workloads.

The methods in [30] and [31] proposed a workload estimation technique which requires that the contents (video clips) should include the decoding time of each frame provided by the content providers. An MPEG decoder in each client machine includes the decoding time translation scheme to scale the content provider's decoding time to the target client machine. However, the extra work given to the content providers limits the practical use of these methods.

The techniques in the final category aimed at estimating the time-varying workloads using adaptive filters.

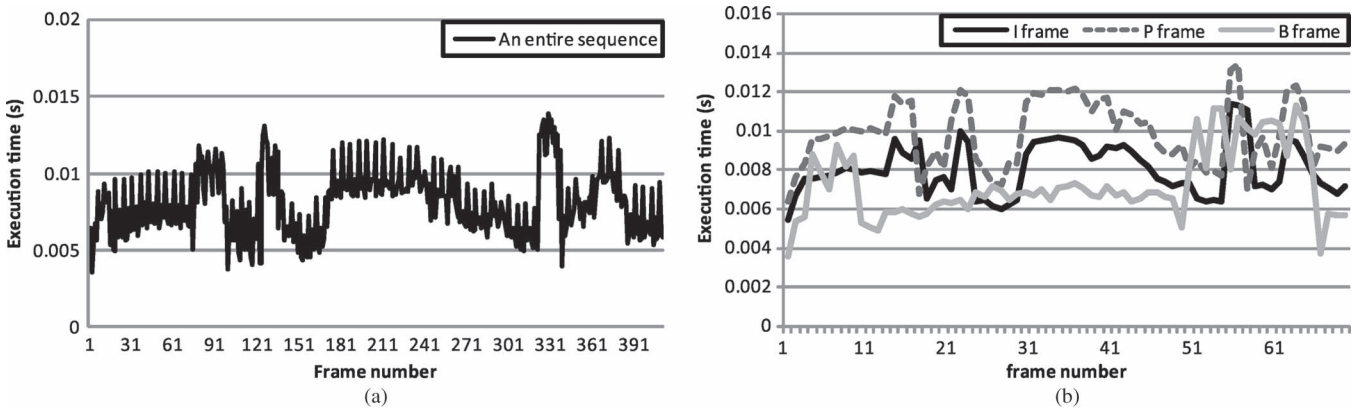


Fig. 1. Comparison of execution time variations in a multimedia application. (a) Execution time variation with all types of frames considered. (b) Execution time variation for each frame type.

Proportional–integral (PI)–derivative (PID) controller was popularly used in these methods for workload estimation. In [32], they used the PID controller to estimate the frame decoding time in multimedia applications. In addition, it was used in [33] for 3-D games. PI controller, which is a variant of PID controller, was adopted in [34] and [35], where it was used to estimate the buffer occupancy for DVS targeting data buffered systems. An integral controller, which is also a variant of PID controller, was used to estimate the workloads. Based upon the workload estimation, they compute the data arrival rate and processing rate which eventually yields system throughput used for performing DVS [36]. Even though the PID controller is an adaptive filter, it is well known that it suffers from overshooting and undershooting, depending on the selected coefficients. Hence, the tuning of coefficients critically determines the prediction accuracy.

Compared to previous approaches, this paper is similar to the techniques using PID controllers in the sense that both methods adopt adaptive filters. However, our technique utilizes Kalman filter, which is more robust than the PID controller in the estimation, since our method does not require any coefficient tuning through the offline profiling. Our method performs self-learning during run time by experiencing the workloads and tunes its coefficients as the workload property is changing as time goes by.

III. Background

A. Real-Time Multimedia Applications

The focus of this paper is on estimating the processing time of multimedia workloads, particularly those that have large variations in processing time. In a real-time multimedia environment, a server system transmits streaming workloads to client systems, and each client system must process received workloads, fulfilling any real-time constraints. Applying a DVS technique to real-time multimedia applications is challenging because we typically do not have sufficient information for workload estimation. There exist various standards for multimedia applications, including MPEG-2/4, H.264, and MP3. Among these, we consider MPEG-2 in this paper, which is a general standard for coding audio and video data in a compressed format [37]. MPEG-2 is one of the most popular standards, and MPEG-2 workloads are typically known to exhibit large variations in processing time.

There are three distinct frame types in MPEG video data, namely, intraframe (I frame), predictive interframe (P frame), and bipredictive interframe (B frame). An I frame is encoded using a source frame in its entirety. A P frame is encoded using differences between the current and previous frames. A B frame is created using the previous and next I or P frames. In the case of the P and B frames, a lossy compression technique is employed for motion estimation in order to reduce the transmitted frame size. It is the different MPEG decoding procedures for different frame types that cause large variations in the processing time.

Fig. 1 shows the processing time variations for decoding an MPEG-2 video clip. When we do not make any distinctions of frame types and consider the clip as it is, the processing time of the clip exhibits a large variation during the whole decoding procedure, as shown in Fig. 1(a). In contrast, when we consider the processing time of each frame type separately, the degree of variation is reduced to some extent, as shown in Fig. 1(b). In this context, various techniques have been proposed, which aim at estimating the processing time of each frame type, one type a time [23], [25]–[28]. However, when the variation of processing time for each frame type is large even after the frame-type-based separation, these techniques often fail to provide satisfactory estimation performance. This is because they are based upon relatively simple estimation models and the resulting real-time tracking ability is limited.

B. Adaptive Filters

For more accurate estimation of workload processing time, we can use adaptive filters that employ feedback control mechanisms for correcting estimation errors. For a system that processes workloads $w_1, w_2, \ldots, w_n$, an (adaptive) filter considers all the previous workloads $w_1, w_2, \ldots, w_{n-1}$ and (noisy) measurement $\hat{w}_n$ in order to estimate w_n.

Filters using the time-invariant *moving average* (MA) and WM are the simplest and most widely used [38]. The MA filter examines a fixed number of prior workloads to estimate the current workload. When the window size is L and the ith workload is $w[i]$, the estimate by the MA filter is given by $w[n+1] =$

$(1/L)\sum_{i=0}^{L-1} w[n-i]$. That is, previous L workloads are used to estimate the next workload, giving equal weight to each workload. One the other hand, the WM filter uses the weighted average of the previous workloads for estimation. When the order of the current workload is N and the weight for the most recent workload is α $(0 < \alpha < 1)$, the estimate by the WM filter is $w[n+1] = \sum_{i=1}^{N} \alpha^{N-i+1} w[i]$, where we require all weights to have a sum of one, i.e., $\sum_{i=1}^{N} \alpha^i = 1$. The WM filter may be more attractive than the MA filter in that the weights can be adjusted for faster adaptation, but finding the optimal weights for different workloads is not trivial.

The *PID* control technique is also generally for error correction in estimation. When the actual measurement and estimation of the ith workload are denoted by w_i and $\hat{w}_i$, respectively, error ϵ_i is simply calculated as $w_i - \hat{w}_i$. Then, a *PID* controller calculates correction value Δw_i as $\Delta w_i = k_P \epsilon_i + (1/k_I)\Sigma_{W_I} \epsilon_i + k_D((\epsilon_i - \epsilon_{i-W_D})/W_D)$, where k_P, k_I, and k_D are the coefficients for proportional, integral, and derivative controls, respectively (W_I and W_D are window sizes for each feedback control). We can easily estimate the processing time of the next workload by adding Δw_i to the previous workload estimate.

C. Kalman Filter

Kalman filter is another popular adaptive filter used in many practical and theoretical fields for accurately estimating the previous, current, and future states of a discrete-time controlled process [10]–[12]. The state x of this process and its corresponding measurement z are governed by the linear stochastic difference equations

$$x_n = Ax_{n-1} + Bu_{n-1} + w_{n-1} \tag{1}$$
$$z_n = Hx_n + v_n \tag{2}$$

respectively. The matrix A relates the states at time steps $n-1$ and n, in the absence of process noise or control input. The matrix B relates the optional control input u to the state x, and the matrix H relates the state x to the measurement z. The random variables w_n and v_n denote the process and measurement noises, respectively, and we assume that w_n and v_n are independent white Gaussian noise following

$$w \sim N(0, Q) \tag{3}$$
$$v \sim N(0, R) \tag{4}$$

where Q and R are the process and measurement noise covariances, respectively.

We then define *a priori* and *a posteriori* estimate errors as $e_n^- \equiv x_n - \hat{x}_n^-$ and $e_n \equiv x_n - \hat{x}_n$, respectively, where $\hat{x}_n^-$ is our *a priori* state estimate given the knowledge on the process prior to step n and $\hat{x}_n$ is our *a posteriori* state estimate after measurement z_n has been made. Based upon these estimates, the *a priori* and *a posteriori* estimate error covariances are given by

$$P_n^- = E\left[e_n^- e_n^{-\mathrm{T}}\right] \tag{5}$$
$$P_n = E\left[e_n e_n^{\mathrm{T}}\right] \tag{6}$$

respectively.

In the Kalman filter framework, an *a posteriori* state estimate $\hat{x}_n$ is represented by a linear combination of an *a priori* state estimate $\hat{x}_n^-$ and a weighted difference between measurement z_n and predicted measurement $H\hat{x}_n^-$

$$\hat{x}_n = \hat{x}_n^- + K\left(z_n - H\hat{x}_n^-\right) \tag{7}$$

where the difference $(z_n - H\hat{x}_n^-)$ is called the *measurement innovation*. The matrix K is chosen to maximize the *a posteriori* error covariance P_n, and a solution is

$$K_n = P_n^- H^{\mathrm{T}}\left(HP_n^- H^{\mathrm{T}} + R\right)^{-1} \tag{8}$$

where K_n is often termed as the *Kalman gain*.

To estimate the states of a process with measurements, the Kalman filter employs a feedback control technique in which the state at some time is estimated first and feedback is then provided in the form of noisy measurements. The Kalman filter thus works in two phases, as shown in Fig. 2. In the time update phase, the previous state and estimation error covariance are updated with the current information by the following *time update equations*:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_{n-1} \tag{9}$$
$$P_n^- = AP_{n-1}A^{\mathrm{T}} + Q. \tag{10}$$

In (9), the conventional Kalman filter first projects the state ahead from the previous state $\hat{x}_{n-1}^-$ and certain input matrix Bu_{n-1}. The filter then projects the error covariance ahead with process noise covariance Q in (10).

After the time update phase, the measurement update phase (i.e., feedback update phase) starts with measuring actual workloads related to the state parameters and computes the following *measurement update (or feedback update) equations*:

$$K_n = P_n^- H^{\mathrm{T}}\left(HP_n^- H^{\mathrm{T}} + R\right)^{-1} \tag{11}$$
$$\hat{x}_n = \hat{x}_n^- + K_n\left(z_n - H\hat{x}_n^-\right) \tag{12}$$
$$P_n = (I - K_n H)P_n^-. \tag{13}$$

In this measurement update phase, the Kalman gain is first computed by using the *a priori* estimate error covariance P_n^- and measurement noise covariance R. The filter then updates the current state matrix $\hat{x}_n$ and *a posteriori* estimate error covariance P_n, using the Kalman gain. Note that we use the terms "measurement update" and "feedback update" interchangeably in this paper.

IV. Comparison Between the Proposed Method and the Conventional Kalman Filter

In this section, we compare and contrast the operation of the proposed estimation technique with that of the conventional Kalman filter. As shown in Fig. 2, both methods are based upon the two-step cyclic operations—time and feedback update phases. The equations in each phase are almost identical, except for the following: The process and measurement noise covariances are time varying in the proposed method and are denoted by Q_n and R_n, respectively, whereas they are constant

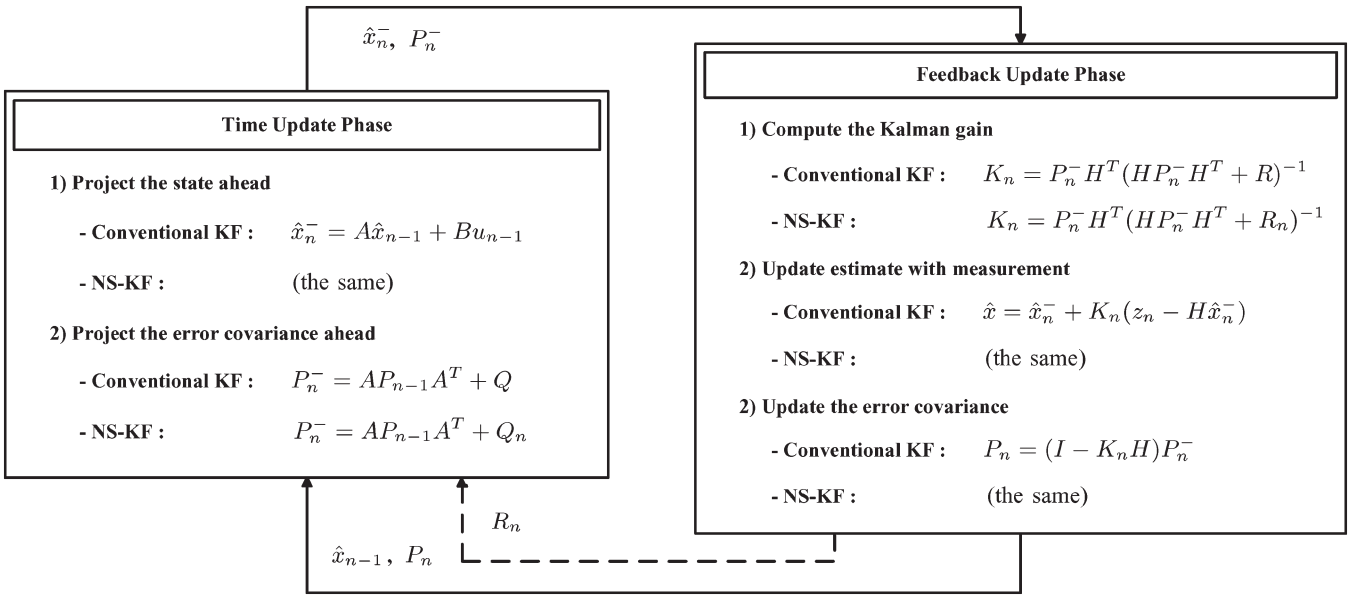


Fig. 2. Comparison of the operations of the proposed and conventional Kalman filters.

in the conventional Kalman filter and are represented by Q and R, respectively. With the most notable difference of the proposed technique being the ability to handle nonstationarity, our approach is referred to as the nonstationary Kalman filter (NS-KF) in what follows.

The physical meaning of Q_n and R_n in this paper also differs from that of Q and R in other domains such as communications. In the applications of the conventional Kalman framework to communication systems, the process noise models the uncertainty of a signal source, and the measurement noise models the noise effect of a channel. Obviously, these two factors are independent, and most approaches in this area assume that they are constant. In contrast, we use the process noise to model the uncertainty of hardware, such as pipeline stalls, branch prediction misses, and cache misses. Such uncertainty affects the frame decoding time with a different amount for each frame. In addition, we model the imperfect prediction ability of the estimator as the measurement noise, which is the only error source in our environment.

In other words, the measurement noise covariance in this paper corresponds to the prediction error (i.e., the difference between the predicted and measured values) covariance. As for the process noise, we assume that it is caused by the unpredictability of the hardware system used. Under this assumption, we can further assume that the process noise accounts for a part of the measurement noise, since the estimator itself is also executed in the same hardware on which workloads are processed. Consequently, we let the process noise covariance be proportional to the prediction noise covariance with a proportional coefficient being less than one, as shown in (22) of Section V. In NS-KF, additional feedback by R_n thus exists from the feedback to time update phase, as represented by the dotted arrow in Fig. 2.

According to our experimental study, using time-varying noise covariance values is critical to increase the estimation performance. As presented in Section VI, the aforementioned assumptions on the process and measurement noise are well supported by the experimental result we obtained. For instance, the result labeled with TKF in Table IV was obtained by setting the process noise covariance as a constant independent of the measurement noise but was inferior to the result obtained from the proposed NS-KF method with respect to all the criteria used. Further details on the proposed NS-KF method are available in Sections V-B and C.

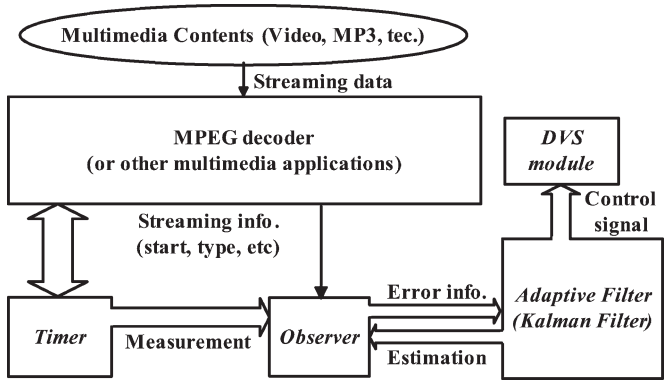


Fig. 3. Proposed estimator using adaptive filter.

V. Workload Estimation for Real-Time Multimedia Applications

In the previous section, we briefly compared NS-KF to the conventional Kalman filter from the conceptual point of view. In this section, we describe the details of NS-KF when we use it for DVS as a workload processing time estimator in MPEG applications.

A. Overall Architecture

Fig. 3 shows the overall architecture of the proposed estimator. On receiving workloads, the client system continuously processes them in order. At the same time, the proposed estimator embedded in the client system continuously estimates

the processing time of the next workload and adjusts the voltage and frequency level in order to achieve power savings by exploiting the idle time of each workload. The proposed estimator consists of four functional modules named *Timer*, *Observer*, *Adaptive filter*, and *DVS module* in addition to the main processing unit (not shown in the figure). All modules can be implemented at the application level without any hardware support. We exploit the fact that the variation in processing time reduces if we divide frames according to their types and analyze each frame type separately, as discussed in Section III. The proposed estimator can thus be regarded as a collection of three estimators, one for each MPEG frame type. All equations and algorithms presented in this section apply equally to every frame type.

The *Timer* module is to measure the actual processing time of each workload and transfers it to *Observer*. The *Observer* module receives the actual processing time information from the *Timer* and additional workload information (e.g., start/end time and workload type) from the multimedia application currently in execution. The *Observer* module then calculates various statistics, including the average, variance, and error covariance for the correction of estimation errors. These statistics are then transferred to the *Adaptive filter*. Using the measurement update equations and the error information received from the *Observer*, the *Adaptive filter* module performs feedback control for correcting error covariances and estimating workload processing time. This module is based upon the Kalman filter. Finally, the *DVS module* sets appropriate voltage and frequency to be used by the client system based upon the estimated workload processing time. More details of the *Observer*, *Adaptive filter*, and *DVS module* are provided next. We also explain how to implement an MPEG decoder based on our DVS technique.

B. Observer

The major role of the *Observer* is to compute R, which is the measurement noise (or error) covariance. To handle nonstationary workloads, we assume that R is time varying and denote the measurement error covariance at time step n by R_n. Since the state x in this paper is a scalar variable (i.e., processing time), so is the covariance R_n.

Let $\mathbf{Z}_n = \{Z_i \mid i = 1, 2, \ldots, n\}$ be he set of measurement values provided by the *Timer* block and $\hat{\mathbf{X}}_n^- = \{\hat{X}_i^- \mid i = 1, 2, \ldots, n\}$ the set of *a priori* estimates obtained from the *Adaptive filter* block. To consider the time-varying nature of the workloads, we set Z_i and $\hat{X}_i^-$ to $\beta^{n-i+1} z_i$ and $\beta^{n-i+1} \hat{x}_i^-$, respectively, where β is a weighting factor like in *WM*. Then, the measurement error covariance at time step n is given by

$$R_n = E\left[\left(\mathbf{Z}_n - \hat{\mathbf{X}}_n^-\right)^2\right] \tag{14}$$

$$= \frac{\sum_{i=1}^{n}\left(Z_i - \hat{X}_i^-\right)^2}{n}. \tag{15}$$

To reduce the computation complexity of (15), we approximate it as follows:

$$R_n = (1-\beta) \times R_{n-1} + \beta \times \left(z_i - \hat{x}_i^-\right)^2. \tag{16}$$

Using (16), we can therefore evaluate R_n by only using R_{n-1} and the current measurement without storing all prior values.

C. Adaptive Filter

This module estimates the processing time of workloads and provides *DVS module* with the estimated processing time so that it can adjust the voltage level adaptively. As estimator, we employ the Kalman filter and implement its time and measurement update equations with some modifications.

Most importantly, we use time-varying process and measurement error covariances and denote them by Q_n and R_n to handle nonstationary workloads. Their necessities in our environment are already discussed in Section IV. This may be a major difference between our approach and the conventional Kalman filter framework, which usually assumes that the process and measurement error covariances Q and R are constant. It is known that the stabilization of the *a posteriori* error covariance P_n and the Kalman gain K_n are both related to Q and R, and it matters how to model Q and R appropriately. The large degree of variation in workload processing time and the result from our empirical studies indicate that using constant Q and R leaves room for improvements in accuracy.

Example 1: Fig. 4 shows how the quality of estimation is affected as the value of Q_n varies. The workload used is an MPEG video clip called *bike*. Four different blocks, each of which has 30 frames, were selected, and the average estimation error was measured for each block using different Q_n values. It is evident that the estimation quality does get affected by using different Q_n values. □

Other minor adjustments we made are as follows. The matrices used in the update equations become scalar because the state variable x and its *a priori* and *a posteriori* estimates $\hat{x}^-$ and $\hat{x}$ are scalar. Hence, we set $A = H = 1$. In addition, we assume that no control input exists and set $B = 0$.

After reflecting these changes into the original Kalman equations, the time update equations now become

$$\hat{x}_n^- = \hat{x}_{n-1} \tag{17}$$

$$P_n^- = P_{n-1} + Q_n \tag{18}$$

and the measurement update equations are given by

$$K_n = P_n^-\left(P_n^- + R_n\right)^{-1} \tag{19}$$

$$\hat{x}_n = \hat{x}_n^- + K_n\left(z_n - \hat{x}_n^-\right) \tag{20}$$

$$P_n = (1 - K_n) P_n^- \tag{21}$$

where the time-varying measurement noise covariance R_n is easily provided by the *Observer* module, as already explained in Section V-B. In order is the explanation on determining the value of Q_n.

According to our experiments, the optimal value of Q_n varies with respect to the location of the frame whose processing time is under estimation.

Example 2: As shown in Fig. 4, we measured the average estimation error by varying the value of Q_n in four different regions of the video clip used. Each region consists of 30 frames. Clearly, the best value of Q_n is different, depending upon the frame location. □

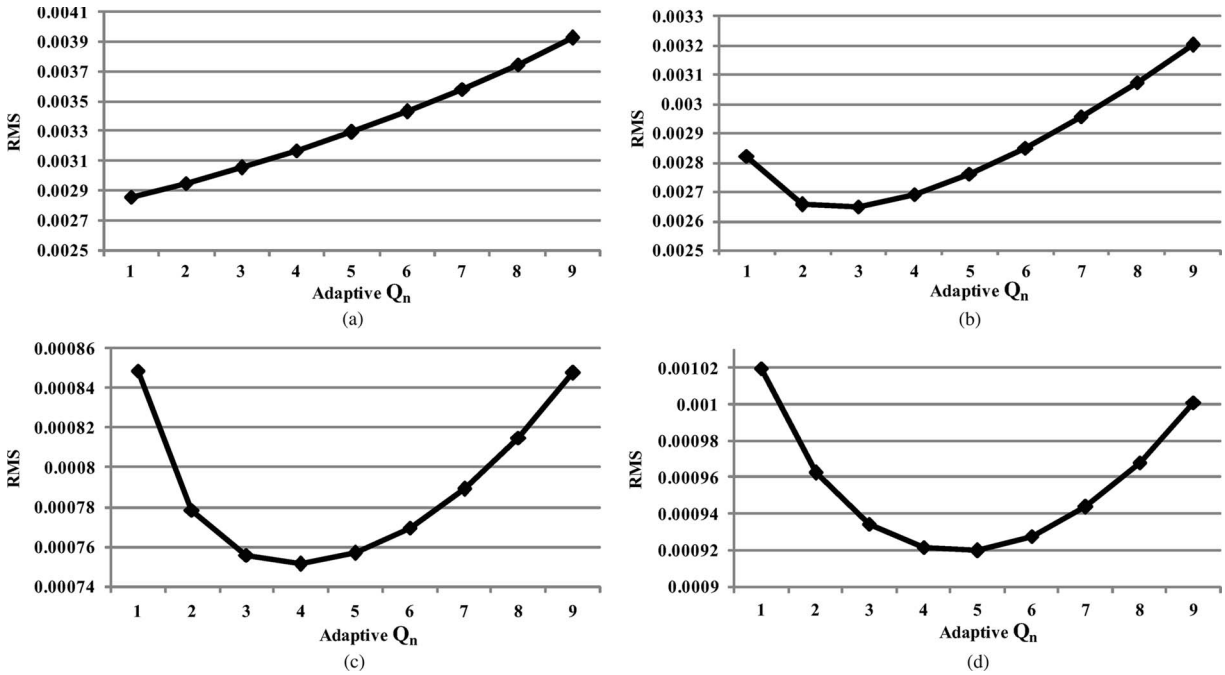


Fig. 4. Impact of varying the Q_n of measurement update phase on the estimation performance. The curve in each pane represents the rms measured in a block of 30 frames taken out of *bike*, which is an MPEG video clip, and the range of frames using this experiment is from the first frame to 120th frame in *bike*.

As shown in Example 2, the value of Q_n for the minimum estimation error varies, depending on the frame sequences. For this reason, Q_n should be considered as a time-varying factor proportional to the measurement error covariance R_n, as discussed in Section IV. Hence, Q_n can be expressed as

$$Q_n = \gamma R_n \tag{22}$$

where γ is a positive coefficient whose value is initially set to unity and is then adjusted adaptively as the estimation process proceeds. The role of γ is to reinforce the weakness of a weighting factor β. More details of determining the proportional coefficient γ will follow shortly.

Algorithm I outlines the algorithm used by the *Adaptive filter* module. For each frame type, the following operations are performed:

1) Line 1. The parameter and Q_n values are initialized. The proportional coefficient γ and Q_n are set to one, and another parameter δ is set to 0.1.
2) Lines 2–12. For each time step n, the procedures described in Lines 3–11 are executed.
3) Lines 3–4. When the value of Q_n is plugged into the time update equation (18), two adjacent values of Q_n are considered as well. This is for increasing the accuracy of prediction by examining additional values of Q_n. These two values are determined by

$$Q_n^R = \frac{Q_n}{1-\delta} \tag{23}$$

$$Q_n^L = Q_n(1-\delta) \tag{24}$$

where δ is a fixed constant $(0 < \delta < 1)$, and Q_n^R and Q_n^L represent the right- and left-adjacent values of Q_n, respectively.

Algorithm I : Pseudo-code of Proposed Estimation Algorithm

```
1  Initialization() ; /* Q_n = γ = 1, δ = 0.1 */
2  For each time step n do the following:
   /* Update Q_n^R and Q_n^L */
3  Q_n^R = Q_n/(1−δ);
4  Q_n^L = Q_n(1 − δ);
   /* Time update phase */
5  {x̂_n^−, P_{n,Q_n}^−, P_{n,Q_n^R}^−, P_{n,Q_n^L}^−} = Time_update(Q_n, Q_n^R, Q_n^L);
6  Send_estimate_to_DVS_module(x̂_n^−) ;
   /* Get actual processing time after workload is processed */
7  z_n = Obtain_observations_from_Observer();
   /* Calculation of estimation error covariance */
8  R_n = (1 − β) × R_{n−1} + β × (z_i − x̂_i^−)^2;
   /* Measurement update phase */
9  {x̂_{Q_n}, x̂_{Q_n^R}, x̂_{Q_n^L}} = Measurement_update(R_n);
   /* Update γ per M frames */
10 if (Current_Frame_Number % M == 0)
11 γ =
   arg min_{Q_n,Q_n^R,Q_n^L} {(z_n − x̂_{Q_n})^2, (z_n − x̂_{Q_n^R})^2, (z_n − x̂_{Q_n^L})^2} ;
12 Q_n = γR_n;
```

4) Line 5. The time update equations are evaluated using three different choices of the estimation error covariance, namely, Q_n, Q_n^R, and Q_n^L. The *a priori* processing time estimate $\hat{x}_n^-$ is returned along with three different values of P_n^-. The return values P_{n,Q_n}^-, $P_{n,Q_n^R}^-$, and $P_{n,Q_n^L}^-$ are computed by evaluating (18) using Q_n, Q_n^R, and Q_n^L, respectively.
5) Line 6. The estimated processing time $\hat{x}_n^-$ is sent to the *DVS module*.

TABLE I
DVS TABLE

Row #	voltage [V]	frequency [MHz]	power [W]
1	1.55	624	0.925
2	1.45	520	0.747
3	1.35	416	0.570
4	1.25	312	0.390
5	1.15	208	0.279

6) Line 7. Once the frame has been processed, the *Observer* module measures the actual processing time z_n and transmits it to the *Adaptive filter*.
7) Line 8. The estimation error covariance R_n is calculated using current estimation and actual measurement.
8) Line 9. The estimation error covariance R_n is plugged into the measurement update equations, which then returns three *a posteriori* state estimates $\hat{x}_{Q_n}$, $\hat{x}_{Q_n^R}$, and $\hat{x}_{Q_n^L}$. These estimates are the result of evaluating (21) using the predetermined P^-_{n,Q_n}, P^-_{n,Q_n^R}, and P^-_{n,Q_n^L} values, respectively.
9) Lines 10–11. The squared error $(z_n - \hat{x}_n)^2$ values are calculated for each of the three state estimates. We then set the value of γ to the process error covariance of the estimate giving the lowest squared error. This operation is performed every M frames.
10) Line 12. The value of Q_n is updated using the new value of γ and R_n.

D. DVS Module

The idea of DVS is to dynamically adjust the supply voltage depending upon the processing need, thus reducing the overall power consumption. A DVS controller typically maintains a table in which each line contains a pair of a supply voltage and its corresponding operating frequency, as shown in Table I. Given a processing time, the operating frequency is first determined, and the corresponding supply voltage is then retrieved from the table using the frequency as a key.

In the proposed architecture, the processing time of the frame at time step n is estimated by the *Adaptive filter* module and is provided as the *a priori* state estimate $\hat{x}_n^-$. Let f_i denote the frequency of row i in the DVS table. Assume that the rows in the DVS table are arranged in descending order of frequency. That is, $f_i > f_j$ iff $i < j$. Then, the row index for $\hat{x}_n^-$ is set to the minimum i such that the inequality

$$f_i \geq \frac{1}{\min\{\hat{x}_n^- + SO, DL\}} \tag{25}$$

is fulfilled. In (25), SO and DL represent the overhead of switching frequencies and the given deadline, respectively. It typically takes only tens of microseconds to switch frequencies, and SO is thus negligible in most cases. The *DVS module* sets the supply voltage of the processor used by reading the information in row i.

A DVS table usually contains power information, such as dynamic power (P_D) and leakage power (P_L) [39] for each line. Dynamic power P_D is given by $P_D = C_s V_{dd}^2 f_i$, where C_s, V_{dd}, and f_i are the switching capacitance, supply voltage, and operating frequency corresponding to row i in the DVS table. Leakage power P_L is calculated by $P_L = (V_{dd} I_{\text{sub}} + |V_{bs}| I_j) L_g$, where I_{sub}, $|V_{bs}|$, I_j, and L_g are device-specific parameters defined in [39].

The DVS table we used in this paper is shown in Table I. It has five different voltage–frequency pairs, and each value in the last column corresponds to the total power consumption $(P_D + P_L)$ for each row.

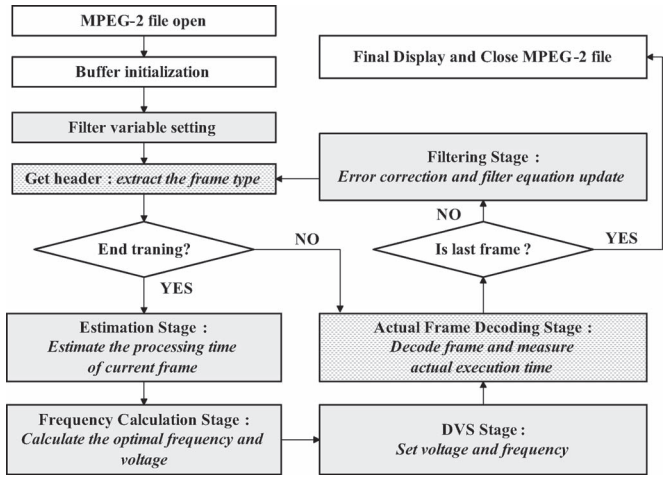


Fig. 5. Entire workflow of libmpeg-2 extended for DVS.

E. MPEG Decoder Implementation

We extended *libmpeg-2* to perform DVS with the proposed workload estimator. The entire workflow of the extended *libmpeg-2* is shown in Fig. 5, where some blocks are added for filter operations (blocks with dot pattern) and some are for performing the estimation and error correction (gray blocks) for performing workload estimation and DVS. First, the decoder initializes internal buffers and filter-related variables. Note that we implemented a filter for each frame type. In other words, there are three independent workload estimators, and each of them is dedicated to each frame type. After initialization, it starts to decode a series of MPEG-2 frames. As a first step, it extracts the frame type from the frame header and selects the corresponding workload estimator. If it is still under training, the decoder decodes the frame with the maximum voltage/frequency pair and measures the decoding time. The measured time is used for the decoding time of the next same-type frame. The training period is defined in terms of the number of frames. In our case, we set it to one in order not to waste the idleness of the initial frames. After training, it performs the following steps iteratively until the last frame is completely decoded. First, the decoder selects the corresponding workload estimator and performs the frame decoding time estimation (estimation stage) for the next frame. Based upon the estimation, it calculates the optimal voltage/frequency pair (frequency calculation stage) and sets the voltage/frequency pair (DVS stage). Then, it decodes the frame with the selected voltage/frequency pair and measures its actual decoding time (actual frame decoding stage). Finally, it performs the error correction and updates the filter equation.

TABLE II
VIDEO CLIPS USED FOR SIMULATION

clips	resolution	frame#	Var	I_{var}	P_{var}	B_{var}
video clips with small workload variation (unit : $\times 10^{-7}$)						
airwolf	192x144	412	44	18	29	43
bobo	199x144	680	35	2.6	5.7	17
berger	240x180	309	45	1.4	1.0	52
cdrs	320x240	301	210	4.7	16	2.4
video clips with large workload variation (unit : $\times 10^{-7}$)						
RedsNight	320x240	1211	210	22	340	170
logo-UC	336x244	359	98	5.6	126	35
bike	352x240	150	170	10	12	110
us	352x240	730	136	14	124	106

VI. EXPERIMENTAL RESULTS

A. Experiments Setup

We conducted three sets of experiments to appreciate the impact of our method. In the first set of experiments, we evaluated the workload estimation accuracy of several filter-based methods, including ours for MPEG video clips, as well as synthetic workloads. In the second set of experiments, we tested how each estimation technique well incorporates with DVS in terms of energy saving and real-time constraint satisfaction in MPEG environment. For these two sets of experiments, we used a cycle-accurate simulator called *SoC Designer* from Carbon Design Systems [40]. In the final set of experiments, we validated our method by measuring the EC of XScale-based test bed (*HBE-SM270 DVB*) adopting PXA270 processor [41], while playing real multimedia application. In both simulation and actual measurements, we used *libmpeg-2* [42] decoder, which is modified to perform DVS incorporating with workload estimation techniques compared.

We used eight video clips in all experiments except those with synthetic workloads. The clips are classified into two groups, depending on the magnitude of workload variation. In this purpose, we measured the workload variation of each type of frames, as well as the variation of all types of frames together, as shown in Table II, where Var denotes the variation of the overall frame decoding time, while $I_{\rm var}$, $P_{\rm var}$, and $B_{\rm var}$ correspond to the execution time variations of I-, P-, and B-type frames, respectively.

To run the video clips in Table II, we used the extended *libmpeg-2* mentioned in Section V-E as our method. All other filter-based workload estimators are similarly embedded in *libmpeg-2* for comparison purposes. We used the Monte Carlo simulation method to find the best parameter set for each technique by stimulating all eight video clips. The chosen best parameter set for each technique yields the minimum average mean squared error (MSE) among all tested parameter sets.

B. Workload Estimation Accuracy Comparison

We compared several filter-based methods [(MA, WM, PID, NS-KF, and traditional Kalman filter (TKF)] for appreciating their relative estimation accuracy for the time-varying workloads. All these methods have an estimator for each frame type for fair comparison. Among these methods, TKF uses a time-varying measurement noise covariance and a constant process noise covariance unlike our method. The objective of this experiment is to measure how quickly and precisely each method tracks the variation of workload.

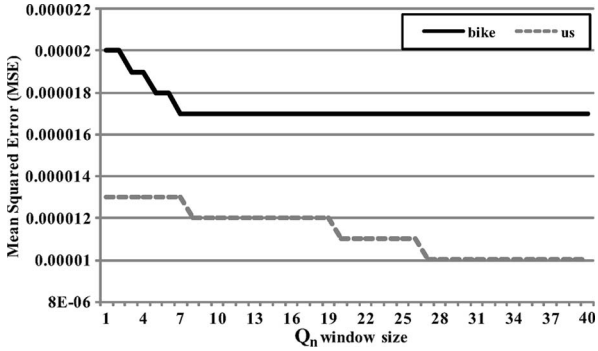


Fig. 6. MSE variation according to Q_n window size.

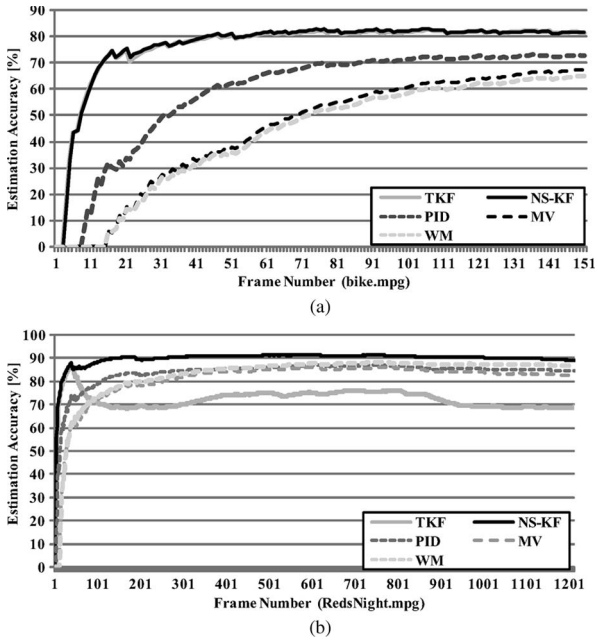


Fig. 7. Convergence speed comparison. (a) bike.mpg. (b) RedsNight.mpg.

First, we conducted an experiment to decide how frequently we must update Q_n to tune the performance of NS-KF appropriately. As shown in Fig. 6, we vary the Q_n window size from 1 to 40 frames while measuring MSE for two video clips—"bike.mpg" and "us.mpg." A window size of seven is good enough for "bike.mpg," while a window size of 27 saturates the MSE of NS-KF for "us.mpg." From these experimental results, we set the window size to 30 for the entire experiments conducted in this section.

We next compared the methods in MPEG environment and then compared two most dominant methods (NS-KF and PID) more in details using synthetic workloads.

Fig. 7 shows the convergence speed of each method for two video clips—"bike.mpg" and "RedsNight.mpg" which have highly time-varying property, where the estimation accuracy is defined by the average absolute difference of the actual and estimated decoding times over the actual decoding time for each frame. In Fig. 7(a), NS-KF outperforms all other methods except TKF. Both NS-KF and TKF reach 80% of estimation

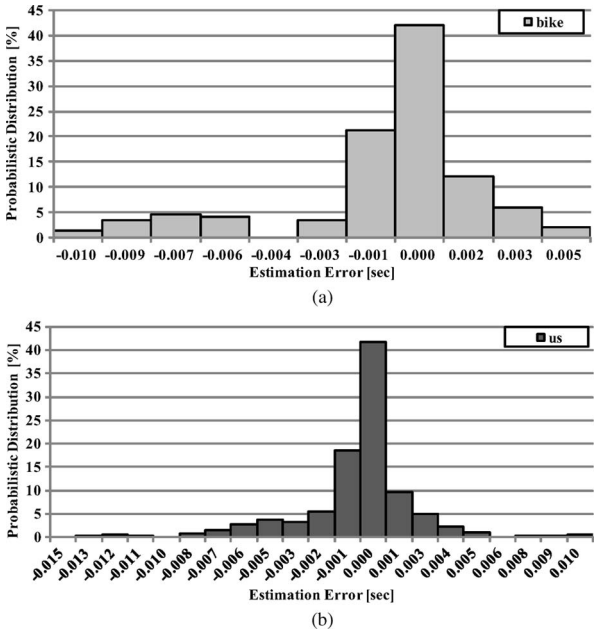


Fig. 8. Probability distribution of estimation errors for NS-KF. (a) bike.mpg. (b) us.mpg.

TABLE III
MSE COMPARISON OF ALL FILTER-BASED METHODS ($\times 10^{-6}$)

clips	MA	WM	PID	NS-KF	TKF
airwolf	7	6	4	3	5
bobo	3	3	2	1	2
berger	12	13	9	6	5
cdrs	17	17	7	2	3
RedsNight	13	10	8	5	24
logo-UC	21	19	9	3	8
bike	77	80	43	17	16
us	15	16	14	10	11

accuracy after experiencing about 40 frames, while others are far inferior to these two methods. On the contrary, other methods are comparable (even still inferior) to NS-KF except TKF. It means that TKF behaves well when it is well customized for the given workload, but its performance is severely degraded when the workload characteristics are changed.

The estimation accuracy of NS-KF is analyzed from the other perspective, i.e., variation as shown in Fig. 8. In Fig. 8(a), NS-KF shows that the estimation error is mostly bounded within 10% for 90% of frames.[1] A similar trend was observed in Fig. 8(b), meaning that NS-KF does not incur severe estimation accuracy in average case, as well as in instantaneous case. We also compared all the methods for all the tested video clips in terms of MSE, as shown in Table III. It confirms that NS-KF outperforms all other methods, particularly for the large variation video clips.

For a detailed comparison of NS-KF and PID, we measured MSE for the workload given in

$$w = A \times \sin(2\pi f t) + n(\mu, \sigma^2) \tag{26}$$

[1]The decoding deadline (rate) was given as 33 ms (30 frames/s); hence, ± 0.033 s corresponds to $\pm 10\%$ of the estimation error.

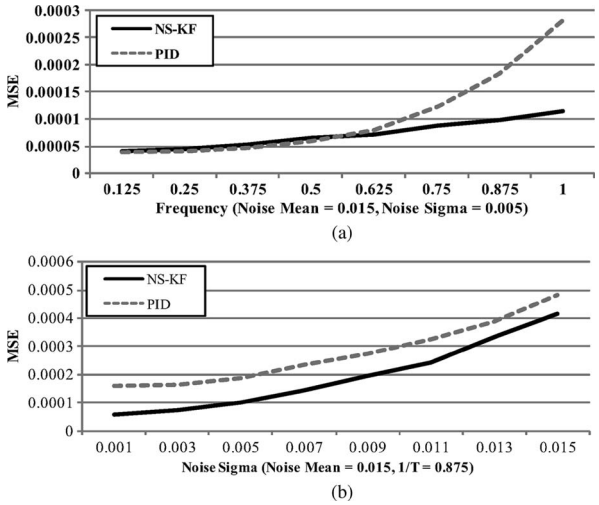


Fig. 9. Comparison of NS-KF and PID for synthetic time-varying and noisy workload. (a) MSE for frequency variation. (b) MSE for noise variation.

where w is the expected frame decoding time at time t, A is the amplitude, and f is the frequency. Moreover, $n(\mu, \sigma^2)$ is a Gaussian noise of which average value and standard deviation are μ and σ, respectively.

First, we measured the MSE of both methods while changing f from 0.125 to 1 to appreciate their workload tracking capability as the time-varying speed of the workload increases. We set A to 0.015; hence, its peak-to-peak value is 0.03, which approximately corresponds to a single MPEG frame decoding deadline in millisecond unit. Moreover, we set μ to 0.015, which is half of the single frame decoding time, and finally, we set σ to 0.003 by assuming that most of the frame decoding times are bounded within $\pm 30\%$ of μ. Fig. 9(a) clearly shows the superiority of NS-KF over PID, since the MSE of NS-KF much slowly increases compared to that of PID as the frequency increases. More specifically, NS-KF completely outperforms PID when the frequency becomes larger than 0.7. In other words, NS-KF tracks the fast time-varying workload much better than PID. In addition, we performed the sensitivity analysis of both methods for the noise, as shown in Fig. 9(b), which clearly shows that NS-KF is more robust to the noise than PID.

C. Impact on DVS Comparison

In these experiments, we tested how each method cooperates well with DVS to save energy while satisfying the given real-time constraint. We compare the five filter-based methods in terms of the following performance metrics.

1) **Energy Consumption (EC):** the EC with DVS adopting the target workload estimation method over the EC without DVS. Note that it includes not only the energy consumed by the application program but also the energy consumed by each DVS method.
2) **Decision Accuracy (DA):** How closely the DVS with the target workload estimation method selects a voltage/frequency pair to the optimal voltage/frequency (v/f) pair. More precisely, $DA = 1 - (|OI - SI|/NI)$, where OI, SI, and NI are the index of the optimal v/f pair,

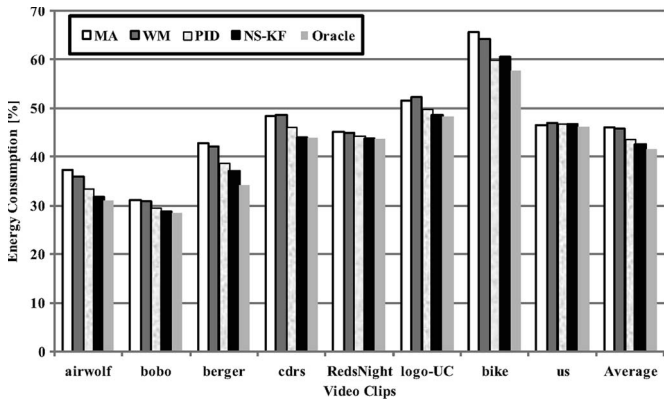


Fig. 10. EC comparison of filter-based DVS methods.

the index of the selected v/f pair, and the number of v/f pairs, respectively. Moreover, note that the index of each v/f pair is from one to the number of total v/f pairs and the index of the v/f pair with the highest voltage is one, as in Table I.

3) **Hit Ratio (HR):** similar to DA, but HR only considers the DVS that selects a v/f pair that is exactly the same to the optimal case. Hence, HR $=$ (number of selections of optimal v/f pair)/(number of total decisions).
4) **Deadline Miss Ratio (DMR):** the number of frames missing deadline due to DVS over the total number of frames.
5) **Computation Overhead Ratio (COR):** the increase of the execution time of libempeg-2 equipped with the target workload estimation method over the execution time of original libmpeg-2.

We also compared the five methods with the *oracle method*, which is an ideal method and only possible with offline workload analysis. Note that the *oracle method* guarantees the minimum EC. In addition, it guarantees that MSE, DMR, and COR are zero, while DA and HR are 100%.

Fig. 10 shows the EC of all the aforementioned methods, while Table IV shows the comparison for all other metrics.

NS-KF outperforms all other filter-based methods and comparable to *oracle method* in terms of EC, and the average energy saving of NS-KF is 57.5% when we compare ours with the original decoder which does not use a DVS scheme. It means that NS-KF is very effective to reduce the EC by providing the high workload estimation accuracy for DVS. However, it shows a little bit (less than 2%) more EC than PID for the video clip "bike." The reason for this result can be easily found in Table IV, where NS-KF shows 1.3 of DMR while DMR of PID is 17.3, meaning that PID misses the deadline 13.3 times more frequently than NS-KF. In other words, PID achieves higher energy saving while sacrificing the real-time constraint satisfaction. All other methods show similar trend in DMR, particularly for the video clips with a large variation (clips in lower four rows in Table IV). Moreover, an average DMR for NS-KF is just 6.1%, which is an acceptable value for human visual system.

TKF was comparable to NS-KF in Section VI-B for some video clips. However, it shows very poor performance in all metrics due to the constant process noise covariance Q, while NS-KF uses Q_n, which is time varying as the workload varies.

DA and HR also support the superiority of NS-KF over the other methods. HR indicates that NS-KF consistently selects the optimal v/f pair with 80% or higher accuracy for all video clips except "us," while others show a large variation and lower average value in HR compared to NS-KF. More interestingly, the DA of NS-KF is always higher than 90%, meaning that NS-KF selects a v/f pair close to the optimal v/f pair, even when it misses the optimal v/f pair, owing to the high MSE unlike other methods. We can also confirm that NS-KF is stronger than the others in largely time-varying environment through a comparison of average values between workloads with small and large variances. The difference of estimation performance values is larger in the case of largely varied workloads.

Finally, we compared COR for all these methods. Surprisingly, NS-KF shows less COR than the other methods, even though it shows better performance in all other metrics. The low overhead of our method is mainly due to the absence of complicated operations like matrix manipulations and efficient computations of the update equations, because the size of a matrix used in our method is 1×1, which only causes simple additions and multiplications, as shown in Section V. Also note that our method is designed for intertask level rather than intratask level. Hence, the estimation is performed only once per frame.

D. Validation With XScale-Based Test Bed

We validated our method in real environment by the use of XScale-based test bed. The board supports DVS with five v/f pairs by adopting PXA270 processor. The five v/f pairs supported in this board are identical to Table I. In addition, it can display the decoded video sequence on its own liquid crystal display (LCD). The detailed specification of the board can be found in [43]. On this board, we ran *mplayer*, which is an extension of *libmpeg-2*, a software MPEG decoder in our simulation-based experiments. Note that *mplayer* is identical to *libmpeg-2* in decoding capability. However, it has more functionalities to interface with other peripheral devices, which is important in real board-level validation. For instance, it transmits each decoded frame to the LCD controller. The details of *mplayer* can be found in [44].

In this board environment, we compared our method, PID, and oracle method when it decodes "bike.mpg," which is one of the highly time-varying workloads, as indicated in Table IV. We compared these methods from a QoS perspective. In our case, QoS can be measured by the decoding time and EC. When a frame is decoded over the given timing constraint, two types of treatments are possible. First, the decoder sends it to the display, which increases the overall decoding time with the display quality degradation. Second, the decoder drops the next frame to compensate the delay caused by the current frame. Both treatments will not satisfy the users' expectation when the delay of frame decoding frequently and severely occurs. In this validation, *mplayer* chose the first option to handle the delayed frame decoding. If the decoder selects a v/f pair that is slower than the ideal v/f pair, the overall decoding time will increase.

TABLE IV
ESTIMATION PERFORMANCE

clips	DA					HR					DMR					COR				
Small Var	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF
airwolf	88.3	90.3	93.9	95.7	88.9	73.1	81.1	84.5	84.9	58.6	5.5	4.6	5.0	6.5	30.7	0.78	0.92	0.79	0.76	0.76
bobo	92.0	92.8	95.0	96.0	93.1	79.4	82.9	85.4	85.4	74.0	8.0	6.9	6.1	6.7	25.2	0.41	0.47	0.41	0.38	0.38
berger	87.7	87.4	91.9	94.9	95.4	77.1	78.8	79.3	82.5	84.8	0.3	2.6	3.5	3.5	0.3	0.31	0.35	0.30	0.29	0.29
cdrs	90.4	90.7	95.3	98.4	96.6	88.4	89.7	93.3	96.6	89.4	2.3	0.9	0.7	1.6	9.2	0.21	0.24	0.21	0.20	0.20
Average	89.6	90.3	94.0	96.3	93.5	79.5	83.1	85.6	87.4	76.7	4.0	3.8	3.8	4.6	16.4	0.43	0.50	0.43	0.41	0.41
Large Var	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF	MA	WM	PID	NS-KF	TKF
RedsNight	90.9	92.8	92.1	95.0	85.5	71.7	79.3	73.0	82.1	48.7	11.8	9.0	12.7	8.7	35.2	0.19	0.21	0.19	0.19	0.19
logo-UC	87.2	88.1	91.4	94.7	89.3	71.6	75.2	75.3	81.6	59.7	11.1	8.0	10.2	8.3	36.9	0.17	0.19	0.17	0.16	0.16
bike	77.8	75.1	84.1	93.7	93.8	64.9	54.3	60.2	80.7	81.4	2.0	12.5	17.8	1.3	0.6	0.17	0.19	0.18	0.17	0.17
us	88.5	87.8	88.6	92.2	89.4	66.7	64.1	61.9	72.5	61.4	20.3	20.2	19.8	11.7	30.9	0.17	0.19	0.17	0.16	0.16
Average	86.1	86.0	89.1	93.9	89.5	68.7	68.2	67.6	79.2	62.8	11.3	12.4	15.1	7.5	25.9	0.18	0.20	0.18	0.17	0.17
Total Avg.	88.0	88.2	91.6	95.1	91.5	74.2	75.7	76.7	83.4	62.9	7.7	8.1	9.5	6.1	21.1	0.38	0.30	0.34	0.30	0.29

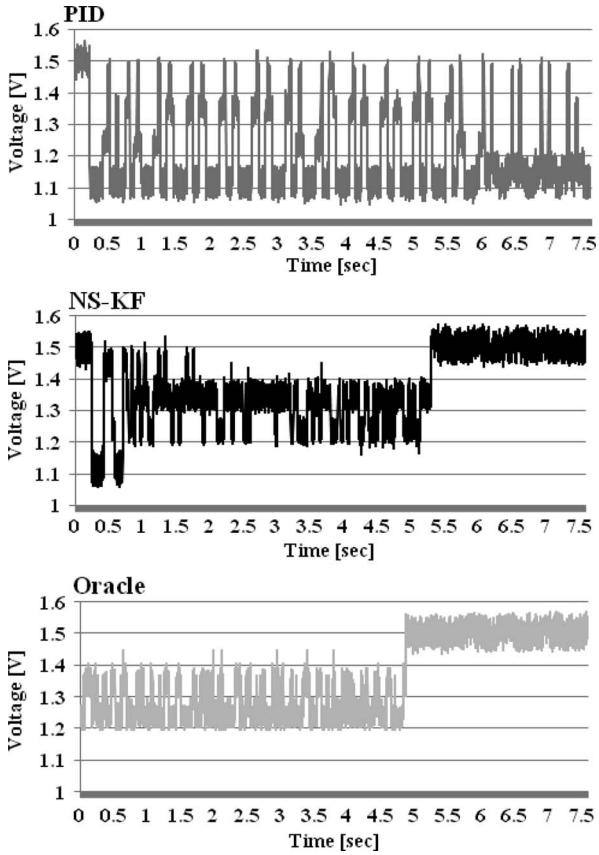


Fig. 11. Voltage selection and decoding time comparison of PID, NS-KF, and oracle method.

Otherwise (faster v/f pair than the ideal v/f pair), the overall EC will increase.

Fig. 11 shows our method, PID, and oracle method from the decoding time perspective.[2] First, the oracle method stays at around 1.5 V after 5 s, meaning that it completes the decoding of all frames and stays in idle state. Our method completes the decoding a little bit later than the oracle method (at approximately 5.3 s) and stays in idle state like the oracle

[2]We dynamically measured not only the voltage supplied to V_{dd} pin of PXA270 but also the current flowing into the V_{dd} pin of PXA270. The voltage measurement was used for Fig. 11, while the current measurement was used for Fig. 12.

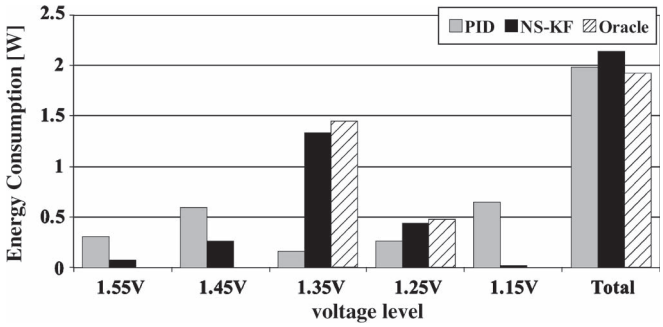


Fig. 12. EC in practical environment.

method. On the contrary, PID method completes the decoding at around 7.5 s, which is 2.5 s longer than the oracle method. In our case, the timing constraint of a single frame decoding is given as 33 ms; hence, it misses roughly 75 frames out of 150 frames if we chose a frame dropping scheme. Fig. 11 also shows how each method tracks the workload variation dynamically by seeing the voltage selection sequence. Obviously, the line shape of our method and that of oracle method are quite similar, whereas the line shape of PID is quite different. More specifically, PID frequently selects the maximum v/f pair right after selecting the minimum v/f pair (or vice versa), meaning that its decision is too much sensitive to the workload variation. On the other hand, our method tracks the workload in a stable form except for the initial periods. In other words, our method shows a lower estimation accuracy during the initial period due to the insufficient information. However, it shows very accurate and consistent workload estimation after the sufficient training period which is equivalent to the decoding time of tens of frames.

Finally, we related the workload estimation accuracy to the EC, as shown in Fig. 12. Fig. 12 shows the energy consumed at each v/f pair, as well as the total EC. As shown in Fig. 12, the oracle method rarely selects two maximum v/f pairs, while PID method consumes half of the total EC at these two pairs. Our method is in between these two methods, but its tendency is closer to the oracle method rather than PID. Even though the total EC of PID is lower (about 4%) than that of our method, our method is favored from the QoS perspective, since PID sacrifices the quality satisfaction for energy saving.

VII. Conclusion

We have proposed a novel DVS approach that can save EC significantly by estimating the processing time of workloads in a robust and accurate manner. This method exploits the Kalman filter that can estimate the future processing time based upon the previous processing time observation. We implemented an MPEG decoder that utilizes the proposed scheme for experimental studies. According to our results through both a cycle-accurate simulator and an Xscale-based test board, the EC of the MPEG decoder decreased by 57.5% on average with a negligible DMR of 6.1%. In addition, the proposed technique was superior to the competing methods we compared in terms of the ability to fulfill the given timing and QoS constraints. Our results indicate that the proposed method can indeed be a very effective means for accurate DVS.

References

[1] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer, 1997.

[2] B. Brock and K. Rajamani, "Dynamic power management for embedded systems," in *Proc. IEEE SoC Conf.*, 2003, pp. 416–419.

[3] E. Y. Chung, L. Benini, and G. D. Micheli, "Dynamic power management using adaptive learning tree," in *Proc. ICCAD*, 1999, pp. 274–279.

[4] Z. Ren, B. H. Krogh, and R. Marculescu, "Hierarchical adaptive dynamic power management," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 409–420, Apr. 2005.

[5] C. Gniady, A. R. Butt, Y. C. Hu, and Y.-H. Lu, "Program counter based techniques for dynamic voltage power management," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 641–658, Jun. 2006.

[6] A. Acquaviva, L. Benini, and B. Ricco, "An adaptive algorithm for low-power streaming multimedia processing," in *Proc. DATE*, 2001, pp. 273–279.

[7] E.-Y. Chung, L. Benini, A. Bogliolo, Y. Lu, and G. De Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1345–1361, Nov. 2002.

[8] C. Xian and Y. H. Lu, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," in *Proc. ACM Great Lakes Symp. VLSI*, 2006, pp. 392–397.

[9] S. Hong, S. Yoo, B. Bin, K. M. Choi, S. K. Eo, and T. Kim, "Dynamic voltage scaling of supply and body bias exploiting software runtime distribution," in *Proc. DATE*, 2008, pp. 242–247.

[10] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*. Chapel Hill, NC: Univ. North Carolina, Chapel Hill, 1995.

[11] D. Simon, "Kalman filtering," *Embed. Syst. Program.*, vol. 14, no. 6, pp. 72–79, Jun. 2001.

[12] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[13] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. Found. Comput. Sci.*, 1995, pp. 374–382.

[14] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," in *Proc. ISLPED*, 2001, pp. 34–39.

[15] R. Jejurikar and R. Gupta, "Energy-aware task scheduling with task synchronization for embedded real-time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1024–1037, Jun. 2006.

[16] D. Shin and J. Kim, "Dynamic voltage scaling of mixed task sets in priority-driven systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 3, pp. 438–453, Mar. 2006.

[17] A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy/power," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 2, pp. 270–276, Apr. 2003.

[18] S. Hua, G. Qu, and S. S. Bhattacharyya, "Energy reduction techniques for multimedia applications with tolerance to deadline misses," in *Proc. DAC*, 2003, pp. 131–136.

[19] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Des. Test. Comput.*, vol. 18, no. 2, pp. 20–30, Mar./Apr. 2001.

[20] D. Shin and J. Kim, "A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications," in *Proc. ISLPED*, 2001, pp. 271–274.

[21] J. Seo, T. Kim, and K. S. Chung, "Profile-based optimal intra-task voltage scheduling for hard real-time applications," in *Proc. DAC*, 2004, pp. 87–92.

[22] D. Shin and J. Kim, "Optimizing intra-task scheduling using data flow analysis," in *Proc. ASP-DAC*, 2005, pp. 703–708.

[23] A. C. Bavier, A. B. Montz, and L. L. Peterson, "Predicting MPEG execution times," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 26, no. 1, pp. 131–140, Jun. 1998.

[24] J. Pouwelse, K. Langendoen, I. Lagendijk, and H. Sips, "Power-aware video decoding," in *Proc. Picture Coding Symp.*, 2001, pp. 303–306.

[25] E. Nurvitadhi, B. Lee, C. Yu, and M. Kim, "A comparative study of dynamic voltage scaling techniques for low-power video decoding," in *Proc. Int. Conf. Embed. Syst. Appl.*, 2003, pp. 23–26.

[26] D. Son, C. Yu, and H. N. Kim, "Dynamic voltage scaling on MPEG decoding," in *Proc. ICPADS*, 2001, pp. 633–640.

[27] K. Choi, W. C. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for an MPEG player," *J. Low Power Electron.*, vol. 1, no. 2, pp. 27–43, Apr. 2005.

[28] K. Choi, R. Soma, and M. Pedram, "Off-chip latency-driven dynamic voltage and frequency scaling for MPEG decoding," in *Proc. DAC*, 2004, pp. 544–549.

[29] Y. Tan, P. Malani, Q. Qui, and Q. Wu, "Workload prediction and dynamic voltage scaling for MPEG decoding," in *Proc. Conf. ASPDA*, 2006, pp. 911–916.

[30] E. Y. Chung, L. Benini, and G. D. Micheli, "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," in *Proc. ISLPED*, 2002, pp. 42–47.

[31] K. Bang, S. Y. Bang, and E. Y. Chung, "Extended MPEG video format for efficient dynamic voltage scaling," *IEICE Trans. Fundam.*, vol. E91-A, no. 5, pp. 1283–1287, May 2008.

[32] C. J. Hughes and S. V. Adve, "A formal approach to frequent energy adaptations for multimedia applications," in *Proc. Int. Conf. Comput. Des.*, 2003, pp. 489–496.

[33] Y. Gu and S. Chakraborty, "Control theory-based DVS for interactive 3D games," in *Proc. DAC*, 2008, pp. 740–745.

[34] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE Micro*, vol. 25, no. 5, pp. 52–62, Sep./Oct. 2005.

[35] Z. Lu, J. Lach, M. Stan, and K. Skadron, "Reducing multimedia decode power using feedback control," in *Proc. Int. Conf. Comput. Des.*, 2003, pp. 489–496.

[36] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron, "Control-theoretic dynamic frequency and voltage scaling for multimedia workloads," in *Proc. Int. Conf. Compilers, Architecture, Synthesis Embed. Syst.*, 2002, pp. 156–163.

[37] *MPEG*. [Online]. Available: http://www.mpeg.org

[38] A. Sinha and A. P. Chandrakasan, "Dynamic voltage scheduling using adaptive filtering of workload traces," in *Proc. Int. Conf. VLSI Des.*, 2001, pp. 221–226.

[39] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for low power microprocessors under dynamic workloads," in *Proc. DAC*, 2004, pp. 721–725.

[40] [Online]. Available: http://carbondesignsystems.com/products_socd.shtml

[41] *Intel PXA27x Processor Family Power Requirements*, Intel Corporation, Santa Clara, CA, 2004.

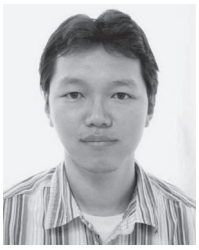
[42] *libmpeg-2*. [Online]. Available: http://libmpeg2.sourceforge.net

[43] *Embedded System HBE-SM2*. [Online]. Available: http://www.hanback.co.kr/htm/sub3_12.htm

[44] *The Movie Player*. [Online]. Available: http://www.mplayerhq.hu/design7/dload.html

Sung-Yong Bang received the B.S. and M.S. degrees from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea, in 2007 and 2009, respectively.

He is currently an Engineer with the China GSM R&D Group, Mobile Communication Division, Samsung Electronics, Suwon-City, Gyeonggi-do, Korea. His research interests include system-level low-power design, low-power techniques for mobile communication, and device drivers.



Kwanhu Bang (S'06) received the B.S. degrees in computer science and in electronic engineering and the M.S. degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2006 and 2008, respectively, where he is currently working toward the Ph.D. degree with the School of Electrical and Electronic Engineering.

His research interests include biocomputation, Flash memory applications, and system-level low-power design.

Sungroh Yoon (S'99–M'06) received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 1996 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 2002 and 2006, respectively.

From 2006 to 2007, he was with Intel Corporation, Santa Clara, CA, where he participated in developing Atom and Core i7 microprocessors. Previously, he held research positions at Stanford University and Synopsys Inc., Mountain View, CA. He is currently an Assistant Professor with the School of Electrical Engineering, Korea University, Seoul. His research interests include system-level low-power design, Flash memory applications, and biocomputation.

Eui-Young Chung (S'99–M'06) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2002.

From 1990 to 2005, he was a Principal Engineer with the System-on-a-Chip R&D Center, Samsung Electronics, Yongin, Korea. He is currently an Associate Professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul. His research interests include system architecture, biocomputing, and very large scale integration design, including all aspects of computer-aided design with special emphasis on low power applications and Flash memory applications.