Frame-Based Dynamic Voltage and Frequency Scaling for an MPEG Player

Kihwan Choi, Wei-Chung Cheng, and Massoud Pedram
Department of EE-Systems, University of Southern California, Los Angeles, CA90089
{kihwanch, wecheng, pedram}@usc.edu

Abstract — *This paper describes a dynamic voltage and frequency scaling (DVFS) technique for MPEG decoding to reduce the energy consumption while maintaining a quality of service (QoS) constraint. The computational workload for an incoming frame is predicted by using a frame-based history so that the processor voltage and frequency can be scaled to provide the exact amount of computing power needed to decode the frame. More precisely, the required decoding time for each frame is separated into two parts: a frame-dependent (FD) part and a frame-independent (FI) part. The FD part varies greatly according to the type of the incoming frame whereas the FI part remains constant regardless of the frame type. Separation of the FI part from the overall decoding sequence provides two key benefits depending on the hardware platform: better compensation of the error due to workload prediction and higher level of energy saving when given a QoS degradation level. The proposed DVFS scheme has been implemented on two platforms, a low performance StrongArm-1110-based evaluation board from Intel and a high performance XScale-based testbed designed at USC. In the StrongArm-1110-based system, the FI part is used to compensate for the prediction error that may occur during the FD part, whereas in the XScale-based system, the FI part is used to reduce energy consumption by employing the lowest CPU frequency during the corresponding time intervals. Detailed current measurements in these two platforms demonstrate larger than 87 % and 80 % CPU energy saving while maintaining a user-provided frame rate, respectively.*

Index Terms — Dynamic voltage and frequency scaling, MPEG decoding, workload decomposition.

I. INTRODUCTION

DEMAND for portable computing and communication devices has been increasing rapidly. Because portable devices are battery-operated, a design objective is to minimize the energy dissipation (and thus maximize the battery service time) without any appreciable degradation in the *QoS*. DVFS is a highly effective method to achieve this design goal. This is because energy consumption in CMOS VLSI circuits is quadratically proportional to the supply voltage. Therefore, reducing the supply voltage results in a large energy saving. Reducing the voltage level, however, slows the circuit down. The key idea behind DVFS techniques is to perform dynamic voltage scaling so as to provide "just-enough" circuit speed to process the workload while meeting the total compute time and/or throughput constraints, and thereby, reduce the energy dissipation.

Previous DVFS works can be divided into two categories, one for non real-time operation and the other for real-time operation. The most important step in implementing DVFS is prediction of the future workload, which allows one to choose the minimum required voltage/frequency levels while satisfying key constraints on energy and *QoS*. As proposed in [1] and [2], a simple interval-based scheduling algorithm can be used in non real-time operation. This is because there is no timing constraint. As a result, some performance degradation due to workload misprediction is allowed. The defining characteristic of the interval-based scheduling algorithm is that uniform-length intervals are used to monitor the system utilization in the previous intervals and thereby set the voltage level for the next interval by extrapolation. This algorithm is effective for applications with predictable computational workloads such as audio [3] or other digital signal processing intensive applications [4]. Although the interval-based scheduling algorithm is simple and easy to implement, it often predicts the future workload incorrectly when a task's workload exhibits a large variability. One typical example of such a task is MPEG decoding. In MPEG decoding, because the computational workload varies greatly depending on each frame type, repeated mispredictions may result in a decrease in the frame rate, which in turn means a lower *QoS* in MPEG.

There are also many studies to apply DVFS in real-time application scenarios [5][6][7][8]. In [5][6] the multi-task scheduling in the operating system (OS) is the focus. More precisely, the scheduling is performed so as to reduce energy consumption while meeting given hard timing constraints. In these coarse-grained DVFS approaches, it is assumed that the total number of CPU cycles needed to complete each task is fixed and known *a priori*. This is an assumption that is difficult to satisfy in practice. In [7], an intra-task voltage scheduling technique was proposed in which the application code is divided into many segments and the worst-case execution time of each segment (which is obtained from a static timing analysis) is used to determine a suitable voltage for the next segment. In [8] a method based on a software feedback loop was proposed. In this method, a deadline for each time slot is provided. The authors calculate the operating frequency of the processor for the next time slot depending on the slack time generated in the current slot and again the worst-case execution time of the next time slot. In these two fine-grained DVFS approaches, it is assumed that the worst-case execution time of each segment of a task is known. This assumption is again difficult to meet for many applications, for example, for MPEG decoding where the worst-case execution

time cannot be accurately determined on a per-frame basis. Note that a single worst-case execution time for all video frames (which may be calculated rather easily) will not be useful because it tends to be too pessimistic and therefore will significantly reduce the energy saving potential of a DVFS technique that uses it.

In this paper, an effective DVFS algorithm for MPEG decoding is proposed in which the future workload is predicted *on a per-frame basis*. This is accomplished by using a frame-type-based workload-averaging scheme where the prediction error due to statistical variation in the workload of the frame-dependent part of the decoder is effectively compensated for by using the frame independent part of the decoding time as a "buffer zone." This allows us to obtain a significant energy saving without any notable *QoS* degradation. This algorithm has been implemented on two different platforms: an Intel-designed StrongARM-1110 based for low performance and a USC-designed XScale-based high performance platform and has resulted in the CPU energy reduction of more than 87% and 80%, respectively.

Notice that when lowering the supply voltage to reduce energy consumption, the clock frequency should be decreased first to prevent timing failure due to the increased gate delay. Because a minimum voltage is assigned to each operating frequency value, in this paper, the term "voltage and frequency scaling" will be used rather than either "voltage scaling" or "frequency scaling."

The remainder of this paper is organized as follows. Related works on DVFS and MPEG are described in Section II. In Sections III and IV, a new DVFS algorithm is presented. Details of the implementation, including both hardware and software, are described in Section V. Experimental results and conclusions are given in Sections VI and VII, respectively.

II. BACKGROUND

A. Fundamentals of DVFS

Many kinds of application programs, which may require real-time or non real-time operations, are executed on a general-purpose processor. In general, DVFS techniques are very effective in reducing the energy dissipation while meeting a performance constraint in real-time applications such as video decoding. The energy consumption per task running on a CMOS VLSI circuit is given by the following well-known equation [9]:

$$E = C_{switched} \cdot V^2 \cdot f_{clk} \cdot T \qquad (1)$$

where V is the supply voltage level, $C_{switched}$ is the switched capacitance per clock cycle, f_{clk} is the clock frequency, and T is the total execution time of the task.

Fig. 1 illustrates the basic concept of DVFS for real-time application scenarios. In this figure, T_2 and T_4 denote deadlines for tasks W_1 and W_2, respectively (in practice, these deadlines are related to the *QoS* requirements.) W_1 finishes at T_1 if the CPU is operated with a supply voltage level of V_1. The CPU will be idle during the remaining (slack) time, S_1. To provide a precise quantitative example, let us assume T_2-T_0=T_4-T_2=ΔT, and T_1-T_0=ΔT/2; the CPU clock frequency at V_1 is f_1=n/ΔT for some integer n; and that the CPU is powered down or put into standby with zero power dissipation during the slack time. The total energy consumption of the CPU is $E_1 = CV_1^2 f_1 \Delta T/2 = nCV_1^2/2$ where C is the effective switched capacitance of the CPU per clock cycle. Alternatively, W_1 may be executed on the CPU by using a voltage level of $V_2=V_1/2$, and is thereby completed at T_2. Assuming a first-order linear relationship between the supply voltage level and the CPU clock frequency, $f_2=f_1/2$. In the second case, the total energy consumed by the CPU is $E_2=CV_2^2 f_2 \Delta T=nCV_1^2/8$. Clearly, there is a 75 % energy saving as a result of lowering the supply voltage (this saving is achieved in spite of "perfect" – i.e., immediate and with no overhead - power down of the CPU). This energy saving is achieved without sacrificing the *QoS* because the given deadline is met. An energy saving of 89 % is achieved when scaling V_1 to $V_3=V_1/3$ and f_1 to $f_3=f_1/3$ in case of task W_2.

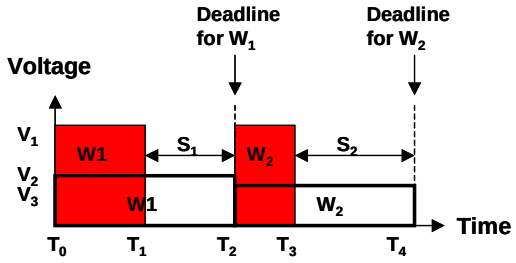


Fig. 1. An illustration of the DVFS technique

A major requirement for implementation of an effective DVFS technique is accurate prediction of the time-varying CPU workload for a given computational task. A simple interval-based scheduling algorithm is employed in [10] to dynamically monitor the global CPU workload and adjust the operating voltage/frequency based on a CPU utilization factor, i.e., decrease (increase) the voltage when the CPU utilization is low (high). Two prediction schemes have been used in interval-based scheduling: the moving-average (MA) and the weighted-average (WA) schemes [10]. In the MA scheme, the next workload is predicted based on the average value of workloads during a predefined number of previous intervals, called window size. In the WA scheme, a weighting factor, ω, is considered in calculating the future workload such that severe fluctuation of the workload is filtered out, resulting in a smaller average prediction error. Their operations are represented in the following equations.

MA :

$$WindowSize = n$$

$$Workload(t+1) = \begin{cases} \dfrac{\sum_{\tau=0}^{n-1} Workload(t-\tau)}{n}, & t \geq n-1 \\ \dfrac{\sum_{\tau=0}^{t} Workload(t-\tau)}{t+1}, & otherwise \end{cases} \qquad (2)$$

WA :

$$
\begin{aligned}
&Workload_{avg}(0) \equiv Workload(0) \\
&Workload(t+1) = \varpi \cdot Workload(t) + (1-\varpi) \cdot Workload_{avg}(t) \\
&\quad = (1-\varpi)^{t} \cdot Workload(0) + \varpi \cdot \sum_{\tau=0}^{t-1} (1-\varpi)^{\tau} \cdot Workload(t-\tau)
\end{aligned} \tag{3}
$$

These two workload prediction schemes are easy to implement and result in effective DVFS algorithms when the workload fluctuation is not too severe. To illustrate this point, two popular software applications, MP3 and MPEG playback, were tested using the WA scheme. Experimental results are shown in Fig. 2 and Fig. 3. Fig. 2 shows the CPU usage measured during each time interval (300 msec) whereas Fig. 3 depicts the workload prediction errors for both cases.

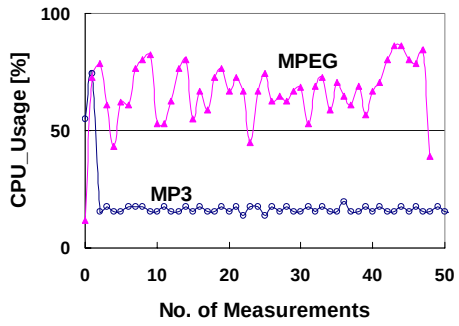


Fig. 2. CPU usage of MP3 and MPEG

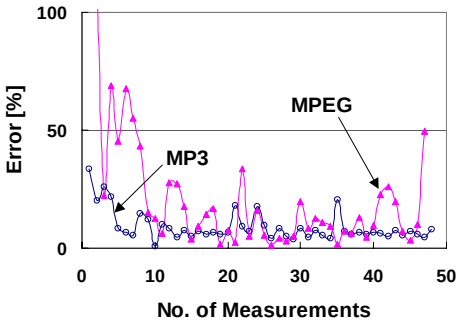


Fig. 3. Workload prediction error

These results show that interval-based voltage scaling which solely depends on the global state of the system is quite effective for the MP3 playback where the workload variation is rather small. On the other hand, it becomes ineffective (see the large prediction errors) for MPEG decoding due to the large variation in the CPU workload for this application. More precisely, the global system status monitoring interval-based DVFS algorithm for MPEG decoding cannot track the workload variation, resulting in a significant *QoS* degradation such as frame rate fluctuation.

B. MPEG Terminology

An MPEG video stream consists of three frame types: *I-frame* (Intra-coded), *P-frame* (Predictive-coded), and *B-frame* (Bi-directionally-coded). I-frames can be decoded independently. P-frames have to be decoded based on the previous frame. B-frames require both the previous and the next frames in order to be decoded. Sequences of frames are grouped together to form a *Group of Pictures* (GOP). A GOP contains 12-15 frames, starting with an I-frame. It takes several steps to decode each frame: *Parsing*, *Inverse Discrete Cosine Transformation* (IDCT), *Reconstruction*, *Dithering*, *and Display* [11]. Among these steps, the IDCT and Reconstruction take up half of the decoding time [12]. The IDCT is CPU-intensive (i.e., requires iterative multiplication-accumulation computation over an 8 × 8 array of integer or floating-point values) whereas the reconstruction, dithering, and display steps are memory-intensive (i.e., require data movement between the processed video stream and display frame buffer). Each frame type results in a different workload during the IDCT and reconstruction step, meaning that the execution time of different frame types varies by a large amount while the time for dithering and display steps is same over all types of frames. Based on these observations, the decoding process may be divided into two parts: a frame-dependent part (parsing, IDCT and reconstruction) and a frame-independent part (dithering and display) as shown in Fig. 4. The operations performed during frame-independent part are off-chip transactions such as SDRAM access and write to frame buffer of display device. CPU is stalled until these operations are finished without doing any useful work. Execution time during frame-independent is constant over all frame types in a given video stream and this property can be useful in the implementation of effective DVFS.

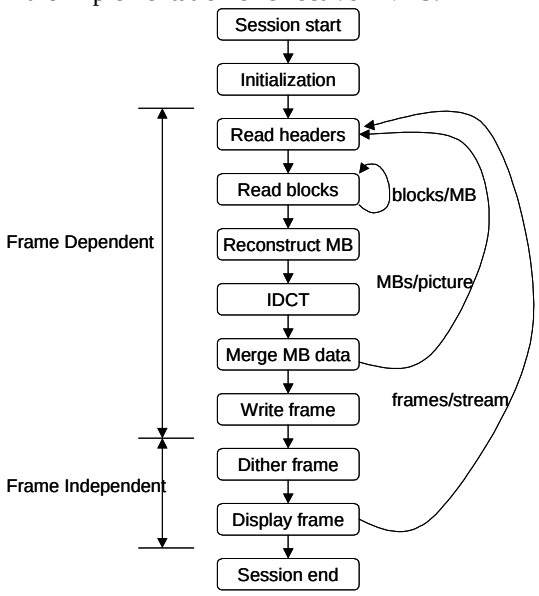


Fig. 4. MPEG decoding sequence

C. Prior Work

As stated in the previous section, it is very difficult to accurately predict the execution time of each frame in MPEG decoder due to the high variability in the computational workload of each frame.

A number of researchers have applied DVFS to MPEG video decoding in order to achieve lower energy consumption. In [10] and [13], DVFS is performed based on the ratio of the number of idle and busy cycles of the CPU while the MPEG stream is decoded. Although significant energy reduction is achieved, there is no guarantee that deadline for each frame is met because this interval-based prediction technique cannot

capture large fluctuations in decoding time for each frame. In [14], the authors studied the empirical relationship between the decoding time and the code size of each frame (i.e., the number of bits in the encoded frame). Their results showed a strong correlation between these two parameters with an error of less than 25%. However, the code size of each frame cannot be obtained before starting the IDCT step. To overcome this limitation, a method using feedback control was proposed in [15] in which the set of macro blocks[1] in a frame are first divided into two groups, each containing half of the macro blocks. The authors assume a static (fixed) relationship (in the form of a linear equation) between the IDCT time and code size of each group of macro blocks. By analyzing the first group of macro blocks, they obtain the code size of that group. Next, they assume that the code size of the second group of macro blocks is the same as this value and calculate the IDCT time of the second group based on the IDCT time and this code size. This code size prediction scheme is however inaccurate and may result in frequent deadlines misses. Furthermore, the linear prediction equation must be changed when different resolutions of the video image or different frame pixel sizes are encountered.

In [16], the estimation of decoding time is performed in units of *GOP*. In this approach, sizes and types of the frames of an incoming *GOP* are observed and the time needed to decode the next GOP is estimated based on statistics of the previous *GOP*s. This approach also suffers from a rather high occurrence of mispredictions. Furthermore, even more severe *QoS* degradation may occur when the prediction is inaccurate because the same frequency (voltage) is applied for all frames in a GOP. There is a different approach in which the decoding time prediction is not needed [17]. This is accomplished by including the execution time information of each frame to the video content itself (e.g., as part of the frame header). However, this approach adds to the computational workload of the video encoding. As acknowledged in the same reference, this approach is only worthwhile if the encoded video stream is sent to many clients so that the extra cost of adding decoding time information to the frame headers is compensated by energy savings on many mobile clients. In addition, this scheme requires modification of currently used standard video stream format.

In [18], an application-independent DVFS approach, *Vertigo*, was proposed, which uses multiple performance-setting algorithms that are organized into a decision hierarchy for various types of applications. This algorithm was applied to MPEG decoding. Vertigo is an interval-based approach where workload in the next time interval is estimated based on the history of previous intervals. This algorithm estimates the deadline for each interval based on the estimated workload in the previous intervals. This approach is different from [10] and [13] where the length of each time interval is fixed. The authors compared Vertigo with LongRun policy [19] and reported that Vertigo has higher performance in terms of the match between the actually achieved frame rate and the target frame rate. Note, however, that actual frame rates with Vertigo are still far less than the target frame rate, i.e., the actual times are 17% to 30% shorter than the target times), which will in turn result in lower energy saving.

There have been a number of studies on using buffers in multimedia processing. One of the most important advantages of using buffers is that no explicit frame decode time prediction is needed, thus missed deadlines due to prediction errors are avoided. Reference [20] used an off-line algorithm to schedule the frame decoding rate and respective frequency, and they did not consider multimedia streams that include B frames. Reference [21] focused on the estimation of the input/output buffer size for the decoder and it is assumed that the worst case execution time is known in advance. In [22] a feedback control scheme using PI controller at the decoder output buffer such that constant frame rate is achieved by monitoring the occupancy of the frame buffer. But, it is difficult to control the gain of PI controller and a slight mismatch in the controller gain might cause underflow/overflow at the buffer. Also, frequency/voltage setting is linearly subdivided into 40 discrete levels, which is not true in actual situation.

Techniques that utilize buffers introduce some amount of delay when a video session starts due to buffer filling as well as severe modification of application source code itself to implement a control scheme. In spite of these problems, buffers can be quite useful. In this paper, we have not considered the effects of the buffers, which would make the deadline for processing any frame a soft deadline. This will have significant implication on the proposed DVFS approach, but falls outside the scope of the present paper. More precisely, we expect that with buffers, one can achieve even higher energy savings by dynamically changing the deadline for processing each frame without overflowing the buffer.

To develop an effective DVFS technique for low-energy MPEG decoding, two prerequisites must be met: existence of an accurate *workload prediction method* and availability of *error compensation methods* for handling the case when a prediction error occurs. Prediction error compensation is important in MPEG decoding because a certain level of video quality, such as frame rate, should be guaranteed as well as energy reduction. However, most approaches concentrate on the prediction only, not the prediction error compensation method. This lack of prediction error compensation method comes from the fact that the characteristics of each step in decoding sequence were not considered carefully, FD part only not FI part. In this paper, a DVFS technique for low-energy MPEG decoding, in which a frame-based workload prediction and two effective prediction error compensations, intra-frame compensation and inter-frame compensation, are provided and these two compensation techniques can be used either individually or together depending on the used hardware. Intra-frame compensation uses the FI part as a “buffer zone” to

[1] A macro block corresponds to a 16 by 16 pixel area of the original image and consists of six 8 by 8 blocks on which IDCT is performed.

recover prediction errors in FD part, whereas inter-frame compensation uses "error diffusion" to distribute a prediction error in a frame to following frames such that the error is localized, resulting in smooth variation in the frame rate at run time. If prediction error can be recovered effectively, then there is a lower requirement for accurate prediction technique.

In this paper, we propose a DVFS method for an MPEG decoding which enables easy implementation with the least modification of MPEG decoder program. It is assumed that there is no display buffer, i.e., a frame should be decoded and displayed in a given time, determined by a frame rate. Also, our method uses no special hardware such as dynamic memory access (DMA) to perform display operation, which is common in portable mobile application.

We mention here that if the FD and FI decoding steps are pipelined and performed in parallel, then the proposed intra-frame compensation technique may not be effective. However, to achieve such pipelined operation of the FD and FI decoding steps, special hardware such as DMA is required. In addition, significant modification of the application source code as well as the interrupt handler in OS kernel will be needed. In this work, we have targeted a DVFS method for a software MPEG decoder, which can run on any computer system without DMA support and without any modification of MPEG decoder source code or the OS interrupt handler (with the exception of some code to predict the workload and to find the optimal voltage setting.)

III. DECOMPOSITION OF MPEG DECODING SEQUENCE

As stated previously, the decoding process of a frame is divided into two parts based on the required execution time and the expected energy consumption. One part captures the frame-dependent (FD) portion of the decoding process whereas the other part captures the frame-independent (FI) portion of the decoding process as shown below:

$$T_{Decoding} = T_{FD} + T_{FI};\ \ E_{Decoding} = E_{FD} + E_{FI} \qquad (4)$$

where $T_{Decoding}$ is the whole decoding time of a frame, T_{FD} and T_{FI} are the elapsed time during the FD and FI parts, respectively, and E_{FD} and E_{FI} are the CPU energy consumption during FD and FI, respectively.

The parsing, IDCT and reconstruction steps are included in the frame-dependent time whereas the dithering and display steps are included in the frame-independent time. A large variation in decoding time for each frame is caused by variation in frame-dependent time, not by frame-independent time. This is because the dithering and display time are dependent upon the frame pixel size and are otherwise constant for a given video stream.

To determine the FD and FI times for a given frame, the source code for a software MPEG decoder, i.e., ***mpeg_play*** [23], was modified, and a timestamp function was inserted at each decoding step. The measurement was performed on SA1110-based platform using a test video clip. Fig. 5 shows the FD and FI time distributions for each frame when playing MPEG with a frames-per-second (fps) rate of 2. Fig. 6 depicts the same distributions for the maximum fps rate that the CPU can sustain (as high a fps rate as the CPU can sustain). In Fig. 5, with fps = 2, the deadline is fixed at 0.5sec. Considering that decoding sequence of (IBBPBBPBBPBBPBB) for a GOP in Fig. 5 and Fig. 6, one can observe that the FD time varies greatly depending on the frame type and that it is longer for the I-frames and shorter for the B-frames.

In Fig. 6, where a frame rate is not set, the decoding time varies depending on the frame type. Here the FI time is constant (~50 msec at the maximum clock frequency of 206 MHz). Notice that there is a large amount of slack in the FI time in Fig. 5. Furthermore, notice that although the FD time varies considerably depending on the frame type, the FI time is nearly constant for a given frame type (the FI time depends on the pixel size of the given movie stream, which is obviously constant for the same movie.) These plots provide empirical evidence of the claims made earlier with regards to the FD and FI parts of the decoding steps and their relationship to the frame type.

The typical operations performed in the FI part are memory-intensive. Examples include reading and writing the pixel data in the dithering step and wiring the decoded frame data to the frame buffer in the display device. These operations result in many CPU stalls for the external memory transactions to be completed. The time required for a memory transaction is directly related to the memory clock frequency. Depending on the hardware design, memory bus clock frequency may or may not be affected by the CPU frequency scaling. For example, in the SA1110-based system the memory access timing is changed in lockstep with the CPU frequency [15] whereas in the XScale-based system [24], the memory bus frequency is determined by the external memory controller component independently of the CPU frequency.

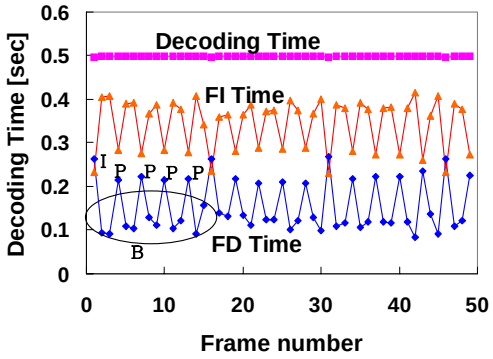


Fig. 5. Decoding time with fps = 2

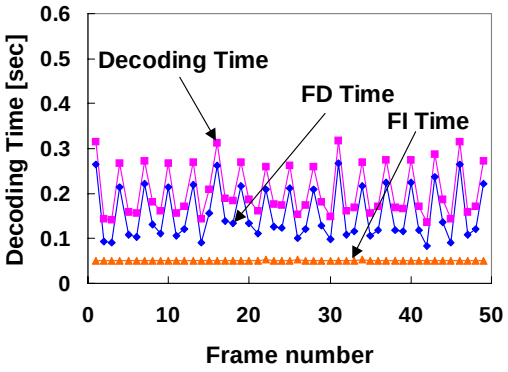


Fig. 6. Decoding time without setting any fps rate (as high a fps rate as the CPU can sustain)

In Fig. 7, actual execution times of the FD and FI parts for each frame type are shown with changing CPU frequencies in the SA1110-based and XScale-based systems. Notice that in case of the SA1110-based system the FI time decreases as the CPU frequency increases (Fig. 7 (a)), whereas in the XScale-based platform the FI time is nearly constant (Fig. 7 (b)). The rates of decoding time ($\Delta T_{Decoding}$) and energy consumption ($\Delta E_{Decoding}$) for processing a frame as a function of the CPU frequency change in both target systems are given below:

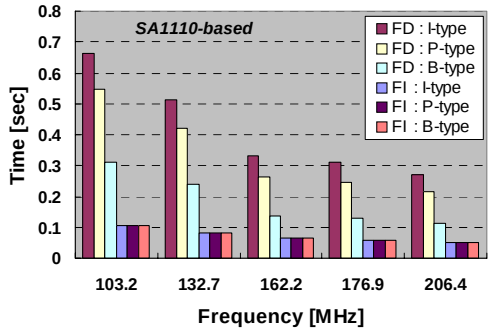
SA1110-based：

$$\Delta T_{Decoding} = \Delta T_{FD} + \Delta T_{FI};\ \ \Delta E_{Decoding} = \Delta E_{FD} + \Delta E_{FI} \qquad (5)$$

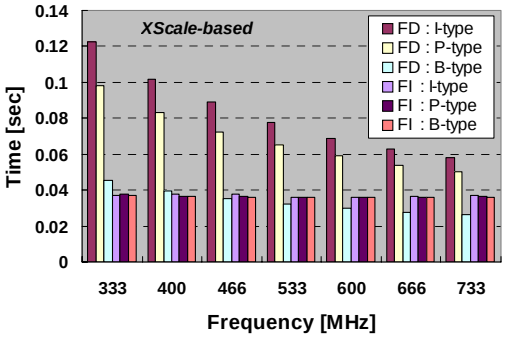
XScale-based：

$$\Delta T_{Decoding} = \Delta T_{FD},\ (\frac{\Delta T_{FI}}{\Delta f} = 0);\ \ \Delta E_{Decoding} = \Delta E_{FD} + \Delta E_{FI} \qquad (6)$$

From these observations, it can be seen that the FI part can be used as a “timing buffer zone” in the SA1110-based system and an “energy saving means” in the Xscale-based system. More precisely, in the SA1110-based system where the memory access time varies according to the CPU frequency, the FI part can used as a kind of timing buffer zone to compensate for the prediction error of T_{FD} because the workload during T_{FI} is constant, and T_{FI} can be adjusted by changing the CPU frequency. In the XScale-based system where memory clock is set independently of the CPU clock frequency, the CPU frequency during the FI part can be set to its lowest allowed value without causing an increase in the latency, resulting in significant amount of energy saving. However, note that in this case the timing error in the FD part cannot be compensated in the FI part.



(a) SA1110-based platform



(b) XScale-based platform

Fig. 7. FD and FI time variation over CPU frequencies on two different platforms

IV. WORKLOAD PREDICTION AND ERROR COMPENSATION

A. *Workload prediction*

A DVFS algorithm for low-power MPEG decoding with large workload variation is presented in this section. The frame-dependent time prediction is performed by maintaining a moving-average of the frame-dependent time for each frame type (three averages, one per frame type). The frame-independent time is not predicted since it is constant for a given video sequence as explained in the previous section. The expected frame-dependent time for an incoming frame is thus determined based on the moving average for the appropriate frame type. The effectiveness of the proposed frame-based workload prediction scheme is verified by calculating the prediction error ratio in B-frames, which usually exhibit the largest variation among the frame types. For the prediction, we tested both MA and WA scheme with different test video clips and found that both schemes showed similar prediction accuracies. So, we chose the MA scheme with a window size of six for the prediction. Results are shown in Fig. 8. The movie clip used in the experiment has 660 frames (320 × 240) including I-, P-, and B-type frames. Based on the measured FD time, the prediction error was calculated. Prediction errors for I-, P-, and B-type were 5 %, 3 %, and 10 %, respectively. In practice, because of the way the predictor function is constructed and the dynamic nature of its updating, the probability of such an occurrence is very small. However, these error rates could be different according to movie type, so

it is required to compensate the prediction error such that energy saving is maximized while a given deadline is kept. We considered two methods for prediction error compensation by separating the FD and FI parts; *intra-frame compensation* and *inter-frame compensation*.

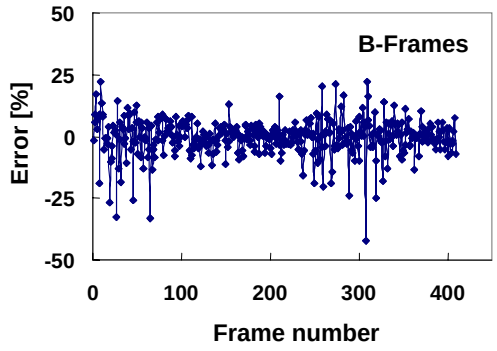


Fig. 8. Errors in B-frame workload prediction

We point out that the FI workload is always constant independent of the hardware platform. Now, the main motivation for intra-frame compensation in the SA1110-based system is that we can adjust the FI time by changing the CPU frequency. This is possible because the FI workload can easily be known after decoding only one frame. For the XScale-based system, the constant FI workload results in a constant FI time due to the asynchrony between the off-chip memory access and the CPU as explained in the text. In the XScale developer's manual, it is stated that, for stable operation, the CPU frequency should be at least three times larger than the memory clock frequency (100MHz). This is a platform design requirement and is the reason that the minimum CPU frequency is 333 MHz instead of the 266 MHz when the memory clock frequency is 100 MHz. For any valid setting of cpu and memory clocks in the XScale-based platform, we have observed a constant FI time.

B. Intra-frame compensation with frequency dependent FI

Intra-frame compensation method recovers FD time prediction error inside that frame itself, i.e., during FI part, such that the decoding time of each frame can be maintained as a given frame rate. For the implementation of intra-frame compensation method, it is required that the FI time should be varied as CPU frequency changes and this method can be applied to SA1110-based system. Prediction error in the FD part is compensated in the following FI zone by changing the CPU frequency/voltage. This is possible because the workload of the FI part is constant for a given video stream and easily obtained after decoding the first frame. The basic operation of the proposed intra-frame DVFS algorithm is shown in Fig. 9.

The FD part comes first. Based on the frame type and the prediction of the required time for the FD part, voltage/frequency scaling is performed to minimize energy dissipation while meeting the predicted time. When a misprediction occurs (which is detected by comparing the predicted FD time with the actual FD time), an appropriate action must be taken during the FI part to minimize the impact of the misprediction. If the actual FD time was smaller than the predicted value, there will be no *QoS* degradation. Hence, we can scale down voltage during the FI time and further save energy while meeting the deadline (cf. "Over-predicted" of Fig. 9). On the other hand, if the actual FD time was larger than the predicted value, corrective action must be taken to preserve the required *QoS*. This is accomplished by scaling up the voltage and frequency during the FI part so as to make up for the lost time (cf. "Under-predicted" of Fig. 9).

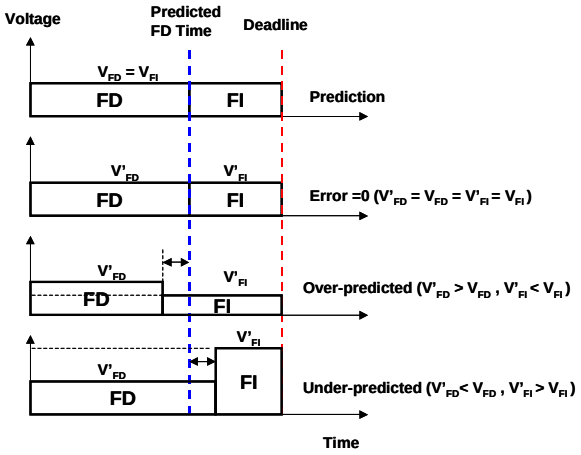


Fig. 9. Intra-frame compensation

To meet a given frame rate without deadline miss when FD workload is under-predicted, the following condition should be met;

$$T_{Decoding} = \frac{W_{FD}}{f_{FD}} + \frac{W_{FI}}{f_{FI}} \le D \Rightarrow \quad \varepsilon \le \frac{W_{FI}}{W_{FD}} \cdot \left(1 - \frac{f_{FD}}{f_{FI}}\right) \text{ where } f_{FD} = \frac{W_{FD} \cdot (1-\varepsilon) + W_{FI}}{D} \tag{7}$$

where D is deadline of a frame to be decoded, given as an inverse of a given frame rate, W_{FD} is actual workload of the FD part, ε is error rate in the workload prediction of the FD part, W_{FI} is a constant workload during FI, f_{FD} and f_{FI} are CPU frequency during the FD and FI parts, respectively.

In order to find out how much prediction error in the FD part can be recovered by using intra-frame compensation, the maximum tolerable error rate (ε_{max}) was considered by setting f_{FI} as f_{max} and it is given as follows;

$$\varepsilon_{\max} = \alpha + \frac{1}{1-\beta} \tag{8}$$

where α and β are defined as the ratio of W_{FI}/W_{FD} and $D \cdot f_{max}/W_{FI}$, respectively, and f_{max} is the maximum CPU frequency supported.

α value is different from different video sequence as well as different frame types of a given video clip and this intra-frame scheme cannot guarantee that one will never encounter a *QoS* degradation because it is possible that the under-prediction of the time needed for the FD part is so large that even the highest voltage/frequency level for the FI part is unable to make up for the lost time. This problem usually occurs more frequently in I-type frames (lower α) compared to P- or B-type frames (higher α), in the higher resolution video (lower α)

than lower resolution one (higher α), and at higher frame rate than at lower frame rate.

We performed measurements of both FD and FI workloads using six different video clips and calculated average error rates for each frame type are summarized in Table I. Based on the measurement results, ε_{max} for each test video is calculated as a function of deadline, i.e., frame rate, α, and W_{FI} as shown in Fig. 10. In Fig. 10, positive ε_{max} represents the maximally allowed FD prediction error, whereas negative ε_{max} value means that any prediction error (under-prediction) cannot be recovered using intra-frame compensation and causes deadline-missed frame. ε_{max} increases as frame rate decreases and α value increases and it is found that intra-frame compensation method using the FI part as timing buffer is quite useful for test video (6). For this video sequence, up to 100 % prediction error in the FD part can be recovered during the FI part at frame rate 1 and all frames would not miss the deadline considering that the maximum prediction error rate of this video is 13.39 in Table I. As we can see in Fig. 10, however, intra-frame compensation cannot guarantee stable frame rate for all kinds of video sequences and it is required another compensation technique called *inter-frame compensation* in which prediction error effect can be minimized.

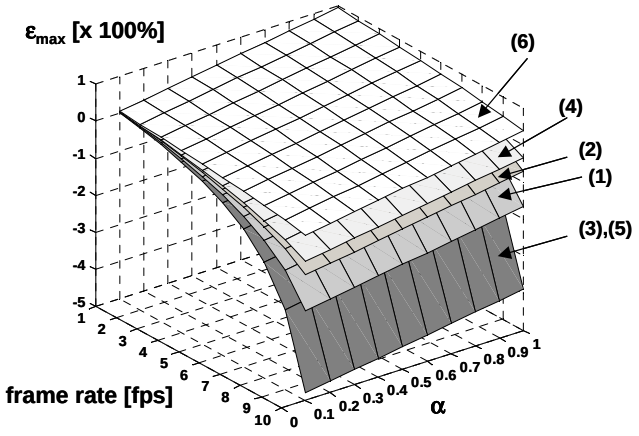
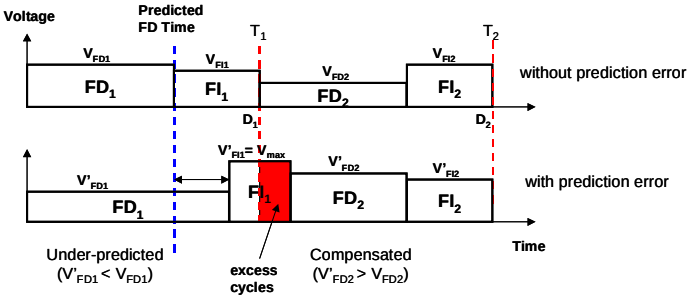


Fig. 10. The maximum tolerable error rate for different video sequences

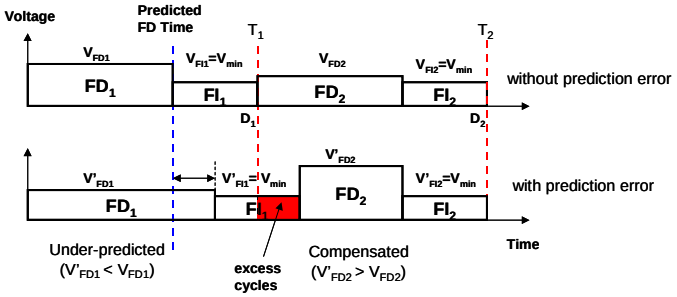
C. *Inter-Frame Prediction Error Compensation*

The concept of proposed inter-frame compensation technique is similar to that of considering "excess cycles" in the interval-based workload prediction techniques [1][2]. "Excess cycles" may be generated as a result of inaccurate estimation of the workload for the next time interval. They are positive (negative) if the selected CPU speed is lower (higher) than the required speed for the next time interval. The same situation can arise during the MPEG decoding. In particular, for the current frame, we calculate the number of excess cycles due to an incorrect CPU speed setting and make use of this information to set the CPU frequency for the next frame. There is a commonly used technique in video rendering called *error diffusion* [25] in which the quantization error of previously quantized pixel is filtered and distributed forward to unquantized pixels such that much smoother image can be achieved. This technique can also be used for inter-frame error compensation and helps eliminate severe fluctuations in the video frame rate due to the prediction errors.

When a prediction error (under-prediction or over-prediction) occurs in the FD part of a frame, this error propagates to the subsequent frames and may cause an overall frame rate degradation (for the under-predicted decoding time) or energy waste (for the over-predicted decoding time) unless this error is compensated in following frame slots. Fig. 11 illustrates the operation of the proposed inter-frame compensation technique for both the SA1110-based and the XScale-based systems. More precisely, the amount of error encountered in one frame is *diffused* over the succeeding frames whereby the CPU frequencies of the following frames are calculated by considering both their individual predicted workload averages and the number of transferred prediction error cycles from the previous frames. This *error diffusion* makes the prediction error to be *localized* into a small number of ensuing frames and allows it to be effectively compensated for by increasing (decreasing) the CPU frequency in case of over-prediction (under-prediction), resulting in smooth and graceful change in the frame rate and/or higher energy saving. As mentioned before, the I-type frame is the most sensitive to under-prediction error because it needs the largest time for the FD part. Notice that considering the typical frame sequence in the MPEG decoding where an I-type frame is followed by B-type or P-type frames, under-prediction error in an I-type frame can be very well compensated in the subsequent frames. The proposed error diffusion technique may of course become ineffective if there are long sequences of under-predicted (or over-predicted) frames. However, in practice, this is an unlikely scenario to occur.



(a) SA1110-based system



(b) XScale-based system

Fig. 11. Illustration of the inter-frame compensation technique

The CPU frequency for the FD part of (i+1)st frame, f^{i+1}_{FD}, is calculated as follows:

$$f_{FD}^{i+1} = \frac{W_{FD}^{avg}}{D - T_{err}^{i} - T_{FI}^{i+1}}$$
$$T_{err}^{i} = \begin{cases} 0, & \text{if } (T_{FD}^{i} + T_{FI}^{i}) = D \\ (T_{FD}^{i} + T_{FI}^{i}) - D, & otherwise \end{cases} \quad (9)$$

where W^{avg}_{FD} is the average FD workload for the frame that is of the same type as the (i+1)st frame, T^{i}_{err} is the time difference between a target deadline D and the actual decoding time for the ith frame. D is in turn calculated as an inverse of the target frame rate.

D. Simulation results

In order to verify the effectiveness of proposed error compensation method, we performed simulations using profiled data of the FD and FI parts for six different video sequences. In this simulation, it is assumed that overhead for scaling is negligible.

Energy consumption during the whole video session, E, is calculated from the operating voltage and frequency values of both SA1110-based and XScale-based systems as follows:

$$E \triangleq \sum_{i=1}^{N} \left[C_{FD} \cdot \left(v_{FD}^{i}\right)^2 \cdot f_{FD}^{i} \cdot T_{FD}^{i} + C_{FI} \cdot \left(v_{FI}^{i}\right)^2 \cdot f_{FI}^{i} \cdot T_{FI}^{i} \right] \quad (10)$$

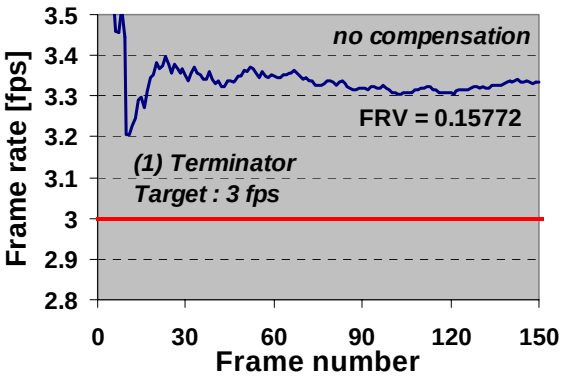
where N is the total number of displayed frames, C_{FD} (C_{FI}) is the *effective switched capacitance* during FD (FI), $T_{FD}{}^{i}$ ($T_{FI}{}^{i}$)is the elapsed time for FD (FI) of the ith frame, $v_{FD}{}^{i}$ ($v_{FI}{}^{i}$) and $f_{FD}{}^{i}$ ($f_{FI}{}^{i}$) are CPU operating voltage and frequency set during FD (FI) for ith frame, respectively.

C_{FD} and C_{FI} are in general different. Furthermore, we expect that C_{FI} is lower than C_{FD} because the CPU is stalled during off-chip memory accesses which are commonplace during FI. Based on the measured CPU power dissipation in the XScale-based system, the ratio of C_{FI} to C_{FD} is calculated as ~0.76 at the maximum CPU frequency of 733 MHz. In our experimental results, however, we set this ratio to one.

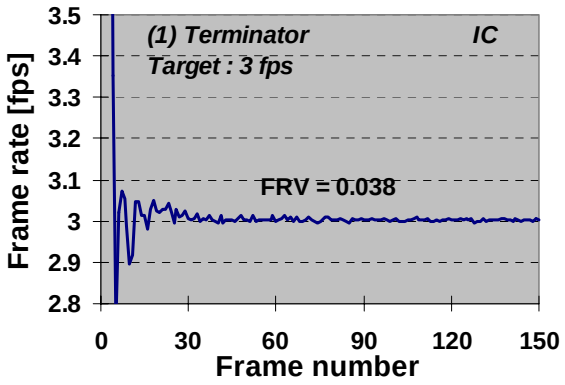
As a criterion to decide about the effectiveness of error compensation methods, we used the frame rate variance (FRV) at run time. FRV is defined as follows:

$$FRV \triangleq \frac{\sum_{i=1}^{N} \left(fps_{act}^{i} - fps_{target}\right)^2}{N}, \quad fps_{act}^{i} \triangleq \frac{i}{T^{i}} \quad (11)$$

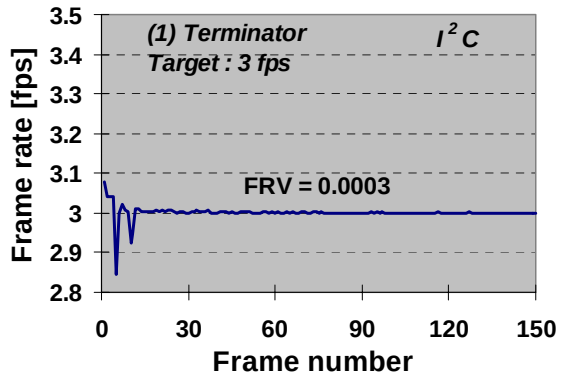
where fps_{target} denotes the target frame rate. FRV represents the frame rate fluctuation during a whole session and it is one of key *QoS* factors in video applications. For the proposed DVFS policy we considered the following three cases: applying 1) no compensation (NC) 2) inter-frame compensation only (IC) 3) both intra-frame and inter-frame compensation (I^2C). Simulation results for SA1110-based system with the test movie (1) are depicted in Fig. 12. From this figure, it can be seen that IC and I^2C result in much lower FRV values compared to the NC case and achieve the target frame rate (3 fps) after some "warm-up time". This warm-up time is needed to collect statistics about the decoding time of various frame types so that the workload prediction for the next frame becomes accurate. Note that the NC case results in a frame rate of 3.3 fps which is larger than target frame rate, 3fps, resulting in higher energy expenditure. In addition, the FRV is much larger for the NC case compared to the IC and I^2C cases. Finally, as expected the FRV is lower for I^2C case, lower frame rate fluctuation can be obtained due to intra-frame compensation, resulting in more stable frame rates.



(a) NC



(b) IC



(c) I^2C

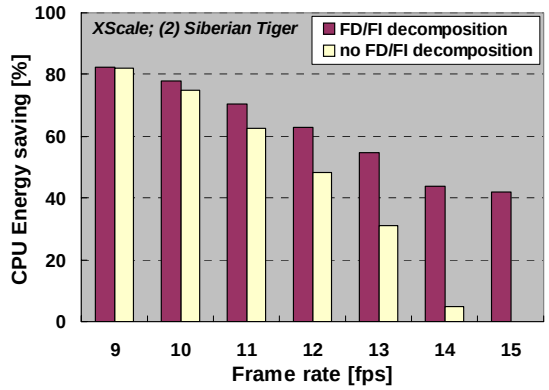
Fig. 12. Frame-rate fluctuation comparison : NC, IC, and I^2C

The FRV values for other test videos are summarized in Table II. Energy savings for all test videos are calculated with respect to the case without any DVFS and are summarized in Table III. Both IC and I^2C achieve nearly the same degree of energy saving for the test videos (more than 80 % saving at

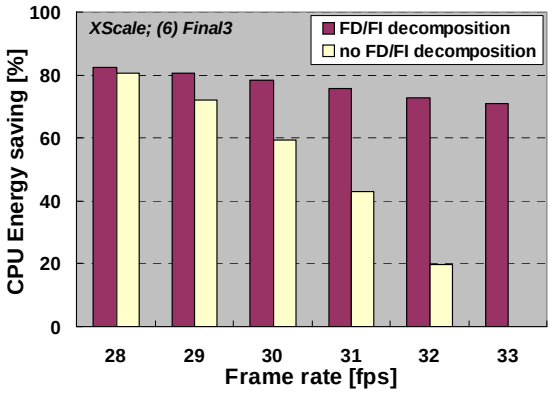
low target frame rates.) For the XScale-based system, we only used inter-frame compensation because the FI time is constant for all frequencies as mentioned earlier.

Energy saving comparisons between two scenarios (i.e., with and without FD/FI decomposition) are presented for two different test movies in Fig. 13. Notice that the energy savings with or without FD/FI decomposition are calculated with respect to the case of no DVFS. All other energy saving data reported in subsequent figures or tables are also calculated in the same way.

For the FD/FI decomposition scenario, the minimum frequency is set during the FI part while for the scenario without FD/FI decomposition, the same frequency is used for the FD and FI parts. From Fig. 13, we found that with the same target frame rate, a much higher CPU energy saving is possible at higher frame rates with the FD/FI decomposition. FRV and energy savings results of the proposed DVFS technique are reported in Table IV.



(a) Test movie (2)



(b) Test movie (6)

Fig. 13. CPU energy saving for with/without FD/FI decomposition (XScale-based)

E. Energy consumption during the FI part

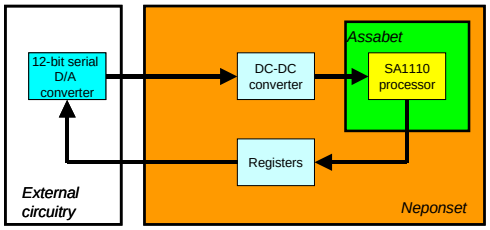
The CPU power consumption during the FI part may be lower than that consumed during the FD part since during the FI part the CPU is stalled until a requested memory transaction is serviced. Meanwhile, the functional units inside the CPU are not utilized. In the XScale-based system, it is possible to set the minimum frequency during the FI part without causing any latency increase, which results in significant CPU energy saving. Note that the FI part cannot be used as a "timing buffer zone" to compensate for the workload prediction error using intra-frame method as can be done in the SA1110-based system. Instead of using the intra-frame compensation, the inter-frame compensation can be applied for the XScale-based system.

V. IMPLEMENTATION

To implement the frame-based prediction algorithm for low-power MPEG decoding, the ***mpeg_play*** program was used and the required functions for calculating the moving averages and calculating the clock speeds and voltages were inserted in the player program. A device driver operating under the Linux OS environment was written to implement the CPU clock speed changes. Prediction of the decoding time for the next frame is based on the moving average (over the last six frames of the same type) as explained in Section II. In selecting the proper frequency value, the overhead of DVFS itself was also considered. For the hardware, we used two types of test beds having different performance. One is Intel's StrongARM1110 evaluation board [26] and the other is Intel's XScale based low power platform designed at USC [27].

A. StrongARM1110 based evaluation board

The hardware used is the Intel's StrongARM 1110 evaluation board, which supports 12 different frequencies ranging from 59 MHz to 221 MHz. A D/A converter was used as a variable operating voltage generator to control the reference input voltage to a DC-DC converter that supplies operating voltage to the CPU. Inputs to the D/A converter are generated using the general purpose input output (GPIO) signals. The extra hardware was designed, built and interfaced to the standard Intel Assabet board as a separate module. In Fig. 14, the block diagram of the variable voltage generator is shown. Regarding color representation, components having same color are in the same board.



Freq. [MHz]	Volt. [V]
59	1.113
74	1.126
89	1.156
103	1.165
118	1.216
133	1.248
148	1.326
162	1.394
177	1.464
192	1.536
206	1.605

Fig. 14. The variable voltage generator implementation in StrongArm 1110-based system

When the CPU clock speed is changed, a minimum operating voltage level should be applied at each frequency to avoid a system crash due to increased gate delays. In our implementation, these minimum voltages are measured through extensive experiments and stored in a table so that these values are automatically sent to the variable voltage generator when the clock speed changes. Voltage levels mapped to each frequency are distributed from 1.1 V @59 MHz to 1.605 V @206 MHz and shown in Fig. 14.

In anticipating the workload for the next frame, there is a discontinuity in the calculated workload between the lower frequencies (upper) and the higher frequencies (bottom) because when the CPU frequency changes, the memory clock characteristics are also affected, resulting in non-linear performance scaling, which is a typical occurrence in a StrongARM-based processor [15]. This phenomenon is illustrated in Fig. 15. To correct for this non-linearity, a weight factor for each frequency is extracted from the measurement and included in the workload calculation.

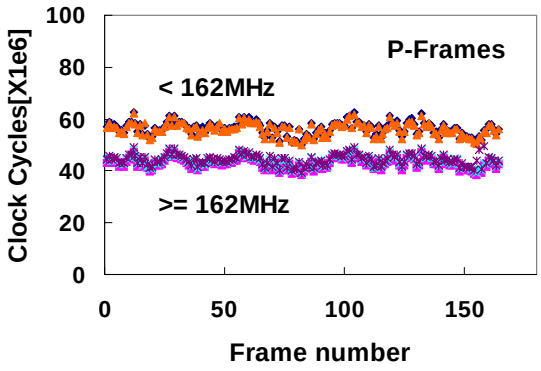


Fig. 15. Non-linearity in memory performance as a function of the CPU clock frequency

B. *XScale based testbed – Apollo Testbed 2*

The Apollo Testbed 2 (AT2), developed at USC, is a high-performance and low-power embedded platform with high-bandwidth wireless communication capability [27]. The photo of the main PC Board of the AT2 system is shown in Fig. 16. AT2 supports a number of peripheral devices such as a Web CAM, external FLASH, 16-channel, and a 100 K samples/second data acquisition system. AT2 system is a complete Linux box in the sense that it provides support for a wide variety of I/O interfaces, allows multi-processing capability and in-system reconfiguration, permits accurate and high speed data acquisition for the complete system as well as for the individual modules in system (this is possible due to careful separation of the system power planes on the PC board layouts).

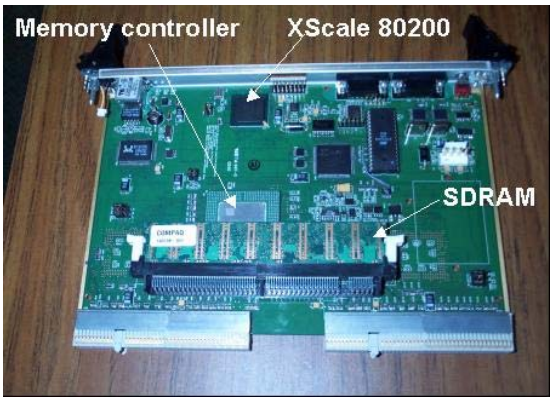


Fig. 16. Main board with the CPU, memory, and memory controller

Fig. 17 shows the data acquisition system in which the voltage drop across a precision resistor inserted between the external power line and the "design under test" (DUT) power line is used to measure the power consumption.

A programmable clock multiplier (PLL) in the XScale processor generates the internal CPU clock which can be adjusted from 200 up to 733MHz in steps of about 66 MHz with the development-board speeds available only from 333 MHz. The lower bound results from a constraint to the memory bus speed which is at 100 MHz in our system. The bus speed has to be less than one third of the CPU clock speed. This would yield a minimum speed of 333 MHz. Running the system at CPU speeds slower than 333 MHz causes immediate halts. The main PCB of AT2 includes an on-board variable voltage generator which provides suitable operating voltage at each clock frequency level. The block diagram of the variable voltage generator and voltage levels for each frequency are shown in Fig. 18.

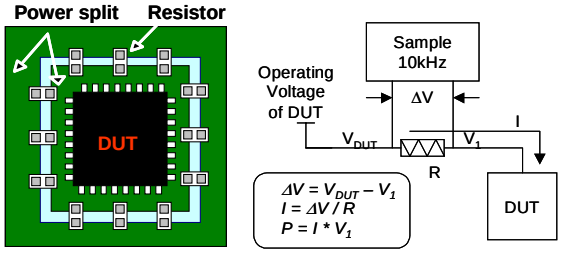
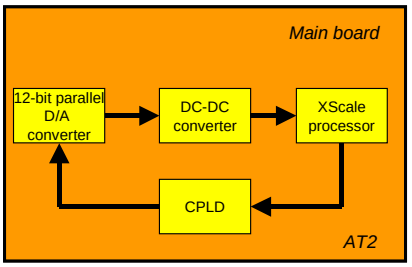


Fig. 17. Data acquisition system with split power plane



Freq. [MHz]	Volt. [V]
333	0.91
400	0.99
466	1.05
533	1.12
600	1.19
666	1.26
733	1.49

Fig. 18. The variable voltage generator implementation in XScale-based system

VI. EXPERIMENTAL RESULTS

The DVFS policies for MPEG decoding were implemented on the StrongARM SA1110-based evaluation board and XScale-based platform and results are discussed below.

A. *StrongARM-based Platform*

Due to the performance limitation of the StrongARM SA1110 processor and large scaling overhead, frame rates higher than 3 fps were not achievable. The overhead of DVFS itself is about 10 msec which is much larger compared to the reported value (~140 usec) [15] because in our implementation, a 12-bit serial D/A converter is used to control a DC-DC converter reference input to change CPU operating voltage. To set data value to the D/A converter it was required to generate a clock chain of 13 cycles (12 cycles to set 12-bit data and 1 cycle to latch data) and this clock chain was generated via general purpose input output (GPIO) registers, causing large

timing overhead. Also, this D/A converter is located in the external circuitry connected to Neponset board via external wires as shown in Fig. 14 and this wire connection would cause additional timing overhead. One more reason for such large overhead is that we can only scale frequency at most two step-wide. If frequency is changed more than two steps, for example change frequency from minimum to maximum directly, our system crashed and we thought this phenomenon occurred due to unstable variable voltage generator we made. As a result, we performed measurements at frame rates of 1 and 2 fps, which are somewhat low for real video applications, but are sufficient to demonstrate the capability of DVFS. For prediction error compensation, only intra-frame compensation was used since there were many time slacks during the FI part due to low frame rates. Fig. 19 and Fig. 20 show the power consumption in the system without and with DVFS while playing MPEG at fps = 1. The power consumption is measured at a 2 kHz sampling frequency. The CPU frequency is 206 MHz without DVFS and, depending on the frame type, it is reduced down to 89MHz with the proposed DVFS technique. Average board-level power consumptions for both cases are 2.94 W (0.49 A @6 V) and 2.46 W (0.41 A @6 V), respectively, which represent a 16 % reduction in the total system energy. By considering the efficiency of two DC-DC converters [28] (one is for 6 V to 3.3 V and the other is for 3.3 V to the CPU supply voltage) 85 % and the power consumption of SA1110 is about 400 mW at 206 MHz according to the Intel SA1110 reference manual [26], then it may be concluded that the CPU energy consumption was reduced by about 87 % as a result of applying the proposed frame-type-based DVFS technique. Notice that it is not possible to directly measure the CPU power consumption in the Intel Testbed. Fig. 21 and Fig. 22 show the power consumption at fps = 2 and about 43 % of the CPU energy saving is achieved without no deadline missed frame.

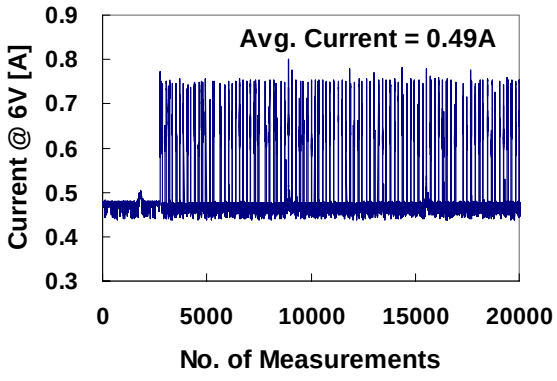


Fig. 19. Power consumption without DVFS at fps = 1

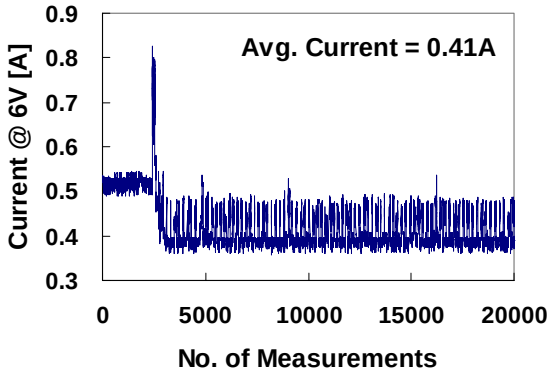


Fig. 20. Power consumption with DVFS at fps = 1

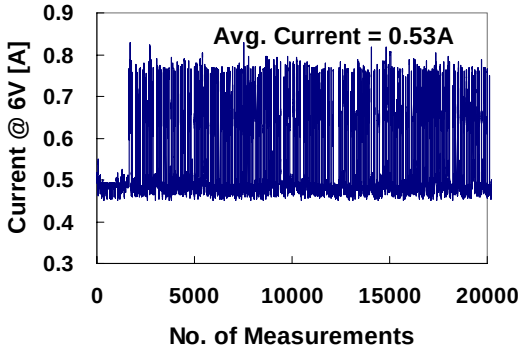


Fig. 21. Power consumption without DVFS at fps = 2

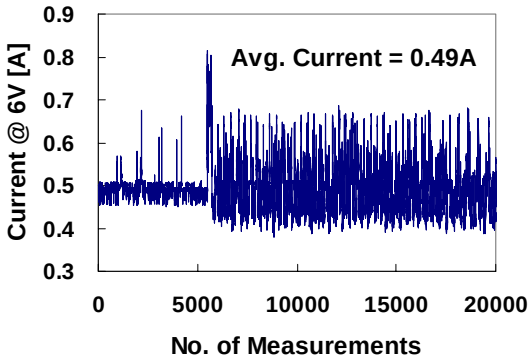


Fig. 22. Power consumption with DVFS at fps = 2

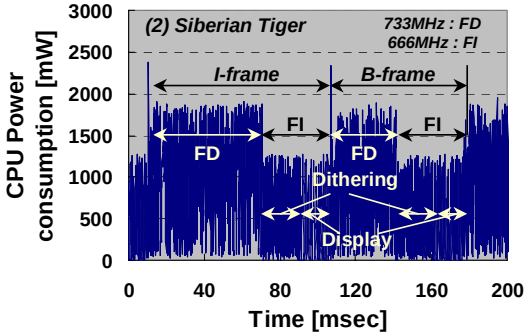
B. XScale-based Platform

The same measurements were performed in the AT2 system. Unlike StrongArm-based platform, there is linear performance scaling in the XScale processor. This is because the AT2 main board contains an external memory controller that isolates the CPU from the memory, and therefore, the memory bus clock speed (100 MHz) and the CPU clock speed become decoupled. For the 80200 XScale processor, the latency for switching the CPU voltage/frequency is 6 μsec at 333 MHz [24]. By using our data acquisition system, the power consumption of each component on the AT2 main board can

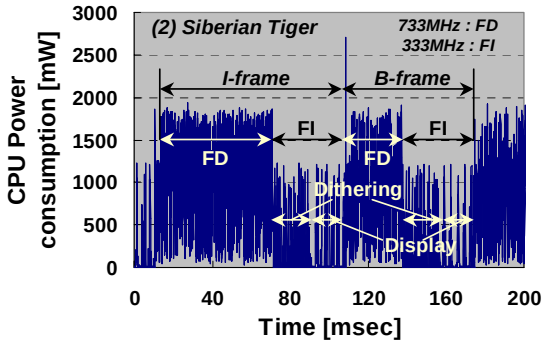
individually be measured and reported.

Fig. 23 depicts the CPU power consumption while decoding an I-frame followed by a B-frame in which two different frequencies are set during the FI part (a) 666 MHz and (b) 333 MHz. A 733 MHz is used for the FD part. As mentioned in the previous section, FI time, which contains the off-chip access latencies during "dithering" and "display", does not change with the CPU frequency, 37 msec at both frequencies, whereas the average power consumption during the FI part is significantly reduced from 510 mW to 186 mW (64 % reduction) as a result of voltage scaling.

We measured the actual CPU power consumptions while playing back six test video clips on the AT2 system with the proposed DVFS method and the results are summarized in Table V. From this table, it is seen that less energy is required by separating the FD and FI parts (cf. the *FD/FI Decom* columns) compared to the case without workload decomposition (cf. the *No-Decom* columns). Furthermore, as frame rate becomes higher, the energy saving difference between the two cases (with and without decomposition) becomes higher. This is because FI time portion in the allowed decoding time for a frame increases as frame rate increases. We acknowledge that we have not exactly implemented any of the previous DVFS approaches on our platform to see how they perform. However, as stated above, our key contribution is to recognize that the frame workload can be divided into FI and FD parts, which behave quite differently with respect to CPU voltage and frequency scaling. None of the previous work does anything like this (they basically consider the FD part only and ignore the fact that the FI part may be used as either "timing buffer" or "energy saving means" to increase the energy saving.) We show in Table V the relative performance of DVFS without decomposition and DVFS with workload decomposition as compared to the base line without any DVFS. We can think of previous work on DVFS for the MPEG decoder as being in essence similar to the reported results without workload decomposition.



(a) FD : 733 MHz, FI : 666 MHz



(b) FD : 733 MHz, FI : 333 MHz

Fig. 23. Decoding time and power consumption at different CPU frequencies

Finally, Fig. 24 shows the effectiveness of inter-frame compensation method. With this compensation scheme, the run-time frame rate smoothly converges to the target frame rate (here, 13 fps). Notice that the frame rate diverges from the target rate without this compensation, resulting in wasted CPU energy. The reason that the divergent rate is higher (rather than lower) than the target frame rate is that the I- and P-frames need the maximum frequency to meet the deadline, and they are unaware of the positive timing slacks that are carried over from the previous B-frames.

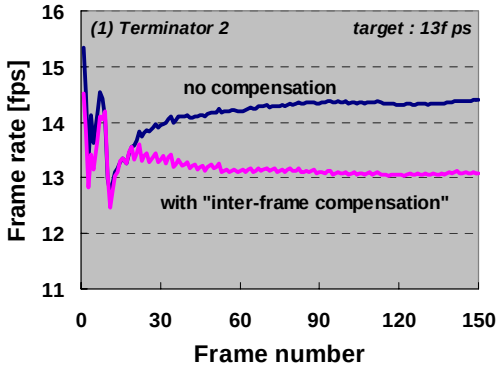


Fig. 24. Frame rate variation with "inter-frame" compensation method

VII. Conclusion

A frame-based workload prediction algorithm for DVFS in MPEG decoding was proposed and implemented on two different platforms, a StrongARM-based portable system for a low performance and an XScale-based AT2 system for a high performance system. In this DVFS, each frame type is handled individually for more accurate decoding time prediction. The whole decoding time for a frame is divided into two parts: frame-dependent and frame-independent. During the FI part the required operation is memory intensive and the amount of workload is same over all frame types in a given video sequence. To avoid *QoS* degradation due to misprediction, two error compensation methods are proposed: intra-frame and inter-frame compensation. Using this property, the FI period is used as a timing buffer when misprediction occurs in FD

workload prediction. When applied to a dedicated MPEG player, more than 87 % and 80 % of CPU energy was saved by the proposed DVFS scheme in both low and high performance platforms, respectively.

REFERENCES

[1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," Proc. 1st Symp on Operating Systems Design Implementation, 1994, pp. 13-23.

[2] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low power CPU," Proc. 1st *ACM Int. Conf. Mobile Computing Networking*, 1995, pp.13-25.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE Journal of Solid-State Circuit*, vol. 35, no.11, Nov. 2000, pp.1571-1580.

[4] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach or energy efficient computing," *Proc. International Symposium on Low Power Electronics and Design*, 1996, pp.347-352.

[5] F. Yao, A. Demers, and S. Shenker, " A Scheduling Model for Reduced CPU Energy," *IEEE Annual Foundations of Computer Science*, 1995, pp.374-382.

[6] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *International Symposium on Low Power Electronics and Design*, 1999, pp.197-202.

[7] D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," *Proc. Design Automation Conference*, 2001, pp.438-443.

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

[9] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," *Proc. IEEE Symp. on Low Power Electronics*, 1994, pp.8-11.

[10] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," *Proc. International Symposium on Low Power Electronics and Design*, 1998, pp.76-81.

[11] J. Mitchell, W. Pennebaker, C. Fogg, and Didier LeGall, *MPEG video compression standard*, Chapman and Hall, 1996.

[12] K. Patel, B. Smith, and L. Rowe, "Performance of a software MPEG video decoder," *Proc. First ACM Int'l Conf. on* Multimedia, 1993, pp.75-82.

[13] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld, "Policies for Dynamic Clock Scheduling," *Symposium on Operating Systems Design & Implementation*, Oct. 2000

[14] A. Bavier, A. Montz, and L. Peterson, "Predicting MPEG execution times," *Proc. Int'l Conf. On Measurement and Modeling of Computer Systems*, 1998, pp. 131-140.

[15] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips, "Power-aware video decoding," Presented at 22nd Picture Coding Symposium, Seoul, Korea, 2001.

[16] D. Son, C. Yu, and H. Kim, "Dynamic voltage scaling on MPEG decoding," *International Conference of Parallel and Distributed System (ICPADS)*, June 2001

[17] E. Chung, L. Benini, and G. Micheli, "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," *International Symposium on Low Power Electronics and Design*, Aug. 2002, pp.42-47.

[18] K. Flautner and T. Mudge, "Vertigo: automatic performance-setting for Linux," In OSDI, Boston, MA, Dec. 2002, USENIX.

[19] Transmeta Crusoe. http://www.transmeta.com/technology/index.html

[20] Y. Lu, L. Benini, and G. D. Micheli, "Dynamic frequency scaling with buffer insertion for mixed workloads," *IEEE Transactions on computer-aided design of integrated circuits and systems*, 21(11): Nov. 2002, pp.1284-1305.

[21] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," *International Symposium on Low Power Electronics and Design*, pp. 34-39, Aug. 2001.

[22] Z. Lu, J. Lach, M. Stan, K. *Skadron*, "Reducing multimedia decode power using feedback control," *Proc. of International Conference on Computer Design*, San Jose, CA, Oct. 2003

[23] http://bmrc.berkeley.edu/frame/research/mpeg

[24] "Intel 80200 Processor Based on Intel XScale Microarchitecture," http://developer.intel.com/design/iio/manuals/273411.htm

[25] R. Floyd and L. Steinberg, "An adaptive algorithm for spatial grayscale," *Proc. SID*, 17(2), pp.75-77

[26] http://www.intel.com/design/strong/manuals/278240.htm

[27] http://atrak.usc.edu/~apollo

[28] http://pdfserv.maxim-ic.com/arpdf/MAX1692.pdf

TABLE I

TEST VIDEO SEQUENCES SUMMARY (SA1110-BASED)

Video clip name	Frame size	# of frame	FD workload ($X10^6$)			FI workload ($X10^6$)	α			Avg. prediction error		
			I	P	B		I	P	B	I	P	B
(1) Terminator2	352 × 240	150	43.11	34.15	17.33	14.98	0.35	0.44	0.86	7.22	4.29	7.57
(2) Siberian Tiger	320 × 240	634	64.91	55.11	29.97	13.00	0.20	0.24	0.43	2.7	5.77	6.62
(3) Deploy	352 × 288	725	22.80	12.73	13.19	17.10	0.75	1.34	1.30	6.66	4.91	3.02
(4) Wg_wt	304 × 224	331	32.33	15.24	-	11.80	0.36	0.77	-	8.34	15.71	-
(5) Badboy2	480 × 208	666	22.00	19.28	18.71	17.10	0.78	0.89	0.91	17.27	16.01	9.02
(6) Final3	160 × 120	500	14.25	12.64	7.89	7.92	0.56	0.63	1.00	5.06	7.5	13.4

TABLE II

FRV SIMULATION SUMMARY – SA1110 (* : NUMBERS IN PARENTHESIS ARE FOR TEST CASE (6))

fps*	(1)		(2)		(3)		(4)		(5)		(6)	
	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C
2 (4)	0.12	0.003	0.002	1E-04	0.116	0.003	0.135	0.011	0.016	1E-04	0.259	0.007
3 (5)	0.038	3E-04	0.002	0.002	0.069	9E-05	0.08	6E-04	0.004	7E-05	0.181	1E-04
4 (6)	0.007	0.001	0.022	0.022	0.038	3E-05	0.042	3E-04	0.001	5E-04	0.121	8E-04
5 (7)	0.02	0.019	0.416	0.416	0.017	8E-05	0.019	2E-04	0.016	0.016	0.077	3E-04
6 (8)	-	-	-	-	0.005	5E-04	0.013	0.008	0.081	0.081	0.042	2E-04

TABLE III

ENERGY SAVING SUMMARY – SA1110 (* : NUMBERS IN PARENTHESIS ARE FOR TEST CASE (6))

fps*	(1)		(2)		(3)		(4)		(5)		(6)	
	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C	IC	I^2C
2 (4)	78.57	78.9	69.36	69.41	83.36	83.38	80.26	80.27	80.94	80.89	82.53	82.52
3 (5)	63.95	64.54	47.99	46.62	72.46	72.46	66.01	65.73	67.94	67.7	77.13	77.06
4 (6)	46.18	45.82	22.77	20.34	59.45	59.29	52.22	52.86	52.16	52.27	70.82	70.82
5 (7)	29.11	28.01	-	-	46.58	46.45	33.68	31.62	38.8	39.04	63.7	63.66
6 (8)	-	-	-	-	33.66	33	6.455	5.869	13.81	13.64	57.4	57.55

TABLE IV

SIMULATION RESULTS SUMMARY – XSCALE (* : NUMBERS IN PARENTHESIS ARE FOR TEST CASE (6))

fps*	(1)		(2)		(3)		(4)		(5)		(6)	
	FRV	Energy Saving	FRV	Energy Saving	FRV	Energy Saving	FRV	Energy Saving	FRV	Energy Saving	FRV	Energy Saving
10 (28)	-	-	0.016	77.715	-	-	-	-	-	-	0.031	82.444
11 (29)	-	-	0.032	70.554	-	-	-	-	-	-	0.02	80.648
12 (30)	0.132	81.861	0.073	63.038	-	-	-	-	-	-	0.03	78.461
13 (31)	0.051	78.139	0.163	54.858	-	-	1.144	81.214	0.028	80.473	0.07	75.901
14 (32)	0.049	72.226	0.364	43.974	-	-	0.442	77.489	0.075	76.212	0.146	72.69
15	0.125	66.622	-	-	0.03	79	0.148	73.164	0.177	71.059	-	-
16	0.327	60.283	-	-	0.007	74.911	0.077	67.099	0.417	63.914	-	-
17	-	-	-	-	0.016	68.241	0.044	60.112	-	-	-	-

TABLE V

ACTUAL MEASUREMENT RESULTS SUMMARY – XSCALE (* : NUMBERS IN PARENTHESIS ARE FOR TEST CASE (6))

fps*	(1) t2		(2) st		(3) dp		(4) wg		(5) bd		(6) fi	
	FD/FI Decom	No-Decom	FD/FI Decom	No-Decom	FD/FI Decom	No-Decom	FD/FI Decom	No-Decom	FD/FI Decom	No-Decom	FD/FI Decom	No-Decom
10 (27)	-	-	77.26	73.147	-	-	-	-	-	-	81.337	80.881
11 (28)	80.791	80.463	68.448	55.49	-	-	-	-	-	-	82.168	82.042
12 (29)	79.883	79.681	60.26	43.391	-	-	-	-	79.508	79.33	81.616	81.849
13 (30)	74.763	71.599	48.213	25.361	-	-	77.043	75.272	78.988	78.848	81.272	81.654
14	68.69	40.45	40.804	2.5972	73.27	57.94	69.311	60.593	75.734	71.339	-	-
15	57.845	22.166	-	-	65.311	35.526	60.003	41.333	60.721	46.986	-	-
16	52.701	4.0561	-	-	60.208	4.2396	54.281	28.227	59.128	0.585	-	-
17	52.399	3.3786	-	-	59.241	0.7792	44.884	9.4696	-	-	-	-