Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder

Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram
Department of EE-Systems, University of Southern California, Los Angeles, CA90089
{kihwanch, dantu, wecheng}@usc.edu, pedram@ceng.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 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. In the DVFS scheme presented in this paper the FI part is used to compensate for the prediction error that may occur during the FD part such that a significant amount of energy can be saved while meeting the frame rate requirement. The proposed DVFS algorithm has been implemented on a StrongArm-1110 based evaluation board. Measurement results demonstrate a higher than 50% CPU energy saving as a result of DVFS.*

1 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.

DVFS techniques [2-6] 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 [2] and [3], a simple interval-based scheduling algorithm can be used in non real-time operation. This is because there is no hard 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 [12] 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, frequent mispredictions may result in a decrease in the frame rate, which in turn means a lower QoS in MPEG.

There are also many ways to apply DVFS in real-time application scenarios [5-6]. In general, some information is given by the application itself, and the OS can use this information to implement an effective DVFS technique. In [5], an intra-task voltage scheduling technique was proposed in which the application code is split into many segments and the worst-case execution time of each segment (which is obtained by static timing analysis) is used to find a suitable voltage for the next segment. A method using a software feedback loop was proposed in [6]. In this scheme, a deadline for each time slot is provided. Furthermore, the actual execution time of each slot is usually shorter than the given deadline, which means that a slack time exists. The authors calculated the operating frequency of the processor for the next time slot depending on the slack time generated in the current slot and the worst-case execution time of each slot.

In both cases, real-time or non real-time, prediction of the future workload is quite important. This prediction is also the most difficult step in devising and implementing an effective DVFS technique, especially when the workload varies dramatically from one time instance to the next.

In this paper, an effective DVFS algorithm for MPEG decoding is proposed in which the future workload is accurately predicted 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 a StrongARM-1110 based platform and results in an energy reduction of more than 50%.

When lowering the supply voltage to reduce energy consumption, frequency should be decreased first in order to prevent malfunction 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 then either "voltage scaling" or "frequency scaling."

The remainder of this paper is organized as follows. Related works on DVFS and MPEG are shown in Section 2. In Section 3, the proposed DVFS algorithm is presented. The details of the actual implementation, including both hardware and software, are described in Section 4. Experimental results and conclusion are given in Sections 5 and 6, respectively.

2 Background

2.1 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 [1]:

$$\mathbf{E = C_{switched} V^2 f_{clk} T}$$

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's 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 $E_1=CV_1^2f_1\Delta T/2=nCV_1^2/2$ 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 $E_2=CV_2^2f_2\Delta T=nCV_1^2/8$. 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.

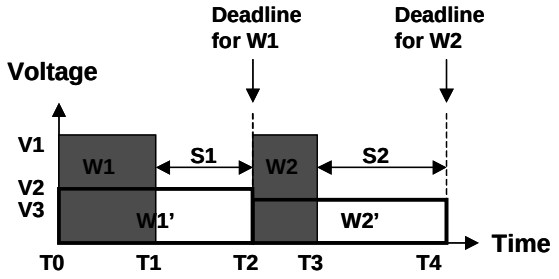

Fig. 1. An illustration of the DVFS technique

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 [14] 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 [14]. 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, α, 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.

$$\text{WindowSize} = n$$

MA :

$$\text{Workload}(t+1) = \frac{\sum_{\tau=0}^{n-1} \text{Workload}(t-\tau)}{n}, \quad t \geq n-1$$

$$= \frac{\sum_{\tau=0}^{t} \text{Workload}(t-\tau)}{t+1}, \quad \text{otherwise}$$

WA :

$$\text{Workload}_{avg}(0) \equiv \text{Workload}(0)$$

$$\text{Workload}(t+1) = \alpha \cdot \text{Workload}(t) + (1-\alpha) \cdot \text{Workload}_{avg}(t)$$

$$= \alpha \cdot \sum_{\tau=0}^{t} (1-\alpha)^{\tau} \cdot \text{Workload}(t-\tau)$$

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 whereas Fig. 3 depicts the workload prediction errors for both cases. 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.

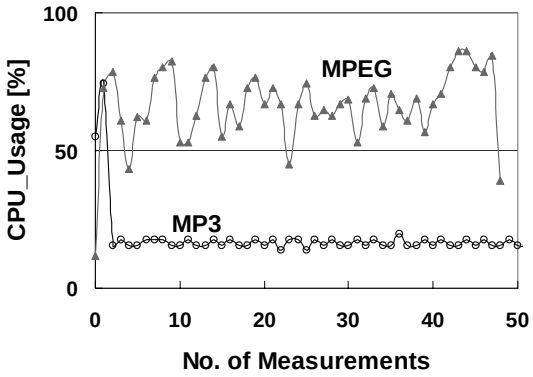

Fig. 2. CPU usage of MP3 and MPEG

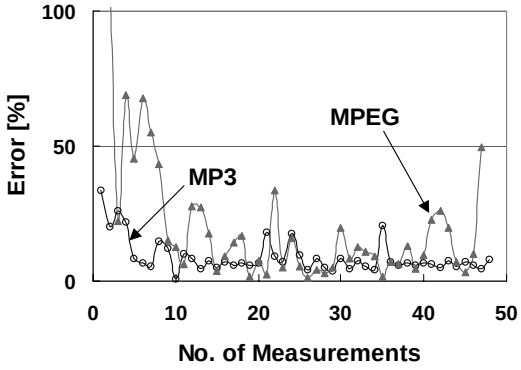

Fig. 3. Workload prediction error

2.2 MPEG Terminology

In general, an MPEG2 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*, and *Dithering* [7]. Among these steps, the IDCT and Reconstruction take up half of the decoding time [9]. The IDCT is CPU-intensive (i.e., requires iterative multiplication-accumulation computation over an 8x8 array of integer or float-point values) whereas the reconstruction and dithering 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 step, meaning that the CPU utilization of different frame types varies by a large amount. 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).

2.3 Prior Work

To develop an effective DVFS technique for MPEG decoding, each frame decoding time should be accurately determined since the supply voltage should vary based on the expected decoding time for the frame and the given deadline.

In [10], the authors empirically studied the relationship between the decoding time and the data size of each frame. The results showed a strong correlation between the two parameters with an error of less than 25%. The code size of each frame, however, cannot be obtained before starting the IDCT step whereas the frame type can be known immediately after the parsing step. To overcome this limitation, a method using feedback control was proposed in [11] in which macro blocks[1] in a frame are first divided into two parts, and the decoding time of the second part is predicted based on the decoding time and the code size of the first part. If the decoding time of the first part exceeds the predicted decoding time, the voltage for the next part is increased so that a deadline violation can be avoided. This technique thus scales up the voltage (and correspondingly the clock speed) during the second part to meet the deadline when a prediction error occurs in the first part. This results in higher CPU energy consumption for performing the IDCT step in the second part when the prediction error occurs in the first part. However, the authors did not consider the energy-related characteristics of each step in the decoding process. It is desirable to scale voltage up during memory intensive steps since this does not impact the CPU energy consumption by much. On the other hand, scaling voltage up during CPU-intensive steps such as IDCT leads to higher energy consumption. From this observation, it can be concluded that it is better to compensate for the workload prediction error by raising the supply voltage level during a memory-intensive step as opposed to a CPU-intensive step. This would achieve the required QoS with maximal energy saving.

3 Proposed Algorithm

A DVFS algorithm for low-power MPEG decoding with large workload variation is presented in this section. The decoding time prediction is performed by maintaining a moving-average of the decoding time for each frame type (three averages, one per frame type). The expected decoding time for an incoming frame is thus determined based on the moving average for the appropriate frame type. As stated previously, the decoding process 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:

$$\mathbf{T_{Decoding} = T_{FD} + T_{FI}\,;\quad E_{Decoding} = E_{FD} + E_{FI}}$$

The parsing, IDCT and reconstruction steps are included in the frame-dependent time whereas the dithering step is included in the frame-independent time. This is because the dithering time is dependent upon the frame pixel size and is otherwise constant for a given video stream. Since the FI part tends to be memory intensive, the energy consumption during the FI step, E_{FI}, is nearly constant [11]. In contrast, the energy consumption during the FD part, E_{FD}, varies considerably. Variations in energy consumption and decoding time due to DVFS are captured by the following equations:

$$\mathbf{\Delta T_{Decoding} = \Delta T_{FD} + \Delta T_{FI}\,;\quad \Delta E_{Decoding} = \Delta E_{FD}}$$

From the above equations, it can be seen that the FI section can be used as a kind of "buffer zone" to compensate for prediction error of $\mathbf{T_{FD}}$ since changing voltage/frequency in the FI section does not significantly alter the energy consumption of the CPU for the decoding process (although it will change the FI time). The basic operation of the proposed DVFS algorithm is shown in Fig. 4.

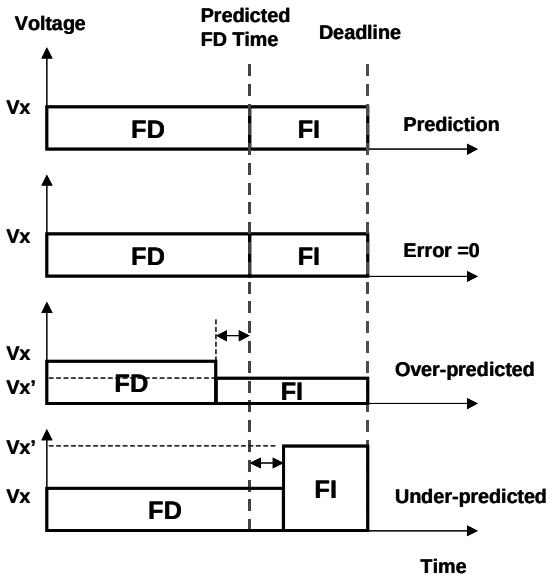

Fig. 4. Proposed DVFS policy

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. 4). 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. 4). Note that this compensation is done by scaling up the supply voltage during the memory intensive part and hence does not result in much CPU energy dissipation. However, this scheme cannot guarantee that one will never

[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.

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. But 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 minute (as can be seen in the results). Note also that even if this case occurs, the penalty is the loss of some video quality for a short period of time and is not a catastrophic failure as would have been the case if the application had a hard real-time deadline.

To determine the FD and FI times for a given frame decoding time, the source code for a software MPEG decoder, that is, ***mpeg_play*** [13], was modified, and a timestamp function was inserted at each decoding step.

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. 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 206MHz). 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 IDCT and their relationship to the frame type.

The effectiveness of the proposed frame-based workload prediction scheme is verified by calculating the prediction error ratio in B-frames, which exhibit the largest variation among the frame types. The MA scheme with a window size of six is used for the prediction. Results are shown in Fig. 7. The movie clip used in the experiment has 660 frames (320 X 240) including I-, P-, and B-type frames. Based on the measured FD time, the prediction error was calculated. 95% of the decoded frames were within an error rate of 15% while 97% of the frames are within a 20% error rate. I- frames and P-frames see prediction error of less than 10%. Prediction error for B-frames is 20%. Note that although the prediction error for B-frames is high, it takes less than half the time to decode as compared to an I-frame thereby giving us double the time to correct it in the FI time.

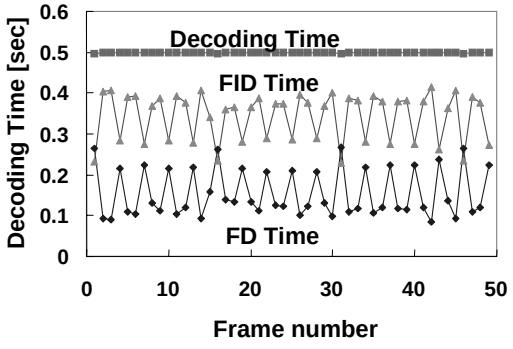

Fig. 5. Decoding time with fps = 2

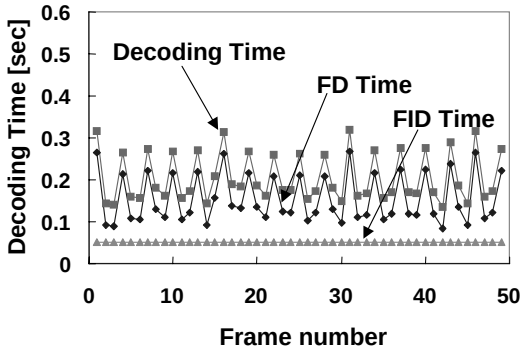

Fig. 6. Decoding time without setting a fps rate (as high a fps rate as the CPU can sustain)

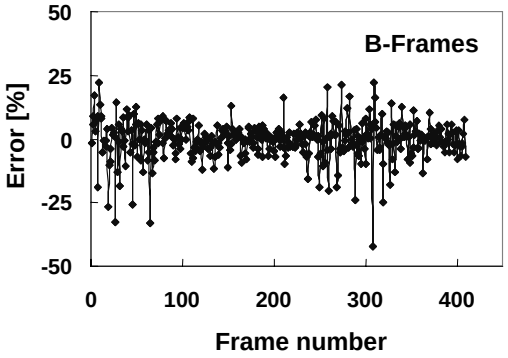

Fig. 7. Errors in B-frame workload prediction

QoS for the MPEG decoding can be defined as the ratio of the number of deadline-missed frames to the total number of decoded frames in the movie stream. The QoS value was calculated by counting the number of frames that violate the deadline, and the results are shown in Fig. 8. Two cases were compared: with and without using the FI region as a timing buffer. In case of without using the FI buffer zone, voltage applied during the FD time is kept maintaining during the FI. Both of the MA and WA schemes were used for each case in the workload prediction. There is little QoS difference between the MA and WA schemes while a considerably better QoS can be achieved by using the FI time as a buffer zone. At a frame rate of three, about 20% better QoS was obtained by using our proposed scheme.

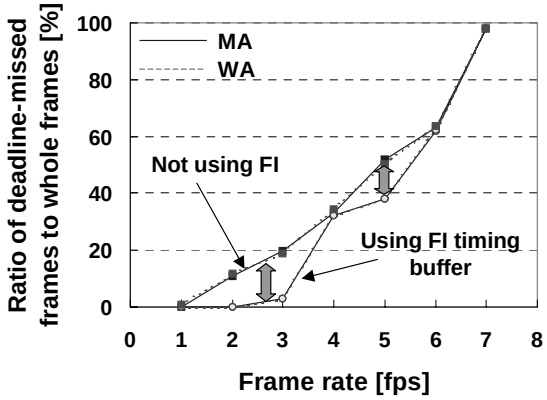

Fig. 8. QoS in MPEG decoding

4 Implementation

To implement the frame-based prediction algorithm for low-power MPEG decoding, a software MPEG player program, ***mpeg_play*** [13], was used and the required functions for calculating the moving averages and calculating the clock speeds and voltages were inserted. A device driver operating under the Linux OS environment was written to implement the CPU clock speed changes. Pseudo code for maintaining statistics and prediction error compensation is shown in Fig. 9.

```
frame_type = Parsing(next frame);

switch(frame_type) {

    case(I- type) :
      next_freq_FD = predict_next_freq(I- type);
      changeScale(next_freq_FD);
      errorCorrect(I- type);
      break;

    case(P- type) :
      next_freq_FD = predict_next_freq(P- type);
      changeScale(next_freq_FD);
      errorCorrect(P- type);
      break;

    case(B- type) :
      next_freq_FD = predict_next_freq(B- type);
      changeScale(next_freq_FD);
      errorCorrect(B- type);
      break;

}
```

```
float predict_next_freq(frame_type)
{
    next_freq_FD = (AvgWorkLoad(frame_type)
                    + DVFS_Overhead
                    + FIWorkLoad) / deadLine;
    return next_freq_FD;
}
```

```
void changeScale(float next_freq)
{
    int min_diff = 100;
    int freq_map[11] = {59, 74, 89, 103, 118,
          133, 148, 162, 177, 192, 206, 221 };
    for(I=0; I<12;I++) {
      diff = freq_map[ I] – next_freq;
      if((diff >= 0) && (diff <= min_diff)) {
        next_freq = freq_map[ I] ;
      }
    }
    set_GPIO_register(next_freq);
}
```

```
void errorCorrect(frame_type)
{
   error = AvgWorkLoad(frame_type) –
            MeasuredWorkLoad;

   if(error != 0) {
     next_freq_FI = ( FIWorkLoad -  error +
          DVFS_Overhead ) / (deadline – FD time);
     changeScale(next_freq_FI);
   }
}
```

Fig. 9. Pseudo code for DVFS

The decoding time prediction for the next frame is based on the moving average (over the last six frames of the same type) as explained in Section 2. In selecting the proper frequency value, the overhead of DVFS itself was also considered.

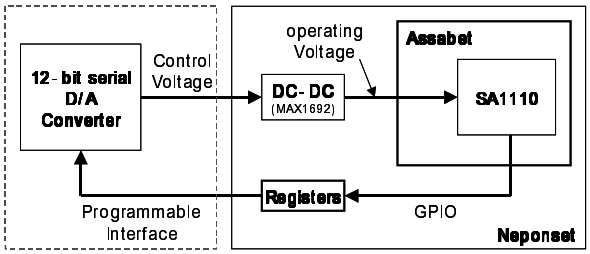

Fig. 10. Variable voltage generator on test bed

The hardware used is the Intel's StrongARM 1110 evaluation board [8], which supports 12 different frequencies from 59MHz to 221MHz. 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. 10, the block diagram of the variable voltage generator is shown.

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 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 1.1V @59MHz to 1.67V @221MHz.

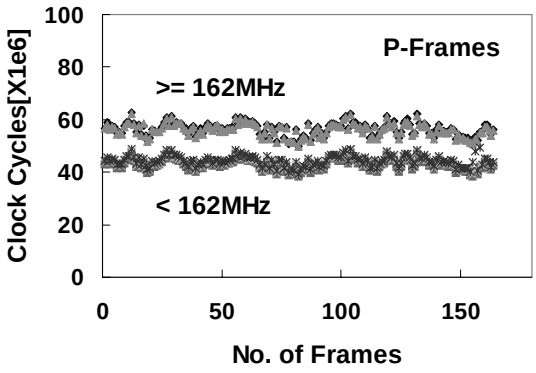

Fig. 11. Non-linearity in memory performance as a function of the CPU clock frequency

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 [11]. This phenomenon is illustrated in Fig. 11. To correct for this non-linearity, a weight factor for each frequency is extracted from the measurement and included in the workload calculation.

5 Experimental Results

The DVFS policies for MPEG decoding were implemented on the StrongARM evaluation board. Due to the performance limitation of the StrongARM processor, frame rates higher than 3 fps were not achievable. Frame rates of 1 and 2 fps, which are very low for real video applications, but are sufficient to demonstrate the capability of DVFS, were chosen. Fig. 12 and Fig. 13 show the power consumption drawn from the system supply rail (6 volts) without and with DVFS while playing MPEG2 at fps=1. The power consumption is measured at a 2 KHz sampling frequency. While the CPU frequency is 206 MHz without DVFS, the frequency is lowered down to 89MHz with the proposed DVFS technique, depending on the frame type. Average board-level power consumptions for both cases are 2.94 W (0.49A @6V) and 2.46 W (0.41A @6V), respectively, which represent a 16% reduction in the total system energy. Since the StrongARM CPU consumes about 30% of the total energy, it can safely be concluded that the CPU energy consumption was reduced by about 53% as a result of applying the proposed frame-type-based DVFS technique.

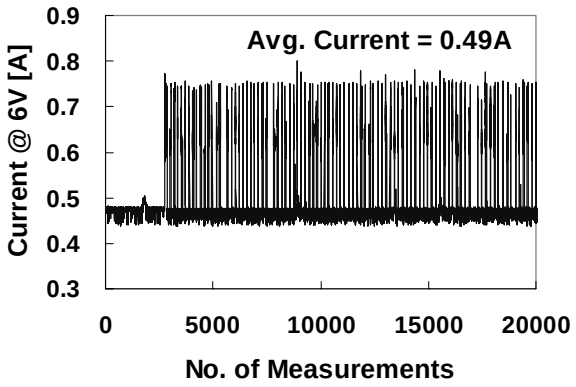

Fig. 12. Power consumption without DVFS at fps=1

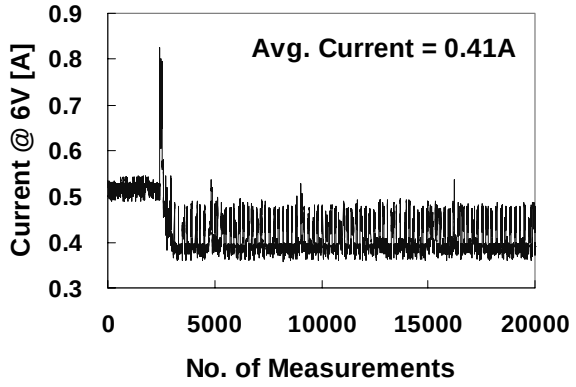

Fig. 13. Power consumption with DVFS at fps=1

6 Conclusion

A frame-based workload prediction algorithm for DVFS in MPEG decoding was proposed and implemented on a StrongARM-based portable system. In this DVFS, each frame type is handled individually for more accurate decoding time prediction, less than 10% ~ 20% prediction error ratio in all frame types. To avoid QoS degradation due to misprediction, the whole decoding time for a frame is divided into two parts: frame-dependent and frame-independent. During the FI step, in which the required operation is memory intensive, CPU voltage increase does not affect the energy consumption since the power supply of the memory subsystem is separated from that of the CPU. 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 50% of CPU energy was saved by the proposed DVFS scheme.

7 References

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

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

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

[4] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach or energy efficient computing," *ISLPED-96: ACM/IEEE International Symposium on Low Power Electronics and Design*, 1996, pp.347-352.

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

[6] S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," in *Proc. ASP-DAC*, 2000, pp. 381-386.

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

[8] http://developer.intel.com/design/strong.

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

[10] A. Bavier, A. Montz, and L. Peterson, "Predicting MPEG execution times," *SIGMETRICS / PERFORMANCE '98, Int'l Conf. On Measurement and Modeling of Computer Systems*, 1998, pp. 131-140.

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

[12] 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.

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

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