Workload Characterization and Cost-Quality Tradeoffs in MPEG-4 Decoding on Resource-Constrained Devices

Yanhong Liu[1] Samarjit Chakraborty[1] Wei Tsang Ooi[1] Ashish Gupta[2] Subramanian Mohan[1]
[1]Department of Computer Science, National University of Singapore
[2]Department of Computer Science, Colorado State University

Abstract

Recently there has been a lot of interest in the embedded systems community on architectures and design methods that are targeted towards multimedia applications. This trend is primarily motivated by the proliferation of resource and power-constrained portable devices (such as mobile phones and PDAs), a major portion of whose workload is made up of multimedia applications. In this paper, we investigate the tradeoffs between video quality and the processor cycle requirements in such resource-constrained devices, in the particular context of MPEG-4 decoding using an open source codec called XviD. The XviD codec implements a number of powerful coding tools from MPEG-4, which are organized as profiles *and* levels*. Given the specification of an architecture on which an XviD decoder is implemented, the work presented here would guide a multimedia applications developer in selecting appropriate profiles and levels for the corresponding encoder application. While the selection of such profiles has so far been primarily influenced by the network bandwidth in the case of video streaming, our work stresses the importance of additionally taking into account the architecture of the device running the decoder application. Although the relevance of this observation is increasingly being realized, sufficient work has not yet been done to provide guidelines on how to systematically make such selections. This work attempts to address this shortcoming.*

1 Introduction

With the increase in the availability and demand of video data on the Internet, there arises a need to cater to the entire spectrum of devices that are vying to access such data. These devices vary from high-speed processors in desktop computers to power- and resource-constrained devices like mobile phones and PDAs. The process of decoding video data to be played on these devices involves some computationally intensive tasks, whose processor cycle demands depend on the encoding schemes and encoding parameters used. A large portion of the video data available today is compressed according to the MPEG standard. With the development of MPEG-4, much attention has been given to various aspects of this standard. MPEG-4 aims to cover virtually every possible aspect of multimedia data compression. It consists of a tool box of coding instruments, where the combination of specific coding tools is organized as *profiles* and *levels*. Each profile, and a level in that profile is targeted towards a specific application domain (e.g. portable devices, DTVs, etc.) and the cost/quality constraints in that domain. As more multimedia applications continue to emerge, new profiles are being developed.

Traditionally, the selection of an encoder profile has been primarily influenced by the available network bandwidth. For example, there has been a considerable amount of work with the goal of maintaining a high video quality even at low bit-rates. Another direction of work attempts to use scalable encoding techniques where the video quality can adapt to changing network conditions in the case of video streaming. With the proliferation of portable devices such as mobile phones and PDAs, it becomes common to stream a video clip over a wireless network and play it on such devices. As a result, the video quality not only depends on network-related issues but also on the resource and power constraints of the device running the decoder application. Such constraints do not exist in the case of more powerful desktop computers. To address this new development, increasingly there is a need to take into account the power and resource constraints of the device running a decoder application while selecting the encoder parameters. While such issues are addressed in the MPEG-4 standard, we do not know of any work which quantitatively studied how the resource requirements change with the different encoder parameters using open source MPEG-4 implementations.

In this paper, we attempt to initiate such a study and report some initial experimental results. While the effects of encoding bit-rate and quantization on the video quality is well-known in the multimedia systems domain, our results provide insights into the tradeoffs between video quality and processor cycle requirements. More specifically, the

results presented in this paper show the following.

- The processor cycle requirement does not linearly increase with the specified video quality. This brings up the problem of choosing an encoder parameters that results in a good tradeoff between the video quality and the resource/power requirements from the device running the decoder application.
- We also see that decreasing the video quality beyond a certain point does not result in significant decrease in the processor cycle requirements.
- There is a significant difference in processor cycle requirements for video clips having different amounts of global motion.
- The relative distribution of the processor cycle requirements among different tasks of the decoder application depend on the encoding parameter values and also on the amount of global motion in the video clips being decoded.

The last two points can be used as a guideline for embedded system designers while designing MPEG-4 specific decoder hardware and also for partitioning and mapping a decoder application on multiprocessor architectures. The first two observations, on the other hand, can be exploited by multimedia application developers (e.g. when designing a web site with video content that will be browsed using resource-constrained devices like PDAs).

Many of our conclusions can be derived intuitively if one understands how MPEG-4 compressions work. As far as we know, however, no such quantitative results based on open source MPEG-4 implementation is available.

The rest of the paper is organized as follows. In the next section we describe our experimental setup, following which we present our results in Section 3 along with a discussion of those results. In Section 4 we discuss some related work and Section 5 concludes the paper.

2 Experimental Setup

We have conducted experiments using version 1.0.2 of XviD codec, a popular open source implementation of the MPEG-4 standard (see `www.xvid.org`). We have profiled the various components of the XviD decoder to obtain the number of instructions executed for each component. This was done using the SimpleScalar instruction set simulator [1] for the ARM processor model. The rationale behind using the ARM processor is its widespread use in various portable devices like PDAs. A variety of test video sequences were encoded using different encoding schemes and policies, after which the processing requirements involved in decoding these encoded video clips were measured.

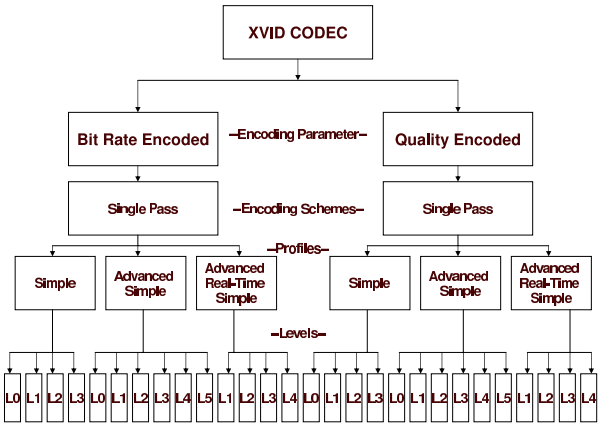

Figure 1: Different encoding schemes in the XviD codec.

2.1 Encoding Options

We first briefly introduce the various encoding options available in the XviD codec. As shown in Figure 1, different encoding levels are available in the XviD codec, which are specified by a profile and a level in this profile. Given the profile and the level, we can specify which type of encoding parameters (Quality or Bit-rate) are used. The quality encoding parameter specifies the target quantization value and bit-rate encoding parameter specifies the target bit-rate of the encoded video clip. We can also choose to use one of the two encoding schemes: Single Pass and Two Pass. Different profiles have different encoding and decoding complexities, each of which can use a different set of MPEG-4 encoding tools. In this paper, we focus only on the *Simple* profile. This profile is used for low-end devices like mobile phones and PDAs which have limited capabilities in terms of display, computational capacity, and memory. It is limited to VCD-like resolutions (352$\times$) and bit-rates up to 384 Kbps. It has four levels ranging from L0 to L3, the details of which are explained in Section 2. It uses only one advanced MPEG-4 tool, namely Lumimasking.

2.2 Functional Components

We partitioned the XviD decoder application into ten major functional components/tasks. This partitioning was done by inserting C code stubs into the XviD decoder code. As shown in Figure 2, these components are classified into two main groups: per-frame processing components and per-macroblock processing components. Since the per-macroblock processing components impose a higher computational demand and also lead to a more accurate workload characterization, in this paper we mostly concentrate on these components.

2.3 Test Sequences

In order to see the effect of all the combinations of encoding parameters on the XviD decoder, we encode video

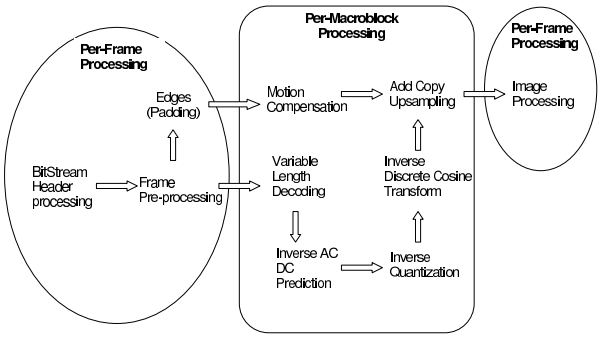

Figure 2: Main tasks of the XviD decoder application.

clips using all combinations of encoding schemes and parameters, and then decoded them to evaluate the processing requirements of the XviD decoder. Towards this, for each video clip we generated a trace recording the number of processor instructions used for decoding each *macroblock* of this video clip. The number of instructions were measured using the "sim-profile" configuration of the SimpleScalar instruction set simulator for ARM.

The video clips we used in our experiments are standard test sequences, which represent a wide range of video clips, with different amounts of details and motion. We restrict our discussion to four different clips, whose characteristics are listed below:

- *Mobile*: very colorful image with constant movement from left to right.
- *Stefan*: more colorful than Mobile, with a moving player in a crowd (movement is random).
- *Coastguard*: dull colored, less movement than in *Stefan*.
- *Foreman*: bright background with zoomed facial movements (comparatively less movement than in *Coastguard*).

Therefore, the amount of motion was the highest in *Mobile* and the lowest in *Foreman*.

3 Results and Discussion

The results we obtained for two of the four clips are shown in Tables 1 and 2. These two tables illustrate two extreme cases—the clip with the least amount of global motion and the other with the highest amount of global motion. Each of these tables contains two parts. The first half shows the results for bit-rate-encoded clips and the second half shows the results for quality-encoded clips (see Figure 1). For the bit-rate-encoded clips, L0 denotes the case with target bit-rate 64Kbps and no other tools enabled, L1 denotes the case with target bit-rate 64Kbps, L2 denotes target bit-rate 128Kbps, and finally L3 denotes target bit-rate 384Kbps. The tool "adaptive quantization" was enabled for L1, L2 and L3. For each level chosen, the corresponding columns show the average bit-rate, the average quantization value, the average peak signal-to-noise ratio (PSNR) per frame, and the average number of instructions required to process a macroblock. The frame rate for all the clips is 30 fps.

It may be noted that as the level is increased from L0 to L3, the average number of instructions required to process a macroblock increases. Similarly, the number of instructions required to process a macroblock decreases as the quantization level is increased. However, both these changes occur in a non-linear fashion.

Another thing to note from Tables 1 and 2 is the mismatch between the bit-rate of the encoded clips and target bit-rate of each level. This mismatch is due to the nature of the clips we used. The large amount of motion in the clip *Mobile* limits the amount of compression possible, causing the resulting bit-rate to be much larger than the target bit-rate of the encoded levels.

3.1 Quality versus Processing Requirements

As an illustration, here we study the video quality versus the processing requirements of the decoder application when different quality parameter values are used to encode the clips. To measure the video quality, instead of using the absolute value of the average PSNR per frame, we use a relative value of average PSNR per frame. The relative PSNR value for a video clip encoded using quality parameter value Q*i*, is defined as the ratio of the *average PSNR per frame encoded using Qi to the average PSNR per frame encoded using Q1*, where *i* ranges from 1 to 31. The encoded video clip achieves the highest quality value when encoded using Q1. The reason behind using a relative PSNR value is as follows. It may be noted that the absolute PSNR value for a higher-motion clip is lower than a lower-motion clip when the same encoding parameters are used (see Tables 1 and 2). But it can be observed that the subjective quality of the higher-motion clips is still comparable to that of the lower-motion clips when the same encoding parameters are used. Hence, we measured the objective quality of each video clip using the relative PSNR value.

Figure 3 shows the average number of instructions required to decode each macroblock of the different video clips, for different relative PSNR values. The encoder parameters corresponding to each such PSNR value can be inferred from Figure 4, which also shows how the relative PSNR value decreases as the quality parameter value is increased.

Given the processor frequency that is supported by a portable device, from Figure 3 it is possible to identify the maximum PSNR value for each video clip that the device can support (assuming each instruction requires one processor cycle, which is valid for RISC-like processors). For

Level	Bit-rate(Kbps)	Quantization	PSNR	# instr.
L0	421	29.65	24.39	165750
L1	519	29.74	24.87	175621
L2	519	29.74	24.87	175624
L3	536	29.43	24.90	180468
Q1	20305	1.00	35.19	917331
Q6	3311	6.00	30.87	372704
Q11	1528	11.0	28.44	259046
Q16	887	16.0	26.75	199342
Q21	600	21.0	25.62	162306
Q26	431	26.0	24.84	142552
Q31	334	31.0	24.64	140009

Table 1: The number of processor instructions required by the decoder for different encoder parameters for the *Mobile* clip.

Level	Bit-rate(Kbps)	Quantization	PSNR	# instr.
L0	143	29.65	33.25	96120
L1	171	29.74	33.64	96193
L2	183	24.80	34.33	102291
L3	445	8.42	37.12	134924
Q1	8454	1.00	42.17	625846
Q6	729	6.00	37.73	134896
Q11	305	11.0	36.02	101680
Q16	193	16.0	34.88	91721
Q21	154	21.0	34.11	87779
Q26	137	26.0	33.61	87019
Q31	128	31.0	33.37	88498

Table 2: The number of processor instructions required by the decoder for different encoder parameters for the *Foreman* clip.

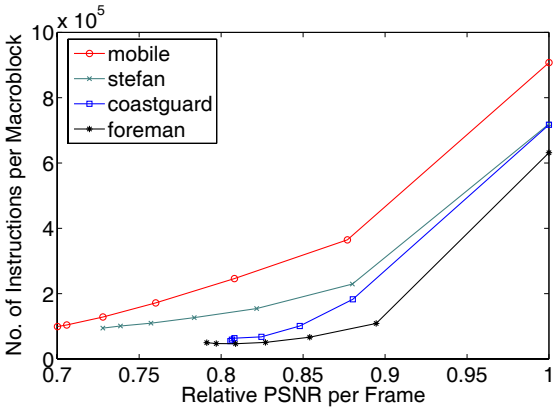

Figure 3: Tradeoff curves showing how the average number of instructions required to process a macroblock increases as the average PSNR value per frame is increased.

example, high-end PDAs with a 400-500MHz processor can decode the *Mobile* clip with a relative PSNR value of about 0.88. A laptop with a 1.5-2GHz processor can decode all the four video clips with the highest possible video quality. When encoding video clips that are meant to be decoded

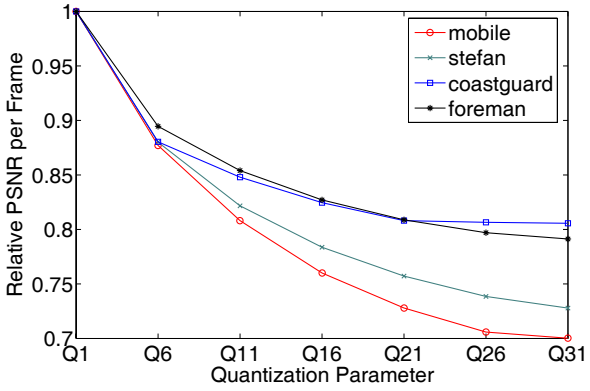

Figure 4: The relative average PSNR value per frame corresponding to each quality parameter value.

on such devices, Figures 3 and 4 can be used to estimate appropriate encoding parameter values.

From Figure 3, it is also possible to identify a good tradeoff between the video quality and the corresponding processing requirements of the decoder application. It may be noted that the relative PSNR values between 0.85 and 0.90 represent the "sweet-spots" for the four selected video clips. The corresponding quality parameter values to these PSNR values can be identified from Figure 4. These "sweet-spots" suggest the most cost-effective frequency values at which a processor should be run in order to decode these video clips. Typically, a processor would also run other applications, in which case these sweet-spots indicate the processor bandwidth that should be allocated to the decoder application.

For the *Foreman* clip, a PSNR value of around 0.89 represents the "sweet-spot". A processor frequency of around 100MHz is required to support this PSNR value, which is well within the reach of low-end PDAs. For the *Mobile* clip, the optimal choice of the average PSNR value would be around 0.87, which can be supported by a high-end PDA with a 400MHz processor.

3.2 Component Types versus Processing Requirements

We also studied the processing requirements of the different tasks of the decoder at the macroblock granularity. For the clips *Mobile* and *Foreman*, Figures 5 and 6 show the average number of instructions required by each task of the decoder to decode each macroblock for different quality parameter values. Table 3 lists the different tasks shown in these two figures.

From Figures 5 and 6, it may be noted that the processor cycle requirements stabilize beyond a certain value of the quality parameter. As the value of this parameter is increased, video clips having a lower amount of global-motion approach this point earlier than those having higher

Abbreviation	Full Component Name
MC	Motion Compensation
VLD	Variable Length Decoding
I-Pred	Inverse AC DC Prediction
IQ	Inverse Quantization
IDCT	Inverse Discrete Cosine Transform
ACU	Add Copy Upsampling

Table 3: Decoder tasks and their abbreviations.

amounts of global motion. For example, the processor cycle requirements rapidly decrease for the high-motion *Mobile* clip till Q26, beyond which there is no significant decrease. However, for the low-motion *Foreman* clip there is almost no decrease in the cycle requirements beyond the quality parameter value of Q11. This trend indicates that there is no reason to encode such clips with parameter values greater than Q11, with the aim of decreasing the computational workload generated by the decoder application.

It may also be noted that the relative distribution of the processor cycle requirements among the different tasks changes with the different quality parameter values. This distribution may also change with the amount of global motion present in a video clip. For example, when the quality parameter value is set to Q1, I-Pred consumes less cycles than IQ for both *Mobile* and *Foreman* clips. However, I-Pred consumes more cycles than IQ for the *Foreman* clip beyond the quality parameter value of Q1, while this happens for the *Mobile* clip from Q26 onwards.

The above information on the processing requirements of the different tasks would allow a designer to understand which tasks of the decoder account for large differences in the instruction counts for different encoding parameters. This information can also help in partitioning and mapping an MPEG-4 decoder application onto a multiprocessor architecture and also identifying candidate tasks which are more suitable for hardware implementations. Further, for resource-constrained devices where the architecture is prespecified, our results provide insights into the appropriate choice of the encoder parameters. Lastly, they can also be useful for designing schedulers when the decoder concurrently runs with other applications.

We also measured the processor cycle requirements of each task for the different video clips. In general, the cycle requirements of a task increase as the quality parameter value increases and when more global motion is present in the clip, as shown in Figure 7. Exceptions, however, exist. As shown in Figure 8, the I-Pred task consumes more cycles for the higher-motion *Mobile* clip than for the lower-motion *Stefan* clip when the quality parameter value is less than or equal to Q11, but when the quality parameter value is beyond that, more cycles are consumed by the lower-motion *Stefan* clip.

4 Related Work

Lately, there has been a considerable amount of work on various aspects of the MPEG-4 standard. Akramullah et. al. [2] designed and developed a real-time MPEG-4 video AS Profile encoder at Level 5 on a single chip platform which has a programmable VLIW-based processor. Similarly, Bereković et. al. [3] describes the implementation of an optimized software MPEG-4 decoder on a programmable multi-core system-on-chip architecture, based on a detailed analysis of the bit-stream statistics.

Complexity analysis of both the MPEG-4 encoder and decoder has also been a topic of extensive study (see reference [4]). Techniques have also been proposed to increase the computational scalability of video encoding algorithms, while considering the coding efficiency (in terms of encoded bit-rate) or video quality (in terms of PSNR) (see references [5, 6]).

Bhatkar et. al [7] and Zhao et. al. [8] study the effects of the encoder parameters (as we do in this paper) on the system energy, but focus only on the encoder application. This is useful when, for example, a mobile phone is used to capture a video image and encode it into an MPEG-4 clip. However, in our opinion, studying these effects on the *decoder* is often more relevant since the decoder is used more frequently on a portable device. A closely related work has recently been presented by Zaccarin [9], which proposes a method for selecting encoding options based on power constraints to be satisfied by the decoder. Finally, Lu and Sheinin [10] presents a memory architecture (based on a *Dedicated Video Internal Memory* for Systems-on-Chip architectures) that is specifically designed for video decoding. The encoder is supposed to be aware of this architecture while encoding video clips, such that the decoder can exploit this architecture to reduce off-chip memory traffic.

However, none of the above papers quantitatively study the tradeoffs between video quality versus the computational workload generated by the decoder, as we do in this paper.

5 Concluding Remarks

In this paper we have studied the effects of different encoder parameters in MPEG-4 on the processor cycle requirements of the decoder application, and quantitatively identified the tradeoffs between video quality and the processor cycle requirements. Our results can provide insights to multimedia application developers and also guide the hardware-software co-design of MPEG-4 decoders on resource constrained devices. The MPEG-4 standard is likely to be supported in a wide range of devices in the recent future, and thus we believe that more work in the direction of this paper will be useful. In future, we plan to more accurately characterize the MPEG-4 decoder workload (e.g. measure the *variability* in the workload)

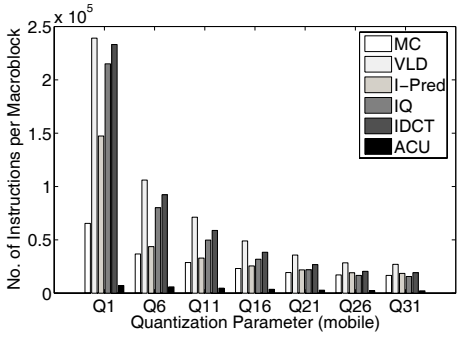

Figure 5: Average number of instructions required by each macroblock by the different tasks of the decoder for the *Mobile* clip.

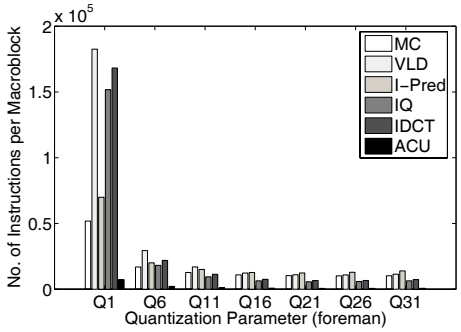

Figure 6: Average number of instructions required by each macroblock by the different tasks of the decoder for the *Foreman* clip.

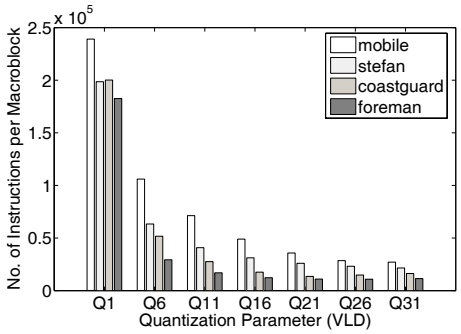

Figure 7: Average number of instructions required by the VLD task to decode a macroblock, for the different video clips.

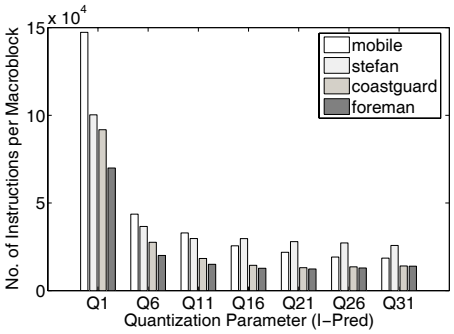

Figure 8: Average number of instructions required by the I-Pred task to decode a macroblock, for the different video clips.

using techniques in the spirit of those described in [11]. It would also be useful to develop analytical models to predict the workload generated for different encoder parameters, especially considering the high simulation times required to obtain the results presented in this paper. Another important direction is to study how different encoding parameters can affect the memory access pattern of the MPEG-4 decoder, especially since memory accesses are often a bottleneck in multimedia applications.

Acknowledgements: The authors gratefully acknowledge the insightful comments provided by the reviewers which helped in improving the quality of the paper.

References

[1] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[2] S. M. Akramullah, R. K. Giduthuri, and G. Rajan, "MPEG-4 advanced simple profile video encoding on an embedded multimedia system," in *SPIE Visual Commun. Image Processing*, San Jose, CA, Jan. 2004.

[3] M. Bereković, H.-J. Stolberg, and P. Pirsch, "Multicore system-on-chip architecture for MPEG-4 streaming video," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, no. 8, August 2002.

[4] P. M. Kuhn and W. Stechele, "Complexity analysis of the emerging MPEG-4 standard as a basis for VLSI implementation," in *SPIE Visual Commun. Image Processing*, Jan. 1998.

[5] S. Mietens, P.H.N. de With, and C. Hentschel, "Resource-aware complexity scalability for mobile MPEG encoding," *SPIE Visual Commun. Image Processing*, Jan. 2004.

[6] J. Jung and A. Bourge, "Power scalable video encoder for mobile devices based on collocated motion estimation," in *SPIE Visual Commun. Image Processing*, San Jose, CA, Jan. 2004.

[7] A. Bhatkar et al., "Computation and transmission energy modeling through profiling for MPEG4 video transmission," in *IEEE ICME*, Baltimore, MD, July 2003.

[8] J. Zhao et al., "Influence of MPEG-4 parameters on system energy," in *Proc. IEEE ASIC/SOC Conference*, Rochester, NY, Sept. 2002.

[9] A. Zaccarin, "Methods and apparatuses for selecting encoding options based on decoding energy requirements," in *U.S. Patent Application Publication, No.: 2003/0198294 A1*, 2003.

[10] L. Lu and V. Sheinin, "Video coding for decoding power constrained embedded devices," in *SPIE Visual Commun. Image Processing*, 2004.

[11] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong, "Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs," in *CODES+ISSS*, Stockholm, Sweden, Sept. 2004.