J. Li, “Embedded Audio Coding (EAC) With Implicit Auditory Masking”, ACM Multimedia 2002, Nice, France, Dec. 2002.

Embedded Audio Coding (EAC) With Implicit Auditory Masking

Jin Li
Microsoft Research, One Microsoft Way, Bld. 113, Redmond, WA 98052.
Tel. + 1 (425) 703-8451 Email: jinl@microsoft.com

ABSTRACT

An embedded audio coder (EAC) is proposed with compression performance rivals the best available non-scalable audio coder. The key technology that empowers the EAC with high performance is the *implicit auditory masking*. Unlike the common practice, where an auditory masking threshold is derived from the input audio signal, transmitted to the decoder and used to quantize (modify) the transform coefficients; the EAC integrates the auditory masking process into the embedded entropy coding. The auditory masking threshold is derived from the encoded coefficients and used to change the order of coding. There is no need to store or send the auditory masking threshold in the EAC. By eliminating the overhead of the auditory mask, EAC greatly improves the compression efficiency, especially at low bitrate. Extensive experimental results demonstrate that the EAC coder substantially outperforms existing scalable audio coders and audio compression standards (MP3 and MPEG-4), and rivals the best available commercial audio coder. Yet the EAC compressed bitstream is fully scalable, in term of the coding bitrate, number of audio channels and audio sampling rate.

Keywords

Audio compression, scalable, JND threshold, implicit auditory masking, entropy coding, sub-bitplane, bitstream assembler

1. INTRODUCTION

The availability of high performance audio codec brings digital music into reality, and revolutionizes our audio experiences. The most popular audio compression technology today is probably MP3[4], which stands for layer III of the MPEG-1 audio compression standard. MP3 device is quickly replacing cassette and CD player as the choice for music playback; and swapping MP3 compressed songs over the internet has become a national hobby for young college students. Developed in the early 1990s, MP3 does not perform very well in terms of the compression efficiency. More advanced audio compression technologies have been proposed later, such as the MPEG-4[5][11], Real™ and Windows Media Audio (WMA™). The later two are commercial audio coders developed by RealNetworks and Microsoft, respectively.

Most existing audio coders optimize only on a single target compression ratio, striving to deliver the best perceptual audio quality given the length of the bitstream, or deliver the shortest length of the bitstream given a constraint on playback quality. However, such goal is far from enough, especially considering the unique characteristics of media (including audio) compression. Unlike data compression, where all content must be exactly preserved during the compression, media compression is elastic and tolerates distortion. It is always possible to compress the media a little more or a little less, with corresponding larger or smaller distortion. In fact, in many applications, it is difficult to foresee the exact compression ratio required at the time the media is compressed. The ability to quickly change the compression ratio afterwards has important applications, and led to better user experience in media storage and transmission. For example, if the compression ratio on the stored media is adjustable, the compressed media can be quickly tailored to meet the exact requirements of the customer. The storage device can use the highest possible compression ratio so long as all the compressed media fits in the device, and thus waste no storage space. When more media needs to be stored, the device can simply increase the compression ratio of the existing media, free up the storage space and squeeze in new content. The ability to quickly adjust the compression ratio is also very useful in the media communication/streaming scenery, where the server and the client may adjust the size of the compressed media to match the instantaneous bandwidth and condition of the network, and thus reliably deliver the best possible media quality over network.

Transcoding technologies have been developed to adjust the compression ratio of standard compressed bitstream, such as MP3. Comparing with decoding and then re-encoding the media, transcoding achieves modest computation saving by skipping part of the compression operations, mainly the inverse and forward transform and motion estimation (for video transcoding). Almost all existing transcoding techniques still need to perform the entropy coding, therefore, the speed of the transcoding is not very fast, usually at least 25% of that of media encoding.

The scalable/embedded media coders have attracted much attention recently. Pioneered by Shapiro[12] in the work of the embedded zerotree wavelet (EZW) image coding, embedded coder has the attractive property that the high compression ratio bitstream (called application bitstream) is embedded in the low compression ratio bitstream (called master bitstream). The conversion of compression ratio can thus be done very quickly by extracting the subset of the compressed bitstream that corresponds to the application bitstream from the master bitstream. In the case of embedded image compression, this operation can be further simplified to truncate the existing bitstream. In the domain of image compression, it has been shown[13][10][14] that embedded coding can not only achieve flexible bitstream adjustment, but also obtain state-of-the-art compression performance and reasonable computation complexity. In fact, the most recent image compression standard – the JPEG-2000[7] is an embedded image coder. Inspired by the success in embedded image coding, a number of embedded audio[11][22][23][24] and video[15] compression algorithms have been developed. However, none achieves the popularity and success of the embedded image coding. A primary reason is that the existing scalable audio and video coders lag quite a bit behind the state-of-the-art in compression performance. There is thus a misconception that you have to pay for the scalable

ACM Multimedia '2002, Dec.1-6, 2002, Nice, France.

functionality with compression performance, which renders the scalable audio and video coder unattractive.

However, we believe that the misconception is not true. Just as the embedded image coder does not take off until highly efficient bitplane entropy coder is developed, highly efficient embedded audio (and video) coder needs unique technologies that suit its need for embedded coding. In this work, we have developed an embedded audio coder (EAC) with performance exceeds or matches that of the best available audio coders. The key technology that empowers EAC with such high performance is the use of the *implicit auditory masking*. Unlike all existing audio coders and the existing embedded audio coding approaches, EAC does not send the auditory masking thresholds to the decoder. Instead, they are derived from already coded coefficients. Furthermore, rather than quantizing (changing) the audio coefficients according to the auditory masking thresholds, the masking thresholds control the order that the coefficients are encoded. The implicit auditory masking approach has several advantages. By deriving the auditory masking thresholds from the coded coefficients, the overhead of sending the masking thresholds is eliminated. The audio compression efficiency can thus be substantially improved, especially at low bitrate. It is also feasible to adopt different psychoacoustic models at different stages (bitrate) of encoding, and improve the perceptual quality of the compressed audio over a wide range of bitrate. Moreover, the implicit auditory masking approach produces no error sensitive header. The bitstream is thus more robust to be transmitted over error prone channels, such as a wireless channel.

The rest of the paper is organized as follows. The framework of the EAC coder is outlined in Section 2. Two most important modules of the EAC, the sub-bitplane entropy coder with implicit auditory masking and the bitstream assembler are examined in Section 3 and 4. Experimental results are shown in Section 5. And finally, we give a conclusion in Section 6.

2. FRAMEWORK OF THE EMBEDDED AUDIO CODER (EAC)

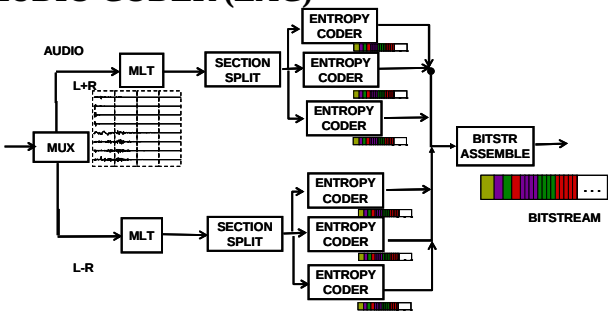

Figure 1 Framework of the embedded audio coder

The embedded audio coder (EAC) is a fully scalable generic waveform coder. There are three components of the EAC: an encoder that turns the input audio waveform into a compressed bitstream, a decoder that turns the compressed bitstream into a playback audio waveform, and a parser that extracts a subset of the master compressed bitstream to form an application bitstream with increased compression ratio, reduced sampling rate or reduced number of audio channels. The framework of the EAC encoder can be shown in Figure 1. The EAC decoder is simply the reverse.

The input audio waveform first goes through a multiplexer (MUX). If the input audio is stereo, it is separated into L+R and L-R components, where L and R represent the waveform of the left and right audio channel, respectively. If the input audio is mono, the MUX simply passes through the audio. The MUX operation ensures that the compressed bitstream can be scaled by audio channels, as the mono audio compressed bitstream can be obtained from the compressed stereo master bitstream by simply dropping the L-R components.

The waveform of each audio component is then transformed by a modulated lapped transform (MLT)[16] with switching windows. Assuming the input audio is sampled at 44.1kHz, the size of the MLT window is adaptively adjusted between 2048 and 256 samples. The long window is used for homogeneous audio segments, while the short window is used for audio segments with large energy fluctuation to reduce the effect of pre-echoing. After the MLT transform, the coefficients are optionally split into a number of sections. A sample split is shown in Table 1, where the long and short windows are split into three sections of 0-0.25π, 0.25-0.50π and 0.50-1.00π. Such section split enables the sampling rate of the reconstructed audio to be at 11.025, 22.05 and 44.1 kHz, respectively. In case the decoder does not have a speaker with high frequency response, or it wants to save the computation power, the section corresponding to the high frequency coefficients can be dropped, while the remaining audio coefficients are inversely transformed by a MLT with a half (dropping section 3) or a quarter of the window size (dropping section 2 and 3). Using again our example, if the section 3 coefficients are not used, the decoded coefficients in section 1 and 2 can be inversed transformed by a MLT of window size 1024 and 128 for the long and short window, respectively; and result in a decoded audio waveform sampled at 22.05kHz (11.025kHz bandwidth). Such audio sampling rate reduction can be considered as passing the audio waveform through a low pass filter that first transforms the audio by MLT, throws away half the coefficients, and then inversely transforms the coefficients with MLT at half window size. The section split provides an effective means of sampling rate reduction of the compressed audio.

Each section of the MLT transform coefficients is then entropy encoded into an embedded bitstream, which can be truncated and reassembled later. To improve the efficiency of the entropy coder, we group the MLT coefficients of a certain number of consecutive windows into a timeslot. In our experimental configuration, a timeslot consists of 16 long MLT windows or 128 short windows, though this number can be easily changed. A timeslot therefore consists of 32,768 samples, which is about 0.74 second if the input audio is sampled at 44.1kHz. The functionality of the entropy coder is thus to convert the coefficients of a section of a timeslot into an embedded bitstream.

Finally, a bitstream assemble module allocates the available coding bits among the timeslots, channels, and sections, and produces the final compressed bitstream.

Among the EAC modules, the MUX and MLT modules are conventional. The entropy coding module employs a unique technology termed the *implicit auditory masking*, which enables the performance of the embedded audio coder to rival the state-of-the-art generic audio coders. The bitstream assemble module ensures that the EAC compressed bitstream can be quickly reshaped to produce an application bitstream of different compression ratio, number of audio channels and audio sampling rate. Our discussion in the following is hence focused on the entropy coding module and the bitstream assemble module.

Table 1 MLT coefficient split.

Window size	Section 1 (0-0.25π)	Section 2 (0.25-0.50π)	Section 3 (0.50-1.00π)
2048 (long)	0-511	512-1027	1028-2047
256 (short)	0-63	64-127	128-255

3. SUB-BITPLANE ENTROPY CODER WITH IMPLICIT AUDITORY MASKING

In this section, we discuss the motivation, framework and implementation of the entropy coder with the implicit auditory masking that enables the high performance of the embedded audio coder (EAC). We first explain the human auditory system in Section 3.1. The difference between the implicit and the conventional auditory masking is elaborated in Section 3.2. The basic bitplane entropy coder is discussed in Section 3.3. Implementation of the implicit auditory masking on top of the bitplane entropy coder is shown in Section 3.4. The speed up of the implicit auditory masking operation is explained in Section 3.5.

3.1 Background – Human auditory masking

Human ear does not respond equal to all frequency components. The auditory system can be roughly divided into 26 *critical bands*, each of which is a bandpass filter-bank with bandwidth in the order of 50 to 100Hz for signal below 500Hz, and up to 5000Hz for signal at high frequencies. Within each critical band, an auditory masking threshold, which is also referred as the psychoacoustic masking threshold or the threshold of the just noticeable distortion (JND)[1], can be determined. Audio waveform with energy level below the threshold will not be audible.

The auditory JND threshold is highly correlated to the spectral envelope of the signal. For different audio input waveforms, the auditory JND threshold level can be completely different. This is rather dissimilar to the JND threshold in the human visual system, where the masking of weak signal by the nearby strong signal occurs in a very short range, and the dominant visual sensitivity is the same for a certain frequency regardless of the input.

Let the auditory JND threshold of a critical band k at time instance i be $TH_{i,k}$. The JND threshold can be calculated as the maximum of a quite threshold and a masking threshold. The quite threshold TH_ST_k dictates the sensitivity of the auditory system for critical band k without the presence of any audio signal. It can be calculated through an equal loudness curve, such as the Fletcher-Munson curve [2] shown in Figure 2. According to the quite threshold, the sensitivity of the ear is nearly linear for a large range (1-8kHz), and drops dramatically before 500Hz and after 10kHz .

In the presence of input audio signal, the auditory JND threshold is largely shaped by masking, which states that a low-level signal (the maskee) can be made inaudible by a simultaneously occurring strong signal (the masker) as long as the masker and the maskee are close enough to each other in time and frequency. The most basic form of the auditory masking is the simultaneous intra-band masking, where the maskee and the masker are at the same time instance and within the same critical band. The intra-band masking threshold $TH\_INTRA_{i,k}$ is directly proportional to the spectral envelope of the masker (its average energy) in the critical band of the same time instance $AVE_{i,k}$, and can be expressed as:

$$TH\_INTRA_{i,k}\ (dB) = AVE_{i,k}(dB) - R_{fac}, \qquad (1)$$

where R_{fac} is a constant offset value determined through psychoacoustic experiment. The masker also masks small signal in the neighbor critical bands. The level of such inter-band masking $TH\_INTER_{i,k}$ can be formulated as:

$$TH\_INTER_{i,k} = max\ (TH_{i,k-1} - R_{high},\ TH_{i,k+1} - R_{low}), \qquad (2)$$

where R_{high} and R_{low} are the attenuation factor towards the high and low-frequency critical band, respectively. The higher frequency coefficients are more easily masked; thus the attenuation R_{high} is smaller than R_{low}. Combining the quite, intra- and inter-band auditory masking, the auditory masking threshold created by a strong audio signal identified as the "masker" can be illustrated in Figure 2, where the auditory JND threshold is shown as the dashed line. Any sound below the JND threshold, e.g., compression distortion, will not be audible by the human ears.

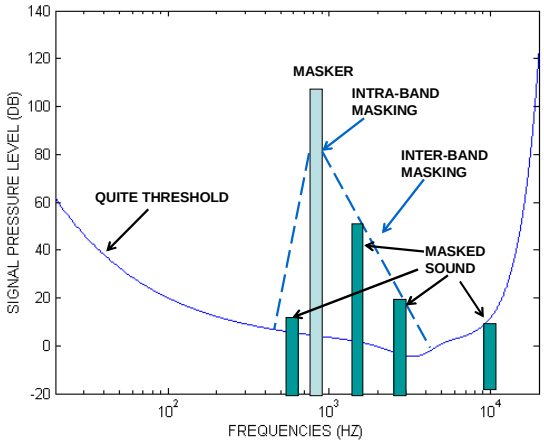

Figure 2 Auditory masking threshold.

Strong masker can also mask weak signal in the immediate proceeding or following time interval. The effect is called *temporal masking*. The duration within which premasking applies is less than one tenth of that of the postmasking, which is in the order of 50 to 200 ms. The temporal masking threshold $TH\_TIME_{i,k}$ can be calculated as:

$$TH\_TIME_{i,k} = max(\ TH_{i-1,k} - R_{post},\ TH_{i+1,k} - R_{pre}), \qquad (3)$$

where R_{pre} and R_{post} are the attenuation factor for the proceeding and following time interval, respectively. A sample temporal masking generated by a masker can be shown in Figure 3.

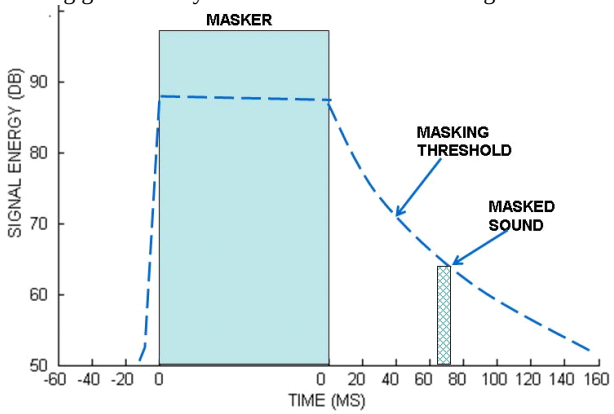

Figure 3 Temporal auditory masking

The combined auditory JND threshold is the maximum among the quite threshold, the intra- and inter-band masking, and the masking of the neighborhood time instance:

$$TH_{i,k} = max\ (TH\_ST_k,\ TH\_INTRA_{i,k},\ TH\_INTER_{i,k},\ TH\_TIME_{i,k}), \qquad (4)$$

Calculation of the JND threshold requires the iteration of (2)-(4). Thus, if the input audio consists of several strong maskers, the

combined JND threshold will be the maximum of the masking threshold generated by the individual masker.

3.2 Auditory masking – regular versus the implicit approach

By using the auditory masking, the audio coder may devote fewer bits to the coefficients that are less sensitive to the human ear, and more bits to the auditory sensitive coefficients; and thus improve the quality of the coded audio. Since the auditory JND threshold is strongly dependent on the input signal, in all existing psychoacoustic audio coders, the auditory masking operation is applied in the following way. It is the encoder that calculates the JND threshold based on the spectral envelope of the input audio waveform, which is then encoded as a part of the compressed bitstream and transmitted to the decoder. The encoder quantizes the transform coefficients with a step size proportional to the JND threshold. Therefore, the coefficients are quantized coarsely in the critical bands with a larger JND threshold, and are quantized finely in those with a smaller JND threshold. The approach can be illustrated with Figure 4.

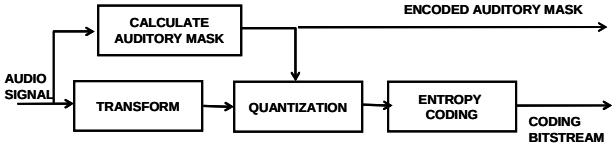

Figure 4 Auditory masking in a traditional coder.

Such approach may be fine for a non-scalable coder. However, using the same framework for scalable audio coders, as practiced by existing scalable audio coders such as [11][22][23][24], is not efficient. First, sending the auditory JND threshold consumes a non-trivial amount of bits, which can be as much as 10% of the total coded bits. Since the auditory masking module is applied before the entropy coding module, the JND threshold must be transmitted with the same precision regardless of the compression ratio. The JND threshold overhead thus eats significantly into the bit budget, especially if the compressed bitstream is later reshaped to a low bitrate. Second, as shown in Section 3.1, the JND threshold is shaped by the energy distribution of the input audio, while the same energy distribution is revealed through the bitplane coding process of the entropy coder. As a result, the information is coded twice, which wastes the precious coding bits.

In the embedded audio coder (EAC), we integrate the auditory masking module with the embedded entropy coding module. Once integrated, the auditory masking can be done in an entirely new way, with two unique features. First, we derive the auditory JND threshold from the spectral envelope of the partially encoded coefficients, rather than the original coefficients. Second, we use the auditory JND threshold to determine the order that the audio coefficients are encoded, rather than to change the coefficients (by quantizing them). We call the approach implicit auditory masking, because the auditory JND threshold is never transmitted to the decoder.

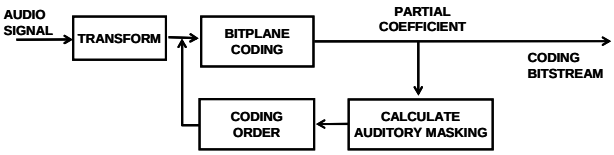

Figure 5 Process of the implicit auditory masking.

The framework of the implicit auditory masking can be shown in Figure 5. Compared to Figure 4, the auditory masking operation is now integrated into the loop of the entropy coding module, and is performed as follows. We first set the initial auditory JND threshold, e.g., using the quite threshold. A portion of the transform coefficients, e.g., the top bitplanes, is then encoded. Afterwards, an updated auditory JND threshold is calculated based on the spectral envelope of the partially coded transform coefficients. Since the decoder may derive the same auditory JND threshold from the same coded coefficients, the value of the auditory JND threshold needs not to be sent to the decoder. Using this implicitly calculated JND threshold, both the encoder and the decoder figure out which portion of the transform coefficients is encoded next. After the next portion of the coefficients has been encoded, the auditory JND threshold is updated again, which is then used to guide the coding order of the remaining portion of the coefficients. The process iterates among the operation of sending a portion of MLT coefficients, updating the JND threshold, and using the updated JND threshold to determine the portions to be sent next. It only stops when a certain end criterion has been met, e.g., all transform coefficients have been encoded, or a desired coding bitrate has reached, or a desired coding quality has reached.

By deriving the auditory masking threshold implicitly from the partially coded coefficients, bits normally required for the auditory JND threshold are saved. The saving can be especially large at low bitrate, or when the coding bitstream is later truncated to a lower bitrate. The implicit auditory masking may thus significantly improve the compression efficiency, and enables the embedded audio coder to gain an edge over the non scalable audio coder. Moreover, in all existing audio coders, the auditory JND threshold is carried as a header of the bitstream. The corresponding rate-distortion (R-D) performance curve can be shown as the dashed curve in Figure 6. At first, the coding bits are for the auditory JND threshold. Since just decoding the JND threshold without any coded coefficients provides no valuable information, the R-D curve is at first flat, and then drops as the coding of the auditory coefficients kicks in. In contrast, the R-D curve of the EAC coder with implicit auditory masking can be shown as the dashed dotted curve in Figure 6. Because the coding bits are used from the start for coefficient coding, the R-D curve is smoother, and drops right from the beginning. With no error sensitive header, the EAC compressed bitstream is less susceptible to the transmission error, and therefore offers better error protection in a noisy channel, such as in the wireless environment. The third advantage of the implicit auditory masking results from the fact that instead of coding the auditory insensitive coefficients coarsely, the EAC encodes them in a later stage. By using the auditory masking to govern the coding order, rather than the coding content, the quality of the compressed audio becomes less sensitive to the accuracy of the auditory JND threshold, as slight deviation in the threshold simply causes certain audio coefficients to be coded later. In Section 3.5, this important observation enables the efficient implementation of the implicit auditory masking.

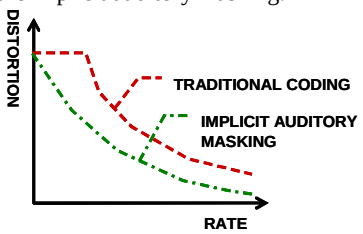

Figure 6 Rate-distortion of the traditional coder (dashed curve) vs. the EAC with implicit auditory masking (dashed doted curve).

There are two key operations in the implicit auditory masking: determining the coding order and implicitly deriving the auditory JND threshold. Each operation may be performed on a different granularity level. For example, we can update the auditory JND threshold right after we encode one more bit of one transform coefficient; and then reorder the coefficient coding based on the newly updated JND threshold. However, such frequent update of the auditory JND threshold and coding order is computational intensive, and may not gain much coding efficiency if the reordering shuffles two units with roughly the same perceptual distortion improvement per coding bit spent. We thus group a number of bits of a set of transform coefficients into an embedded coding unit (ECU), which is the smallest unit in the reordering operation. We also determine an update interval which is the instance that the auditory JND threshold is recalculated. Because a slightly outdated auditory JND threshold only leads to a slight non-optimal coding order of the ECUs, its impact on the compression performance is minimal. We may therefore choose to update the auditory JND threshold infrequently, thereby save the computation complexity. The operation of the implicit auditory masking can be concluded in Figure 7. First, the initial auditory JND thresholds are calculated, e.g., by using the quite threshold. The bits of the audio coefficients are then separated into a number of ECUs. Using the initial threshold, the coding order of the ECU is determined, and a set of high priority ECUs are encoded. When the update interval is reached, the auditory JND threshold is recalculated by both the encoder and decode based on the coded ECUs. The updated threshold is then used to determine the coding order of the remaining ECUs. The process iterates until a certain end condition is met.

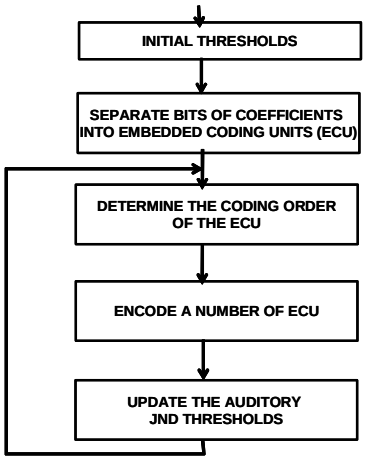

Figure 7 Implicit auditory masking operation.

3.3 Bitplane entropy coder

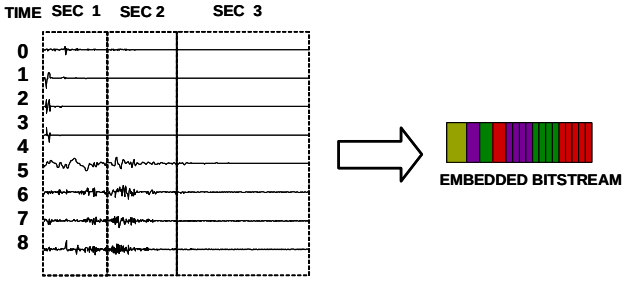

Figure 8 Compression of a section into an embedded bitstream.

Before the discussion of a specific implementation of the implicit auditory masking, we first describe the entropy coder used in the EAC – the bitplane embedded entropy coder. The functionality of the entropy coding module can be illustrated in Figure 8. As mentioned in Section 2, the input audio waveform is MLT transformed. After the transform, a certain number of MLT windows are grouped together to form a timeslot. The coefficients in the timeslot are then optionally divided into sections for sampling rate scalability. The job of the sub-bitplane entropy coding module is thus to compress each section of the timeslot independently into an embedded bitstream, which may be truncated at a later stage.

Since the human auditory masking is based on critical band, we further divide the MLT coefficients in each window into critical bands. Let i index the time instance, j index the frequency component, and k index the critical band. Let $x_{i,j}$ be a coefficient at time instance i, frequency j, and $s_{i,k}$ be a critical band k at time instance i.

The embedded coder encodes the audio coefficient bit by bit. Let each audio coefficient be represented in the binary form as:

$$[\ \pm b_{L-1}\ b_{L-2}\cdots b_0\ ]$$

where b_{L-1} is the most significant bit (MSB), and b_0 is the least significant bit (LSB), ± is the sign of the coefficient. A group of bits of the same significance from different coefficients forms a bitplane. For example, bit b_{L-1} of all coefficients forms the most significant (L-1) bitplane. The principle of the embedded coding is to encode the coefficients bitplane by bitplane, first the most significant bitplane, then the second most significant bitplane, and so on. This way, if the output compressed bitstream is truncated, at least part of each coefficient can be decoded.

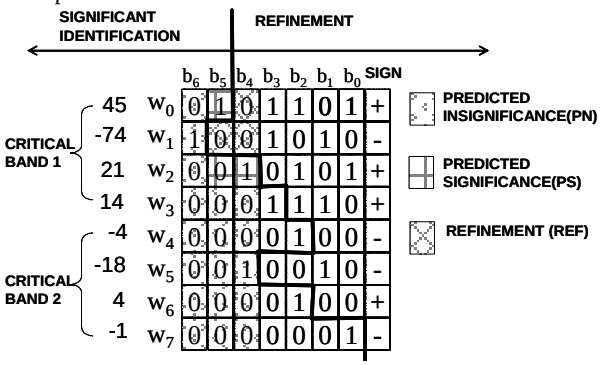

Figure 9 Embedded coding.

All bits of the coefficients in a timeslot form a bit array. We show a sample bit array in Figure 9. Since the coefficient in the EAC is actually arranged in a 2D array indexed by the time instance i and frequency j, the actual bit array is 3D. However, it is difficult to draw a 3D bit array, therefore, we show a slice of the bit array in 2D in Figure 9, which can be considered as a slice of the 3D bit array at a certain time instance. Note that the sign of the coefficient is also part of the bit array, as the '+' and '-' signs can be represented by 0 and 1, respectively. The bits in the bit array are statistically different. Let b_M be a bit in a coefficient x which is to be encoded. If all more significant bits in the same coefficient x are '0's, the coefficient x is said to be insignificant (because if the bitstream is terminated right after bit b_M has been coded, coefficient x will be reconstructed as zero), and the current bit b_M is to be encoded in the mode of significance identification. Otherwise, the coefficient is said to be significant, and the bit b_M is to be encoded in the mode of refinement. We distinguish between sig-

nificance identification and refinement bits because the significance identification bit has a very high probability of becoming '0', and the refinement bit is usually equally distributed between '0' and '1'. The sign of the coefficient only needs to be encoded immediately after the coefficient turns significant, i.e., a first non-zero bit in the coefficient is encoded. For the bit array in Figure 9, the significance identification and the refinement bits are separated by a solid bar. For a critical band $s_{i,k}$, we call it insignificant if all the coefficients in the critical band are insignificant. It becomes significant when at least one coefficient is significant.

The significant identification bits, refinement bits and signs are not statistically equal even within their own category. Statistical analysis demonstrates that if a MLT coefficient $x_{i,j}$ is of large magnitude, its time and frequency neighbor coefficient may be of large magnitude as well. Moreover, its frequency harmonics (at double and/or triple frequency) may be of large magnitude too. To account for such statistical difference, we entropy encode the significant identification bits, refinement bits and signs with context, each of which is a number derived from the already coded coefficients in the neighborhood of the current coefficient. Such technique is called context adaptive entropy coding, and is frequently used in modern image/audio/video coding systems.

We first describe the context for the refinement bits and signs because they are simpler. The context of the refinement coding bits depends on the significant statuses of the four immediate neighbor coefficients, which for coefficient $x_{i,j}$ are the coefficients with the same frequency but at the proceeding ($x_{i-1,j}$) and following ($x_{i+1,j}$) time instance, and coefficients at the same time instance but with a lower ($x_{i,j-1}$) and higher ($x_{i,j+1}$) frequency, as shown in Figure 10. The refinement context is formed according to Table 2. If one of the four neighbor coefficients is unreachable as it falls out of the current timeslot or the current section, it is considered insignificant.

Table 2 Context for the refinement bit.

Context	Description
10	Current refinement bit is the first bit after significant identification and there is at least one significant coefficient in the immediate four neighbors
11	Current refinement bit is the first bit after significant identification and there is no significant coefficient in the immediate four neighbors
12	Current refinement bit is at least two bits away from significant identification.

Table 3 Calculation of sign count.

Sign count	Description
-1	Both coefficients are negative significant; or one is negative significant, and the other is insignificant.
0	Both coefficients are insignificant; or one is positive significant, and the other is negative significant.
1	Both coefficients are positive significant; or one is positive significant, and the other is insignificant.

Table 4 Expected sign and context for sign coding.

Sign count										
	h	-1	-1	-1	0	0	0	1	1	1
	v	-1	0	1	-1	0	1	-1	0	1
Expected sign		-	-	+	-	+	+	-	+	+
Context		13	14	15	16	17	16	15	14	13

To determine the context for sign coding, we calculate a horizontal sign count *h* and a vertical sign count *v*. We separate the four neighbor coefficients into two pairs, a horizontal pair ($x_{i,j-1}$ and $x_{i,j+1}$) and a vertical pair ($x_{i-1,j}$ and $x_{i+1,j}$). For each pair, the sign count is calculated according to Table 3. The expected sign and the context of sign coding can thus be further calculated according to Table 4.

The context for the refinement and sign coding are designed with reference to the context used in the JPEG 2000 standard [7], as we felt that the design suits our need. However, the significant identification context is specially tailored for audio coding. To calculate the context of the significant identification bit, we not only use the significant statuses of the four neighbor coefficients, but use the significant statuses of the half harmonics $x_{i,j/2}$ (shown in Figure 10) and the MLT window size. The use of the half harmonic frequency is due to the fact that most sound producing instruments produce harmonics of the base tone. Therefore, there is strong correlation among the coefficient and its harmonics. The context used for the significant identification can be found in Table 5.

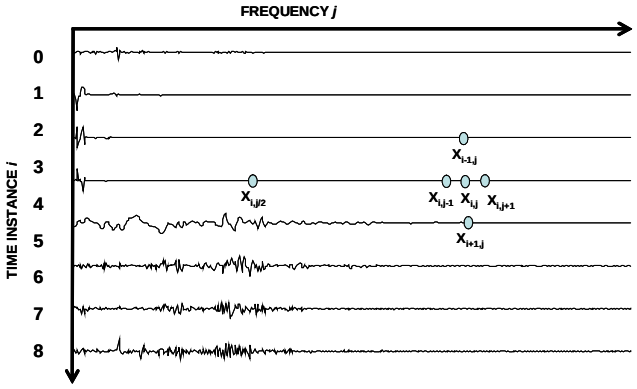

Figure 10 Illustration of neighborhood coefficients.

Table 5 Context for significant identification (S: significant, N: non-significant, *: arbitrary)

Context	MLT window size	Significant status of coefficient			
		$x_{i,j-1}$	$x_{i-1,j}$	$x_{i+1,j}$	$x_{i,j/2}$
0	2048	N	N	N	N
1	2048	*	S	*	*
2	2048	S	N	*	*
3	2048	N	N	S	*
4	2048	N	N	N	S
5	256	N	N	N	N
6	256	*	S	*	*
7	256	S	N	*	*
8	256	N	N	S	*
9	256	N	N	N	S

A total of 18 contexts are used for the embedded audio coefficient coding. Among them, there are 10 contexts for the significance identification, 3 for refinement coding and 5 for sign coding.

3.4 Implicit auditory masking and the sub-bitplane entropy coder

To apply the implicit auditory masking to the bitplane entropy coder, we need to determine the size of the embedded coding unit

(ECU), how the ECUs are reordered, and the update interval of the auditory JND threshold.

Since the human auditory system is based on the critical band, the ECU is formed by bits of the coefficients within the same critical band. We further divide bitplane in a critical band into 3 sub-bitplanes: the predicted significance (PS), the refinement (REF), and the predicted insignificance (PN). The PS sub-bitplane consists of bits of coefficients that are insignificant but has at least one significant neighbor. The REF sub-bitplane consists of bits of coefficients that are already significant, i.e., in the refinement mode. The PN sub-bitplane consists of bits of coefficients that are insignificant with no significant neighbors. The sub-bitplane design is motivated by the author's previous work on image coding [10] and the JPEG 2000 standard [7], which show that bits in different sub-bitplanes contribute different average distortion decrease per coding bits spent. For the sample bit-array in Figure 9, we show the sub-bitplane types with different shades for the first three bitplanes of the bit array. The ECU of the EAC coder is thus a sub-bitplane of bits of a critical band.

We mark the identity of the ECU by the critical band the ECU resides in and an ID that identifies the sub-bitplane. The ID is a fraction number; with the integer part be just the bitplane index, and the fraction part be assigned according to the sub-bitplane class. Currently, the PS, REF and PN sub-bitplanes are assigned with fraction value 0.875, 0.125 and 0.0, respectively. As an example, the ID of the PS sub-bitplane of bitplane 7 is 7.875. The fraction value is designed with the consideration of the average rate-distortion contribution of each sub-bitplane class. Within each critical band, we always encode the ECUs according to the descending order of its ID. For a critical band with a total of L bitplanes, the first ECU to be encoded is the PN sub-bitplane of bitplane *L-1* (ID: *L-1.0*), because all coefficients are insignificant at bitplane *L-1*. The next three sub-bitplanes to be encoded are the PS (ID:L-1.125), REF (ID:L-1.875) and PN (ID:L-2.0) sub-bitplanes of bitplane *L-2*. Subsequently, the sub-bitplanes of bitplane *L-3* are to be encoded.

With the order of ECUs within a critical band already determined, the implicit auditory masking process only needs to determine the order of the ECUs among different critical bands. More conveniently, this can be done by determining the critical bands whose ECUs are next-in-line to be coded. We assign two important properties to each critical band: an instantaneous JND threshold and a progress indicator, both are updated during the embedded coding process. The instantaneous JND thresholds are based on the partial reconstructed coefficient value of already coded ECUs, and the progress indicator records the ID of the next ECU to be encoded. It is the gap between the progress indicator and the instantaneous JND threshold that determines the coding order of ECUs. The coding process of the sub-bitplane entropy coder with implicit auditory masking can thus be described as follows:

Step 1. Initialization.

The maximum bitplane L of all coefficients is calculated. The progress indicators of all critical bands are set to the PN sub-bitplane of bitplane *L-1* (with ID: *L-1*). The initial instantaneous JND threshold of the critical band is set according to the quite threshold. We also mark all critical bands as insignificant.

Step 2. Finding the critical bands whose ECUs are to be encoded next.

For each critical band, we calculate a gap between its progress indicator and the instantaneous JND threshold. Since the progress indicator can be considered as a form of coding distortion measurement; the gap is closely related to the level of the coding noise over the auditory JND threshold, in other words the noise-mask-ratio (NMR). The largest gap among all critical bands is defined as the current gap. The value of the current gap can be negative, which simply means that the coefficients with signal energy level below the auditory JND threshold are encoded. Only the critical bands with the gap value the same as the current gap are chosen to be encoded in this iteration. Such process leads to the encoding of the critical bands with the largest instantaneous NMR level. It can be easily proven that the instantaneous JND threshold is monotonically increasing and the progress indicator is monotonically decreasing. Therefore, the current gap shrinks in every iteration.

Step 3. Critical band skipping

If a chosen critical band is insignificant (not a single coefficient is significant), a status bit is encoded to indicate whether the critical band turns significant after the coding of the current bitplane. This is an optional step. However, it speeds up the coding / decoding operation significantly, as large area of zero-bits can be skipped with this step.

Step 4. Encoding the sub-bitplane of the critical band

We locate the sub-bitplane that is next-in-line to be coded for each critical band. For each bit in the sub-bitplane, its context is calculated according to section 3.3. The string of bit and context pairs are then compressed by a modern context adaptive entropy coder, such as the QM-coder used in the MPEG-4[5] or the adaptive Golomb coder developed in [9]. It is observed that the adaptive Golomb coder has about the same compression efficiency as the QM-coder, with roughly the same complexity. The two entropy coders are interchangeable in the current EAC.

Step 5. Moving the progress indicator.

After the sub-bitplane is encoded, the progress indicator moves forward to the ID of the next sub-bitplane to be encoded.

Step 6. Updating the instantaneous JND threshold.

After all chosen critical bands have been encoded, the instantaneous JND thresholds of all critical bands are updated based upon the already coded ECUs.

Step 7. Repeating steps 2-7.

The steps 2-7 are repeated until a certain end criterion is reached, e.g., the desired coding bitrate/quality has been reached, or all bits in all coefficients have been encoded.

3.5 Calculation of the instantaneous JND threshold

Except step 6, all the above processing steps are either trivial in complexity, or can be found in a regular sub-bitplane entropy coder. The additional computation of the implicit auditory masking process is at the update of the instantaneous JND threshold by both the encoder and the decoder. In the following, we describe speed-up technologies that calculate the instantaneous JND threshold quickly.

Since we need to calculate the gap between the progress indicator and instantaneous JND threshold, both must be expressed in the same unit. We choose to represent the JND threshold in bitplanes, which is related to the *dB* representation through:

$$TH(bitplane) = TH\,(dB)\cdot \log_2(10)/20, \qquad (5)$$

Examining equations (1)-(4), we observe that the auditory JND threshold consists of two parts, the quite threshold which is a constant, and the masking threshold which is determined by the coefficient energy of the current and neighbor critical bands. Among the masking thresholds, it is the intra-band masking threshold (1) that is directly related to the energy of the critical

band. The inter-band masking (2) and temporal masking threshold (3) are then iteratively derived from the intra-band masking threshold.

The instantaneous JND thresholds can thus be calculated as follows:

Step 1. Calculating the energy of each critical band

Since the auditory JND threshold needs to be synchronized between the encoder and the decoder, the energy of the partial reconstructed coefficients is used instead of the energy of the real coefficients. An accurate energy calculation should be performed in the complex MLT domain. However, this requires additional computation of inverse MLT transform and forward complex MLT transform. We again observe that the auditory JND threshold is used to reorder the coding in the EAC, therefore, the compression performance is not so sensitive to the accuracy of the JND threshold. Besides, the error between the real and the complex MLT coefficients tends to average out in a critical band. We therefore choose to use the energy of the MLT coefficients in the real domain, which is simply the mean square value of the reconstructed coefficients in a critical band.

To further speed up the calculation of the energy of the critical band, we introduce an adjusted energy value $E_{i,k}$ for each critical band. The value of $E_{i,k}$ is the total energy of the partial reconstructed coefficients of the critical band $s_{i,k}$ adjusted by the current bitplane, i.e., divided by 2^M, where M is the current coding bitplane. The energy of the critical band is related to the adjusted energy value according to:

$$AVE_{i,k} = E_{i,k} \cdot 4^M / sizeof(s_{i,k}) \quad (6)$$

where $sizeof(s_{i,k})$ is the number of coefficients in critical band si,k.

The advantage of using the adjusted energy value $E_{i,k}$ is that it can be calculated very quickly during the embedded coding process. The value $E_{i,k}$ is first initialized to zero. Then, during the coding process, whenever a coefficient turns significant in the PN and PS sub-bitplane pass, the adjust energy $E_{i,k}$ is incremented by 1, because the partial reconstructed value of a newly significant coefficient is either +1 or -1 after adjusting by the current coding bitplane. For the refinement bit coding, the adjust energy $E_{i,k}$ does not change if the refinement bit is '0', and is incremented by a value of $2 \cdot [b_{L-1} b_{L-2} \ldots b_M] - 1$ if the refinement bit b_M is '1'. The calculation of the adjusted energy is thus only an incremental operation per significant bit '1' coded, and one shift and two addition operations per refinement bit '1' coded. The energy value $E_{i,k}$ is quadrupled (shifted by two bits) whenever an entire bitplane has been encoded, which reduce the current bitplane M by 1.

Step 2. Calculating the intra-band masking threshold.

Combing (1), (5) and (6), and using bitplane to express the auditory JND threshold, we have:

$$TH\_INTRA_{i,k}(bitplane) = M + \log_4 \lfloor E_{i,k} \rfloor - C_k, \quad (7)$$

$$C_k = \log_4 \left(sizeof(s_{i,k}) \right) + R_{fac} \frac{\log_2 10}{20}, \quad (8)$$

where C_k is a constant of critical band k that can be pre-calculated. Calculation of (7) thus needs only one logarithmic and two addition operations per critical band.

Step 3. Calculating the combined auditory JND threshold.

The combined auditory JND threshold can be calculated through the iteration of equation (2)-(4), where large JND threshold of high energy critical band propagates to the critical bands in the frequency or time neighborhood.

Since the steps 2 and 3 only operate on a critical band basis, and are performed once every threshold update interval, the majority of the computation lies in the step 1. And even in the step 1, the added complexity is on average less than one arithmetic operation per coefficient, which is minor compared with the complexity of the entropy coder. It can thus be concluded that the implicit auditory masking operation can be performed very efficiently.

4. BITSTREAM ASSEMBLER AND PARSING

After entropy coding, a bitstream assembler module distributes the available coding bits among audio channels, sections and timeslots, and puts together the EAC compressed bitstream. A companion file is also generated to hold the structure information of the EAC bitstream. The companion file is not necessary for the decoding of the audio waveform, but is used to assist the reshape of the EAC bitstream. The syntax of the EAC bitstream and the companion file can be shown in Figure 11 and Figure 12, respectively.

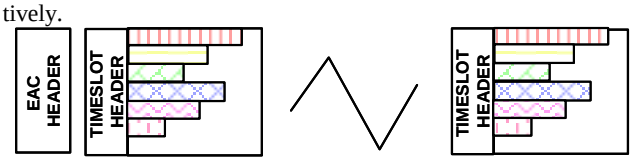

Figure 11 EAC Bitstream syntax.

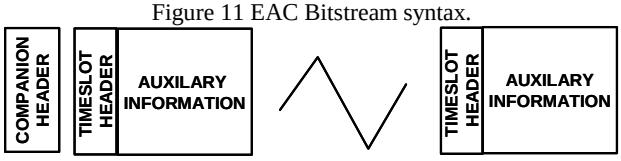

Figure 12 Companion file syntax.

The compressed EAC bitstream is lead by a global header, which identifies the EAC bitstream and stores the global codec information such as the identity of the transform module, entropy coding module, etc. The global header is then followed by the compressed bitstreams of individual timeslot, which consists of the embedded bitstream of each section of each audio channel of that timeslot. The timeslot is again led by a header, which records the length of the compressed bitstream in the timeslot, and the length of the compressed bitstream of the individual section.

Current implementation of the EAC bitstream assembler adopts a pretty simple bit allocation strategy. It assigns each timeslot with the same number of bits B, which can be calculated as:

$$B = R * T / S, \quad (9)$$

where R is the desired coding bits per second, T is the number of audio samples per timeslot, and S is the audio sampling rate. The available bits in the timeslot are distributed among sections so that each section is truncated at approximately the same gap between the instantaneous JND threshold and the progress indicator. Since we have established in Section 3.4 that the gap is related to the noise-mask-ratio (NMR), each section is thus encoded to the approximate same NMR.

The companion file records auxiliary information of the timeslot. In the case of EAC, it records for each section of time slot a sequence of length-gap pair, where the gap is the difference between the instantaneous JND threshold and the progress indicator, and the length is the size of the compressed bitstream corresponds to the incremental gap value. Such information is used only for the quickly scaling the bitstream, and is not used in the decoding operation. We thus store the length-gap pair information in a companion file that is separated from the compressed bitstream.

At the time of scalable parsing, an EAC parser can be invoked to convert the EAC master bitstream and the associated compan-

ion file into an application bitstream of a different coding bitrate, numbers of audio channels, and audio sampling rate. If the compressed audio is converted from stereo to mono, the compressed bitstream of the L-R channel is removed from the bitstream. If the compressed audio is switched into a lower sampling rate, the compressed bitstream of the sections corresponding to higher sampling rate is dropped. If the compressed audio is scaled to a lower bitrate, the EAC parser uses the length-gap pair of the companion file to calculate a new gap value; and then truncates the embedded bitstream of each section accordingly. Since the parsing is achieved by simply stitch segments of compressed bitstream together, it can be performed at a super fast speed.

5. EXPERIMENTAL RESULTS

To evaluate the performance of the embedded audio coder (EAC), we compare it against a number of existing audio coders. The benchmarks include two current audio coding standards - MP3[17] and MPEG-4[18]; two commercial coders - the Microsoft Windows Media Audio (WMA™) codec[20] and the Real™ audio codec[19]; and a scalable audio coder - WINSAC[22] . Please refer to the reference for the specific version of each codec used in the experiment[1,2]. The test audio waveform is the MPEG-4 sound quality assessment materials (SQAM) downloaded from [8]. The original audio is in stereo and sampled at 44.1kHz.

We test for both stereo and mono audio coding. The coding bitrates for stereo are 128, 96, 64, 48, 32 and 16 kbps (kilo bits per second). The coding bitrates for mono are 64, 48, 32, 24, 16 and 8 kbps. We use the noise-mask-ratios (NMR)[3] to provide an objective quality evaluation of the compressed audio. The NMR is calculated using a 4096 convoluted window and complex MLT transform. It measures the level (in dB) of audio coding noise above the auditory JND threshold. The lower the NMR, the less that the coding noise is audible; and thus the better the quality of the reconstructed sound.

Table 6. Average NMR of the comparison coders (Stereo)

Coder	128kbps	96kbps	64kbps	48kbps	32kbps	16kbps
EAC	-2.72	-0.96	1.17	2.64	4.75	6.72
WMA 8.0	-2.94	-0.90	1.85	3.21	4.48	N/A
Real 8.5	N/A	N/A	4.11	4.95	5.51	N/A
MP3	8.15	9.54	12.62	12.51	14.61	14.66

Table 7. Average NMR of the coders (Mono).

Coder	64kbps	48kbps	32kbps	24kbps	16kbps	8kbps
EAC	-3.79	-1.59	1.43	3.29	5.20	6.86
WMA 8.0	N/A	0.25	2.94	N/A	5.94	8.87
Real 8.5	3.10	3.89	4.70	5.81	N/A	N/A
MP3	8.04	9.17	12.75	12.83	15.25	14.72
MPEG-4	3.33	4.59	5.75	6.44	6.86	7.54
WINSAC	6.63	6.99	7.47	7.73	8.19	N/A

[1] The author acknowledges that there are multiple versions and parameter settings for each standard and commercial coder. The test is conducted using the one setting with the best compression performance.

[2] WINSAC does not support stereo audio coding. It should be scalable. However, in the experiment, we just directly encode to the target bitrate. At the time of the submission, the author is unable to get the MPEG-4 reference coder to work in the stereo and scalable BSAC mode. Therefore, only the result of the MPEG-4 TwinVQ coder (non-scalable) mode is provided, which should be better than that of the scalable BSAC mode.

We comment only on the mono audio coding result. Nevertheless, the stereo NMR result is available in Table 6. We have also put all the encoded bitstream on the web [5]. The readers are encouraged to download the clips and make subjective judgment of the quality of each audio coder by themselves.

All benchmark coders compress the audio waveform directly into the bitstream of the target bitrate. However, for the EAC, only one encoding operation is performed to convert the original audio waveform into a master compressed bitstream of the highest bitrate (128kbps for stereo and 64kbps for mono). The rest of the bitstream is parsed from the master bitstream through the EAC parser. Thus, we have already put the scalable functionality of the EAC coder on test.

From Table 7, it is observed that the EAC coder outperforms today's standard by a large margin. It outperforms MP3 by 10.2dB and MPEG4 by 3.9dB. EAC also outperforms WMA by 1.5dB, Real by 4.5dB, and WINSAC by 6.5dB. Therefore, in term of NMR, EAC performs superbly among all coders.

We leave the subjective judgment of the codec quality to the readers, as all clips are available through the web [5]. We do observe that the subjective quality of the EAC coder is far superior to MP3, MPEG-4 and WINSAC. Since the existing scalable audio coders [11][22][23][24] (including WINSAC) claim at most matches the quality of MP3, EAC is superior to all of them. The subjective quality of EAC coded clip is as good as that of the Real and WMA, the two commercial coders. We also observe that at low bitrate, all the benchmark coders falls back to lower sampling rate (narrow bandwidth), because there are not enough bits to transmit all audio coefficients and the JND threshold. However, since the EAC coder does not need to send the auditory JND threshold, it has bits available for all coefficients and does not need the trick. Therefore, the EAC compressed clip sounds more rich (with more high frequency components) compared to the benchmark coders at lower bitrate.

The compressed bitstream of EAC can be reshaped (scaled) to an application bitstream of lower bitrate, number of audio channels and audio sampling rate. On the web[5], we have provided samples that a 512kbps stereo master bitstream sampled at 44.1kHz is reshaped to the application bitstreams of low bitrate (128, 96, 64, 48, 32, 16 and 8kbps), mono channel (with coding bitrate 64, 48, 32, 24, 16, 8 and 4kbps), and lower sampling rate (at 11.025kHz and 22.05kHz). As mentioned in Section 4, EAC uses a companion file to record the bitstream structure information. For the test clips, the companion file is at 2.3kbps for the 64kbps mono coding, and 4.4 kbps for the 128kbps stereo coding. It is about 3.4-3.6% in size with regard to the compressed bitstream it represents, which is not a big overhead. Note that the companion file is not required in the decoding operation. Therefore, it does not need to be sent over the network, and can be deleted if further manipulation of the EAC bitstream is unnecessary. The EAC parsing operation can be performed in lightning speed. In fact, the parsing operation is too fast to be timed accurately on the test data set. In a more extended data set of 2616 seconds (more than 43 minutes) of compressed audio content, the EAC parser is able the reshape the entire dataset within 2 seconds (on a Pentium III 800Mhz computer). This is almost two magnitudes faster than any existing audio encoder / transcoder without the scalable coding technology.

Using the EAC, we have further developed a portable music player that can holds virtually unlimited amount of compressed music. A running screen shot of the player can be seen in Figure

13. During the operation, the user may choose to transfer compressed audio from the host (PC) to the portable player, which has a limited amount of memory. Whenever the memory of the player is full and additional compressed music is still to be transferred to the player, the EAC parser is used to scale the bitstream already in the player and the bitstream to be transferred in. The player can thus dynamically trade the length of music it can hold versus the music compression ratio (playback quality). A streaming application using the scalability of EAC is being developed by my colleagues Chou and Zabinsky[25]. Preliminary experiments also demonstrate superior performance compared with the streaming technologies today.

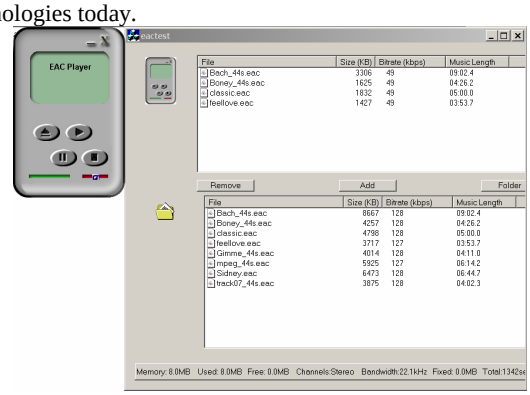

Figure 13 Screen shot of a portable music player with EAC coder.

6. CONCLUSIONS

The embedded audio coder (EAC) pushes the state-of-the-art of the scalable audio compression. The compression performance of the EAC exceeds the existing scalable audio coders, outperforms the audio compression standards MP3 and MPEG-4, and rivals the commercial audio coders- Real™ and Windows Media Audio™. It achieves the compression performance of the best available non-scalable audio coder, yet retains full bitstream scalability. The EAC compressed bitstream can be reshaped 1000x faster than real-time in terms of compression ratio (coding bitrate), number of audio channels and audio sampling rate. The key technology that enables efficient embedded audio coding in EAC is the implicit auditory masking. By deriving the auditory JND threshold from coded coefficients, and reordering the coding of the coefficients according to the threshold, EAC eliminates a substantial fixed overhead, and thus greatly improve the compression performance. As a highly efficient embedded audio coder, EAC has widespread applications in audio storage and streaming.

7. REFERENCES

[1] N. S. Jayant, J. D. Johnson, and R. Safranek, “Signal compression based on model of human perception”, Proc. of IEEE, Vol. 81, No. 10, pp.1385-1422, 1993.

[2] B. Moore, An introduction to the psychology of hearing, Academic Press, Feb. 1997.

[3] B. Beaton, et. al, “Objective perceptual measurement of audio quality”, in *Collected papers on digital audio bit-rate reduction* (N. Gilchrist and C. Crewin, eds.), pp.126-152, Audio Engineering Society, 1996.

[4] P. Noll, “MPEG digital audio coding”, IEEE Signal Processing Magazine, Sept. 2001, pp. 59-81.

[5] J. Li, “The embedded audio coder”, http://research.microsoft.com/users/jinl/2001/eac/eac.htm.

[6] B. Grill, "MPEG-4 General Audio Coder," AES 17th International Conf. on High-Quality Audio Coding, Firenze, Sep. 1999.

[7] JPEG 2000 Image Coding System, ISO/IEC IS 15444-1, Dec. 2000.

[8] “Sound quality assessment material recordings for subjective tests”,http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/.

[9] H. S. Malvar, “Fast adaptive encoder for bi-level images”, Proc. Of Data compression conferences, Snowbird, Utah, Mar. 2001, pp. 253-262.

[10] J. Li and S. Lei, "An embedded still image coder with rate-distortion optimization", IEEE Trans. On Image Processing, Vol. 8, No. 7, pp. 913-924, Jul. 1999.

[11] S. R. Quackenbush, “Coding of natural audio in MPEG-4”, *1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 6, 1998, pp. 3797-3800.

[12] J.M Shapiro, “Embedded image coding using zerotrees of wavelet coefficients”, *IEEE Trans. On Signal Processing*, Vol.41 No. 12, pp. 3445-3462, Dec. 1993.

[13] A. Said, and W. A. Pearlman, “A new, fast, and efficient image codec based on set partitioning in hierarchical trees”, I*EEE Trans. On Circuits and Systems for Video Technology*, Vol.6, No.3, pp. 243-250, June 1996.

[14] D. Taubman, “High performance scalable image compression with EBCOT”, *IEEE Trans. On Image Processing*, Vol. 9, No. 7, pp.1158-1170, July 2000.

[15] D. Taubman and A. Zakhor, “Multirate 3-D subband coding of video”, IEEE Transactions on Image Processing, Vol. 3, No. 5, pp. 572-588, Sept. 1994.

[16] H. S. Malvar, *Signal Processing with Lapped Transforms*, Boston, MA: Artech House, 1992.

[17] “Fraunhofer IIS MPEG layer-3 encoder”, http://www.iis.fhg.de/amm/techinf/layer3.

[18] “ISO/IEC 14496-5/FPDAM 1”, ISO/IEC JTC1/SC29/WG11/N3309, Noordwijkerhout, Holland, March 2000. (options: -m tf -c “-mp4ff -core_coder 10 -mode 0”)

[19] “Real Producer® Basic 8.51”, http://www.realnetworks.com/

[20] “Windows Media Technologies 8.0”, http://www.microsoft.com/windows/windowsmedia/

[21] B. Leslie, C. Dunn and M. Sandler, "Developments with a Zero Tree Audio Codec," Proc. AES 17th International Conf. ‘High Quality Audio Coding’, Florence, pp. 251-257 (1999 Sep.).

[22] C. Dunn, “Winsac 1.0”, http://www.scalatech.co.uk/download.htm

[23] A. Scheuble and Z. Xiong, “Scalable audio coding using the nonuniform modulated complex lapped transform”, *Proc. 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume. 5, pp. 3257-3260

[24] S. Ye, H. Ai, C. Kuo, “A progressive approach for perceptual audio coding”, *Proc. 2000 IEEE International Conference on Multimedia and Expo*, Volume. 2, pp. 815-818.

[25] P. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *submitted to* IEEE Transactions on Multimedia, February 2001.