MUSIC 422 - A Darker Phonetic Audio Coder

Prateek Murgai and Orchisama Das

Abstract

In this project we develop an audio coder that tries to improve the quality of the audio at 128kbps per channel by employing the redundancies and similarities that exist in the two channels along with block switching that improves the coding of transients. As block switching can be a bit hungry process, we employ Huffman coding for mantissa coding that helps us create a decent sized bit reservoir. The bits saved from stereo coding and Huffman coding can now be employed to feed block switching thus improving the quality of our audio at lower data rates. In the paper we present our overall system and the different components that make up our audio coding system and followed by some preliminary results.

1 Project Overview

Figure 1 and Figure 2 display the encoding and decoding steps of our audio coder. In the next section we will discuss all the different features that we have implemented in our coder.

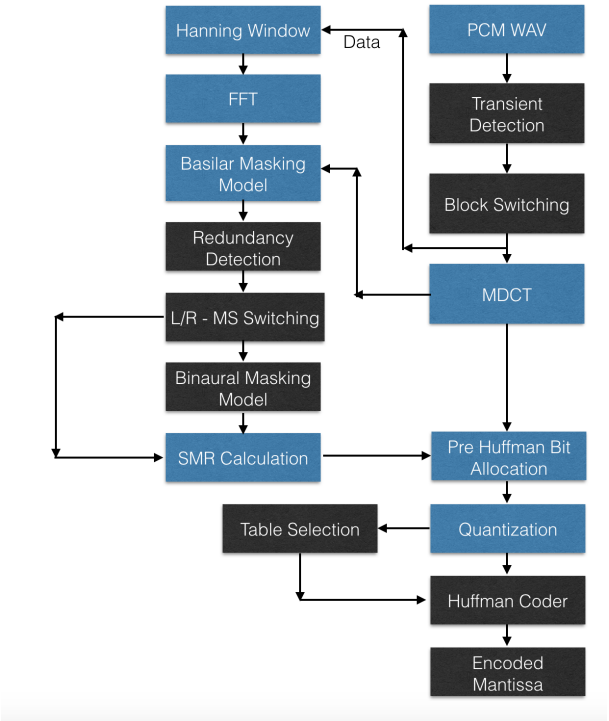


Figure 1: Audio Encoder

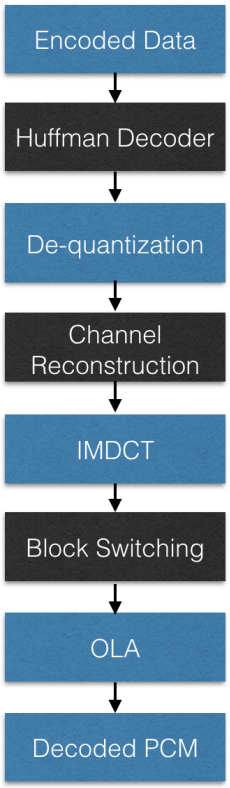


Figure 2: Audio Decoder

The features implemented are shown in black and the other features in blue are a part of the baseline coder.

2 Features

We have implemented the following features on top of our baseline coder:

- Transient Detection and Block Switching
- Huffman Coding and Decoding
- Band-wise MS Stereo Coding
- Bit Reservoir

2.1 Block Switching

Block switching is needed to properly encode onsets in percussive signals. A block size of 1024 is too large for good time resolution of sharp attacks, which have high frequency content changing rapidly. As a result, the transients get smeared and we hear false attacks known as pre-echoes. The way we implement block switching is a hybrid between the way it is done in Dolby AC2 and MPEG-2 AAC block-switching [1] (diagram in Figure 4) which uses 8 shorter windows in the transient block. The main difference from MPEG-2 AAC is that our transition windows are of length 576 instead of 1024.

We detect if each block of data contains a transient or not. If a transient is detected, we divide the large block into smaller sub-blocks of 128 samples and encode each individually. For proper overlap add, two transition windows of size 576 samples are used before and after the shorter blocks. A state machine paired with the buffer controls how the transient informs which window size is to be used next. The window state takes up 2 additional bits (a total of 4 states) that need to be encoded and decoded for each block of data. Based on the window size for the block, we also change how many MDCT lines are to be used in each Bark band. In case there are zero lines in any band (which is common for the shorter blocks), we set the SMR for that band to a minimum value of -30 dB and the scale factor and mantissa to 0.

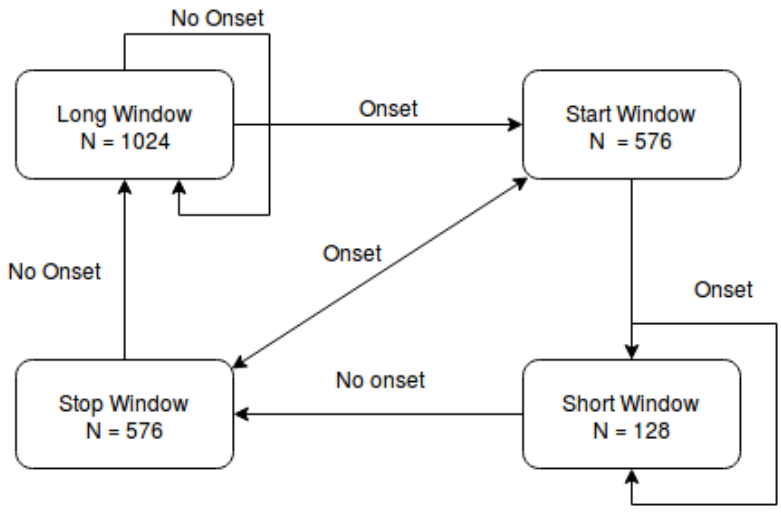


Figure 3: Window selection state machine

2.1.1 Transient Detection

For transient detection, we use the High Frequency Content (HFC) function proposed in [2]. Changes due to transients are noticed at high frequencies. We take the FFT of each block of the signal and form a weighted energy function given by :

$$\tilde{E} = \frac{2}{N} \sum_{k=0}^{\frac{N}{2}-1} W(k)|X(k)|^2 \tag{1}$$

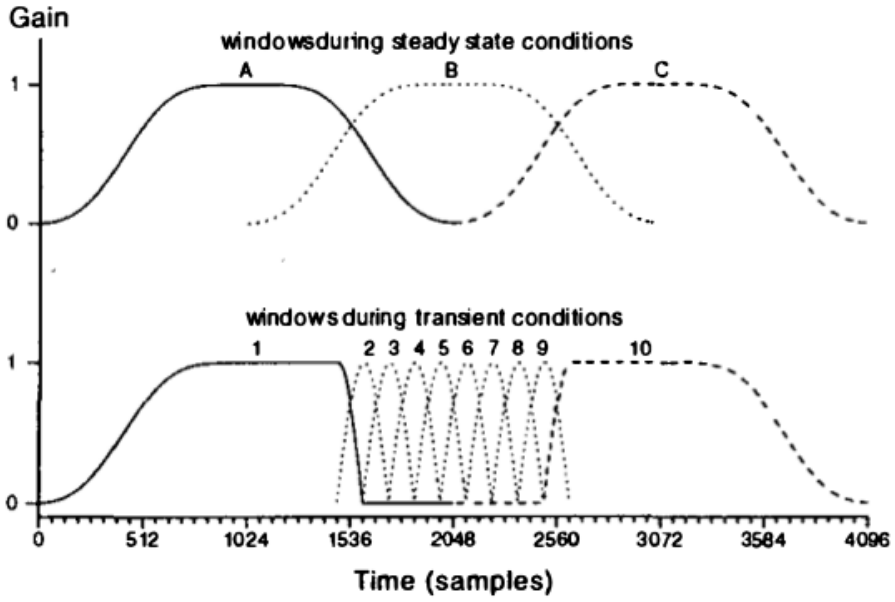


Figure 4: Block Switching in MPEG-2 AAC

where $X(k)$ are the DFT coefficients of the signal, N is the FFT size and $W(k)$ is a weighting function given by $|k|$. This gives a linear weighting to the FFT bin numbers (more weights to higher frequencies). We conclude that the block has a transient is if $\tilde{E} > thres$.

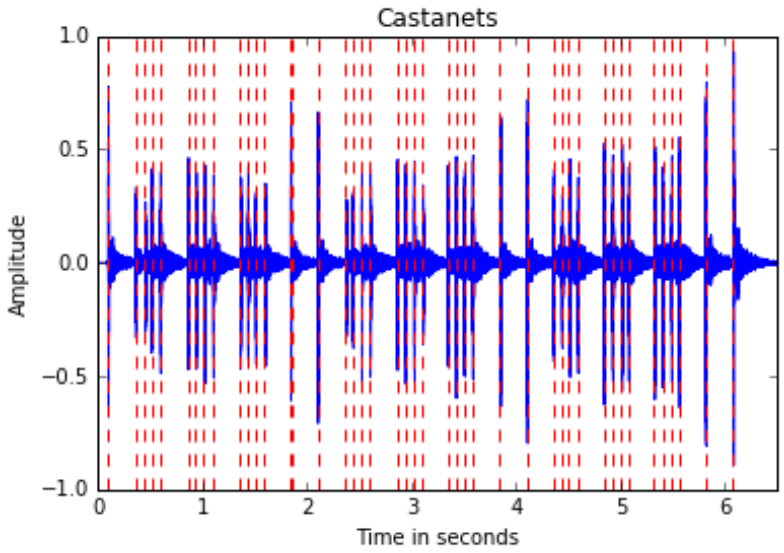


Figure 5: Transient detection

The results of the transient detection function applied to the *Castanets.wav* file is given in Figure 5. Occasionally, two consecutive blocks were classified as having transients. This is because we chose a conservative *thres* value of 50 to make sure we did not miss any transients. To prevent this, we increase *thres* value of current block to 80 if the last block was detected as a transient block. This fixes the problem of consecutive blocks being detected as false transients.

2.2 Stereo Coding

We were interested in implementing stereo coding as most of the audio available to us is present in stereo format and provides us with an opportunity to exploit the redundancies that occur between

two correlated data streams. The main advantage of stereo coding comes when there is high correlation between the two channels and we can employ the difference signal (Left Channel - Right Channel) which is automatically a very small signal thus requiring low bit allocation for representing this difference signal.

For our audio coder we have employed a band-by-band MS coding technique based on how correlated the two channels of audio are. If the correlation is greater than a threshold (estimated experimentally) then we say that for that specific band it is better to apply MS coding (saving a lot of bits!).

For correlation estimation we tried two different methods. The first one is based on finding the mean coherence in a given frequency band. The coherence between two signals is given by :

$$C(\omega) = (|X^*(\omega) * Y(\omega)|^2)/(|X(\omega)|^2 * |Y(\omega)|^2) \tag{2}$$

where $X(\omega)$ is the FFT of the signal in that given frequency band.

The correlation in a frequency band is given by $mean(C)$. For our application we set the threshold to **0.75**. This value was decided after multiple experiments on multiple audio files.

The second correlation method that we employed is the one that MPEG-2 employs [1]. We say that the signals are correlated if the following is true

$$\sum_{n=flower}^{fupper} \left|(L(\omega)^2 - R(\omega)^2)\right| < 0.8 * \sum_{n=flower}^{fupper} \left|(L(\omega)^2 + R(\omega)^2)\right| \tag{3}$$

where $L(\omega)$ and $R(\omega)$ are the FFTs of the left and right channel respectively. On our multiple experiments we found that both the methods gave similar results, thus we ended up using the second method for our system.

2.3 MS Coding

Let L_i and R_i be the left channel and right channel data. M_i and S_i are defined as follows:

$$M_i = (L_i + R_i)/2 \tag{4}$$

$$S_i = (L_i - R_i)/2 \tag{5}$$

2.4 MS Decoding

Let L_i and R_i be the left channel and right channel data. L_i and R_i are reconstructed as follows:

$$L_i = (M_i + S_i) \tag{6}$$

$$R_i = (M_i - S_i) \tag{7}$$

2.4.1 MS Psychoacoustics Model

Now, the only important thing left is the psychoacoustics model that one should employ. For the frequency bands for which we do not have enough correlation we employ L/R. Thus, the psychoacoustics model for those bands is the same as our baseline coder but for the bands in which we perform M/S coding we need to have a different model. We employ the psychoacoustics model and the binaural masking level differences (BMLD) employed in [3] to take care of that. We briefly describe the psychoacoustics model below:

The binaural masking level difference (BMLD) is an empirically estimated curve which describes ability of the ear to differentiate binaural material at low frequencies. The parameterization of this curve is given by the following equation.

$$BMLD(z) = 10^{(1.25(1-\cos(\pi * min(z,15.5)/15.5))-2.5)} \tag{8}$$

Where z is in Bark scale. Fig.3 shows BMLD function in bark scale.

In this case, where BTHR represents the base threshold (as would be computed for the LR channels) and $MLD_{M,S}$ indicates pointwise multiplication of MLD with $BTHR_{M,S}$ with its masker drops eliminated, the final thresholds for the M and S channels are derived [3]

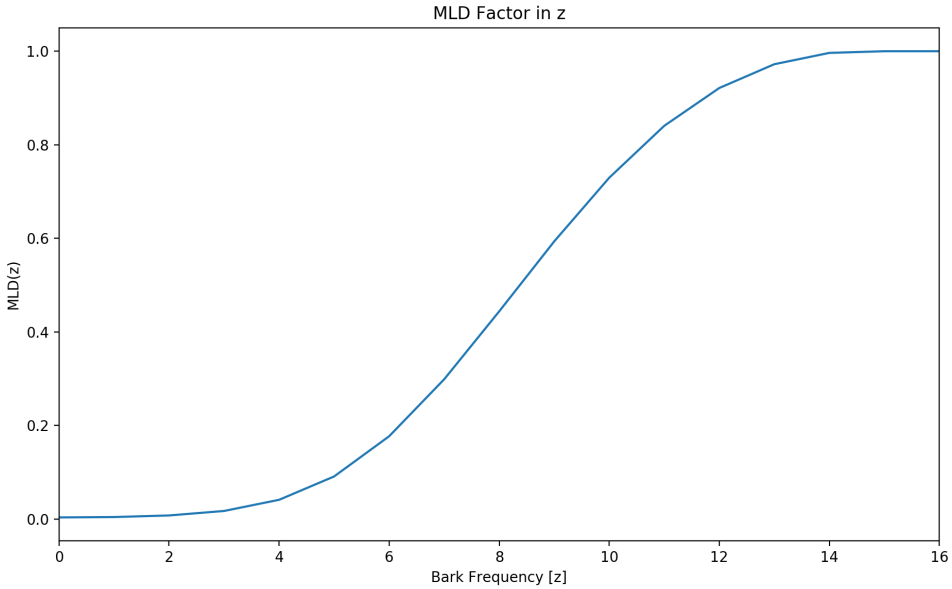


Figure 6: MLD in Bark Scale

$$THR_M = \max(BTHR_M, \min(BTHR_S, MLD_S)) \tag{9}$$

$$THR_M = \max(BTHR_M, \min(BTHR_S, MLD_M)) \tag{10}$$

If one imagines a very small S channel masking curve, it is reasonable to find that the M channel could mask signals on the S channel. We had a lot of trouble deciding on the proper way to implement these curves and eventually arrived at a critical understanding of what is required for M/S encoding to meet or exceed the quality of L/R encoding. In actuality, the coding gains that are derived from SMRs reaching an arbitrarily defined threshold in the bit allocation routine must offset the additional bits that are required to meet the lower masking curves that are a result of binaural unmasking.

2.5 Per-Block Huffman Coding

We employ Huffman to reduce the number of bits required to encode mantissa values, giving us a good amount of extra bits that we can use for other bit demanding features such as block switching. Huffman coding requires one to prepare a probability table of the occurrence of each sample (Huffman coding is a method of entropy coding). As this task of preparing Huffman tables can be a difficult task with audio due to high diversity of audio sample, we need to pre-construct huffman tables of mantissa probabilities using a number of audio samples.

There are usually two steps to such an per block audio Huffman coding technique, they are:

- Training Process
- Table Selection and Encoding/Decoding Process

The training process required one to input a bunch of audio signals, record mantissa's and then create a frequency table of each mantissa occurrence.

As we did not get time to train Huffman tables, we are using an open source collection of ten Huffman tables(different tables containing different genres of audio). [4]

Our system goes through each of the tables to see which table is the correct fit for the audio block that is being processed currently. Once the table selection has taken place the audio sample is encoded by looking at the mantissa probabilities.

2.5.1 What if the mantissa doesn't occur in the table ?

In case a given mantissa doesn't occur within the tables, we use a special code to tell the decoder to decode the specific mantissa sample using using straight-off block floating point and to bypass the Huffman decoding step.

2.6 Bit Reservoir

Savings from Huffman coding and stereo coding leads to a net positive amount of extra bits that we save that can be used later for more demanding data blocks. Every block we calculate the bits we save from stereo coding and stereo coding and add it to a bit reservoir and during the bit allocation process we snatch 10%-20% of the bits from the bit reservoir and add it to our bit budget

3 Other Small Features We Implemented

- KBD Window
- Noise Masking along with Tonal Masking
- Optimal Bit Allocation along with Water-filling algorithm

4 Takeaways

4.1 Block Switching

Pre-echoes in percussive sounds were audibly reduced when block switching was added. However, without bit-reservoir, the performance at 128 kbps actually degraded. This is because the short blocks had very few bits to allocate to each MDCT line. This problem was solved once Huffman coding and bit reservoir were implemented. However, consecutive false transients increased the size of the coded file. More accurate transient detection would be beneficial in the future.

4.2 Huffman Coding

The performance of Huffman coding is pretty decent and gives a significant amount of saving in each block that we can use for the future blocks. It would be to our advantage to have more than 10 tables to chose from which were built using very limited audio samples.

When the targetBitsperSample = 2.66667 (Compression Ratio of 6)

- **Average Compression Ratio without bit snatching from bit reservoir : 9.5**
- **Average Compression Ratio with 10% bit snatching from bit reservoir : 6.5**

4.3 Stereo Coding

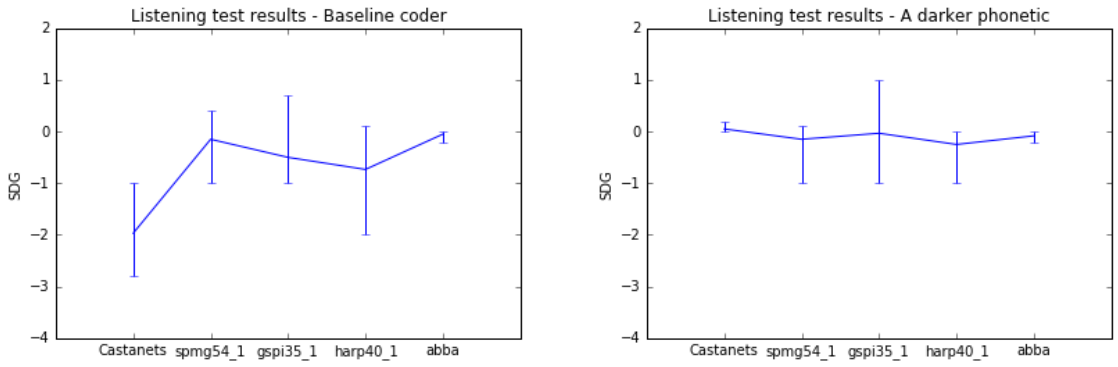
Stereo coding worked well for most of the frequency bands other than the first three lower frequency bins. The BMLD extracted from the empirical data is not accurate in the lower frequencies. Due to this we experience artifacts in the lower frequencies in the output audio signal. A quick fix for this was to simply not perform M/S in the those three bands and that fixed the problem for us but we do not have a fixed solution to this. Neither could we find any solution to this from literature.

4.4 Bit Reservoir

Our simple implementation of the bit reservoir helped in supplying the bits at the right time when they were needed. Currently we snatch only 10% of the bits in the bit reservoir , this percentage is open to tuning and experimentation. On our experimentation if we supply all the bits we save -5 bits (lose 5 bits). If we supply 20% bits we save more than 10K bits (giving a smaller encoded file size) with no reduction in perceived audio quality.

Sr No.	File	Baseline coder	A Darker Phonetic
1	Castanets	6.305	5.696
2	Harpsichord	6.295	5.996
3	Glockenspiel	6.308	16.140
4	German male speaker	6.317	6.433
5	Abba	6.839	6.752

Table 1: Compression Ratios



(a) Baseline coder

(b) A darker phonetic

Figure 7: Listening test results

5 Results

5.1 Compression Ratios

At a data rate of 128 kbps we expect the compression ration of our files to be **6:1**. The compression ratios of 5 audio files from the SQAM database are given in the Table 1. Since we don't employ any bit reservoir in our baseline coder, the compression ratios are higher, because extra bits are being wasted. Ideally, Huffman coding should give significant savings in bits, which should be used by the bit reservoir and the compression ratio should be maintained at 6 (which is what happens in most cases). The surprisingly high data rate of the glockenspiel is unexpected, but it is likely that the mantissas matched one of the 10 Huffman tables very well, and we could encode the mantissas with fewer bits. However, the high bit savings are not being utilized completely by the bit reservoir, and hence we lose a lot of saved bits.

5.2 Listening tests

The listening tests were conducted according to BS.1116 standards with 6 subjects (due to lack of time). The SDG plots for each of the 5 test audio files can be seen in Figure 7. The castanets are rated much better in our coder that includes block switching. The average SDG value of our coder is much closer to 0, indicating that it gives better performance than the baseline coder at 128 kbps overall. The listeners found it difficult to accurately rate the ABBA song, and thought both the coders were perceptually nearly equivalent to the original. Although we get a much higher compression ratio for the glockenspiel file with our coder, it doesn't do much worse than the baseline coder in the listening tests.

6 Future Work and Acknowledgement

Issues we would like to solve in the future :

- Better transient detection.
- M/S coding in lower frequency bands.

- Generate our own Huffman tables.
- More evolved bit reservoir.
- Aim for better compression ratios and perceptual quality at lower data rates.

We would like to thank Prof. Marina Bosi and Mark Hertensteiner for teaching us a tremendous amount about audio codecs in 10 weeks, and for their constant guidance and constructive feedback.

References

[1] Marina Bosi, Karlheinz Brandenburg, Schuyler Quackenbush, Louis Fielder, Kenzo Akagiri, Hendrik Fuchs, and Martin Dietz. Iso/iec mpeg-2 advanced audio coding. *Journal of the Audio engineering society*, 45(10):789–814, 1997.

[2] Paul Masri. *Computer modelling of sound for transformation and synthesis of musical signals.* PhD thesis, University of Bristol, 1996.

[3] JD Johnston and Anibal J Ferreira. Sum-difference stereo transform coding. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 2, pages 569–572. IEEE, 1992.

[4] Mamta Sharma. Compression using huffman coding. *IJCSNS International Journal of Computer Science and Network Security*, 10(5):133–141, 2010.