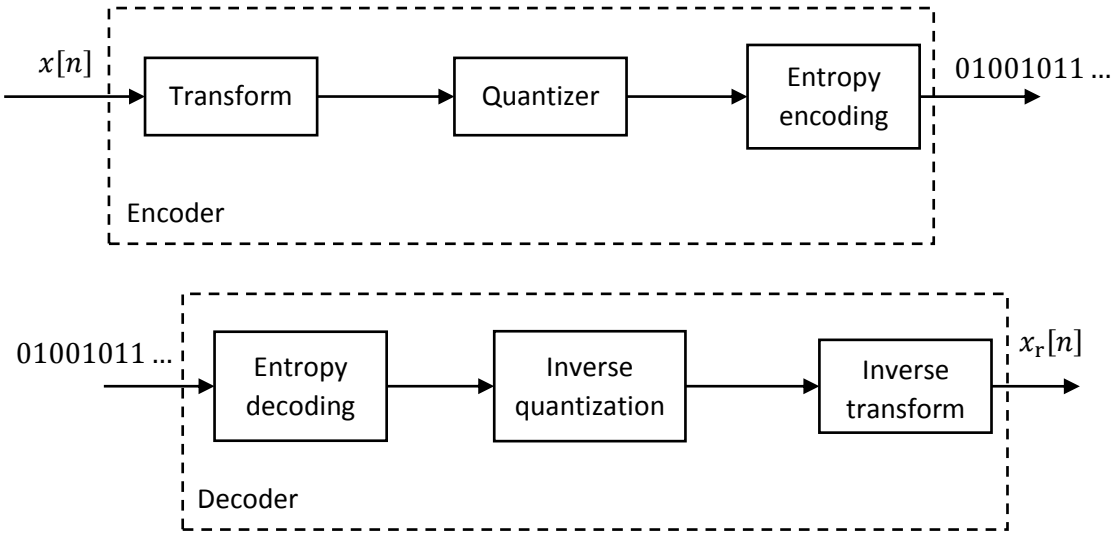
EE430 Term Project Part 2

1. Introduction

In this part of the project, you are going experiment with audio compression. A simplified flow of a complete encoder/decoder system is given below.



There may be two main objectives in an audio compression algorithm. The first is to make the total number of bits at the output of the encoder as small as possible. The second objective is to make the reconstructed signal $x_r[n]$ and the original signal $x[n]$ to be perceptually indistinguishable to the human ear. In this project, we are going to deal only with the encoding part.

You will also complete the missing parts of an almost complete MPEG[1]/Audio encoding software. The software will take captured audio (or an already existing uncompressed audio file) as input and return a compressed file with mp3 extension as an output. For the purpose of this project, we will benefit from some of the related material in an online course created by EPFL: https://www.coursera.org/learn/dsp [1].

MPEG/Audio compression makes use of both statistical redundancies and perceptual redundancies of an input signal. By using entropy encoding, a reduction in the number of bits can be achieved while being able to reconstruct the input signal perfectly. In other words, this is a lossless process. By quantizing/discarding some (frequency) components of the audio file, it can be compressed even more. In this process, some data is lost in the final representation. In this project you will experiment with the lossy part of the compression and you will not deal with entropy encoding. You will also have a simplified MPEG/Audio compression code at the end. A short tutorial about the MPEG/Audio compression is given [2].

For the following sections, try various types of audio inputs such as music (various types, various instruments), speech (male/female voice), and some other sounds. For each audio input, try to make

[1] Moving Picture Experts Group.

a fair comparison of the methods and the effects of different parameters. Prepare a short, descriptive and clearly written (in language and format) report.

For the sections below titled as

- Reducing the number of time samples
- Reducing the bits per sample on time domain
- Transform coding
    - Using DFT
    - Using DCT
    - Using MDCT,

the specifications are somehow open. That is, you may have to specify some more technical details. You are also responsible for selecting a set of parameter values to demonstrate the differences and similarities of the given methods. A good presentation would involve side by side comparisons of plots (for example spectrograms, time domain and frequency domain plots where applicable), listening two audio outputs one after the other, etc.

Also a note about the structure of your code: Although we do not require you to design a graphical user interface, we expect you to be able to change the parameter values easily. In order to do that, give a variable name to each parameter and only set them at the top of your code. (Going to the 842'th line of your code and modifying a number there is not the most efficient method during a presentation!) You have to try various values for the parameter values for technical discussions (in the written report and to be ready for the presentation) but you also have to do this for removing any possible errors in your code.

The remaining sections titled as

- MPEG/Audio encoder
    - Prototype filter design
    - Subband filtering
    - Spectrum estimation
    - Quantization

are more straightforward and should take the lesser of your time.

Overall, you are going to do the following main items:

1. Reduce the size of audio data by decimation of time samples

2. Reduce the size of audio data by quantizing the time samples

3. Transform the audio data block by block, set some transform domain coefficients to zero, and then reconstruct the time domain data. You are not doing any encoding, but if you did, the number of zero coefficients would be related to the overall reduction of the size of audio.

4. Fill in the minor parts of the almost complete mp3 encoder software that is distributed to you.

The details for these tasks are described below.

2. Reducing the number of time samples

In this section, you will reduce the size of the audio data by decimation.

You can reduce the total number of time samples of the input signal by reducing its sampling rate. Since any recorded signal in the computer memory is already sampled, you will just decimate the discrete-time signal. Try different downsampling factors, for example, 2, 3 and 6 (respectively, reducing the total size by 2, 3 and 6) Try also fractional downsampling factors. Do you need any preprocessing before downsampling? Listen to the resulting audio signal and comment on the quality. In order to present the results, you may plot the spectrogram of the input and output signals and write down the corresponding total number of time samples (and the compression percentage) on the figure titles.

3. Reducing the bits per sample on time domain

In this section, you will reduce the size of the audio data by quantizing time-domain samples.

In order to reduce the size of the encoded bit-stream, you may also reduce the number of bits per sample. Quantize each sample of the audio signal in order to reduce the total number of bits. For example, if the number of bits per sample in the original audio file is 16bit/sample, then try for example, 8, 6 and 4 bits per sample. Please note that you may need to adjust the partitions of the quantizer to get the most use of the given desired number of bits. For each case, plot the time signals, the error signals (the difference between the original and the output signal) and compute its average power. Listen to the resulting audio signal and comment on the quality. What are the resulting total number of bits needed? What is the resulting bit rate? What would happen if you tried to store the time samples with floating point precision?

4. Transform coding

In this section, you will first transform the data into another domain and then apply the quantization afterwards.

In the previous section you have applied quantization directly to the time domain samples. Now, you will first transform the signal to another domain and apply the quantization afterwards. You are going to work with discrete Fourier transform (DFT), discrete cosine transform (DCT) and the modified discrete cosine transform (MDCT). The quantizer in this section is somehow a special and unconventional one. You will have a threshold value as a parameter. Only the transform-domain coefficients whose absolute value is greater than that threshold will represent the given signal. The others will be discarded. For a given threshold value T, we may denote this quantizer function as

$$Q(x) \triangleq \begin{cases} x, & |x| > T \\ 0, & |x| \leq T \end{cases}.$$

By reconstructing the time-domain signal through inverse transformations, you will be able to listen to the compressed signal. You are not doing any encoding here. You are just going to flag those coefficients that are less than or equal to the given threshold. Set those flagged coefficients to zero and count the number of nonzero coefficients for reporting the compression amount etc. (Here, we are assuming that encoding of zero coefficients does not require any number of bits which may not be exactly true in practice)

For each transformation, play with the threshold value, listen to the resulting audio signal, and comment on the quality. You may also plot and observe the spectrograms of the input and output signals. You may compute error signals and determine the average power of the error signal.

Instead of supplying a threshold value as an input parameter, it may be more suitable to supply the desired number of nonzero coefficients (or the desired compression ratio). One way to do that is by

sorting the absolute values of the coefficients and then determining the corresponding threshold automatically for each input audio signal. Note that, for a given desired number of nonzero coefficients, you may need to have a different threshold value for each input signal and for each type of transformation.

4.1. Using DFT

In this subsection, you will apply quantization to frequency domain coefficients.

Partition the input signal into non-overlapping blocks of size N and compute the DFT for each block (You are computing STFT). Apply a threshold to the absolute values of the DFT coefficients. Reconstruct the compressed signal by applying inverse DFT to each block.

The signal flow diagram may look like the following:

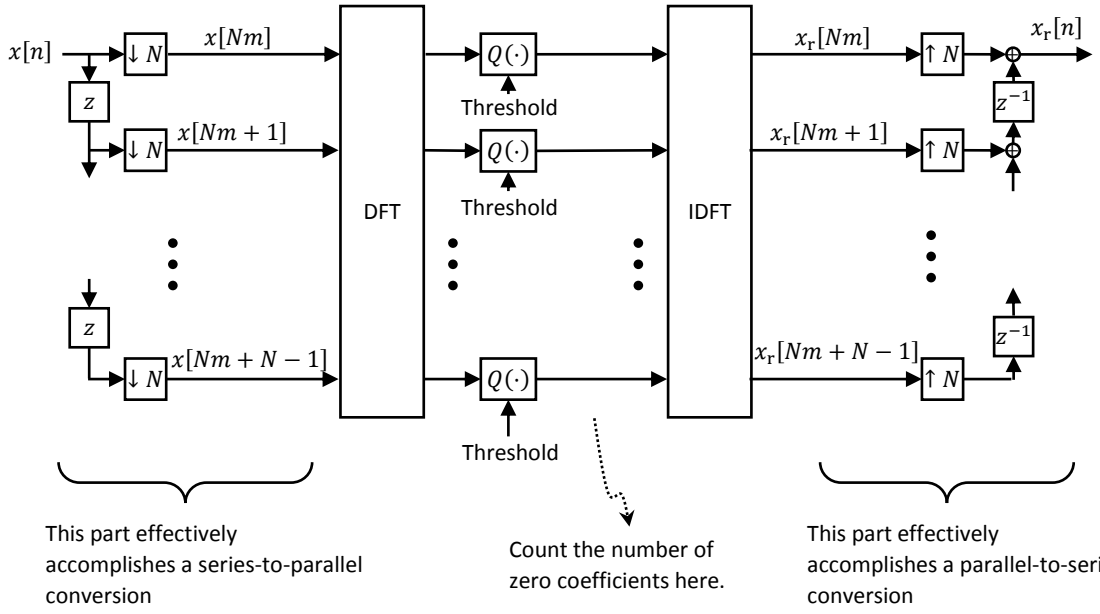


Figure 1: A toy compression method which uses DFT and an unconventional quantizer. Q(·) denotes the quantizer which depends on a threshold and sets the input values whose absolute values are less than or equal to that threshold to zero. The other values are not modified.

For the implementation of the series-to-parallel part shown in Figure 1, you may find MATLAB's `buffer` command useful. Also note that the `fft` and `ifft` commands in MATLAB are able to operate on the columns of a given 2-dimensional array input. Finally, the parallel-to-series part can be implemented by using the colon indexing operator (`:`) of MATLAB.

4.2. Using DCT

In this subsection, you will replace the DFT block with DCT.

DCT is a transformation that is similar to DFT except that real vectors transform into real vectors. Also, together with the periodic assumption, DCT has the underlying even symmetry assumption. It happens that a finite duration sequence can be extended to be periodic and even symmetric in 8 different ways [4]. This leads to 8 different definitions for DCT named as DCT-1, DCT-2, ….

In this section you will use DCT-2 which has a nice energy compaction property for most of the real life audio signals. For a given positive integer N, DCT-2 coefficients of a vector $\boldsymbol{x} = [x_0 \quad x_1 \quad \dots \quad x_{N-1}]^{\mathrm{T}} \in \mathbb{R}^N$ are defined as

$$X_k \triangleq \sqrt{\frac{2}{N}} \frac{1}{\sqrt{1+\delta[k]}} \sum_{n=0}^{N-1} c_k[n] x_n \ , \qquad k \in \{0, 1, \dots, N-1\} \qquad \text{(DCT2)}$$

where

$$c_k[n] \triangleq \cos\left(\frac{\pi k(n+1/2)}{N}\right) , \qquad n, k \in \{0, 1, \dots, N-1\} .$$

The inverse DCT-2 (IDCT-2) is given by

$$x_n = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \frac{X_k}{\sqrt{1+\delta[k]}} c_k[n] \ , \qquad n \in \{0, 1, \dots, N-1\} \qquad \text{(IDCT2)}$$

The forward and inverse transforms have efficient implementations. In MATLAB, you can use the `dct` and `idct` commands with desired "Type" option.

Partition the input signal again into non-overlapping blocks of size N and compute the DCT (specifically, DCT-II) for each block. Apply a threshold to the absolute values of the DCT coefficients. Reconstruct the compressed signal by applying inverse DCT to each block.

The signal flow diagram may look like the following:

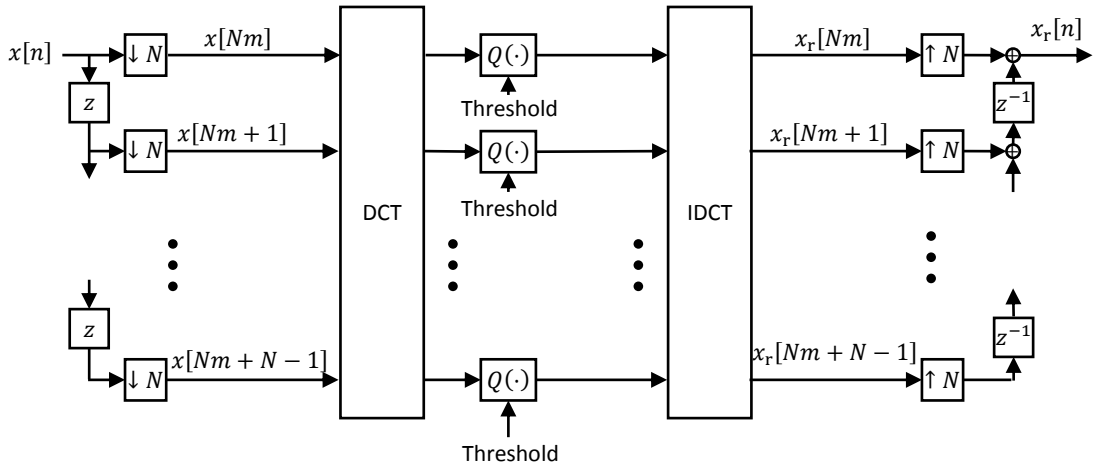


Figure 2: A toy compression method which uses DCT and an unconventional quantizer.

For a given number of desired compression ratio, how does the listening quality of DFT and DCT compare? Also compare the average power values of the signals.

4.3. Using MDCT (This is a bonus part)

In this subsection, you will overlap the time frames to reduce the block edge effects.

In the previous two cases, you possibly have discontinuities in the block edges of the reconstructed signal and it should be audible while listening. In order to reduce that effect, mp3 and some other recent audio compression algorithms uses overlapping blocks together with a window function to smooth out the discontinuities near the block edges. MDCT is widely used in audio coding [3]. For

some specific window types, it is possible to obtain the reconstructed time-domain signal by computing the inverse MDCT and combining the resulting blocks.

For a given positive integer N, the MDCT coefficients of a vector $\boldsymbol{x} = [x_0 \quad x_1 \quad \dots \quad x_{2N-1}]^{\mathrm{T}} \in \mathbb{R}^{2N}$ are defined as

$$X_k \triangleq \sqrt{\frac{2}{N}} \sum_{n=0}^{2N-1} m_k[n] x_n \ , \qquad k \in \{0, 1, \dots, N-1\} \qquad \text{(MDCT)}$$

where

$$m_k[n] \triangleq \cos\left(\frac{\pi}{N}\left(k + \frac{1}{2}\right)\left(n + \frac{1}{2} + \frac{N}{2}\right)\right) , \qquad k \in \{0, 1, \dots, N-1\}, n \in \{0, 1, \dots, 2N-1\}.$$

The inverse MDCT (IMDCT) is defined as

$$\alpha_n \triangleq \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X_k m_k[n] \, , \qquad n \in \{0, 1, \dots, 2N-1\} \, . \qquad \text{(IMDCT)}$$

We are considering the following signal flow diagram:

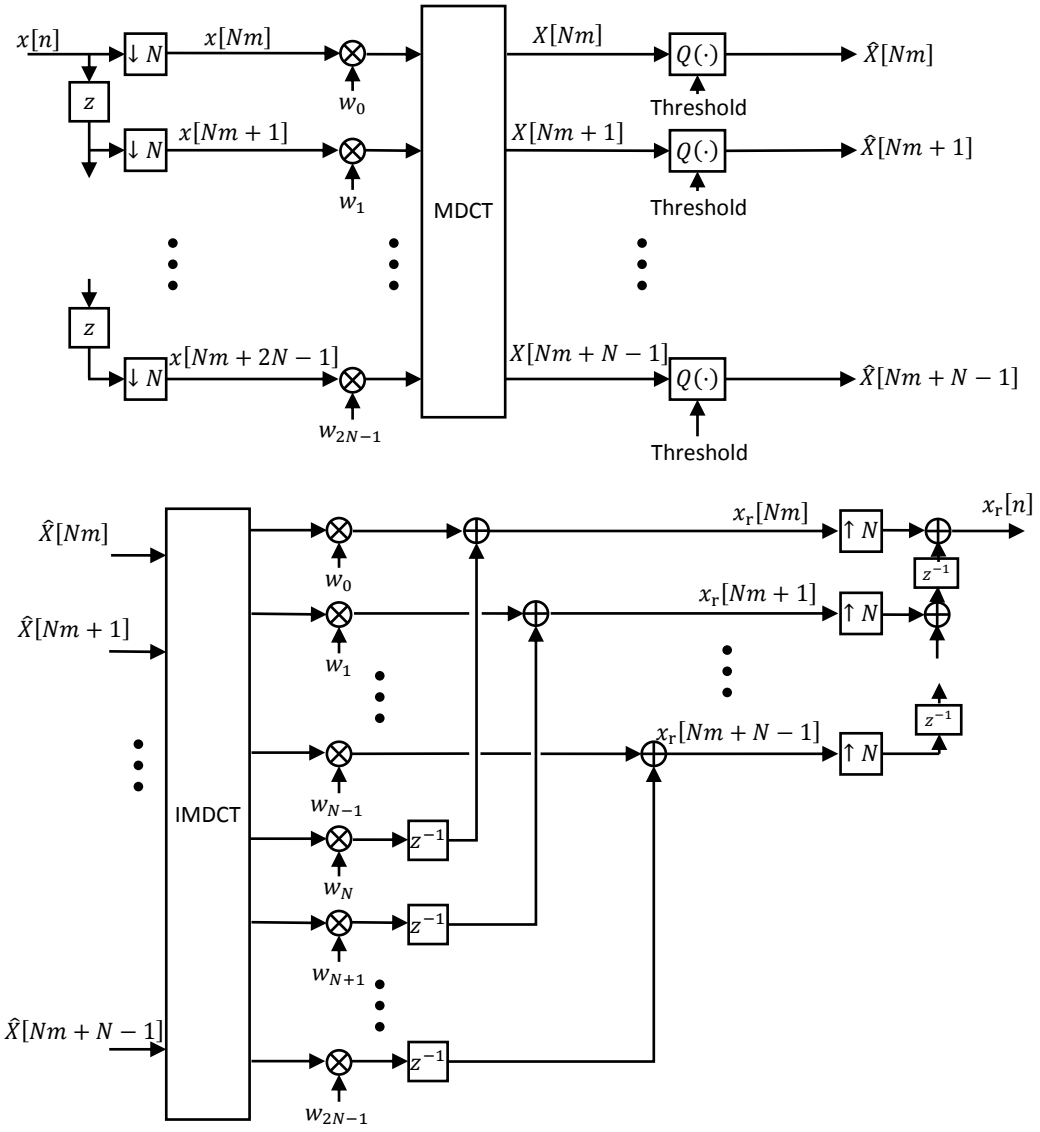


Figure 3: Signal flow diagram for MDCT operation. The window samples are such that $w_n = w_{2N-1-n}$ and $w_n^2 + w_{n+N}^2 = 1$.

It is shown in Appendix that, for a zero threshold value, the structure in Figure 3 allows a perfect reconstruction ($x_r[n] = x[n]$) if the samples of the window function satisfy

$$w_n = w_{2N-1-n}\,, \qquad n \in \{0, 1, \dots, 2N-1\}$$

and

$$w_n^2 + w_{n+N}^2 = 1, \qquad n \in \{0, 1, \dots, N-1\}\,.$$

Two examples of window functions that satisfy the above properties are given below.

$$w_n^{(1)} \triangleq 1/\sqrt{2}\,, \quad n \in \{0, 1, \dots, 2N-1\}\,,$$

$$w_n^{(2)} \triangleq \sin\left(\frac{\pi}{2N}\left(n + \frac{1}{2}\right)\right), \quad n \in \{0, 1, \dots, 2N-1\}\,.$$

Also, in the Appendix, it is shown that MDCT can be implemented by using readily available fast implementations for DCT-4.

Partition the input signal into overlapping blocks of size $2N$ with an overlap of 50%. Compute the MDCT (by using the fast implementation described in Appendix) for each block. Apply a threshold to the absolute values of the MDCT coefficients. Reconstruct the compressed signal by applying inverse MDCT to each block and combining the results as shown in Figure 3.

Report the results and compare with DFT and DCT results of the previous subsections.

5. MPEG/Audio encoder

The MPEG/Audio encoder algorithm quantizes the transform domain coefficients. Using a fixed budget, some number of bits are allocated to each transform coefficient. This allocation is, in general, different for each time-block and computed by using what is known as a psychoacoustic model of human hearing system. This model assumes that some frequency components of the signal are "masked" under strong tonal and non-tonal components. For the purposes of human perception, these masked components are discarded and they are not encoded in the output file. By this way, the total number of encoded bits is reduced.

In MPEG Layer 3, due to some compatibility issues with Layer 1 and Layer 2, MDCT is not applied to the input signal directly, but its is applied to the outputs of a bank of subband filters. The details about this filtering and other related items about the MPEG/Audio encoder is given in the Coursera lecture video

https://www.coursera.org/learn/dsp/lecture/n7mzX/mp3-encoder .

In the Coursera course, an almost complete encoder software is available to download from

https://www.coursera.org/learn/dsp/resources/eBDIq .

This software is written in the language of Python. However, for your convenience, we have adapted this code to MATLAB and you can perform all the steps in MATLAB environment.

Prototype filter design

As part of the MP3 encoder algorithm, a bank of 32 subband filters with real coefficients filter the input signal resulting in 32 subband signals.

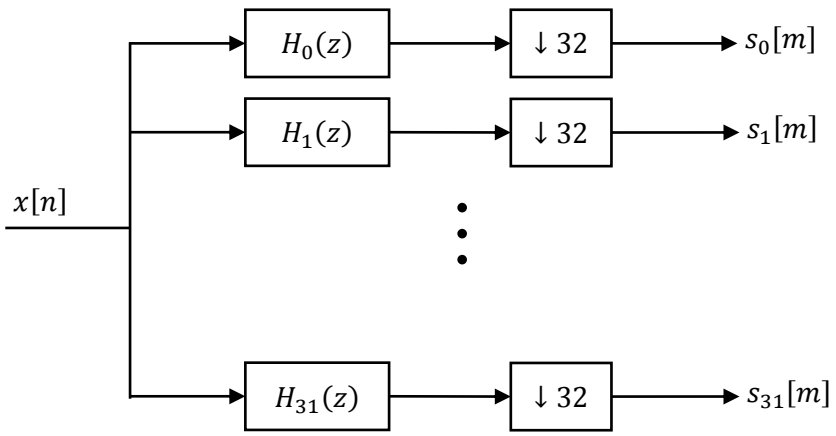
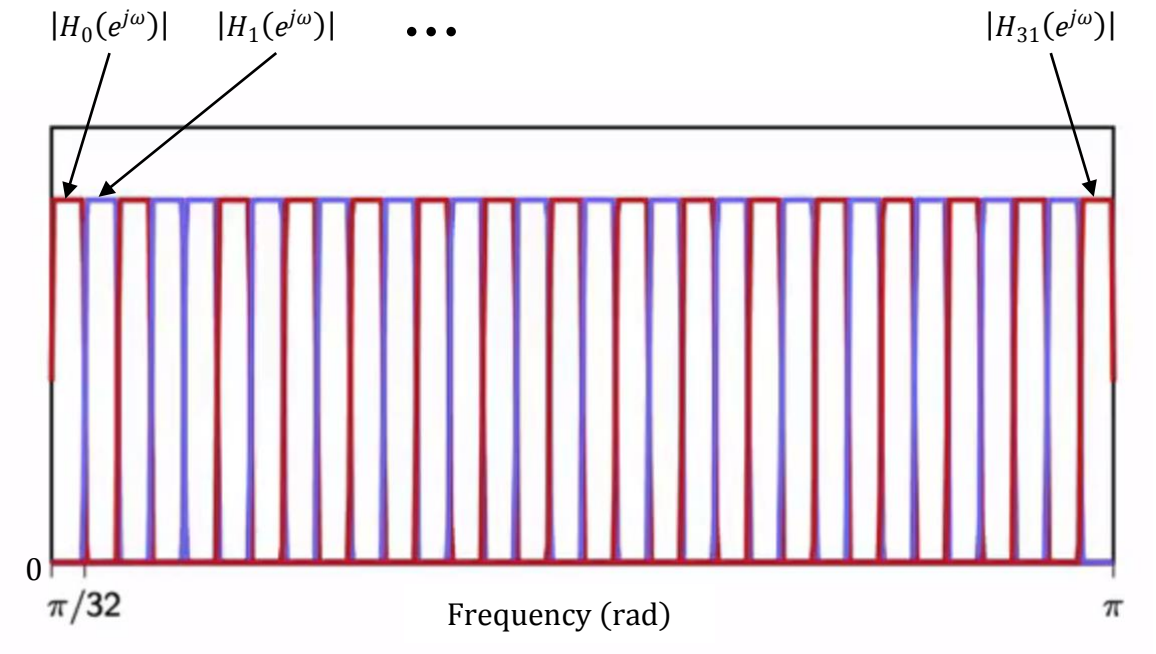


Figure 4: Subband filtering in MPEG/Audio encoding algorithm

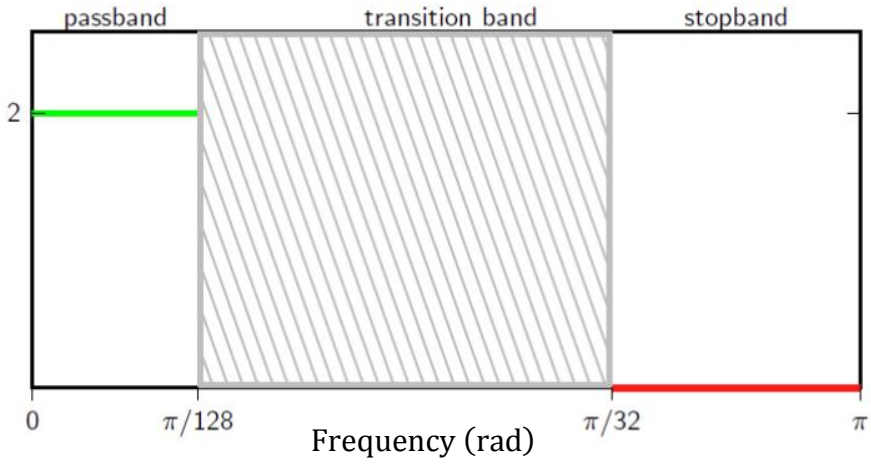
Here, $H_0(z), H_1(z), \ldots, H_{31}(z)$ denote the bandpass filters covering the entire spectrum as shown below.



In order to design these filters, first, a lowpass prototype filter $H(z)$ is designed. Then, H_0 through H_{31} are obtained by cosine modulation:

$$h_k[n] = h[n]\cos\left(\frac{\pi}{64}(2k+1)(n-16)\right), \qquad k \in \{0, 1, \dots, 31\}.$$

The prototype filter $h[n]$ is a 512-tap FIR filter. The desired response of $h[n]$ is given below:



[This desired response is taken from: https://www.coursera.org/learn/dsp/programming/ixExB/mp3-encoder-prototype-filter-design]

You can use `firpm` command of MATLAB to design $h[n]$. This command uses Parks-McClellan optimal FIR filter design algorithm to design a filter with a given desired response.

Things to do:

1) Design $h[n]$. Plot time domain and frequency domain characteristics (magnitude and phase responses).

2) Design $h_0[n]$ through $h_{31}[n]$. Plot their frequency domain characteristics on a single plot. Do they cover the entire frequency spectrum?

3) Create a MATLAB function with its own file named `prototype_filter.m`. This function will take zero input arguments and it will return the samples of $h[n]$ as a 512×1 array. The file `prototype_filter.m` should look like similar to the following:

```
function h = prototype_filter()
h = [...
    8.511229680152609944e-06
    3.585802005545701832e-06
                                        ⋮
    8.511229680152609944e-06];
```

The related programming assignment in the Coursera course is given below:

https://www.coursera.org/learn/dsp/programming/ixExB/mp3-encoder-prototype-filter-design .

5.1. Subband filtering

In Figure 4, the input signal is filtered and then immediately downsampled by 32. This means that 31 out of 32 samples are discarded. Therefore, a direct implementation of Figure 4 is wasteful in terms of computational resources. Fortunately, this can be implemented more efficiently.

For a given input $x[n]$, denote the subband filter outputs as

$$u_k[n] \triangleq \sum_{r\in\mathbb{Z}} h_k[r]x[n-r]\,, \qquad k \in \{0,1,\dots,N-1\}\,.$$

Downsampling the subband filter outputs by N yields

$$\begin{aligned}
s_k[m] &\triangleq u_k[Nm] \\
&= \sum_{r\in\mathbb{Z}} h[r]\cos\left(\frac{\pi}{2N}(2k+1)\left(r-\frac{N}{2}\right)\right)x[Nm-r] \\
&= \sum_{p\in\mathbb{Z}}\sum_{q=0}^{2N-1} h[2Np+q]\cos\left(\frac{\pi}{2N}(2k+1)\left(2Np+q-\frac{N}{2}\right)\right)x[Nm-2Np-q] \\
&= \sum_{q=0}^{2N-1}\cos\left(\frac{\pi}{2N}(2k+1)\left(q-\frac{N}{2}\right)\right)\sum_{p\in\mathbb{Z}}(-1)^p h[2Np+q]x[Nm-2Np-q] \\
&= \sum_{q=0}^{2N-1}\cos\left(\frac{\pi}{2N}(2k+1)\left(q-\frac{N}{2}\right)\right)c_q[Nm]
\end{aligned}$$

where

$$c_q[n] \triangleq \sum_{p\in\mathbb{Z}}(-1)^p d[n,2Np+q]$$

with

$$d[n,i] \triangleq h[i]x[n-i]\,.$$

In the third equality above, the summation index r is replaced by $r = 2Np+q$, $p \in \mathbb{Z}$ and $q \in \{0,1,\dots,2N-1\}$. In the fourth equation, the trigonometric relation

$$\cos\left(\frac{\pi}{2N}(2k+1)\left(2Np+q-\frac{N}{2}\right)\right)$$
$$= \cos\left(\frac{\pi}{2N}(2k+1)\left(q-\frac{N}{2}\right)\right)\cos(\pi(2k+1)p)$$
$$- \sin\left(\frac{\pi}{2N}(2k+1)\left(q-\frac{N}{2}\right)\right)\underbrace{\sin(\pi(2k+1)p)}_{=0}$$

is used. Since $h[n]=0$ for $n \notin \{0,1,\dots,511\}$,

$$c_q[Nm] = \sum_{p=0}^{7}(-1)^p d[32m, 64p+q]\,.$$

The final expression for $s_k[m]$ says that the following implementation is possible:

At a given time index m,

1) Compute $d[32m,i] = h[i]x[n-i]$ for $i \in \{0,1,\dots,511\}$,

2) Compute $c_q[Nm] = \sum_{p=0}^{7}(-1)^p d[32m, 64p+q]$ for $q \in \{0,1,\dots,63\}$,

3) Compute $s_k[m] = \sum_{q=0}^{63}\cos\left(\frac{\pi}{64}(2k+1)(q-16)\right)c_q[Nm]$ for $k \in \{0,1,\dots,31\}$.

The efficiency of the implementation can further be improved by not doing the summation in (3) directly but by using a DCT transformation.

In the MPEG/Audio standard, a 512-point circular buffer is used. At each time index m, 32 samples in the buffer are replaced with new data. You do not need to consider updating of this buffer.

Things to do:
Create a MATLAB function with its own file `subband_filtering.m`. (This function will be called by the main script at each time index m.) This function will take two input arguments and it will return 32 subband samples as a 32×1 array.
Inputs:
x: 512-by-1 array containing the elements of the circular buffer, already updated with 32 new samples and time-reversed.
h: 512-by-1 array containing the samples of the impulse response of the prototype FIR filter.
Output:
s: 32-by-1 array containing the subband samples.
The efficient implementation involves the following steps. You have to take care of the indices since MATLAB indices start from 1 not from 0.
1. The buffer is already updated and time-reversed, so the newest samples are first.
2. Compute $d_i = h_i x_i, \;\; i = 0,1,\dots,511$
3. Compute $c_q = \sum_{p=0}^{7}(-1)^p d_{64p+q}\;,\;\; q = 0,1,\dots,63$
4. Compute the subband outputs as $s_k = \sum_{q=0}^{63}\cos\left(\frac{\pi}{64}(2k+1)(q-16)\right)c_q\,,\;\; k = 0,1,\dots,31\,.$

[This procedure is taken from https://www.coursera.org/learn/dsp/programming/PvXWi/mp3-encoder-subband-filtering]

5.2. Spectrum estimation

In an MPEG/Audio encoder algorithm, the number of bits to be allocated to each subband is determined by the masking levels of each strong tonal and non-tonal components. The masking level is determined by a psychoacoustic model which operates on the magnitude spectrum of the input

signal. Here, we do not go into the details of this model. You will only need to complete the spectrum estimation part.

Things to do:
Create a MATLAB function with its own file `scaled_fft_db.m`. This function will take one input argument and it will return the spectrum estimate as a 257×1 array.
Inputs:
x: 512-by-1 array containing the elements of audio buffer
Output:
X: 257-by-1 array containing the spectral magnitude values in dBs, normalized to 96dB.
The implementation should involve the following steps:
1. Window the input signal by a Hanning window, i.e, compute $y_n = x_n w_n$ where w_n is a Hann window defined as $w_n = \frac{\alpha}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$, where N is the length of the window (512 in our case) and α is a constant such that $\sum_{n=0}^{511} w_n^2 = 511$.
2. Compute the DFT of the windowed input.
3. Normalize the DFT output by the size of the input (i.e, divide by 512).
4. Since the input data is real-valued, keep only the first 257 coefficients and discard the rest.
5. Convert the magnitude to dB. If the original magnitude is zero, set the value in dBs to -100dB.
6. Rescale the output so that the maximum value is 96dB. Keep in mind that rescaling in a log scale is a simple addition.
[This procedure is taken from https://www.coursera.org/learn/dsp/programming/qv3Me/mp3-encoder-spectrum-estimation]

5.3. Quantization

The quantizer function used in the MPEG/Audio encoder is given below:

$$q = \left\lfloor \left(Q_a \frac{s}{S_\mathrm{f}} + Q_b \right) 2^{R-1} \right\rfloor$$

where s is the sample to be quantized, S_f is a scaling factor, R is the number of bits for the sample, Q_a, Q_b are quantization parameters and $\lfloor \cdot \rfloor$ denotes the floor function.

The main script will handle the determination of the required number of bits and other parameters for each subband. Here, you will just apply the quantization of a sample.

Things to do:
Create a MATLAB function with its own file named `quantization.m`. This function will take five input arguments and it will return the quantized sample
Inputs:
s: the sample to quantize
S_f: the scaling factor
R: the number of bits for the sample
Q_a, Q_b: the quantization parameters
Output:
q: uniformly quantized output
This function must implement $q = \left\lfloor \left(Q_a \frac{s}{S_\mathrm{f}} + Q_b \right) 2^{R-1} \right\rfloor$.
[The procedure is taken from https://www.coursera.org/learn/dsp/programming/bzmbL/mp3-encoder-quantization]

6. Appendix: MDCT

MDCT can be implemented efficiently using a DCT-4 implementation. The basis vectors of DCT-4 are slightly different than the ones in DCT-2.

DCT-4 coefficients of a vector $\boldsymbol{y} = [y[0] \quad y[1] \quad \dots \quad y[N-1]]^{\mathrm{T}} \in \mathbb{R}^N$ are defined as

$$Y[k] \triangleq \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} d_k[n] y[n] \ , \qquad k \in \{0, 1, \dots, N-1\} \qquad \text{(DCT4)}$$

where

$$d_k[n] \triangleq \cos\left(\frac{\pi(k+1/2)(n+1/2)}{N}\right) , \qquad n, k \in \{0, 1, \dots, N-1\} .$$

The inverse DCT-4 (IDCT-4) has the same form with the forward transform:

$$y[n] = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} Y[k] d_k[n] \ , \qquad n \in \{0, 1, \dots, N-1\} . \qquad \text{(IDCT4)}$$

Note that, for $0 \le n < N$,

$$d_k[n] = m_k[n - N/2] = -m_k[n + 3N/2] = -m_k[3N/2 - 1 - n] .$$

This relation allows MDCT to be computed in terms of DCT-IV as

$$\begin{aligned}
X_k &= \sqrt{\frac{2}{N}} \sum_{n=0}^{2N-1} m_k[n] x_n \\
&= \sqrt{\frac{2}{N}} \sum_{n=0}^{N/2-1} m_k[n] x_n + \sqrt{\frac{2}{N}} \sum_{n=N/2}^{N-1} m_k[n] x_n + \sqrt{\frac{2}{N}} \sum_{n=N}^{3N/2-1} m_k[n] x_n + \sqrt{\frac{2}{N}} \sum_{n=3N/2}^{2N-1} m_k[n] x_n \\
&= \sqrt{\frac{2}{N}} \sum_{n=N/2}^{N-1} d_k[n] x_{n-N/2} - \sqrt{\frac{2}{N}} \sum_{n=N/2}^{N-1} d_k[n] x_{3N/2-1-n} - \sqrt{\frac{2}{N}} \sum_{n=0}^{N/2-1} d_k[n] x_{3N/2-1-n} \\
&\qquad - \sqrt{\frac{2}{N}} \sum_{n=0}^{N/2-1} d_k[n] x_{n+3N/2} \\
&= \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} d_k[n] y_n
\end{aligned}$$

where

$$y_n \triangleq \begin{cases} -x_{n+3N/2} - x_{3N/2-1-n}, & 0 \le n < N/2 \\ x_{n-N/2} - x_{3N/2-1-n}, & N/2 \le n < N . \end{cases}$$

Similarly, IMDCT can be computed as

$$\alpha_n = \begin{cases} \beta_{n+N/2}, & 0 \le n < N/2 , \\ -\beta_{3N/2-1-n}, & N/2 \le n < 3N/2 , \\ -\beta_{n-3N/2}, & 3N/2 \le n < 2N \end{cases}$$

where

$$\beta_n \triangleq \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X_k d_k[n] , \qquad n \in \{0, 1, \dots, N-1\} .$$

Note that although the name of IMDCT involves the word "inverse", it is not an actual inversion of the transform. In other words, $\alpha_n \neq x_n$ in general. However, for a certain class of window functions and overlapping input sequence blocks by 50%, the input sequence can be reconstructed from MDCT coefficients. In order to see that, assume that in Figure 3, the threshold is equal to zero. Then, at a given time index m, and for $k \in \{0, 1, \dots, N-1\}$,

$$X[Nm+k] = \sqrt{\frac{2}{N}} \sum_{n=0}^{2N-1} m_k[n] w_n x[Nm+n]$$
$$= \sqrt{\frac{2}{N}} \sum_{n=0}^{2N-1} d_k[n] y[Nm+n]$$

where

$$y[Nm+n] \triangleq \begin{cases} -w_{n+3N/2} x[Nm+n+3N/2] - w_{3N/2-1-n} x[Nm+3N/2-1-n], & 0 \le n < N/2 \\ w_{n-N/2} x[Nm+n-N/2] - w_{3N/2-1-n} x[Nm+3N/2-1-n], & N/2 \le n < N \end{cases}.$$

At the output of the IMDCT block, for $n \in \{0, 1, \dots, 2N-1\}$,

$$\alpha_n[m] = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X[Nm+k] m_k[n] = \begin{cases} y[Nm+n+N/2], & 0 \le n < N/2, \\ -y[Nm+3N/2-1-n], & N/2 \le n < 3N/2, \\ -y[Nm+n-3N/2], & 3N/2 \le n < 2N. \end{cases}$$

For $n \in \{0, 1, \dots, N-1\}$,

$$x_{\mathrm{r}}[Nm+n] = w_n \alpha_n[m] + w_{n+N} \alpha_{n+N}[m-1]$$
$$= \begin{cases} w_n y[Nm+n+N/2] - w_{n+N} y[N(m-1)+N/2-1-n], & 0 \le n < N/2, \\ -w_n y[Nm+3N/2-1-n] - w_{n+N} y[N(m-1)+n-N/2], & N/2 \le n < N \end{cases}$$
$$= \begin{cases} (w_n^2 + w_{n+N}^2) x[Nm+n] + (w_{n+N} w_{2N-1-n} - w_n w_{N-1-n}) x[Nm+N-1-n], & 0 \le n < N/2, \\ (w_n^2 + w_{n+N}^2) x[Nm+n] + (w_{n+N} w_{2N-1-n} - w_n w_{N-1-n}) x[Nm+N-1-n], & N/2 \le n < N. \end{cases}$$

Now, assume that the samples of the window satisfy

$$w_n = w_{2N-1-n}, \qquad n \in \{0, 1, \dots, 2N-1\}$$

and

$$w_n^2 + w_{n+N}^2 = 1, \qquad n \in \{0, 1, \dots, N-1\}.$$

Then,

$$x_{\mathrm{r}}[n] = x[n], \quad n \in \mathbb{Z}.$$

References

[1] Coursera. (2017). Digital Signal Processing | Coursera. [online] Available at: https://www.coursera.org/learn/dsp [Accessed 13 Dec. 2017].

[2] D. Pan, "A tutorial on MPEG/audio compression," IEEE MultiMedia, vol. 2, no. 2, pp. 60–74, Summer 1995.

[3] Y. You, Audio coding: theory and applications. New York ; London: Springer, 2010.

[4] A. V. Oppenheim and R. W. Schafer, Discrete-time signal processing, 3. ed., Internat. ed. Upper Saddle River, NJ: Pearson, 2010.