Solutions: Homework Set # 9

Problem 1

(a) In order to apply the Noble identities for downsampling, we need to first rewrite the given filterbank as shown in Figure 1. Here, the downsamplers after the first two filters $h_0[n]$ have been replaced by two "cloned" downsamplers *after* the branch.

Now, repeated application of the Noble identities results in the equivalent system shown on Figure 2.

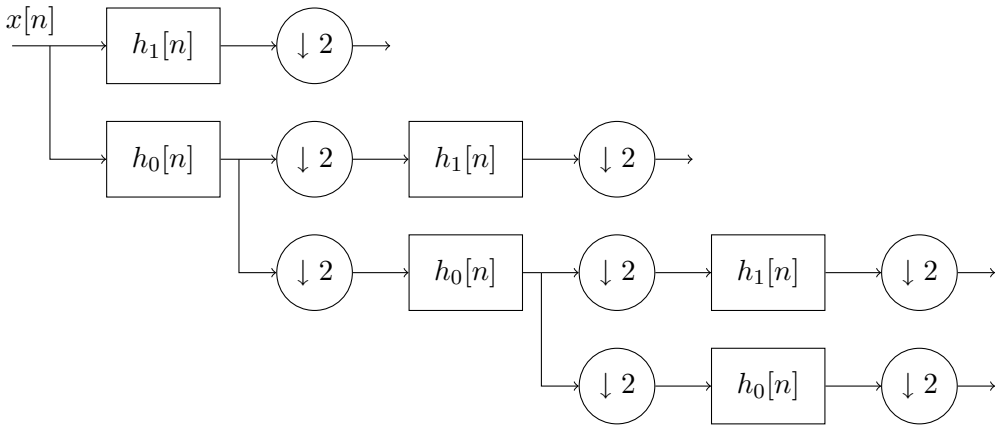


Figure 1: This is the same as the original analysis filter bank, except that the downsamplers before the split branches have been replaced by two downsamplers after the splits.

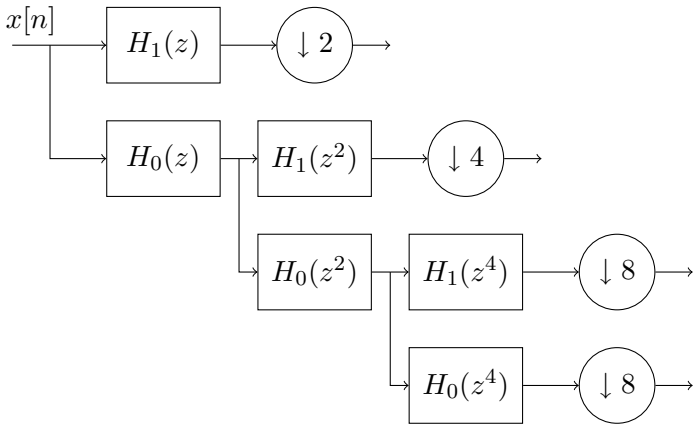


Figure 2: Analysis filter bank after applying the Noble identities.

Comparing Figure 2 with the desired single-level filter bank (Figure 2 in the assignment), we obtain the following relationships (expressed in the Z-domain):

$$\begin{aligned}
H^{(1)}(z) &= H_1(z) \\
H^{(2)}(z) &= H_0(z)H_1(z^2) \\
H^{(3)}(z) &= H_0(z)H_0(z^2)H_1(z^4) \\
H^{(4)}(z) &= H_0(z)H_0(z^2)H_0(z^4).
\end{aligned}$$

In the time domain, $H_i(z^L)$ corresponds to $h_i[n]$ upsampled by a factor L. Computing the respective convolutions, we find the following expressions for $h^{(i)}[n]$, $i = 1, \ldots, 4$.

$$h^{(1)}[n] = \frac{1}{\sqrt{2}} \begin{cases} 1 & \text{if } n = 0 \\ -1 & \text{if } n = -1 \\ 0 & \text{otherwise,} \end{cases}$$

$$h^{(2)}[n] = \frac{1}{2} \begin{cases} 1 & \text{if } n \in \{-1, 0\} \\ -1 & \text{if } n \in \{-3, -2\} \\ 0 & \text{otherwise,} \end{cases}$$

$$h^{(3)}[n] = \frac{1}{2\sqrt{2}} \begin{cases} 1 & \text{if } n \in \{-3, \ldots, 0\} \\ -1 & \text{if } n \in \{-7, \ldots, -4\} \\ 0 & \text{otherwise,} \end{cases}$$

$$h^{(4)}[n] = \frac{1}{2\sqrt{2}} \begin{cases} 1 & \text{if } n \in \{-7, \ldots, 0\} \\ 0 & \text{otherwise.} \end{cases}$$

(b) The plots of the filters $h^{(i)}[n]$, $i = 1, \ldots, 4$are shown on Figure 3. Roughly speaking, $h^{(1)}[n]$ covers the upper half of the spectrum. $h^{(2)}[n]$ then covers the "upper half of the lower half" of the spectrum, and so on. $h^{(4)}[n]$ covers the lowest $1/8^{\text{th}}$ of the spectrum. Observe that the more of the spectrum covers, the shorter is the support of its impulse response, and vice versa. In other words, Precision in frequency and precision in time are traded off against each other.

(c) In 8 time units, the first branch of the system obtained in (a) produces 4 output values, the second branch produces 2 output values, and the last two branches each produce one output.

Using delay elements, we can split the first branch, which produces 4 outputs in 8 time units, into 4 branches, each producing one output in 8 time units, as illustrated on Figure 4. Note that the delays are all *negative*, so they constitute *anticausal* elements. The resulting equivalent filters are

$$\begin{aligned}
\tilde{h}^{(1)}[n] &= h^{(1)}[n] * \delta[n] = h^{(1)}[n] \\
\tilde{h}^{(2)}[n] &= h^{(1)}[n] * \delta[n+2] = h^{(1)}[n+2] \\
\tilde{h}^{(3)}[n] &= h^{(1)}[n] * \delta[n+4] = h^{(1)}[n+4] \\
\tilde{h}^{(4)}[n] &= h^{(1)}[n] * \delta[n+6] = h^{(1)}[n+6].
\end{aligned}$$

It is easy to verify that the outputs of these 4 new branches correspond to the four outputs produced by the original branch in 8 time units.

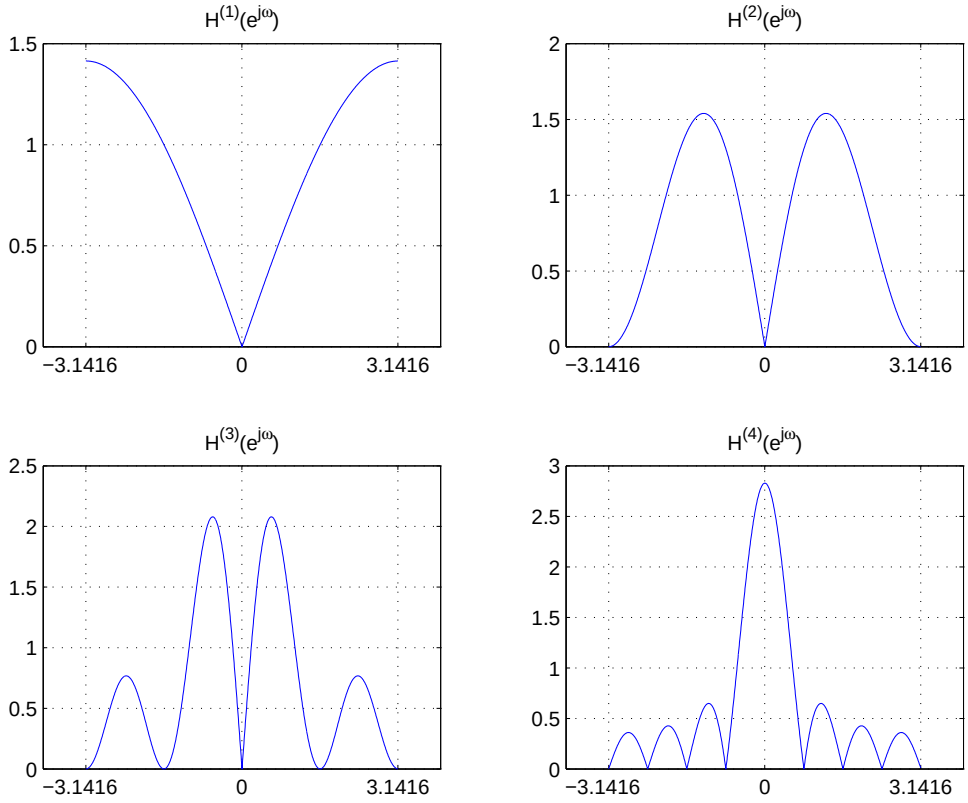


Figure 3: 1024-point FFT plots of the filters $h^{(i)}[n]$, $i = 1, \ldots, 4$, from Problem 1(a). Note how the filters divide the spectrum.

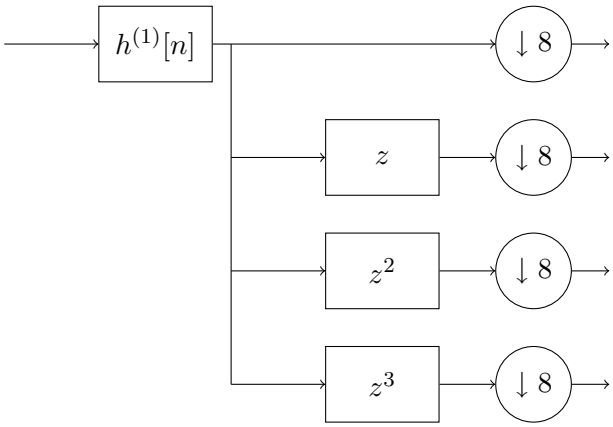


Figure 4: For Problem 1(c), The first branch is split up using delays into four branches, each producing one output every 8 time instants. Properly speaking, the delay elements are not delays but rather "look-ahead" elements, since the delays are negative.

Similarly, we can split up the second branch as

$$\tilde{h}^{(5)}[n] = h^{(2)}[n]$$
$$\tilde{h}^{(6)}[n] = h^{(2)}[n+4].$$

The last two branches remain unaltered:

$$\tilde{h}^{(7)}[n] = h^{(3)}[n]$$
$$\tilde{h}^{(8)}[n] = h^{(4)}[n].$$

(d) It is easy to express $y[n]$ in terms of $x[n]$. Let $z^{(i)}[n] = x[n] * \tilde{h}[n]$. Then $y[i] = D_8(z^{(i)}[n])\,|_{n=0} = z^{(i)}[0]$, *i. e.,*

$$y[i] = z^{(i)}[0] = \sum_m x[m]h^{(i)}[0-m] = \sum_{m=0}^{7} x[m]h^{(i)}[-m].$$

Therefore, we have

$$\begin{aligned}
y[0] &= \frac{1}{\sqrt{2}}[x[0] - x[1]],\\
y[1] &= \frac{1}{\sqrt{2}}[x[2] - x[3]],\\
y[2] &= \frac{1}{\sqrt{2}}[x[4] - x[5]],\\
y[3] &= \frac{1}{\sqrt{2}}[x[6] - x[7]],\\
y[4] &= \frac{1}{2}[x[0] + x[1] - x[2] - x[3]],\\
y[5] &= \frac{1}{2}[x[4] + x[5] - x[6] - x[7]],\\
y[6] &= \frac{1}{2\sqrt{2}}[x[0] + x[1] + x[2] + x[3] - x[4] - x[5] - x[6] - x[7]],\\
y[7] &= \frac{1}{2\sqrt{2}}[x[0] + x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7]].
\end{aligned}$$

Writing the above equations is matrix form, we obtain

$$\mathbf{Q} = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\\
\frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2}\\
\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}}\\
\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}}
\end{bmatrix}.$$

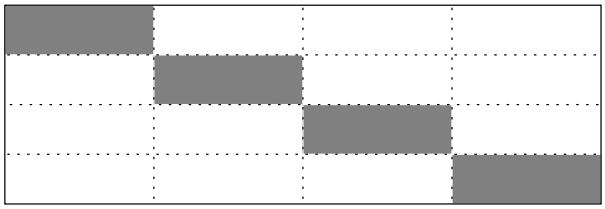
Remark. We have changed the indices of the input and output vectors from $[1, \ldots, 8]$ to $[0, 1, \ldots, 7]$ for simplicity.

(e) One can easily check that the inner product of each two distinct rows of $\mathbf{Q}$ is zero, and the norm of each row is 1. More formally,

$$\mathbf{Q}\mathbf{Q}^* = \mathbf{I}_8.$$

Therefore this filter-bank operates as an orthonormal projection. This shows that we can use exactly the same filter-bank to reverse the operation.

(f) This is not difficult to observe the diagonal structure of the first four rows of $\mathbf{Q}$. This



structure suggests to write this sub-matrix as a Kronecker product as

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} = \mathbf{I}_4 \otimes q_1,$$

where $q_1 = \frac{1}{\sqrt{2}}[1 \ -1]$. Similarly, the next two rows are structured as

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \end{bmatrix} = \mathbf{I}_2 \otimes \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \end{bmatrix} = \mathbf{I}_2 \otimes q_1 \otimes q_0.$$

The 7^{th} row can be also rewritten in Kronecker product form as

$$\begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} & \frac{-1}{2\sqrt{2}} \end{bmatrix} = \mathbf{I}_1 \otimes q_1 \otimes q_0 \otimes q_0.$$

Finally, the last row can be written as

$$\begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \end{bmatrix} = \mathbf{I}_1 \otimes q_0 \otimes q_0 \otimes q_0.$$

[redacted]

(g) Using the structure found in part (f), we can show that for general J, the $2^j \times 2^J$ submatrix induced by rows $2^J - 2^{j+1} + 1, \ldots, 2^J - 2^j$ can be written as

$$\mathbf{I}_{2^j} \otimes q_1 \otimes \underbrace{q_0 \otimes \cdots \otimes q_0}_{J-j-1},$$

and the last row is

$$\mathbf{I}_1 \otimes \underbrace{q_0 \otimes \cdots \otimes q_0}_{J}.$$

Using the MATLABfiles, this matrix can be constructed for general J as follows.

```
>> function H=WT(J)
>> % In this m-file we produce the transfer matrix of the tree-structutred
>> % wave-let transform of depth J.
>> % Consider the structure of the matrix, and note that each incremental
>> % 2^j rows of the matrix have a special form.
>>
>> q1=[1,-1]/sqrt(2);              ;
>> H=zeros(2^J);      %
>> S=0;   % counter to keep track of the number of rows have been produced yet
>>
>> for j=J-1:-1:0  %
>>     A=ones(1,2^(J-1-j))/sqrt(2^(J-1-j));
>>     B=kron(q1,A); % produces the fundamental block of each sub-matrix
>>     H(S+1:S+2^j,:)=kron(eye(2^j),B);
>>     S=S+2^j; % to jump to the right row
>> end
>> H(2^J,:)=ones(1,2^J)/sqrt(2^J); % assignement of the last row of H
```

Problem 2

(a) Each leaf of the filterbank tree corresponds to a frequency subband, so there are $J+1 = 7$ different subbands. The entries of $\tilde{\mathbf{c}}$ that belong to each frequency subband are given in the following table:

entries 1-32	highest frequency band
entries 33-48	second-highest frequency band
entries 49-56	third-highest frequency band
entries 57-60	fourth-highest frequency band
entries 61-62	third-lowest
entry 63	second-lowest frequency band
entry 64	lowest frequency band

The above values are obtained as follows: if we number the frequency bands by $1, 2, \ldots, J+1$, starting with the high-frequency ones, then frequency subband 1 contains the first 2^{J-1}

entries, subband 2 contains the next 2^{J-2} entries and so on. In general, subband j contains the 2^{J-j} entries following the entries of the previous subband. The exception is the last subband which only contains one entry.

(b) In Matlab, we type:

```
>> M = WT(10);
>> [conga,fs]=wavread('conga.wav');
>> c = conga(300001:301024);
>> plot_wt(M*c);
```

The result of the plot is given in Figure 5. We can see that there are regions similar to the ones given in Figure 6. These time-frequency plots show the meaning of each coefficient. Each region of the plot shows the magnitude of a coefficient of the filterbank transform. Also, each row of the plot corresponds to one frequency subband. Hence, we see that every coefficient in the top row contains the information about only two values of k in $c[k]$, but it contains information about roughly the upper half of the frequencies. ("Roughly" because we use the Haar filters. If we used a perfect low- and highpass filterbank, it would be exactly the upper half of the frequencies.) On the second row, every coefficient contains information about 4 samples of c, but it covers only about half as many frequencies. As we go through the rows of the plot (from top to bottom), the number of frequencies that each coefficient covers decreases while the number of samples of c it covers increases. Finally, on the last row, there is only one coefficient, which tells us how much energy is contained in the very low frequencies, but summarized over the whole time-sequence.

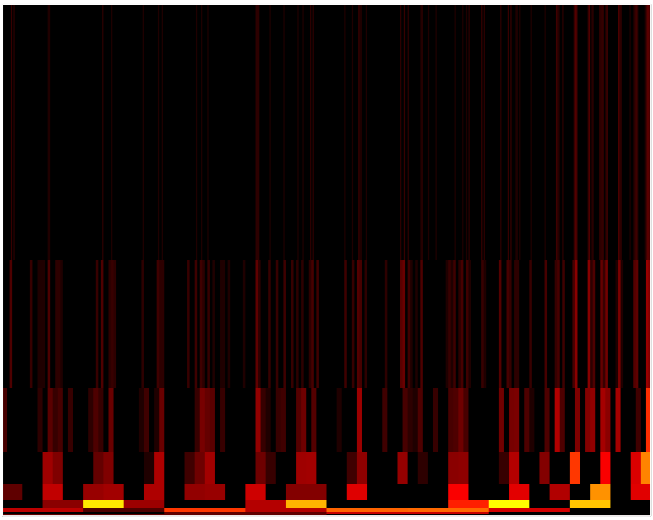


Figure 5: Output of the function "plot_wt".

(c) If we apply M to every block of length 2^J, we obtain again a collection of blocks of length 2^J. No information has been lost (perfect reconstruction), and therefore, if we stored the outcome on the harddisk, exactly the same space would be required. We have not done

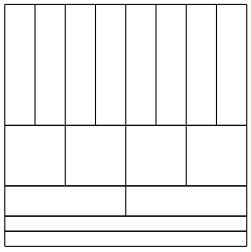


Figure 6: Hybrid transform with coarse frequency resolution in the upper half of the frequency range and finer frequency resolution for the lower half of the frequency range. Consequently, the time resolution is higher for higher frequency ranges and lower for lower frequency ranges.

any compression yet. One might ask what the filterbank is good for if it does not do any compression. The answer is that it splits the time-sequence c into several sets of coefficients (the different subbands) and we know what frequency range of the reconstruction will be affected by changing which coefficient.

(d) This is where the actual compression starts. It should now be clear that (for $J = 10$), we should use the fewest bits for the first subband (containing $2^{J-1} = 512$ entries), and the most bits for the last two subbands (containing only $2^{J-J)} = 2^0 = 1$ entry each).

(e) The two Matlab functions "encode_block" and "decode_block" are given below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function encodes the block c into the file given by the e   %
% file-pointer fid. It uses the function store to store the binary %
% representations of the subblocks.                                %
% You are required to complete this function.                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:                                                       %
% c: the block to be encoded and stored                            %
% J: the number of levels of the filterbank                        %
% M: the projection matrix corresponding to the filterbank         %
% qbits: a vector that specifies the number of bits to be used for %
%  the different subbands (the last 6 subbands are treated as one) %
% fid: the filepointer used to store the data                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function encode_block(c,J,M,qbits,fid)

% apply the projection matrix to the block:
ctrans = M*c; %delete

% find normalization factor for this block:
nf = ceil(max(abs(ctrans)));

% normalize the block:
ctrans_norm = ctrans / nf;

% the entries of ctrans_norm are all in [-1,1]

% store the normalization factor:
fwrite(fid,nf,'uint8');

% apply all quantizers (except the last)
% and store the data directly to the file:
stop = 0;
for j = 1:J-6

% define the quantizer:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(j) qbits(j)-1]);

% set the boundaries of the subblock corresponding to subband j:
start = stop+1; %delete
stop = stop+2^(J-j); %delete

% apply the quantizer and store the binary representation:
store(num2bin(q,ctrans_norm(start:stop)),fid);

end
```

```
% do the same for the last subband:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(J-5) qbits(J-5)-1]);
start = stop+1; %delete
stop = length(ctrans_norm); %delete
store(num2bin(q,ctrans_norm(start:stop)),fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function decodes the the next block from the file given by  %
% the file-pointer fid. It uses the function retrieve to retrieve  %
% the binary representations of the subblocks one by one.          %
% You are required to complete this function.                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:                                                       %
% J: the number of levels of the filterbank                        %
% M: the projection matrix corresponding to the filterbank         %
% qbits: a vector that specifies the number of bits to be used for %
%  the different subbands (the last 6 subbands are treated as one) %
% fid: the filepointer used to store the data                      %
% Return Values:
% b: the decoded block to be encoded and stored                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function b = decode_block(J,M,qbits,fid)
%function [b,bit] = decode_block(J,M,qbits,fid,key) %used for Pb3

% read the normalization factor from the file:
count = 0;
[nf,count] = fread(fid,1,'uint8');
if count ~= 1
b = 0;
return;
end

% allocate space for the sequence:
btrans_norm = zeros(2^J,1);

% read the first J-5 quantized subblocks, convert them
% to numerical values and add them to btrans_norm:
stop = 0;
for j=1:J-6

% define the quantizer:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(j) qbits(j)-1]);

% retrieve the quantized, binary version of bhe subblock:
[bsubquant,error] = retrieve(fid);
if error == 1
b = 0;
return;
end

% determine where to place this subblock in btrans_norm:
start = stop+1; %delete
stop = stop+2^(J-j); %delete
```

```
% convert the binary subblock to numerical values:
btrans_norm(start:stop) = bin2num(q,bsubquant);

end

% do the same for the last subblock:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(J-5) qbits(J-5)-1]);
[bsubquant,error] = retrieve(fid);
if error == 1
b = 0;
return;
end
start = stop+1; % delete
stop = length(btrans_norm); %delete
btrans_norm(start:stop) = bin2num(q,bsubquant);

% undo the normalization:
btrans = btrans_norm*nf; %delete
% note: because of the quantization, some values
% might now be larger than 1.
% This problem is taken care of in the function
% decode_file.m.

% undo the wavelet-projection:
b = M' * btrans; %delete
```

(f) If written correctly, the encoder should work a few minutes, and then write a file "conga.ema". This compressed file is roughly 3 to 4 times smaller than the original file "conga.wav".

(g) If written correctly, the decoder should work a few minutes, also, and then write a file "reconstruction.wav". If we listen to the reconstructed audio file, we hear a clear difference. The overall sound-quality has decreased. Have you tried playing with the program parameters ("J" and "qbits") to improve the quality?

Problem 3

(a) Decode the file by typing

```
>>decode_file('conga_tagged.ema','reconstruction_tagged.wav');
```

and then listen to the wave-file "reconstruction_tagged.wav". To hide the watermark, we have used bits in the EMA file which are not very important (the least significant bit of a 7-bit number). Hence, the difference is not audible.

(b) The new version of the file "decode_block" is given below. Apart from the function definition, the only line that has been added is

```
bit=bsubquant(key,7);
```

Oterwise, no modifications have been made.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function checks the the next block from the file given by   %
% the file-pointer fid. It uses the function retrieve to retrieve  %
% the binary representations of the subblocks one by one.          %
% You are required to complete this function.                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:                                                       %
% J: the number of levels of the filterbank                        %
% M: the projection matrix corresponding to the filterbank         %
% qbits: a vector that specifies the number of bits to be used for %
%  the different subbands (the last 6 subbands are treated as one) %
% fid: the filepointer used to store the data                      %
% Return Values:
% b: the decoded block to be encoded and stored                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function b = decode_block(J,M,qbits,fid)
function [b,bit] = decode_block(J,M,qbits,fid,key) %used for Pb3

% read the normalization factor from the file:
count = 0;
[nf,count] = fread(fid,1,'uint8');
if count ~= 1
b = 0;
return;
end

% allocate space for the sequence:
btrans_norm = zeros(2^J,1);

% read the first J-5 quantized subblocks, convert them
% to numerical values and add them to btrans_norm:
stop = 0;
for j=1:J-6

% define the quantizer:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(j) qbits(j)-1]);

% retrieve the quantized, binary version of bhe subblock:
[bsubquant,error] = retrieve(fid);
if error == 1
b = 0;
return;
end

% determine where to place this subblock in btrans_norm:
start = stop+1; %delete
stop = stop+2^(J-j); %delete
```

```
% convert the binary subblock to numerical values:
btrans_norm(start:stop) = bin2num(q,bsubquant);

end

% do the same for the last subblock:
q = quantizer( 'mode', 'float','roundmode','round',...
   'overflowmode', 'saturate', 'format', [qbits(J-5) qbits(J-5)-1]);
[bsubquant,error] = retrieve(fid);
if error == 1
b = 0;
return;
end
start = stop+1; % delete
stop = length(btrans_norm); %delete
btrans_norm(start:stop) = bin2num(q,bsubquant);

%checking:
bit=bsubquant(key,7);

% undo the normalization:
btrans = btrans_norm*nf; %delete

% undo the wavelet-projection:
b = M' * btrans; %delete
```

(c) When decoding the file "conga_tagged.ema" using the function "check_file" (and having "key" set to 1, we find the message "Copyright reserved by EPFL 2007".

(d) An easy way to check your personal EMA file with many different keys is the following: We define the variable "key" to be global in "check_file.m":

```
%key = 1;
global key;
```

Then, in your Matlab command line, you can type:

```
>> global key
>> for key=1:64
check_file('Diggavi_xmas.ema','output.wav');
key
end
```

Then, you just lean back and observe the command line. Every 20 seconds or so, a new key will be tried. Several times, the output of the program will be a meaningless string of symbols. Finally, when the correct key has been tried, you will see a meaningful message (about Christmas, New Year or some other kind of Congratulations), and you can also determine which key has led to that message.