Lecture 14, Multirate Signal Processing, Low Delay Filter Banks

Last time we saw that an MDCT type filter bank leads to an algorithmic delay of z^{-1} in the lower sampling rate, hence a delay of one block, over the analysis and also synthesis filter bank. (Remember: the z^{-1} came from making the inverse delay matrix $D(z)$ causal). In addition to this one block, we usually also need to take into account the so-called **blocking delay**, which results from assembling our incoming samples into blocks of length N. We first have to wait for N-1 incoming samples on the analysis side, and at the arrival of the N'th sample the block is full, and we can send it immediately to further processing. Hence the blocking delay is **N-1 samples**. Observe that we have no blocking delay on the synthesis side.
This means the total delay is both, this blocking delay, plus the delay resulting from our matrices. In the case of our MDCT type filter bank, the total delay or so-called **system delay** is our blocking delay of N-1 sample plus 1 block delay from the polphase matrices:

$$n_d = N - 1 + N = 2\mathrm{N} - 1$$

We also have the length of our filter $L=2\mathrm{N}$, which results in the system delay to be the length of the filters minus 1:

$$n_d = L - 1$$

This relation holds in **general for para-unitary filter banks**!

Observe that mathematically it only makes sense to **define a delay over the entire analysis and synthesis filter bank**, because here we have the exact same signal as the input signal, just delayed. After only the analysis, we only have subband signals, which look differently and have a different sampling rate. Hence, here we cannot really define a delay.

For para-unitary filter banks, the only possibility to reduce the system delay is to reduce the filter length. But this also degrades the filter characteristics, for instance, it would lead to reduced stopband attenuation.

Often, in applications like real time audio communications, it is desirable to have a lower end-to-end delay, but also good filters with high stopband attenuation, such that we still obtain a good coding or compression performance.

There the solution are the so-called **Low-Delay Filter Banks**.

With them, it is possible to reduce the system

delay without reducing the filter length L. How does this work? Or first, how do we get longer filters than the MDCT type filter banks in the first place? For MDCT type filter banks, we only had filters of length L=2N. How do we make them longer?

To see a solution, we first need to take a look at the polyphase matrix again, this time for filters of arbitrary length (more than 2N).

We can again compute our folding matrix of it, using

$$\boldsymbol{F}_a \cdot \boldsymbol{D}(z) = \boldsymbol{H}(z) \cdot \boldsymbol{T}^{-1}$$

as for the MDCT type filter banks, just that now our analysis polyphase matrix $\boldsymbol{H}(z)$ can contain baseband impulse responses $h_a(n)$ with arbitrary length L (and L can be much larger than 2N here). The resulting form of the analysis folding matrix is the same as for the MDCT type filter banks, but with higher degree polynomials in it, as expected:

$$\boldsymbol{F}_a(z) =$$

$$= \begin{bmatrix} . & 0 & -H_{2N-1}^{\downarrow 2N}(-z^2)\cdot z^{-1} & -H_{N-1}^{\downarrow 2N}(-z^2) & 0 & . \\ . & . & 0 & 0 & . & . \\ -H_{1.5N}^{\downarrow 2N}(-z^2)\cdot z^{-1} & 0 & . & . & 0 & -H_{N/2}^{\downarrow 2N}(-z^2) \\ -H_{1.5N-1}^{\downarrow 2N}(-z^2)\cdot z^{-1} & 0 & . & . & 0 & H_{N/2-1}^{\downarrow 2N}(-z^2) \\ 0 & . & 0 & . & . & 0 \\ . & 0 & -H_{N}^{\downarrow 2N}(-z^2)\cdot z^{-1} & H_{0}^{\downarrow 2N}(-z^2) & . & . \end{bmatrix}$$

eq. **(1)** with

$$H_n^{\downarrow 2N}(z) := \sum_{m=0}^{\infty} h_a(m2N+n)\cdot z^{-m} \quad \text{(eq.2)}$$

which are the *z*-transforms of the downsampled by 2N versions of our baseband prototype h_a for the analysis at phases n . The minus sign in $-z^2$ results from our cosine modulation function changing its sign every 2nd block.

(**eq: 2**)

(From: G. Schuller, MJT. Smith: “New Framework for Perfect Reconstruction Filter Banks”, IEEE Transactions on Signal Processing, Aug. 1996).

Observe that this $\boldsymbol{F}_a(z)$ corresponds to our $\boldsymbol{F}_a \cdot \boldsymbol{D}(z)$.

Observe that the **order of the polyphase polynomials determines the resulting filter length** of our filter bank. If the highest exponent is M, then our filters have length M+1 blocks (of 2N here since we the downsampling of 2N), or

length (M+1)2N.

Here we can observe 2 effects:

- One is that the shape of our analysis folding matrix has again the diamond shape.
- Second, the polynomials in it now have a higher order, depending on the length of our filters.

For the MDCT type with L=2N we still get an easy solution here, but what about longer filters? Our Goal is still to obtain a perfect reconstruction, and also to obtain Finite Impulse Response filters for our synthesis, to make it easier to handle.

The **approach** now is to use our previous matrix decomposition, writing our polyphase matrices as a product of simpler matrices, and simply **extend this product** with **additional matrices**, such that we

- keep the diamond shape of our resulting folding matrix intact (because this was the result of our cosine modulation, which we would like to keep intact),
- that we increase the polynomial order and hence the resulting filter length

- such that we get easy FIR inverses
- and such we can control the resulting system delay.

The trick now is to invent additional matrices which fulfill these requirements.
These additional matrices can be split into separate types. The first type is the so-called **Zero-Delay Matrix**:

$$\boldsymbol{E}_i(z)=\begin{bmatrix} 0 & . & . & . & 0 & 1 \\ . & \ddots & . & 0 & 1 & 0 \\ . & \dots & 0 & 1 & 0 & . \\ . & 0 & 1 & e_0^i z^{-1} & 0 & . \\ 0 & 1 & 0 & . & \ddots & 0 \\ 1 & 0 & . & . & . & e_{N/2-1}^i z^{-1} \end{bmatrix}$$

It has an inverse, which is **already causal,** we don't need a delay to make it causal!:

$$\boldsymbol{E}_{\boldsymbol{i}}^{-\mathbf{1}}(z)=\begin{bmatrix} -e_{N/2-1}^{i}z^{-1} & . & . & . & 0 & 1 \\ . & \ddots & . & 0 & 1 & 0 \\ . & ... & -e_{0}^{i}z^{-1} & 1 & 0 & . \\ . & 0 & 1 & 0 & . & . \\ 0 & 1 & 0 & . & \ddots & . \\ 1 & 0 & . & . & . & 0 \end{bmatrix}$$

In sympy, with N=4 and $[e_{0,}e_{1}]$:
Start python, for instance in a terminal window.
Then type

```
from sympy import *
from numpy import *

z=symbols('z')
#Coefficients
e=symbols('e:2')

#The zero-delay E matrix:
E=Matrix([[0, 0, 0, 1],[0, 0, 1, 0],[0, 1, e[0]*z**(-1), 0],
[1, 0, 0,e[1]*z**(-1) ]])
E
```

```
Matrix([
[0, 0,    0,    1],
[0, 0,    1,    0],
[0, 1, e0/z,    0],
[1, 0,    0, e1/z]])
```

Then its inverse is:

```
E**(-1)
Matrix([
[-e1/z,     0, 0, 1],
[    0, -e0/z, 1, 0],
[    0,     1, 0, 0],
[    1,     0, 0, 0]])
```

we see that this is indeed the form as above.

We can now multiply our analysis polyphase folding matrix (1) with this zero-delay matrix, and get polynomials of higher order, because we have elements with z^-1 in it. The synthesis polyphase matrix is multiplied by this inverse, to still obtain perfect reconstruction. Since the

inverse also has polynomials of degree 1, it is still FIR. Also, since the inverse is already causal, we don't need any multiplication with z^-1 to make it causal, and hence no additional delay. This means, if we use this matrix, we can **increase the filter length**, but **don't increase the delay**! In principle we can repeat this process as often as we like, and get **arbitrarily high filter lengths**, but **no increase in delay**! Zero-delay matrices **increase the filter length by 1 block**, but **don't increase the system delay**. We increase the filter length by 1 block because the increase of the order of the polynomials by one which corresponds to 1 block.

There is a second type with the same properties, where we simply use the other half of the diagonal:

$$\boldsymbol{G_i}(z)=\begin{bmatrix} g_0^i\cdot z^{-1} & . & . & . & 0 & 1 \\ . & \ddots & . & 0 & 1 & 0 \\ . & \dots & g_{N/2-1}^i\cdot z^{-1} & 1 & 0 & . \\ . & 0 & 1 & 0 & . & . \\ 0 & 1 & 0 & . & \ddots & . \\ 1 & 0 & . & . & . & 0 \end{bmatrix}$$

Its (causal) inverse is:

$$\boldsymbol{G_i^{-1}}(z)=\begin{bmatrix} 0 & . & . & . & 0 & 1 \\ . & \ddots & . & 0 & 1 & 0 \\ . & \dots & 0 & 1 & 0 & . \\ . & 0 & 1 & -g_{N/2-1}^i\cdot z^{-1} & 0 & . \\ 0 & 1 & 0 & . & \ddots & 0 \\ 1 & 0 & . & . & . & -g_0^i\cdot z^{-1} \end{bmatrix}$$

Observe: Contrary to diagonal matrices, where we needed to take the inverse of each element of the diagonal, here we obtain the inverse by flipping the diagonal half and flipping just the sign, which keeps it causal!

Python Sympy Example

We would like to see how the product

$\boldsymbol{P_a}(z)=\boldsymbol{Fa}\cdot\boldsymbol{D}(z)\cdot\boldsymbol{G}(z)$ of our folding matrix Fa(z) with the zero delay matrix G(z) looks like. For that we write the following python file (e.g. as “lowdelayFa.py”),

```
from sympy import *
z=symbols('z')
g=symbols('g:2')
h=symbols('h:8')

Fa=Matrix([[0, -h[7], -h[3], 0],[-h[6], 0, 0, -h[2]],[-h[5],
0, 0, h[1]],[0, -h[4], h[0], 0]])
D=Matrix([[z**(-1), 0, 0, 0],[0, z**(-1), 0, 0],[0, 0, 1,
0],[0, 0, 0, 1]])
G=Matrix([[g[0]*z**(-1), 0, 0, 1],[0, g[1]*z**(-1), 1,
0],[0, 1, 0, 0],[1, 0, 0,0 ]])

print("Fa=")
pprint(Fa)
print("D(z)=")
pprint(D)
print("G(z)=")
pprint(G)
print("Fa*D(z)*G(z)=")
```

```
print(Fa*D*G)
pprint(Fa*D*G)
print(latex(Fa*D*G))
```

The call from a terminal window,

```
python lowdelayFa.py
```

which results in:

Fa=

$$\begin{bmatrix} 0 & -h_7 & -h_3 & 0 \\ -h_6 & 0 & 0 & -h_2 \\ -h_5 & 0 & 0 & h_1 \\ 0 & -h_4 & h_0 & 0 \end{bmatrix}$$

D(z)=

$$\begin{bmatrix} \frac{1}{z} & 0 & 0 & 0 \\ 0 & \frac{1}{z} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

G(z)=

$$\begin{bmatrix} \frac{g_0}{z} & 0 & 0 & 1 \\ 0 & \frac{g_1}{z} & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Fa*D(z)*G(z)=

```
[            0, g1*h7/z**2 + h3, h7/z,    0]
[g0*h6/z**2 + h2,             0,    0, h6/z]
[g0*h5/z**2 - h1,             0,    0, h5/z]
[            0, g1*h4/z**2 - h0, h4/z,    0]
```

or using “pprint” (pretty print) instead of “print” we get the somewhat “prettier” form

Fa*D(z)*G(z)=

$$\begin{bmatrix} 0 & -\frac{g_1 \cdot h_7}{z^2} - h_3 & \frac{-h_7}{z} & 0 \\ -\frac{g_0 \cdot h_6}{z^2} - h_2 & 0 & 0 & \frac{-h_6}{z} \\ -\frac{g_0 \cdot h_5}{z^2} + h_1 & 0 & 0 & \frac{-h_5}{z} \\ 0 & -\frac{g_1 \cdot h_4}{z^2} + h_0 & \frac{-h_4}{z} & 0 \end{bmatrix}$$

Or using “*print(latex(Fa*D*G))*” and using the

LaTex equation editor, we get this nice print:

$$\begin{bmatrix} 0 & -\frac{g_1 h_7}{z^2} - h_3 & -\frac{h_7}{z} & 0 \\ -\frac{g_0 h_6}{z^2} - h_2 & 0 & 0 & -\frac{h_6}{z} \\ -\frac{g_0 h_5}{z^2} + h_1 & 0 & 0 & -\frac{h_5}{z} \\ 0 & -\frac{g_1 h_4}{z^2} + h_0 & -\frac{h_4}{z} & 0 \end{bmatrix}$$

We can see that this is indeed the **diamond shaped form** of equation (1). By **comparing** it with **eq. (1)** we could now **read out** the resulting final **baseband prototype filter** for the analysis $h_a(n)$.

Observe that eq. (1) is for an even number of Zero-Delay matrices. For odd numbers, as in this example, we would need to flip the matrix in eq. (1) horizontally (fliplr).

So now we don't specify h_a in our design process, but we start with our coefficients for our matrices, and obtain a baseband prototype filter h_a with the desired length and delay properties.

Q: What is a possible drawback of these zero delay matrices?
A: If we only increase the filter length with our zero delay matrices, we can observe that the far off **attenuation increases easily**, a **clear improvement** over the MDCT case. But the **nearby attenuation**, to the neighboring subbands, increases not as much as for orthogonal filter banks with **similar length but with more delay**. Hence it can make sense to also increase the system delay, to obtain filters with higher nearby attenuation more easily.
This leads us to a matrix, which is very efficient in increasing the system delay, the opposite of zero-delay matrices:
The so-called **Maximum-Delay Matrices**. They basically result from exchanging z^{-1} by z in our zero-delay matrices, and a multiplication with z^{-1} to make them causal:

$$\boldsymbol{H1}\left(z\right):=z^{-1}\cdot\boldsymbol{E}\left(z^{-1}\right)$$

This is the type of matrix, each matrix in our cascade has its own coefficients.
The other type is:

$$\boldsymbol{H2}\left(z\right):=z^{-1}\cdot\boldsymbol{G}\left(z^{-1}\right)$$

Their inverse need a multiplication with z^{-2} to make it causal, and is, correspondingly:

$$\boldsymbol{H1}^{-1}(z)\cdot z^{-2}=z^{-1}\cdot\boldsymbol{E}^{-1}(z^{-1})$$

and

$$\boldsymbol{H2}^{-1}(z)\cdot z^{-2}=z^{-1}\cdot\boldsymbol{G}^{-1}(z^{-1})$$

Here we can see that the matrix and its inverse leads to 2 blocks of delay:

$$\boldsymbol{H1}(z)\cdot\boldsymbol{H1}^{-1}(z)\cdot z^{-2}=z^{-2}$$

and

$$\boldsymbol{H2}(z)\cdot\boldsymbol{H2}^{-1}(z)\cdot z^{-2}=z^{-2}$$

Observe that the Maximum-Delay matrices have a polynomial order of 1, just like the zero-delay matrices, and hence **increase the filter length by 1 block**, but **increase the delay by 2 blocks**! This is twice as much as for an orthogonal filter bank.

Using only maximum-delay matrices would result in similar frequency responses as with only zero-delay matrices, the impulse response would only appear time-reversed. We get increased nearby attenuation only if we **mix both types** of matrices. Orthogonal filter banks can be obtained if we use the same number of Maximum-delay and Zero-delay matrices.

Our resulting polyphase matrix for the analysis then becomes this product:

$$\boldsymbol{P_a}(z)=\boldsymbol{F_a}\cdot\boldsymbol{D}(z)\prod_{n=0}^{P-1}\boldsymbol{G_n}(z)\prod_{m=0}^{Q-1}\boldsymbol{H_m}(z)\cdot\boldsymbol{T}$$

where $\boldsymbol{H_m}(z)$ is either $\boldsymbol{H1}(z)$ (if P is even) or $\boldsymbol{H2}(z)$, if P is odd.

For the synthesis polyphase matrix we get the inverse product:

$$\boldsymbol{P_s}(z)=\boldsymbol{P_a^{-1}}(z)\cdot z^{-(2Q+1)}=$$

$$=\boldsymbol{T^{-1}}\prod_{m=Q-1}^{0}z^{-2}\boldsymbol{H_m^{-1}}(z)\prod_{n=P-1}^{0}\boldsymbol{G_n^{-1}}(z)\boldsymbol{D^{-1}}(z)\cdot z^{-1}\boldsymbol{F_a^{-1}}$$

Now we can design the impulse response length and the system delay of our filter bank. The **total number of our zero-delay and maximum-delay matrices determines the length of our impulse response.** Including the effect of the $\boldsymbol{F_a}\cdot\boldsymbol{D}(z)$ matrix (a length of 2 blocks), we obtain a total filter length of

$$L=(2+Q+P)N\ .$$

Just the number of **maximum-delay matrices** $\boldsymbol{H_m}(z)$ determines the **system delay**. Including the effects of the $\boldsymbol{F_a}\cdot\boldsymbol{D}(z)$ matrix and the blocking delay (with a delay of 2N-1), we obtain a system delay of

$$n_d = 2N - 1 + 2Q \cdot N \ .$$

Each zero-delay and maximum-delay matrix increases the impulse response length by one block of N samples. Each maximum-delay matrix increases the system delay by 2 blocks (2N samples), in this case starting with the MDCT delay of 2N-1 samples. It is also possible to start with a lower delay than an MDCT, by setting the leading samples of the MDCT type part to zero. **Observe**: We obtain the **same analysis and synthesis baseband impulse response**, if the determinants of our submatrices of our polyphase matrix are -1, as seen before. If we look at the **zero-delay matrices**, we find that their determinant of the submatrices is already -1 (very practical). If we look at the **maximum-delay matrices**, we see that their determinant of the submatrices is $-z^{-2}$, which is their resulting delay contribution and a factor of -1, which has the same effect, of leading to the same analysis and synthesis impulse responses. All we need to remain to do, is to make sure that the **matrix** F_a has a **determinant of -1** in their submatrices, to obtain identical analysis and synthesis baseband impulse responses. This means the number of unknown coefficients for

the folding matrix $\boldsymbol{F}_a$ is going down from 2N to 2N-N/2. The remaining N/2 coefficients come out ot the det=-1 condition.
This also makes the design easier, because then we only need to **optimize one side** of our filter bank, for instance the analysis side, and we have a reduced number of coefficients.
We need a numerical optimization, because we now have a long cascade or product, with many unknown coefficients (in the Fa folding matrix, the Zero-Delay and Maximum-Delay matrices). We need to optimize them such that we obtain a good frequency response for the resulting impulse response. We can use eq. (1) in this lecture and our product for the folding matrix $\boldsymbol{F}_a(z)$ (without the transform matrix $\boldsymbol{T}$), to **read out the resulting baseband impulse response** $h_a(n)$. We can then compute the resulting frequency response and use that as a function to optimize, towards some “optimum” frequency response (for instance an ideal low pass filter as the prototype).
An example application of this type of filter bank is the audio coder “Enhanced Low Delay AAC” (ELD-AAC) in MPEG-4. Here it is used to obtain a lower encoding/decoding delay.

Python Fast Implementation Example

For Low Delay filter banks it is useful to process the samples as they come from the sound card. Hence we need a different implementation, an implementation which is not based on the signal already in memory. To obtain it, we interpret the z^{-1} as a delay by one sample, and implement it using a memory element, which stores or delays samples for 1 sampling interval.
The Delay Matrix $D(z)$ for instance can be implemented as follows:

```
#The D(z) matrix:
def Dmatrix(samples):
#implementation of the delay matrix D(z)
   #Delay elements:
   out=np.zeros(N)
   out[0:(N/2)]=Dmatrix.z
   Dmatrix.z=samples[0:(N/2)]
   out[N/2:N]=samples[N/2:N]
   return out


Dmatrix.z=np.zeros(N/2)
```

A real time MDCT or Low Delay filter bank

implementation, producing a waterfall spectrogram, including the synthesis filter bank and playback of the reconstructed signal, can be run with

```
python pyrecplayMDCT.py
```