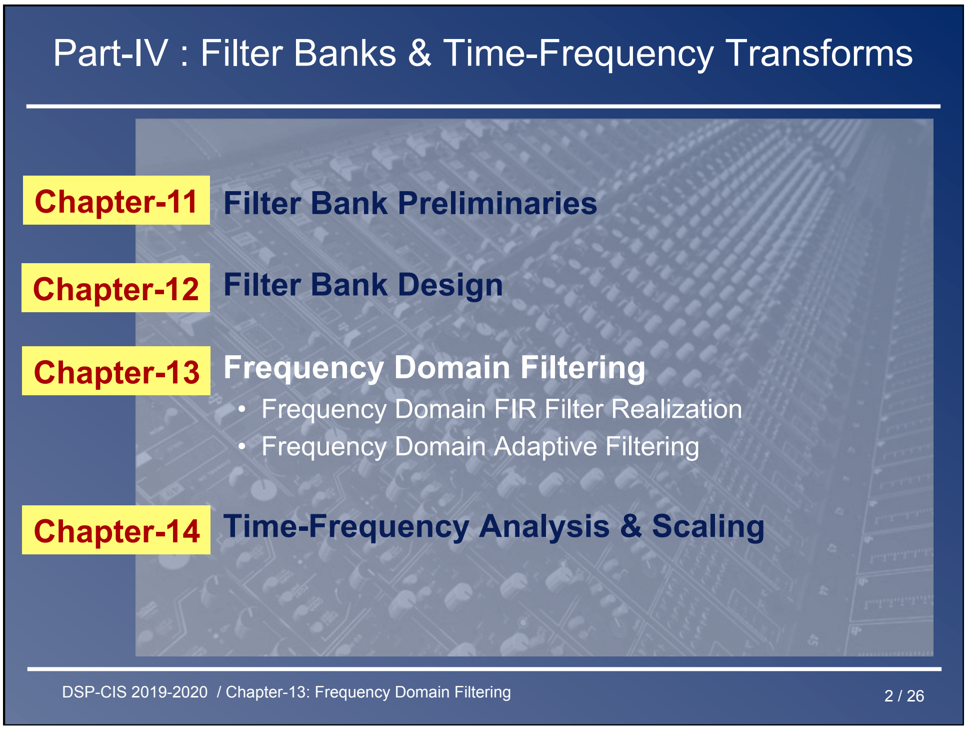
DSP-CIS

Part-IV : Filter Banks & Time-Frequency Transforms

Chapter-13 : Frequency Domain Filtering

Marc Moonen
Dept. E.E./ESAT-STADIUS, KU Leuven
marc.moonen@kuleuven.be
www.esat.kuleuven.be/stadius/



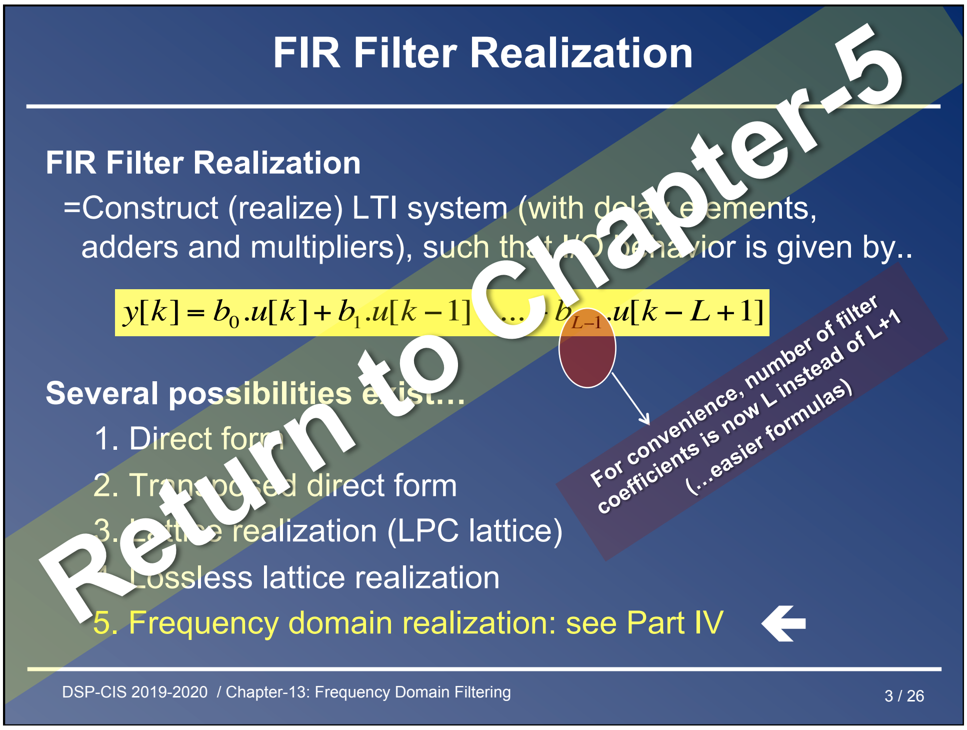
Part-IV : Filter Banks & Time-Frequency Transforms

Chapter-11 **Filter Bank Preliminaries**

Chapter-12 **Filter Bank Design**

Chapter-13 **Frequency Domain Filtering**
- Frequency Domain FIR Filter Realization
- Frequency Domain Adaptive Filtering

Chapter-14 **Time-Frequency Analysis & Scaling**



FIR Filter Realization

FIR Filter Realization

=Construct (realize) LTI system (with delay elements, adders and multipliers), such that I/O behavior is given by..

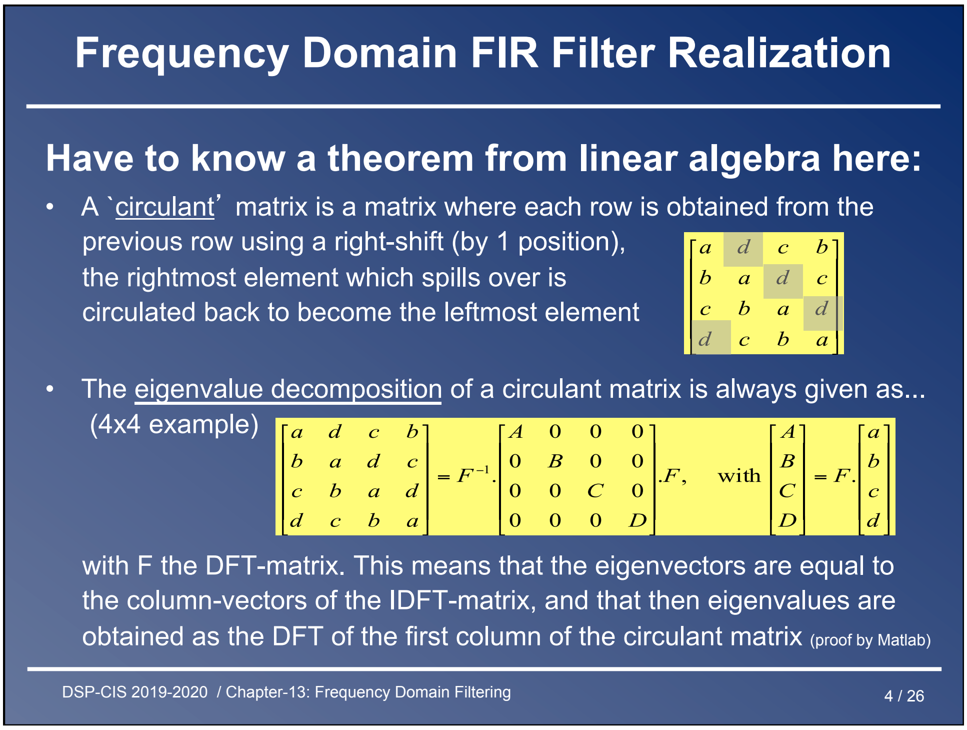
$$y[k]=b_0.u[k]+b_1.u[k-1]+...+b_{L-1}.u[k-L+1]$$

For convenience, number of filter coefficients is now L instead of L+1 (...easier formulas)

Several possibilities exist…

1. Direct form
2. Transposed direct form
3. Lattice realization (LPC lattice)
4. Lossless lattice realization
5. Frequency domain realization: see Part IV ←

Return to Chapter-5



Frequency Domain FIR Filter Realization

Have to know a theorem from linear algebra here:

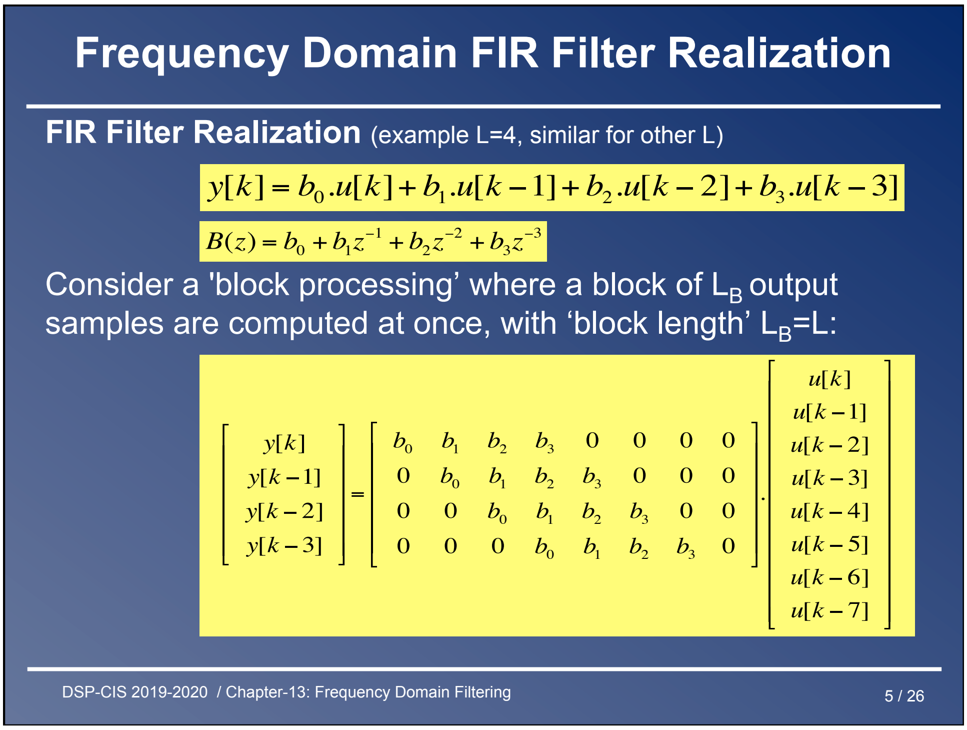
- A `circulant' matrix is a matrix where each row is obtained from the previous row using a right-shift (by 1 position), the rightmost element which spills over is circulated back to become the leftmost element

$$\begin{bmatrix} a & d & c & b \\ b & a & d & c \\ c & b & a & d \\ d & c & b & a \end{bmatrix}$$

- The eigenvalue decomposition of a circulant matrix is always given as... (4x4 example)

$$\begin{bmatrix} a & d & c & b \\ b & a & d & c \\ c & b & a & d \\ d & c & b & a \end{bmatrix} = F^{-1}.\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & D \end{bmatrix}.F, \quad \text{with} \quad \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = F.\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

with F the DFT-matrix. This means that the eigenvectors are equal to the column-vectors of the IDFT-matrix, and that then eigenvalues are obtained as the DFT of the first column of the circulant matrix (proof by Matlab)



Frequency Domain FIR Filter Realization

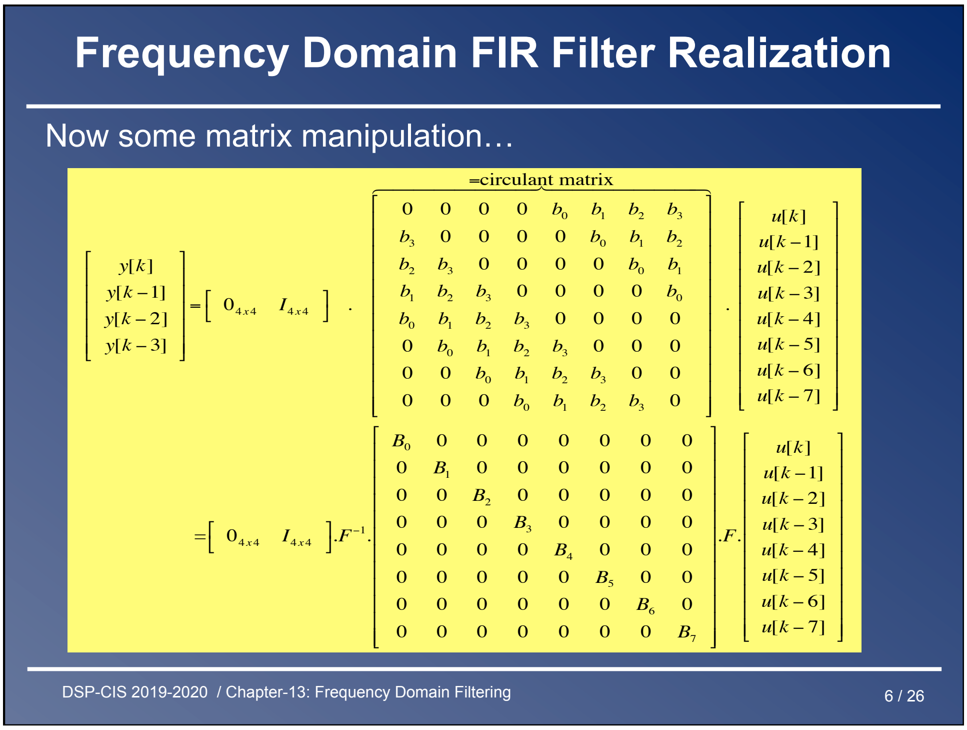
FIR Filter Realization (example L=4, similar for other L)

$$y[k]=b_0.u[k]+b_1.u[k-1]+b_2.u[k-2]+b_3.u[k-3]$$

$$B(z)=b_0+b_1z^{-1}+b_2z^{-2}+b_3z^{-3}$$

Consider a 'block processing' where a block of L_B output samples are computed at once, with 'block length' L_B=L:

$$\begin{bmatrix} y[k] \\ y[k-1] \\ y[k-2] \\ y[k-3] \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 \\ 0 & b_0 & b_1 & b_2 & b_3 & 0 & 0 & 0 \\ 0 & 0 & b_0 & b_1 & b_2 & b_3 & 0 & 0 \\ 0 & 0 & 0 & b_0 & b_1 & b_2 & b_3 & 0 \end{bmatrix} . \begin{bmatrix} u[k] \\ u[k-1] \\ u[k-2] \\ u[k-3] \\ u[k-4] \\ u[k-5] \\ u[k-6] \\ u[k-7] \end{bmatrix}$$



Frequency Domain FIR Filter Realization

Now some matrix manipulation…

$$\begin{bmatrix} y[k] \\ y[k-1] \\ y[k-2] \\ y[k-3] \end{bmatrix} = \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} . \overbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & b_0 & b_1 & b_2 & b_3 \\ b_3 & 0 & 0 & 0 & 0 & b_0 & b_1 & b_2 \\ b_2 & b_3 & 0 & 0 & 0 & 0 & b_0 & b_1 \\ b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 & b_0 \\ b_0 & b_1 & b_2 & b_3 & 0 & 0 & 0 & 0 \\ 0 & b_0 & b_1 & b_2 & b_3 & 0 & 0 & 0 \\ 0 & 0 & b_0 & b_1 & b_2 & b_3 & 0 & 0 \\ 0 & 0 & 0 & b_0 & b_1 & b_2 & b_3 & 0 \end{bmatrix}}^{=\text{circulant matrix}} . \begin{bmatrix} u[k] \\ u[k-1] \\ u[k-2] \\ u[k-3] \\ u[k-4] \\ u[k-5] \\ u[k-6] \\ u[k-7] \end{bmatrix}$$

$$= \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} .F^{-1}. \begin{bmatrix} B_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & B_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & B_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & B_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & B_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & B_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & B_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_7 \end{bmatrix} .F. \begin{bmatrix} u[k] \\ u[k-1] \\ u[k-2] \\ u[k-3] \\ u[k-4] \\ u[k-5] \\ u[k-6] \\ u[k-7] \end{bmatrix}$$

Frequency Domain FIR Filter Realization

- This means that a block of L_B=L output samples can be computed as follows (read previous formula from right to left) :
  - Compute DFT of 2L input samples, i.e. last L samples combined ('overlapped') with previous L samples
  - Perform component-wise multiplication with… (=freq.domain representation of the FIR filter)
  - Compute IDFT
  - Throw away 1st half of result, select ('save') 2nd half

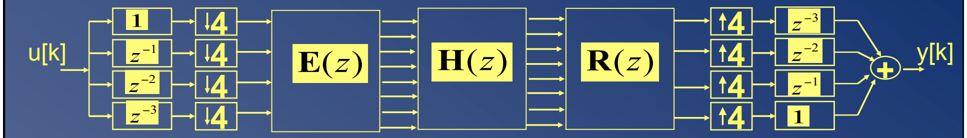
$$\begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} = F.\begin{bmatrix} 0 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This is referred to as an 'overlap-save' procedure

  (and 'frequency domain filter realization' because of the DFT/IDFT)

Frequency Domain FIR Filter Realization

- This corresponds to a filter bank-type realization as follows...



Analysis bank: $\mathbf{E}(z) = F.\begin{bmatrix} I_{4x4} \\ z^{-1}.I_{4x4} \end{bmatrix}.$

Subband processing: $\mathbf{H}(z) = diag\{B_0, B_1, \ldots B_7\}$

Synthesis bank: $\mathbf{R}(z) = \begin{bmatrix} 0 & I_{4x4} \end{bmatrix} F^{-1}$

This is a 2L-channel filter bank, with L-fold downsampling
The analysis FB is a 2L-channel uniform DFT filter bank (see Chapter 11)
The synthesis FB is matched to the analysis bank, for PR: $\mathbf{R}(z).\mathbf{E}(z) = z^{-1}I_{4x4}$

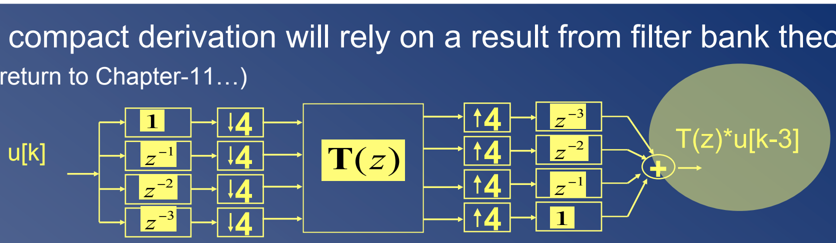
Frequency Domain FIR Filter Realization

- ☺ Overlap-save procedure is very efficient for large L :
  - Computational complexity (with FFT/IFFT i.o. DFT/IDFT) is 2.[α.2L.log(2L)] + 2L multiplications for L output samples, i.e. O(log(L)) per sample for large L
  - Compare to computational complexity for direct form realization: L multiplications per output sample, i.e. O(L) per sample
- ☹ Overlap-save procedure introduces O(L) processing delay/latency (e.g. y[k-L+1] only available sometime after time k)
- Conclusion: For large L, complexity reduction is large, but latency is also large
- Will derive 'intermediate' realizations, with a smaller latency at the expense of a smaller complexity reduction. This will be based on an N^th order polyphase decomposition of B(z)…

Frequency Domain FIR Filter Realization

A compact derivation will rely on a result from filter bank theory

(return to Chapter-11…)



(…and now let B(z) take the place of 'distortion function' T(z))

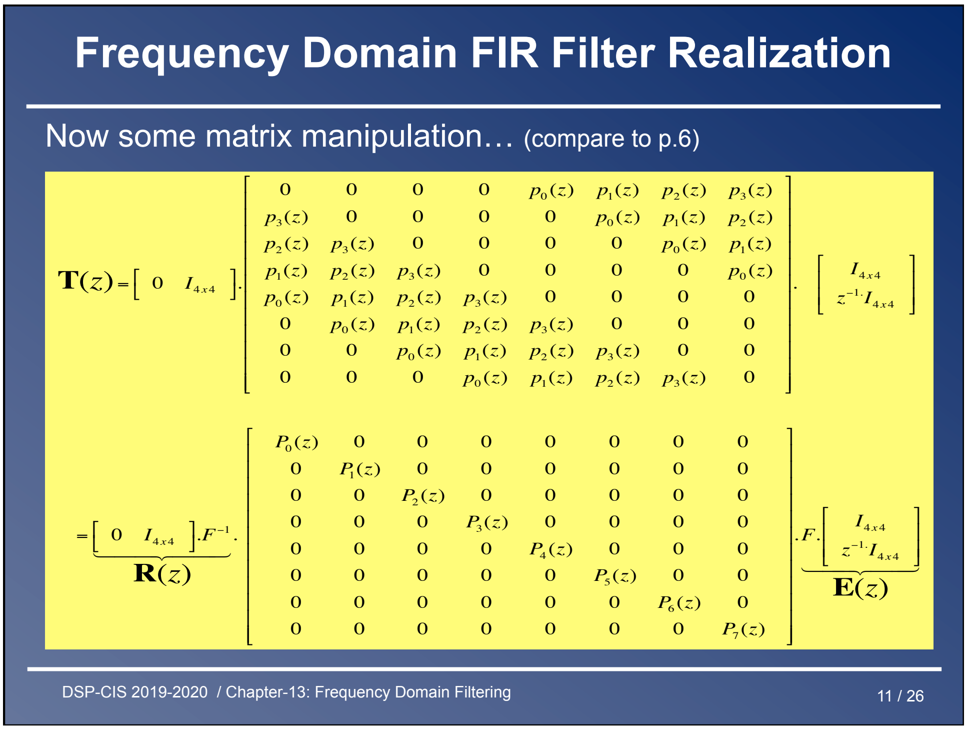
This means that a filter (specified with N-fold polyphase decomposition)

$$B(z)=p_0(z^4)+z^{-1}p_1(z^4)+z^{-2}p_2(z^4)+z^{-3}p_3(z^4)$$

can be realized in a multirate structure, based on a pseudo-circulant matrix

$$\mathbf{T}(z)=\begin{bmatrix} p_0(z) & p_1(z) & p_2(z) & p_3(z) \\ z^{-1}.p_3(z) & p_0(z) & p_1(z) & p_2(z) \\ z^{-1}.p_2(z) & z^{-1}.p_3(z) & p_0(z) & p_1(z) \\ z^{-1}.p_1(z) & z^{-1}.p_2(z) & z^{-1}.p_3(z) & p_0(z) \end{bmatrix}=$$

PS: formulas given for N=4, for conciseness (but without loss of generality)

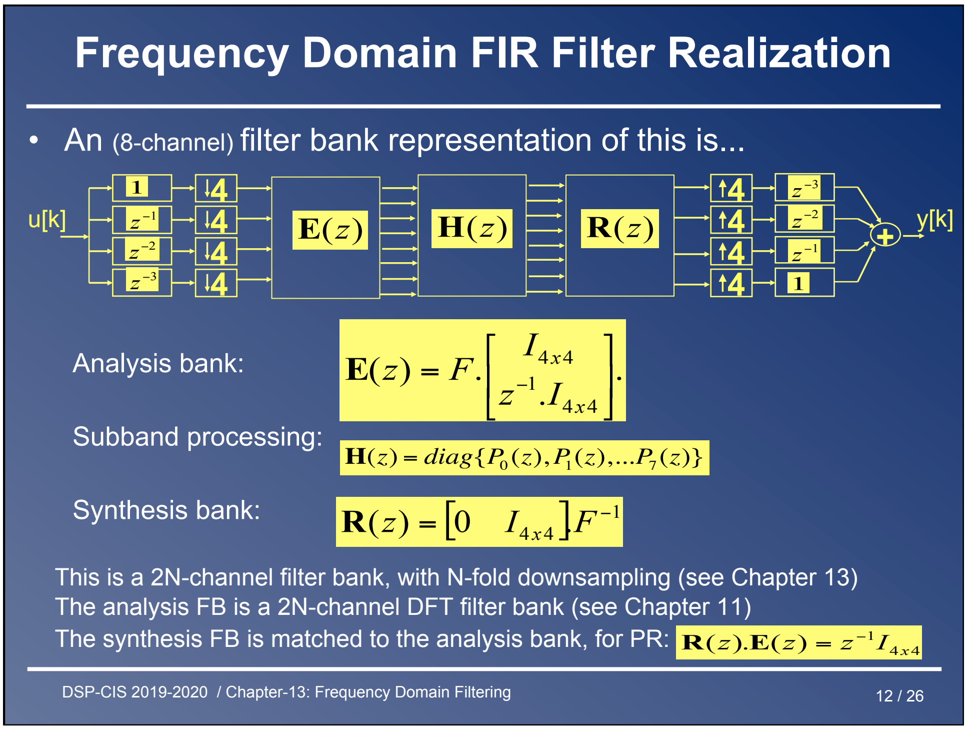


Frequency Domain FIR Filter Realization

Now some matrix manipulation… (compare to p.6)

$$\mathbf{T}(z) = \begin{bmatrix} 0 & I_{4x4} \end{bmatrix} . \begin{bmatrix} 0 & 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) \\ p_3(z) & 0 & 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) \\ p_2(z) & p_3(z) & 0 & 0 & 0 & 0 & p_0(z) & p_1(z) \\ p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 & 0 & p_0(z) \\ p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 & 0 \\ 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 \\ 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 \\ 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 \end{bmatrix} . \begin{bmatrix} I_{4x4} \\ z^{-1}.I_{4x4} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} 0 & I_{4x4} \end{bmatrix} . F^{-1} .}_{\mathbf{R}(z)} \begin{bmatrix} P_0(z) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & P_1(z) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & P_2(z) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & P_3(z) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & P_4(z) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & P_5(z) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P_6(z) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_7(z) \end{bmatrix} \underbrace{. F . \begin{bmatrix} I_{4x4} \\ z^{-1}.I_{4x4} \end{bmatrix}}_{\mathbf{E}(z)}$$



Frequency Domain FIR Filter Realization

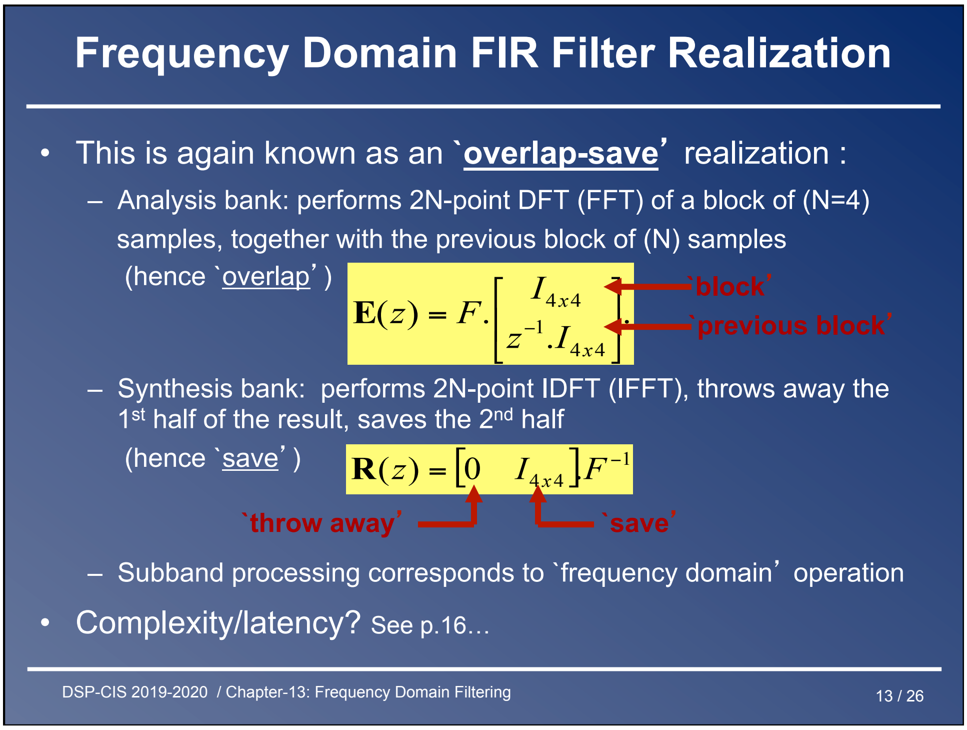
- An (8-channel) filter bank representation of this is...

Analysis bank: $\mathbf{E}(z) = F . \begin{bmatrix} I_{4x4} \\ z^{-1}.I_{4x4} \end{bmatrix}.$

Subband processing: $\mathbf{H}(z) = diag\{P_0(z), P_1(z), ...P_7(z)\}$

Synthesis bank: $\mathbf{R}(z) = \begin{bmatrix} 0 & I_{4x4} \end{bmatrix} F^{-1}$

This is a 2N-channel filter bank, with N-fold downsampling (see Chapter 13)
The analysis FB is a 2N-channel DFT filter bank (see Chapter 11)
The synthesis FB is matched to the analysis bank, for PR: $\mathbf{R}(z).\mathbf{E}(z) = z^{-1} I_{4x4}$



Frequency Domain FIR Filter Realization

- This is again known as an `**overlap-save**' realization :
  - Analysis bank: performs 2N-point DFT (FFT) of a block of (N=4) samples, together with the previous block of (N) samples (hence `overlap')

$$\mathbf{E}(z)=F.\begin{bmatrix} I_{4x4} \\ z^{-1}.I_{4x4} \end{bmatrix}$$

`block'

`previous block'

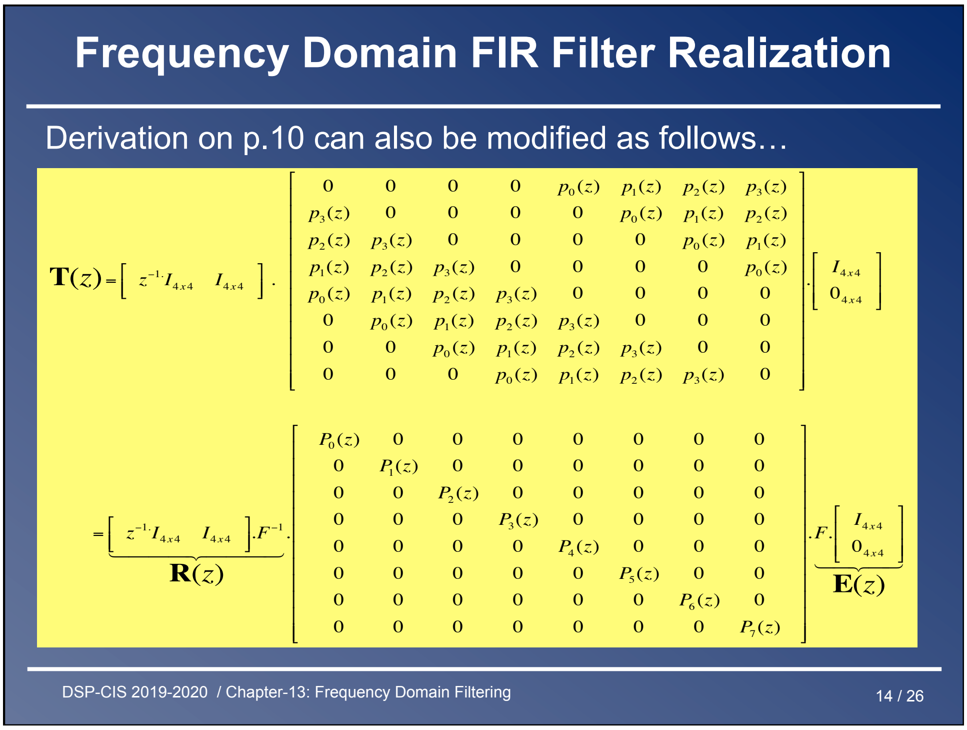
  - Synthesis bank: performs 2N-point IDFT (IFFT), throws away the 1st half of the result, saves the 2nd half (hence `save')

$$\mathbf{R}(z)=\begin{bmatrix} 0 & I_{4x4} \end{bmatrix} F^{-1}$$

`throw away'

`save'

  - Subband processing corresponds to `frequency domain' operation
- Complexity/latency? See p.16…

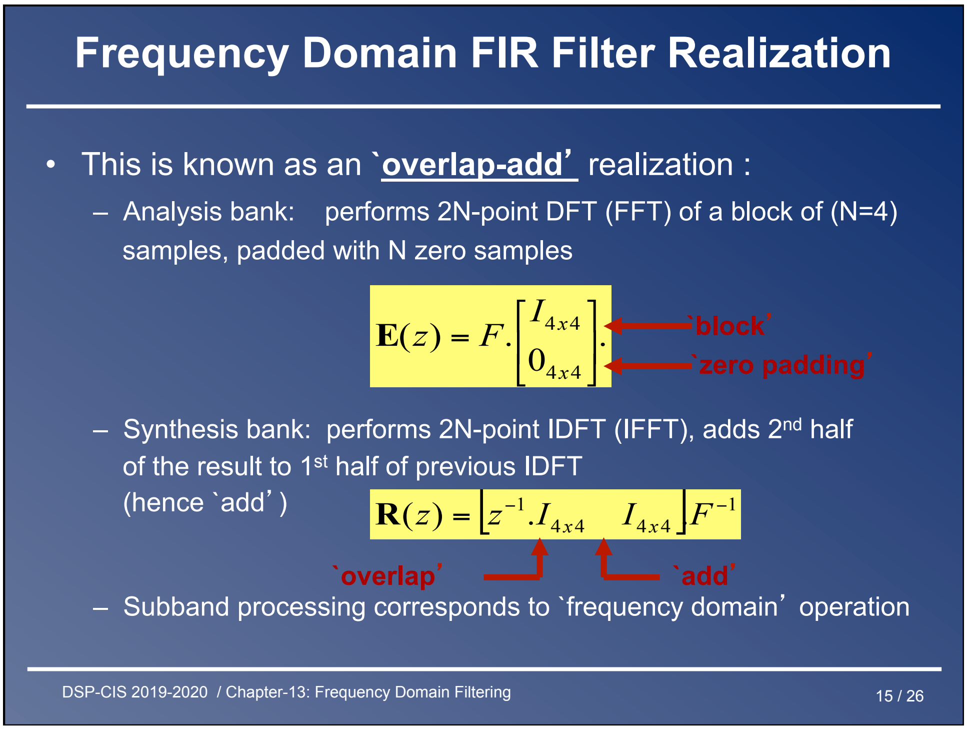


Frequency Domain FIR Filter Realization

Derivation on p.10 can also be modified as follows…

$$\mathbf{T}(z)=\begin{bmatrix} z^{-1}.I_{4x4} & I_{4x4} \end{bmatrix}.\begin{bmatrix} 0 & 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) \\ p_3(z) & 0 & 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) \\ p_2(z) & p_3(z) & 0 & 0 & 0 & 0 & p_0(z) & p_1(z) \\ p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 & 0 & p_0(z) \\ p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 & 0 \\ 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 & 0 \\ 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 & 0 \\ 0 & 0 & 0 & p_0(z) & p_1(z) & p_2(z) & p_3(z) & 0 \end{bmatrix}.\begin{bmatrix} I_{4x4} \\ 0_{4x4} \end{bmatrix}$$

$$=\underbrace{\begin{bmatrix} z^{-1}.I_{4x4} & I_{4x4} \end{bmatrix}.F^{-1}}_{\mathbf{R}(z)}.\begin{bmatrix} P_0(z) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & P_1(z) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & P_2(z) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & P_3(z) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & P_4(z) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & P_5(z) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P_6(z) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_7(z) \end{bmatrix}\underbrace{.F.\begin{bmatrix} I_{4x4} \\ 0_{4x4} \end{bmatrix}}_{\mathbf{E}(z)}$$



Frequency Domain FIR Filter Realization

- This is known as an `**overlap-add**' realization :
  - Analysis bank: performs 2N-point DFT (FFT) of a block of (N=4) samples, padded with N zero samples

$$\mathbf{E}(z) = F.\begin{bmatrix} I_{4x4} \\ 0_{4x4} \end{bmatrix}.$$

`block'

`zero padding'

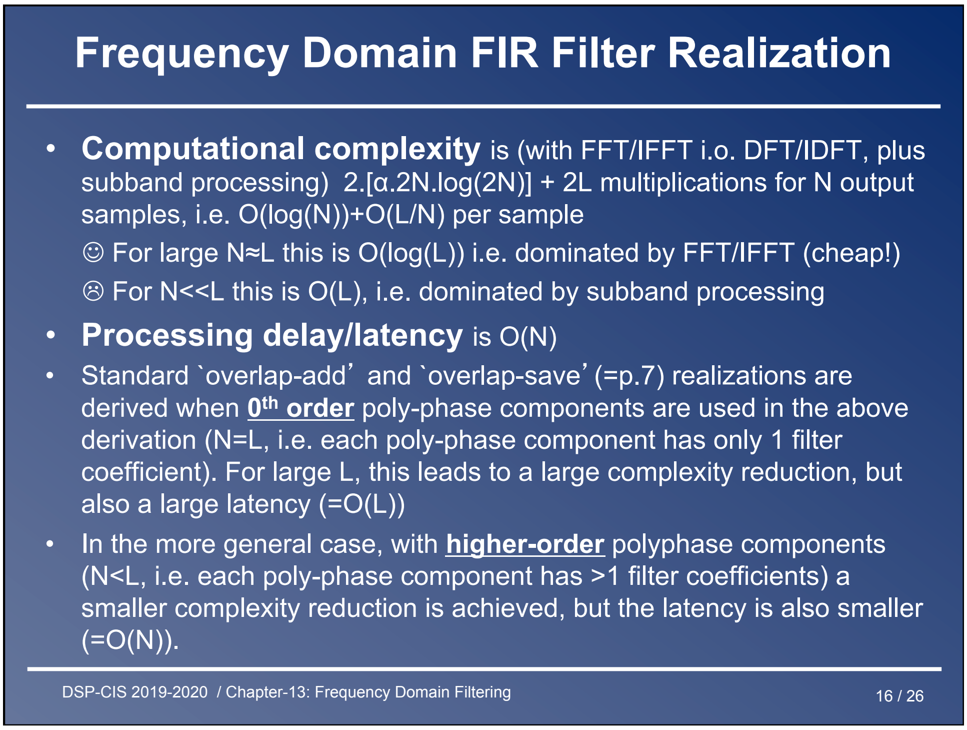
  - Synthesis bank: performs 2N-point IDFT (IFFT), adds 2nd half of the result to 1st half of previous IDFT (hence `add')

$$\mathbf{R}(z) = \begin{bmatrix} z^{-1}.I_{4x4} & I_{4x4} \end{bmatrix} F^{-1}$$

`overlap'

`add'

  - Subband processing corresponds to `frequency domain' operation



Frequency Domain FIR Filter Realization

- **Computational complexity** is (with FFT/IFFT i.o. DFT/IDFT, plus subband processing) 2.[α.2N.log(2N)] + 2L multiplications for N output samples, i.e. O(log(N))+O(L/N) per sample

  ☺ For large N≈L this is O(log(L)) i.e. dominated by FFT/IFFT (cheap!)

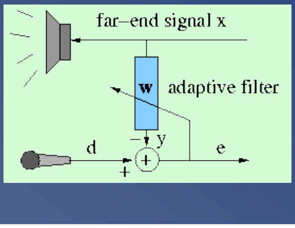
  ☹ For N<<L this is O(L), i.e. dominated by subband processing
- **Processing delay/latency** is O(N)
- Standard `overlap-add' and `overlap-save' (=p.7) realizations are derived when **0th order** poly-phase components are used in the above derivation (N=L, i.e. each poly-phase component has only 1 filter coefficient). For large L, this leads to a large complexity reduction, but also a large latency (=O(L))
- In the more general case, with **higher-order** polyphase components (N<L, i.e. each poly-phase component has >1 filter coefficients) a smaller complexity reduction is achieved, but the latency is also smaller (=O(N)).

Frequency Domain Adaptive Filtering

- A similar derivation can be made for LMS-based adaptive filtering with block processing ('Block-LMS'). The adaptive filter then consist in a <u>filtering operation</u> plus an adaptation operation, which corresponds to a <u>correlation operation</u>. Both operations can be performed cheaply in the frequency domain..
- Starting point is the LMS update equation

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu.\mathbf{x}_k.(d[k] - \mathbf{x}_k^T \mathbf{w}_k)$$

$$\mathbf{x}_k = \begin{bmatrix} x[k] \\ \vdots \\ x[k-L+1] \end{bmatrix}, \qquad \mathbf{w}_k = \begin{bmatrix} w[0] \\ \vdots \\ w[L-1] \end{bmatrix}$$



Frequency Domain Adaptive Filtering

Consider block processing with so-called **<u>'Block-LMS'</u>**

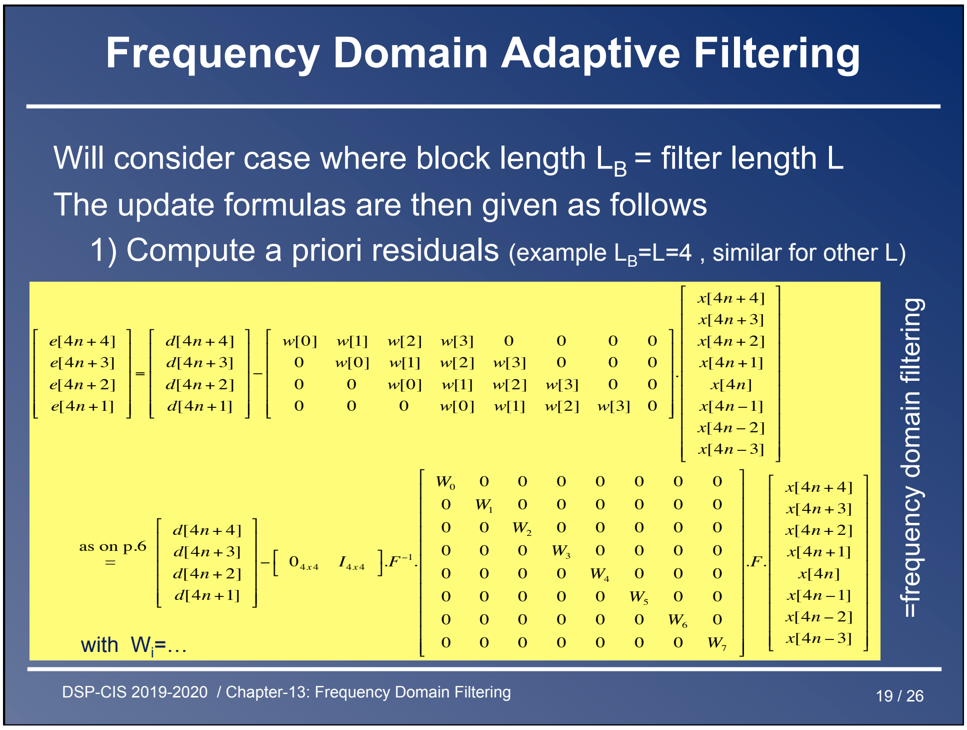
- Remember that LMS is a ‘stochastic gradient’ algorithm, where **<u>instantaneous</u>** estimates of the autocorrelation matrix and cross-correlation vector are used to compute a gradient (=steepest descent vector)
- Block-LMS uses **<u>averaged</u>** estimates, with averaging over a block of L_B (=‘block length’) samples, and hence an averaged gradient.

The update formula is then..

where n is the block index

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu. \sum_{k=nL_B+1}^{nL_B+L_B} \mathbf{x}_k.(d[k] - \mathbf{x}_k^T \mathbf{w}_n)$$

- Compared to LMS, Block-LMS does fewer updates (one per L_B samples), but with (presumably) better gradient estimates. Overall, convergence could be faster or slower (=unpredictable).
- The important thing is that Block-LMS can be realized cheaply…



Frequency Domain Adaptive Filtering

Will consider case where block length L_B = filter length L

The update formulas are then given as follows

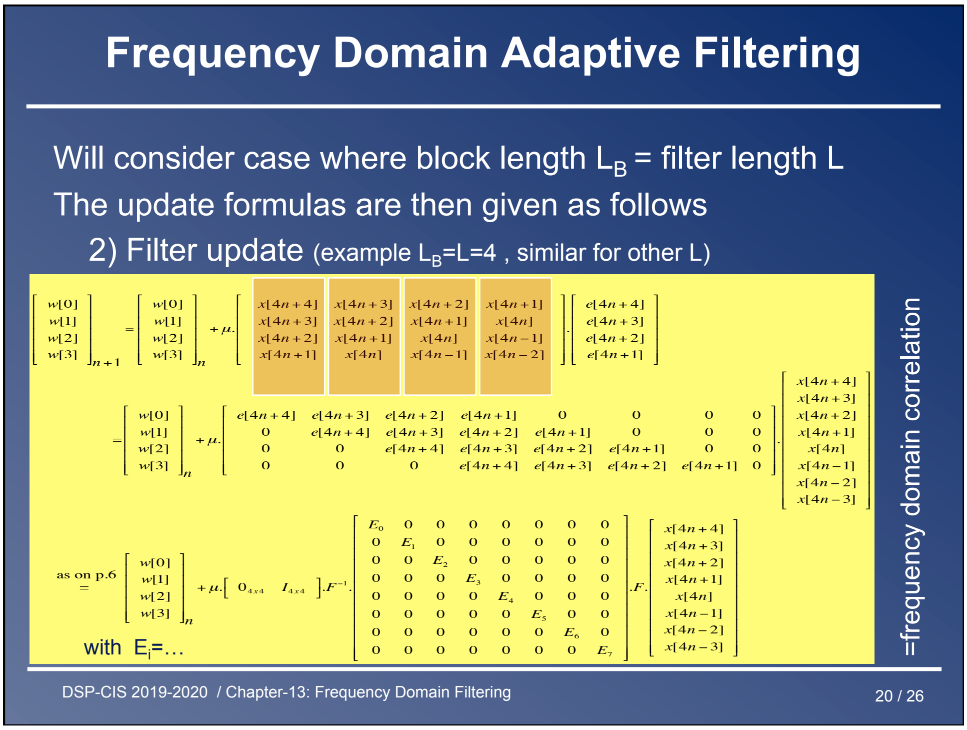
1) Compute a priori residuals (example L_B=L=4 , similar for other L)

$$\begin{bmatrix} e[4n+4] \\ e[4n+3] \\ e[4n+2] \\ e[4n+1] \end{bmatrix} = \begin{bmatrix} d[4n+4] \\ d[4n+3] \\ d[4n+2] \\ d[4n+1] \end{bmatrix} - \begin{bmatrix} w[0] & w[1] & w[2] & w[3] & 0 & 0 & 0 & 0 \\ 0 & w[0] & w[1] & w[2] & w[3] & 0 & 0 & 0 \\ 0 & 0 & w[0] & w[1] & w[2] & w[3] & 0 & 0 \\ 0 & 0 & 0 & w[0] & w[1] & w[2] & w[3] & 0 \end{bmatrix} . \begin{bmatrix} x[4n+4] \\ x[4n+3] \\ x[4n+2] \\ x[4n+1] \\ x[4n] \\ x[4n-1] \\ x[4n-2] \\ x[4n-3] \end{bmatrix}$$

$$\overset{\text{as on p.6}}{=} \begin{bmatrix} d[4n+4] \\ d[4n+3] \\ d[4n+2] \\ d[4n+1] \end{bmatrix} - \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} .F^{-1}. \begin{bmatrix} W_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & W_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & W_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & W_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & W_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & W_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & W_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & W_7 \end{bmatrix} .F. \begin{bmatrix} x[4n+4] \\ x[4n+3] \\ x[4n+2] \\ x[4n+1] \\ x[4n] \\ x[4n-1] \\ x[4n-2] \\ x[4n-3] \end{bmatrix}$$

with W_i=…

=frequency domain filtering



Frequency Domain Adaptive Filtering

Will consider case where block length L_B = filter length L

The update formulas are then given as follows

2) Filter update (example L_B=L=4 , similar for other L)

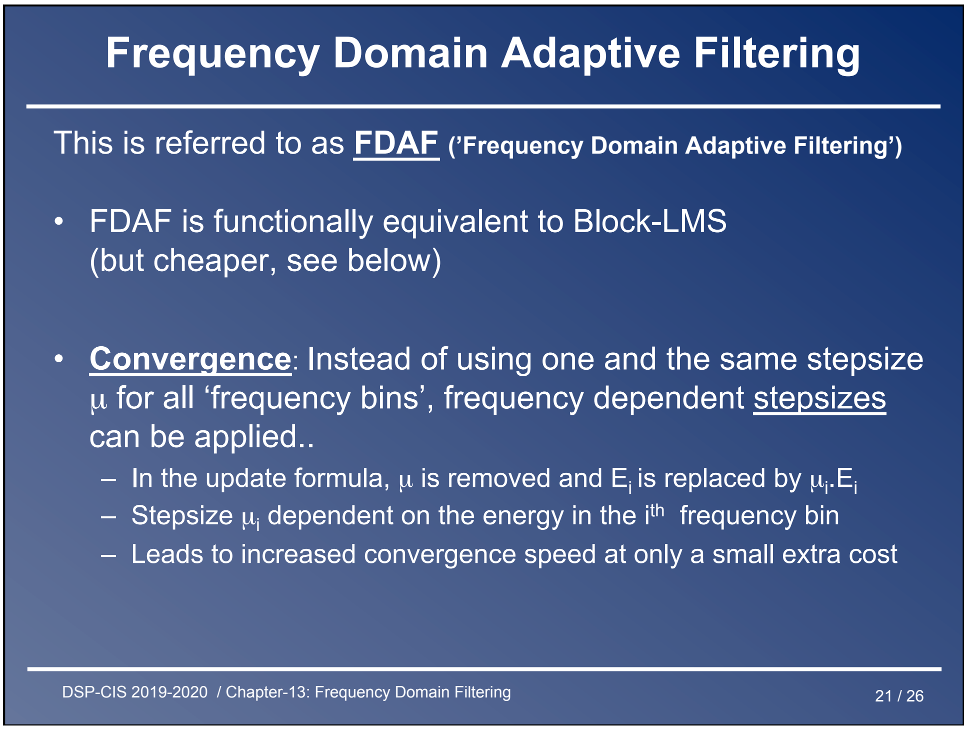
$$\begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \end{bmatrix}_{n+1} = \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \end{bmatrix}_{n} + \mu . \begin{bmatrix} x[4n+4] & x[4n+3] & x[4n+2] & x[4n+1] \\ x[4n+3] & x[4n+2] & x[4n+1] & x[4n] \\ x[4n+2] & x[4n+1] & x[4n] & x[4n-1] \\ x[4n+1] & x[4n] & x[4n-1] & x[4n-2] \end{bmatrix} . \begin{bmatrix} e[4n+4] \\ e[4n+3] \\ e[4n+2] \\ e[4n+1] \end{bmatrix}$$

$$= \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \end{bmatrix}_{n} + \mu . \begin{bmatrix} e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 & 0 & 0 \\ 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 & 0 \\ 0 & 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 \\ 0 & 0 & 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 \end{bmatrix} . \begin{bmatrix} x[4n+4] \\ x[4n+3] \\ x[4n+2] \\ x[4n+1] \\ x[4n] \\ x[4n-1] \\ x[4n-2] \\ x[4n-3] \end{bmatrix}$$

$$\overset{\text{as on p.6}}{=} \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \end{bmatrix}_{n} + \mu . \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} .F^{-1}. \begin{bmatrix} E_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & E_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & E_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & E_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & E_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & E_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & E_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_7 \end{bmatrix} .F. \begin{bmatrix} x[4n+4] \\ x[4n+3] \\ x[4n+2] \\ x[4n+1] \\ x[4n] \\ x[4n-1] \\ x[4n-2] \\ x[4n-3] \end{bmatrix}$$

with E_i=…

=frequency domain correlation

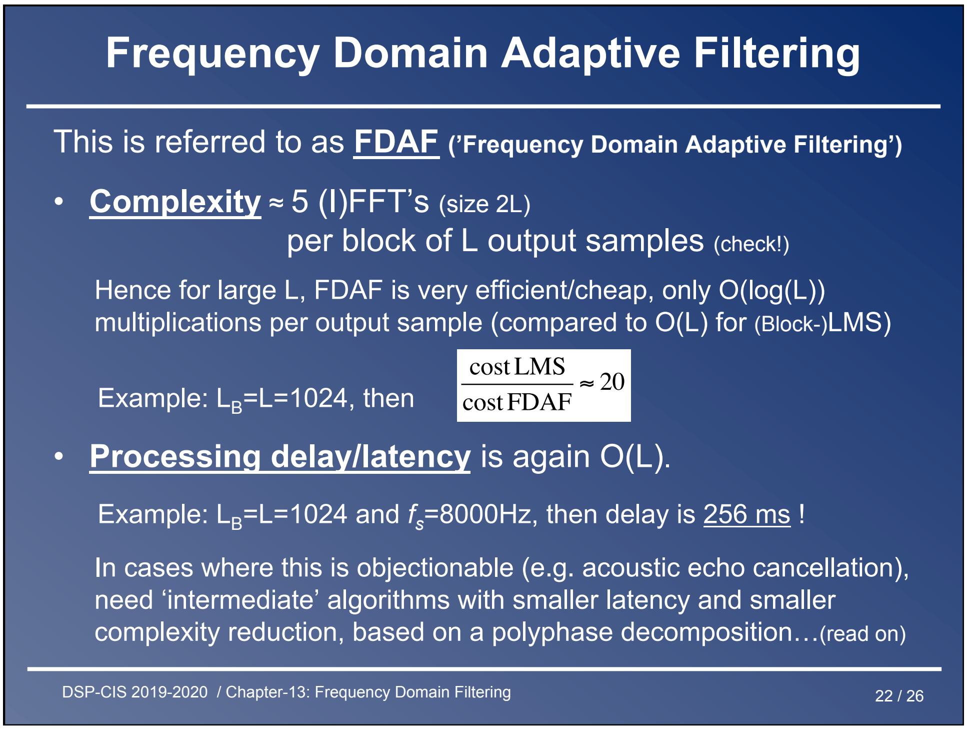


Frequency Domain Adaptive Filtering

This is referred to as **FDAF** **('Frequency Domain Adaptive Filtering')**

- FDAF is functionally equivalent to Block-LMS
  (but cheaper, see below)

- **Convergence**: Instead of using one and the same stepsize μ for all 'frequency bins', frequency dependent stepsizes can be applied..
  – In the update formula, μ is removed and E_i is replaced by $\mu_i.E_i$
  – Stepsize μ_i dependent on the energy in the i^{th} frequency bin
  – Leads to increased convergence speed at only a small extra cost



Frequency Domain Adaptive Filtering

This is referred to as **FDAF** **('Frequency Domain Adaptive Filtering')**

- **Complexity** ≈ 5 (I)FFT's (size 2L)
  per block of L output samples (check!)

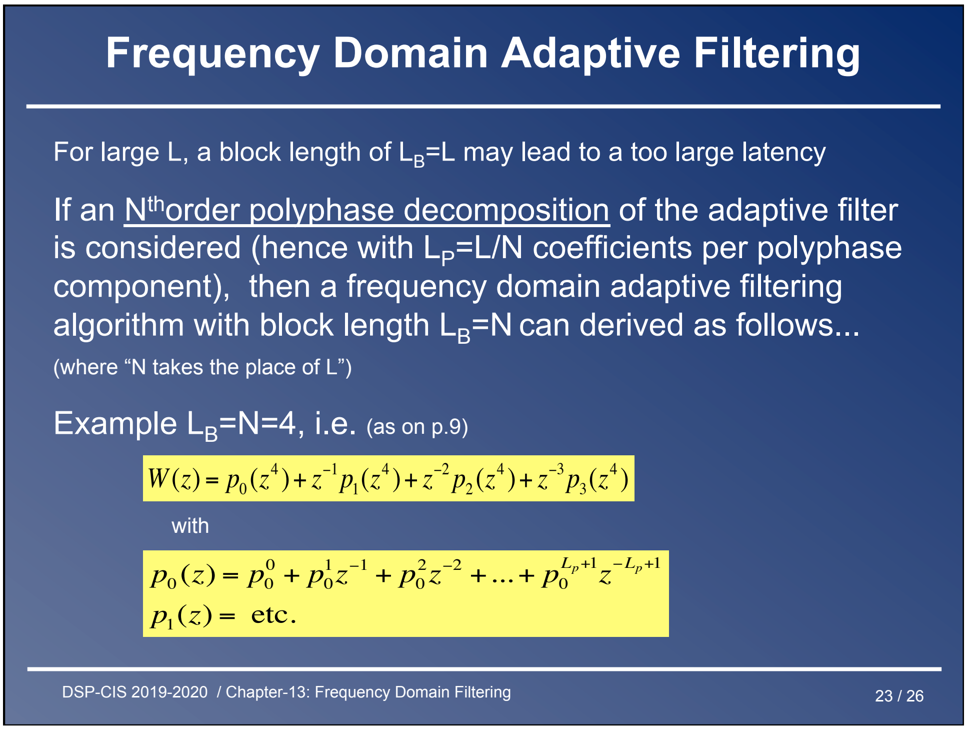
Hence for large L, FDAF is very efficient/cheap, only O(log(L)) multiplications per output sample (compared to O(L) for (Block-)LMS)

Example: L_B=L=1024, then $\frac{\text{cost LMS}}{\text{cost FDAF}} \approx 20$

- **Processing delay/latency** is again O(L).

Example: L_B=L=1024 and f_s=8000Hz, then delay is 256 ms !

In cases where this is objectionable (e.g. acoustic echo cancellation), need 'intermediate' algorithms with smaller latency and smaller complexity reduction, based on a polyphase decomposition…(read on)



Frequency Domain Adaptive Filtering

For large L, a block length of L_B=L may lead to a too large latency

If an N^{th}order polyphase decomposition of the adaptive filter is considered (hence with L_P=L/N coefficients per polyphase component), then a frequency domain adaptive filtering algorithm with block length L_B=N can derived as follows...

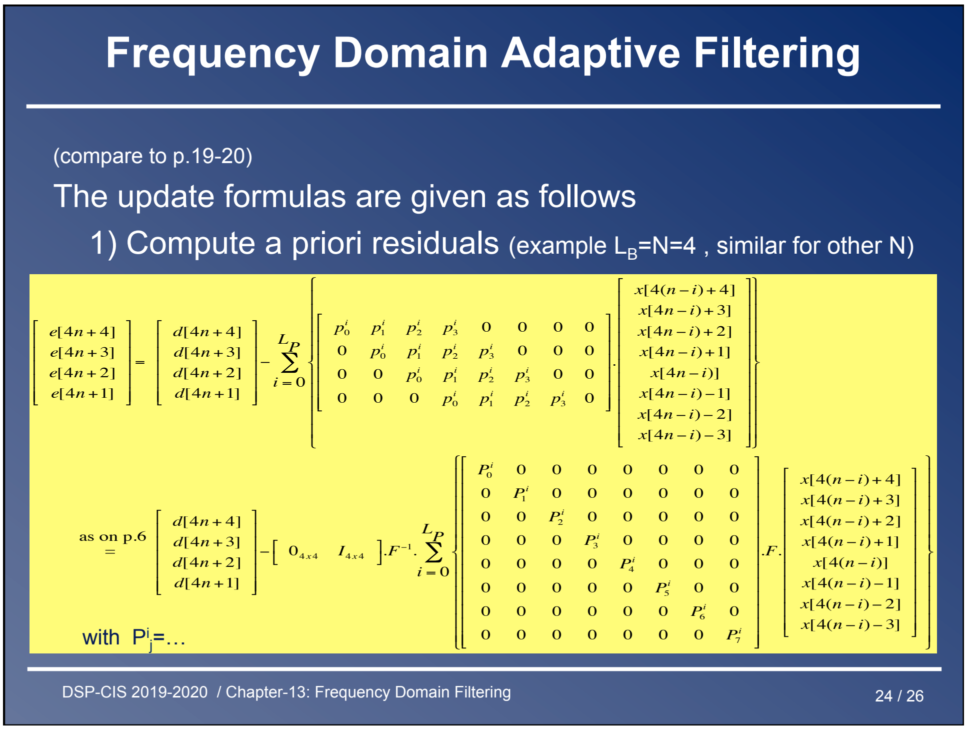
(where “N takes the place of L”)

Example L_B=N=4, i.e. (as on p.9)

$$W(z)=p_0(z^4)+z^{-1}p_1(z^4)+z^{-2}p_2(z^4)+z^{-3}p_3(z^4)$$

with

$$p_0(z)=p_0^0+p_0^1z^{-1}+p_0^2z^{-2}+\ldots+p_0^{L_p+1}z^{-L_p+1}$$
$$p_1(z)=\text{ etc.}$$



Frequency Domain Adaptive Filtering

(compare to p.19-20)

The update formulas are given as follows

1) Compute a priori residuals (example L_B=N=4 , similar for other N)

$$\begin{bmatrix} e[4n+4] \\ e[4n+3] \\ e[4n+2] \\ e[4n+1] \end{bmatrix} = \begin{bmatrix} d[4n+4] \\ d[4n+3] \\ d[4n+2] \\ d[4n+1] \end{bmatrix} - \sum_{i=0}^{L_P} \left\{ \begin{bmatrix} p_0^i & p_1^i & p_2^i & p_3^i & 0 & 0 & 0 & 0 \\ 0 & p_0^i & p_1^i & p_2^i & p_3^i & 0 & 0 & 0 \\ 0 & 0 & p_0^i & p_1^i & p_2^i & p_3^i & 0 & 0 \\ 0 & 0 & 0 & p_0^i & p_1^i & p_2^i & p_3^i & 0 \end{bmatrix} . \begin{bmatrix} x[4(n-i)+4] \\ x[4n-i)+3] \\ x[4n-i)+2] \\ x[4n-i)+1] \\ x[4n-i)] \\ x[4n-i)-1] \\ x[4n-i)-2] \\ x[4n-i)-3] \end{bmatrix} \right\}$$

$$\overset{\text{as on p.6}}{=} \begin{bmatrix} d[4n+4] \\ d[4n+3] \\ d[4n+2] \\ d[4n+1] \end{bmatrix} - \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} . F^{-1} . \sum_{i=0}^{L_P} \left\{ \begin{bmatrix} P_0^i & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & P_1^i & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & P_2^i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & P_3^i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & P_4^i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & P_5^i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P_6^i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_7^i \end{bmatrix} . F . \begin{bmatrix} x[4(n-i)+4] \\ x[4(n-i)+3] \\ x[4(n-i)+2] \\ x[4(n-i)+1] \\ x[4(n-i)] \\ x[4(n-i)-1] \\ x[4(n-i)-2] \\ x[4(n-i)-3] \end{bmatrix} \right\}$$

with P_j^i=…

Frequency Domain Adaptive Filtering

(compare to p.18-19)

The update formulas are given as follows

2) Filter update (example L_B=N=4 , similar for other N)

$$\begin{bmatrix} p_0(z^4) \\ p_1(z^4) \\ p_2(z^4) \\ p_3(z^4) \end{bmatrix}_{n+1} = \begin{bmatrix} p_0(z^4) \\ p_1(z^4) \\ p_2(z^4) \\ p_3(z^4) \end{bmatrix}_{n} + \mu . \left\{ \sum_{i=0}^{L_P} z^{-4i} \begin{bmatrix} x[4(n-i)+4] & x[4(n-i)+3] & x[4(n-i)+2] & x[4(n-i)+1] \\ x[4(n-i)+3] & x[4(n-i)+2] & x[4(n-i)+1] & x[4(n-i)] \\ x[4(n-i)+2] & x[4(n-i)+1] & x[4(n-i)] & x[4(n-i)-1] \\ x[4(n-i)+1] & x[4(n-i)] & x[4(n-i)-1] & x[4(n-i)-2] \end{bmatrix} \right\} . \begin{bmatrix} e[4n+4] \\ e[4n+3] \\ e[4n+2] \\ e[4n+1] \end{bmatrix}$$

$$= \begin{bmatrix} p_0(z^4) \\ p_1(z^4) \\ p_2(z^4) \\ p_3(z^4) \end{bmatrix}_{n} + \mu . \begin{bmatrix} e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 & 0 & 0 \\ 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 & 0 \\ 0 & 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 & 0 \\ 0 & 0 & 0 & e[4n+4] & e[4n+3] & e[4n+2] & e[4n+1] & 0 \end{bmatrix} . \left\{ \sum_{i=0}^{L_P} z^{-4i} \begin{bmatrix} x[4(n-i)+4] \\ x[4(n-i)+3] \\ x[4(n-i)+2] \\ x[4(n-i)+1] \\ x[4(n-i)] \\ x[4(n-i)-1] \\ x[4(n-i)-2] \\ x[4(n-i)-3] \end{bmatrix} \right\}$$

$$\overset{\text{as on p.6}}{=} \begin{bmatrix} p_0(z^4) \\ p_1(z^4) \\ p_2(z^4) \\ p_3(z^4) \end{bmatrix}_{n} + \mu . \begin{bmatrix} 0_{4x4} & I_{4x4} \end{bmatrix} . F^{-1} . \begin{bmatrix} E_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & E_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & E_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & E_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & E_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & E_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & E_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & E_7 \end{bmatrix} . F . \left\{ \sum_{i=0}^{L_P} z^{-4i} \begin{bmatrix} x[4(n-i)+4] \\ x[4(n-i)+3] \\ x[4(n-i)+2] \\ x[4(n-i)+1] \\ x[4(n-i)] \\ x[4(n-i)-1] \\ x[4(n-i)-2] \\ x[4(n-i)-3] \end{bmatrix} \right\}$$

Frequency Domain Adaptive Filtering

This is referred to as **PB-FDAF**

('Partitioned Block Frequency Domain Adaptive Filtering')

- PB-FDAF is functionally equivalent to Block-LMS
- **Complexity** ≈ 3+2.L_P (I)FFT's (size 2N))
  per block of N output samples (check!)

  Example: L=1024, N=128, then $\frac{\text{cost LMS}}{\text{cost PB-FDAF}} \approx 6$
- **Processing delay/latency** is O(N).

  Example: L=1024, N=128 and f_s=8000Hz, then delay is 32 ms
  (used in commercial acoustic echo cancellers)