Digital Filter Structures

Jose Krause Perin

Stanford University

July 25, 2017

Today's lecture

We know that **rational LTI systems** can be either FIR or IIR

FIR

$$H(z) = b_0 + b_1 z^{-1} + \ldots + b_M z^{-M} \qquad (z\text{-transform})$$
$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots + b_M x[n-M] \qquad (\text{difference equation})$$

All poles are at the origin

IIR

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 - a_1 z^{-1} - \ldots - a_N z^{-N}} \qquad (z\text{-transform})$$
$$\begin{aligned} &y[n] - a_1 y[n-1] - \ldots - a_N x[n-N] \\ &= b_0 x[n] + b_1 x[n-1] + \ldots + b_M x[n-M] \end{aligned} \qquad (\text{difference equation})$$

There is at least one pole different from the origin

Careful with conventions

When studying digital filter structures, it is more convenient to express IIR systems according to this convention following convention:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 - a_1 z^{-1} - \ldots - a_N z^{-N}} \tag{1}$$

Note that...

1. the first denominator coefficient is made equal to 1 (i.e., $a_0 = 1$)
2. the coefficients $a_1, \ldots, a_N$ appear with a **minus sign**. This is the same convention from the textbook. This way the coefficients $a_1, \ldots, a_N$ appear without the minus sign in the signal flow graphs.

Today's lecture

Question: how to realize those systems *efficiently*?

- Structures for IIR systems
- Structures for FIR systems
- Pipelining and parallel processing

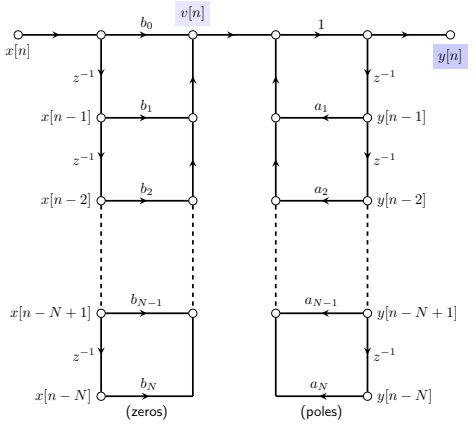
Structure for IIR systems

- Direct form I
- Direct form II
- Cascade form
- Parallel form
- Transposed forms

Ideally, they all produce the same output. However, differences arise when dealing with **finite-precision arithmetic** (next lecture)

Direct form I

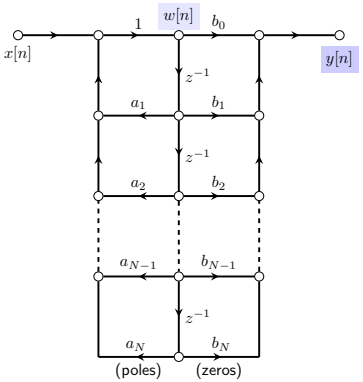
$$v[n] = \sum_{k=0}^{M} b_k x[n-k] \qquad y[n] = \sum_{k=1}^{N} a_k y[n-k] + v[n]$$



Direct form II

Swaps order of poles and zeros. Requires fewer delays z^{-1} (less memory).

$$w[n] = \sum_{k=1}^{N} a_k w[n-k] + x[n] \qquad y[n] = \sum_{k=0}^{M} b_k w[n-k]$$

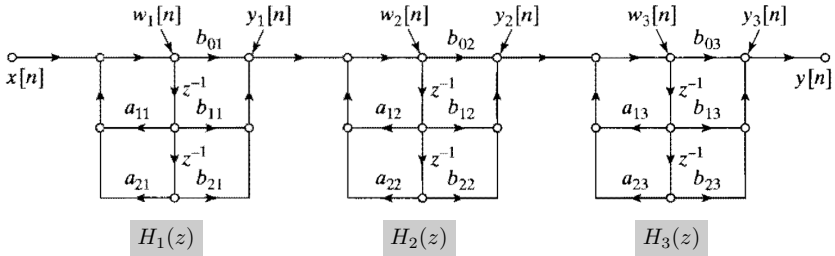


Cascade forms

The overall system transfer function $H(z)$ is factored into 1st or 2nd-order subsystems $\{H_1(z), H_2(z), \ldots, H_K(z)\}$

$$H(z) = H_1(z)H_2(z)\ldots H_K(z)$$

The goal of this factorization is to minimize effects of **finite-precision arithmetic**



This figure shows a 6th-order system factored into three 2nd-order subsystems. Each subsystem is realized according to direct form II.

Parallel forms

Now the factorization is done using **partial fraction expansion** of $H(z)$:

$$H(z) = H_1(z) + H_2(z) + \ldots + H_K(z)$$

The subsystems $H_k(z), k = 1, \ldots, K$ are obtained by grouping second-order factors. These subsystems will either be simple delays or second-order systems:

$$H_k(z) = \begin{cases} C_k z^{-k}, & \text{delay} \\ \dfrac{e_{0k} + e_{1k} z^{-1}}{1 - a_{1k} z^{-1} + a_{2k} z^{-2}}, & \text{2nd-order system} \end{cases}, k = 1, \ldots, K$$

For partial fraction expansion use the Matlab function `residuez`

Transposed forms

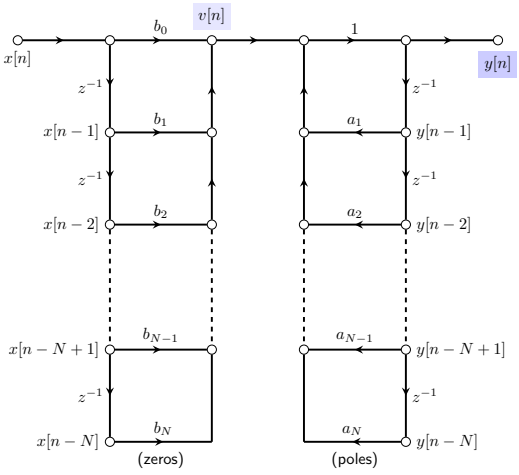
Transposed forms are obtained by performing **flow graph reversal** or **transposition**.
In essence flow graph transposition is done by

1. reversing all branches without changing the gain (transmittance) of each branch
2. reversing the roles of input and output

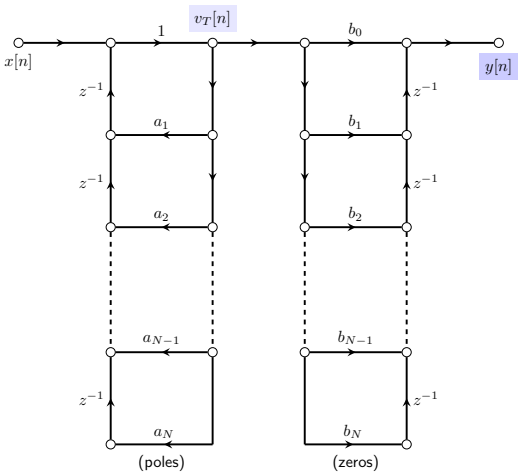
For single-input single-output systems these operations do not change the system function as long as the input and output nodes are interchanged.

Transposed direct form I

Direct form I

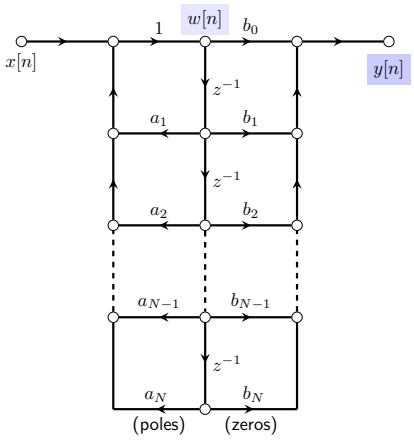


Transposed direct form I

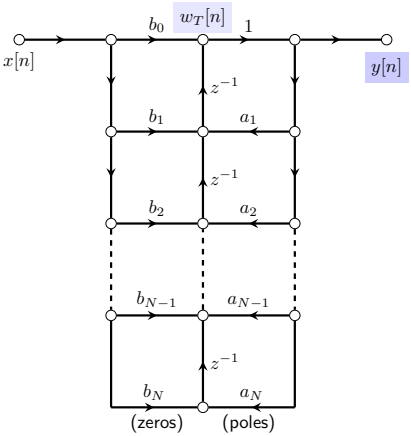


Transposed direct form II

Direct form II



Transposed direct form II

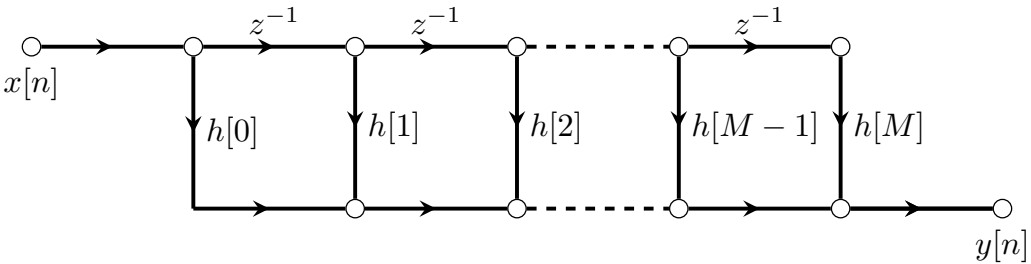


Used in Matlab's `filter` function

Structures for FIR systems

FIR systems do not have the autoregressive component (no feedback).
The **direct form I** simply realizes the convolution sum

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} = \sum_{k=0}^{M} h[k] z^{-k}$$

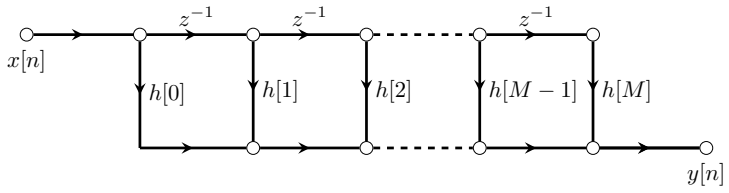


This operation is commonly called **multiply-accumulate (MAC)**

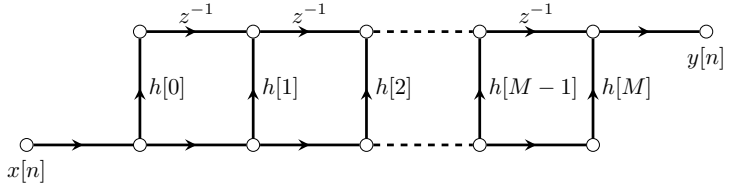
FIR transposed direct form

Similarly to IIR systems, we can apply flow graph transposition to obtain the transposed form

Direct form



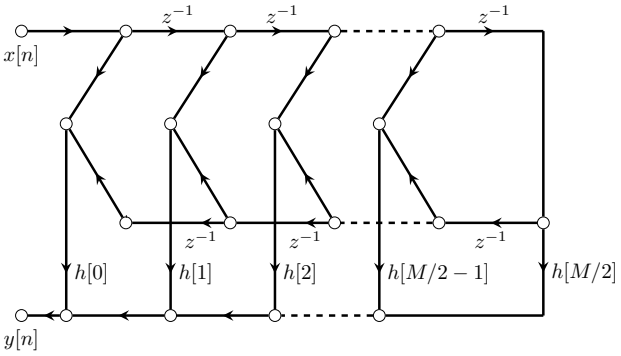
Transposed direct form

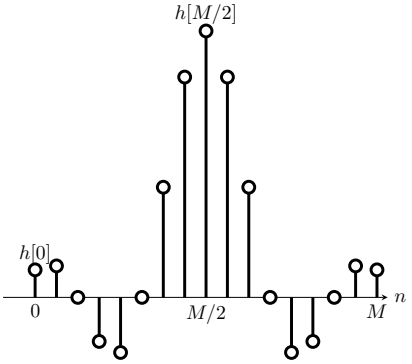


FIR linear phase systems

The impulse response of FIR linear phase systems have either even or odd symmetry
We can halve the number of multiplications by leveraging this symmetry

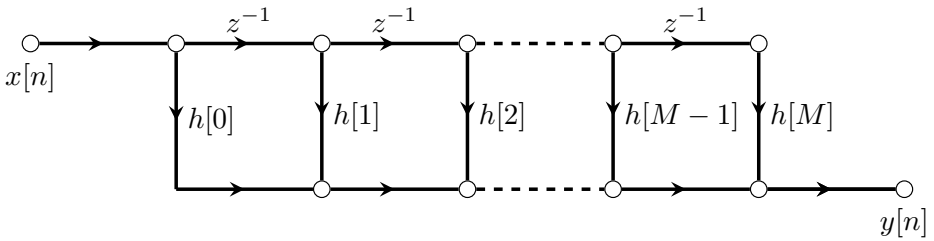
Example: Direct form of a **type I filter** (even symmetry and M even)





Pipelining and parallel processing

Question: how to implement this filter for a signal of rate $1/T = 1$ GHz when the multiplications are performed at $1/T_M = 100$ MHz?



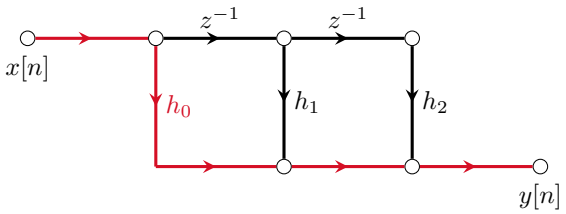
Two solutions:

1. Pipelining
2. Parallel processing

Pipelining and parallel processing solve the problem in two complementary ways. Hence, they can be use together, which is generally done in practice.

Pipelining

Suppose we want to implement this 3-tap FIR filter in real time.



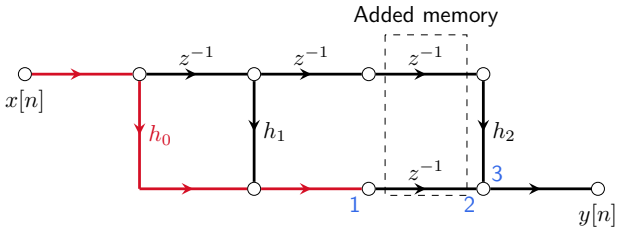
The **critical path** of this filter has one multiplication, which takes T_M seconds, and two additions, which take T_A seconds each
Therefore the sampling period T should satisfy

$$T > T_M + 2T_A,$$

so that all operations in the critical path will have been finished by the time a new input sample comes in.
Question: what can we do if $T < T_M + 2T_A$ i.e., operations are too slow?

Pipelining

Solution: add memory z^{-1}

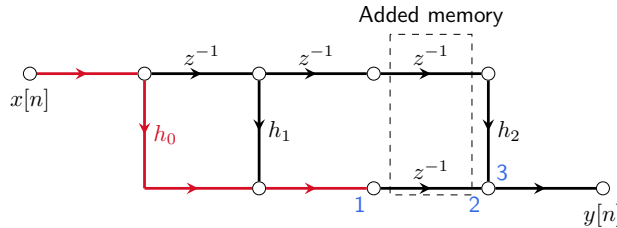


By adding the extra memory the function of the filter is not altered. However, now the **critical path** only has one multiplication and one addition.
Therefore, if

$$T > T_M + T_A,$$

the value at node 1 will be stored in the memory by the time a new input sample comes in. Hence, the first part of the circuit can start *working* on the new sample, while the second part still *works* on the previous sample.

The pipelined FIR filter has the following *schedule* assuming $T > T_M + T_A$



Time	Input	Node 1	Node 2	Node 3	Output
0	$x[0]$	$h_0x[0] + h_1x[-1]$	-	-	-
T	$x[1]$	$h_0x[1] + h_1x[0]$	$h_0x[0] + h_1x[-1]$	$h_2x[-2]$	$y[0]$
2T	$x[2]$	$h_0x[2] + h_1x[1]$	$h_0x[1] + h_1x[0]$	$h_2x[-1]$	$y[1]$
3T	$x[3]$	$h_0x[3] + h_1x[2]$	$h_0x[2] + h_1x[1]$	$h_2x[0]$	$y[2]$

Pipelining increases the **latency**. Note that the outputs are delayed by 1 sample. That is, when the input is $x[1]$, the output will be $y[0]$, which corresponds to the input $x[0]$.

Parallel processing

Processes several inputs at the same time
This difference equation takes one input $x[n]$ and produces one output $y[n]$

$$y[n] = h_0 x[n] + h_1 x[n-1] + h_2 x[n-2]$$

We can rewrite it this way

$$\begin{aligned} y[3k] &= h_0 x[3k] + h_1 x[3k-1] + h_2 x[3k-2] \\ y[3k+1] &= h_0 x[3k+1] + h_1 x[3k] + h_2 x[3k-1] \\ y[3k+2] &= h_0 x[3k+2] + h_1 x[3k+1] + h_2 x[3k] \end{aligned}$$

- Now there are three inputs $\{x[3k], x[3k+1], x[3k+2]\}$ and three outputs $\{y[3k], y[3k+1], y[3k+2]\}$.
- Each difference equation is a FIR filter with the same coefficients as the original filter
- This can be extended to more inputs/outputs.

Pipelining and parallel processing

Additional comments

- Pipelining and parallel processing solve the problem of using a slow hardware to process a fast signal in two complementary ways.
- Pipelining solves the problem by introducing memories and reducing the critical path. These memories are used to store "partial results", so that computations for new input samples can start before all the output has been calculated. The name pipeline comes from the analogy with a water pipe: "continue sending water without waiting for the water in the pipe to come out."
- Pipelining increases the latency and requires more memory
- Parallel processing solves the problem by processing multiple inputs simultaneously
- The effective speed of the hardware is increased by the level of parallelism
- The drawback of parallel processing is that we essentially have to replicate the hardware several times

Summary

- There are different forms of realizing IIR and FIR rational systems
- Their difference becomes evident when considering finite arithmetic precision
- Pipelining and parallel processing solve the problem of using a slow hardware to process a fast signal in two complementary ways.
- Pipelining adds memory (delays) to minimize the critical path. Consequently, pipelining increases latency
- In parallel processing the hardware is replicated to allow processing of multiple input samples simultaneously
- Pipelining and parallel processing can be realized together
- Pipelining and parallel processing are more difficult in IIR systems due to their inherent feedback