Digital Signal Processing

Chapter VI. Implementation of Digital Systems

VI.1. Direct-Form structures

Structures for implementation

- We will see different methods of implementing systems
  - mostly LTI systems
- Differences
  - computational complexity (number of operations)
  - memory requirements = number of unit delays
  - finite-precision effects
  - flexibility
- Block diagrams (structures)
  - can be implemented either in HW or SW

Direct-Form I

- A LTI system is described by the ~~difference~~ system equation

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k]$$
$$= -a_1 y[n-1] - a_2 y[n-2] - \ldots - a_N y[n-N] + b_0 x[n] + b_1 x[n-1] + \ldots + b_M \cdot x[n-M]$$

- **Direct-Form I** structure = directly implementing this equation
- Main disadvantage: too many delay blocks (approx. 2x filter order)

Direct-Form I

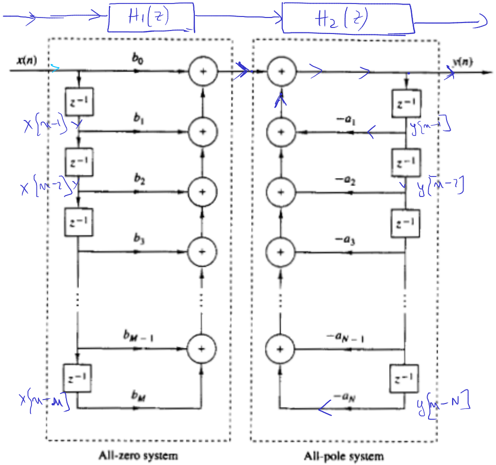


Figure 1: Direct-Form I structure

$$H(z) = \frac{1+3z^{-1}+2z^{-2}}{1-7z^{-1}+9z^{-2}} = \underbrace{(1+3z^{-1}+2z^{-1})}_{H_1(z)} \cdot \underbrace{\frac{1}{1-7z^{-1}+9z^{-2}}}_{H_2(z)}$$

$$\Longleftrightarrow \quad \to \boxed{H_2(z)} \to \boxed{H_1(z)} \to$$

$$y[n] = -a_1 \cdot y[n-1] - a_2 y[n-2] - a_N y[n-N] + b_0 \cdot x[n] + b_1 x[n-1] + \dots + b_M \cdot x[n-M]$$

„Forma directă I"

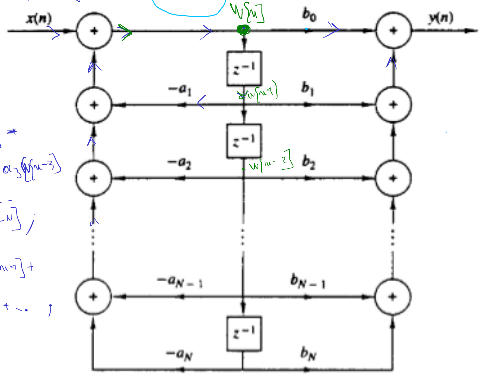
[image from “Digital Signal Processing”, Proakis & Manolakis, 3rd ed.]

Direct-Form II

- Swap the two halves of a Direct-Form I structure
  - (convolution is commutative)
- Advantage: number of delay blocks = filter order
- Is not straightforwardly related to the difference equation
- Known as Direct-Form II or **canonical form**

Direct-Form II

$y[n] = -a_1 \cdot y[n-1] - a_2 y[n-2] - a_N y[n-N] + b_0 \cdot x[n] + b_1 x[n-1] + \dots + b_M \cdot x[n-M]$



for i = 1 : N

$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2] - a_3 w[n-3] - \dots - a_N w[n-N]$;

$y[n] = b_0 \cdot w[n] + b_1 w[n-1] + b_2 w[n-2] + \dots$;

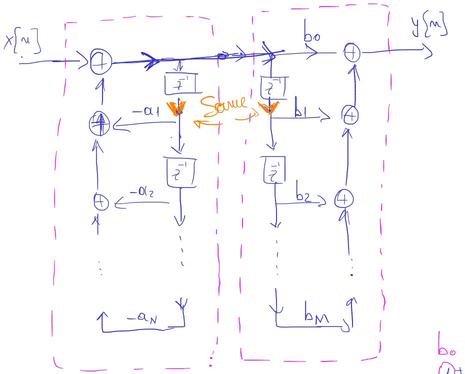
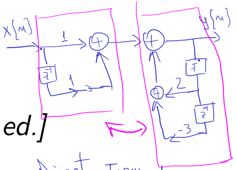


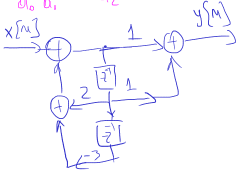
Figure 2: Direct-Form II structure

[image from "Digital Signal Processing", Proakis & Manolakis, 3rd ed.]

Example: $H(z) = \dfrac{1 + 1 \cdot z^{-1}}{1 + 2z^{-1} + 3z^{-2}}$ order = 2



Direct Form 1



Direct-Form 2

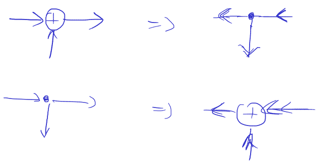
Transposed forms

$\longrightarrow \quad \Rightarrow \quad \longleftarrow$

- **Transposition of a graph** = reverse the direction of all branches, swap input and output
- Theorem: If a structure is transposed, the transfer function stays the same

  same $H(z)$

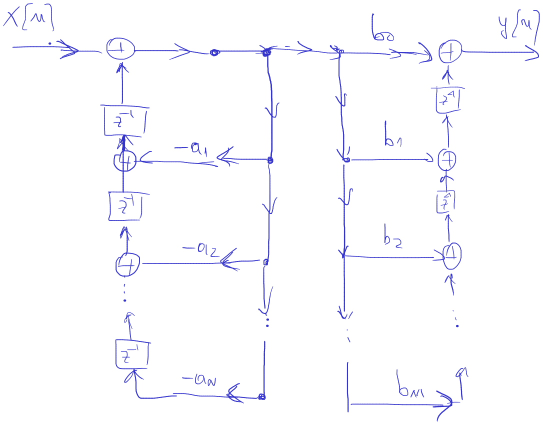
  - summing nodes become branching nodes
  - branching nodes become sum nodes

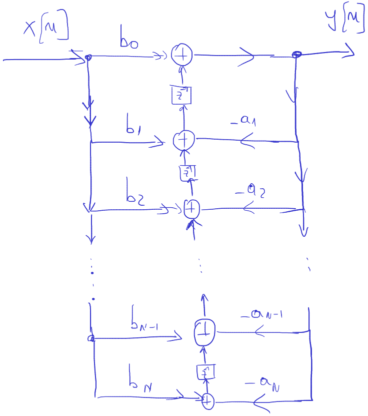


- Direct-Form I and II **Transposed**
  - transpose the form
  - different structures than the originals

Transposed forms

- Draw here: Direct-Form I Transposed, Direct-Form II transposed





FIR systems

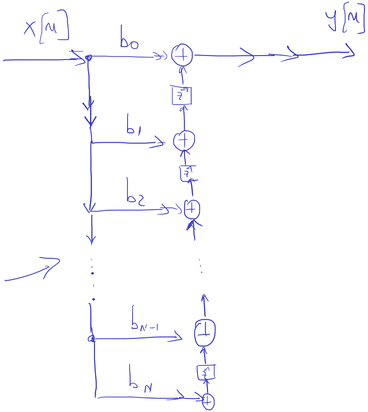
$\boxed{FIR}$

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-N}}{1}$$

$a_1 = 0$

$a_2 = 0$

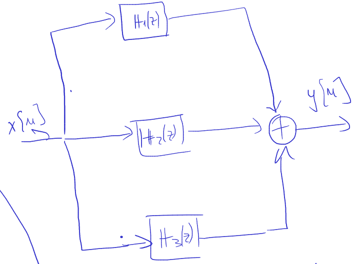
- For FIR systems, $a_i = 0$ so the graphs become simpler
- There is a single Direct-Form, and a single Direct-Form Transposed

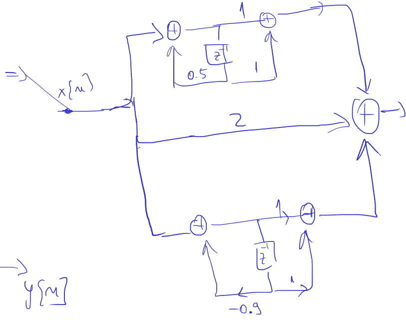


Cascade and parallel implementations

$$H(z) = H_1(z) + H_2(z) + H_3(z) \Rightarrow$$

$$H(z) = \frac{1+z^{-1}}{1-0.5z^{-1}} + 2 + \frac{1+z^{-1}}{1+0.9z^{-1}}$$

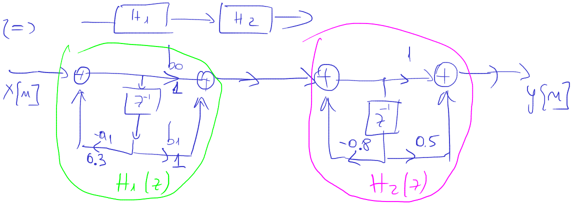




- If a system function $H(z)$ can be written as a **sum** of smaller parts, the system can be implemented in a **parallel structure**
  - implement each smaller part
  - same input, sum the outputs
- If a system function $H(z)$ can be written as a **product** of smaller parts, the system can be implemented in a **cascade structure** (or **series**)
  - implement each smaller part, connect in series
  - order does not matter

$$H(z) = H_1(z) \cdot H_2(z) \Leftrightarrow$$

$$H(z) = \frac{\overbrace{(1+z^{-1})}^{H_1(z)}\overbrace{(1+0.5z^{-1})}^{H_2(z)}}{(1-0.3z^{-1})(1+0.8z^{-1})} =$$



Cascade and parallel implementations

- A system function $H(z)$ can always be written as a sum of **partial fractions**
  - a parallel implementation is always possible
- A system function $H(z)$ can always be written as a product of $\frac{(z-z_k)}{(p-p_k)}$ terms
  - a series implementation is always possible
- To avoid complex-number coefficients, must group conjugate zeros and conjugate poles together
  - resulting in polynomials of degree 2

$H(z) = \_\_ + \_\_ + \_\_ + \_\_$

Second-order sections

- In practice, due to finite-precision calculations, small rounding errors may appear in coefficients or signal values
- The **most robust** implementation to these errors is the **series implementation**
  - using as many terms as possible
  - but always keeping conjugate zeros and conjugate poles together
- **Second-order sections** structure = implementation as a series of small systems of degree at most 2
  - very robust to finite-precision errors

second-order sections (SOS)

$$\underbrace{H(z)}_{\text{order }10} = \underbrace{H_1(z)}_{\text{order }2} \cdot \underbrace{H_2(z)}_{\text{order }2} \cdot \underbrace{H_3(z)}_{\text{order }2} \cdot \underbrace{H_4(z)}_{\text{order }2} \cdot \underbrace{H_5(z)}_{\text{order }2}$$