Lecture 28 DT Filter Implementation

- Given a transfer function $H(z)$, how do we implement a computational systems that performs the filtering.
- Broadly two approaches
  - Frequency Domain implementation using FFT * covered for ex in DSP ECE4624
  - Time Domain implementation * this will be our focus. covered in depth in DSP.

- Recall the Discrete Fourier transform $X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$

  corresponds to the Fourier transform of a finite length signal. It has a fast implementation called FAST Fourier Transform or FFT.

- Consider an FIR filter $h[n] = \sum_{m=0}^{N-1} a_m \delta[n-m] \Leftrightarrow H(z) = \sum_{m=0}^{N-1} a_m z^{-m}$

  ⇓ stable

  $H(e^{j\omega}) = \sum_{m=0}^{N-1} a_m e^{-j\omega m}$

  $= H[k]$ if $\omega = \frac{2\pi k}{N}$ sample ω axis.

  - Let $H_N[k]$ be the N-point FFT of $h[n]$
  - Suppose $X_N[k]$ is the N-point FFT of an input of finite length N.
  - Then $Y_N[k] = H_N[k] \cdot X_N[k]$ is FFT of output.
  - Taking inverse FFT (IFFT) we get the output $y_N[n]$
  - This gives an approach to implement FIR filter given a finite length input of same N.
    1. Compute $H_N[k]$ by taking FFT of $h[n]$ or sampling $H(e^{j\omega})$
    2. Compute $X_N[k]$ using FFT
    3. Compute $Y_N[k] = H_N[k] X_N[k]$
    4. Compute output $y_N[n] = \text{IFFT}\{Y_N[k]\}$

- Some issues:
  - Cannot be done in real time, but can in "pseudo" or "soft" real-time.
  - what about $x[n]$ too large to fit into memory? < overlap save / overlap add — NOT covered see DSP.
  - Quantization if no Floating Point

- So our primary focus will be on time domain methods.

Consider a general transfer function $H(z) = \dfrac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}$ in delay form

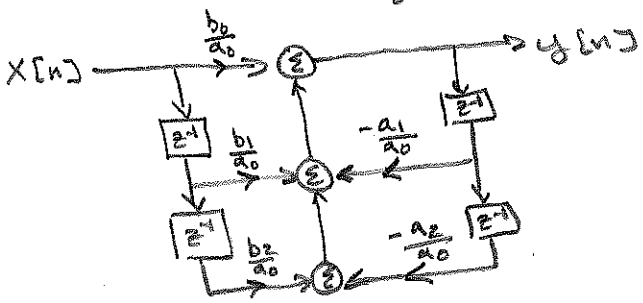
This corresponds to the recursive LCCDE

$$y[n] = -\sum_{k=1}^{N} \frac{a_k}{a_0} y[n-k] + \sum_{k=0}^{M} \frac{b_k}{a_0} x[n-k]$$

- Example $M=2$, $N=2$

$$y[n] = -\frac{a_1}{a_0} y[n-1] - \frac{a_2}{a_0} y[n-2] + \frac{b_0}{a_0} x[n] + \frac{b_1}{a_0} x[n-1] + \frac{b_2}{a_0} x[n-2]$$

This has BD



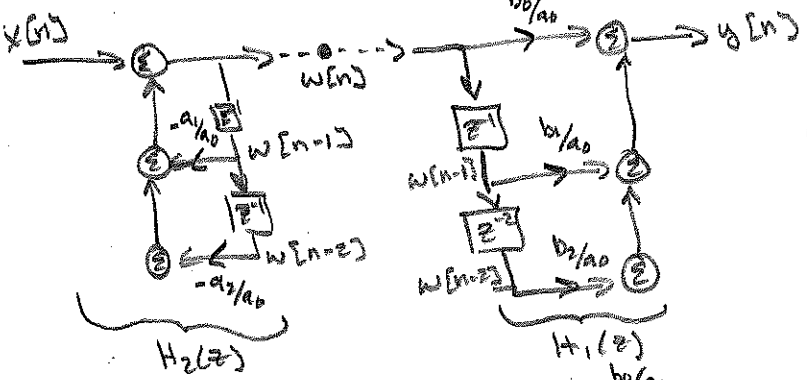
called the Direct Form I realization.

- requires $M+N$ memory locations and $M+N-1$ multiplies and $M+N$ adds.

We note the transfer function and corresponding BD can be split into

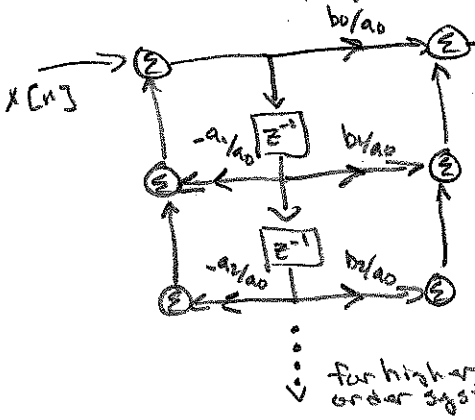
$$H(z) = \underbrace{\left[\sum_{k=0}^{M} b_k z^{-k}\right]}_{H_1(z)} \underbrace{\left[\sum_{k=0}^{N} a_k z^{-k}\right]^{-1}}_{H_2(z)} = \underbrace{\left[\sum_{k=0}^{N} a_k z^{-k}\right]^{-1}}_{H_2(z)} \underbrace{\left[\sum_{k=0}^{M} b_k z^{-k}\right]}_{H_1(z)}$$

or as BD



Note the duplicate memory blocks.

removing the duplication

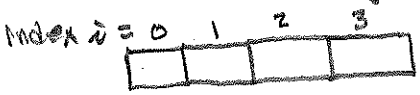


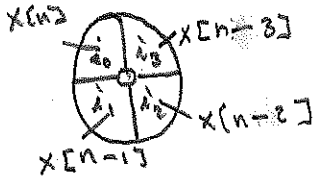
- typically reduces memory required by $\approx 1/2$.

Called Direct Form II

- requires $\max(M,N)$ memory locations $M+N+1$ multiplies $M+N$ adds.

- If you have floating point hardware then Direct Form II is an easy way to implement a Filter.
  - Data Structure required
    - b – array for b_k of length $M+1$ } pre processed so that $a_0 = 1$.
    - a – array for a_k of length N }
    - d – array for delay outputs of length $\max(N, M) + 1$ as ring buffer
  - treat the delay array as a ring buffer. e.g. $\max(N,M) = 3$





for $n = 0, 1, 2, \ldots$

$i_3 = n \% 4$
$i_2 = (n+1) \% 4$
$i_1 = (n+2) \% 4$
$i_0 = (n+3) \% 4$

$n=0,\ i_3=0,\ i_2=1,\ i_1=2,\ i_0=3$

  - Algorithm

```
while (true) {
    x = read()
    d[i0] = x + a[1] d[i1] - a[2] d[i2] - a[3] d[i3]
    y = b[0] d[i0] + b[1] d[i1] + b[2] d[i2] + b[3] d[i3]
    write(y)
}
```

- Example: Consider a second order lowpass butterworth filter with sample frequency $f_s = 10\text{kHz}$ and cutoff frequency of 1kHz.
  - The folding frequency is 5000
  - using matlab `[b, a] = butter(2, 1000/5000)`

    gives $b = [0.1411, 0.2823, 0.1411]$

    $a = [1, -0.6934, 0.2579]$

* Show code in C *

This is how you will implement HW #3 (NOW BONUS)

- It is common to not have floating point hardware.
  - power hungry
  - takes a lot of silicon to implement, particularly IEEE compliant.

• requires we quantize the filter to use integer math.

- b_k, a_k, and signals $x \in \mathbb{R}$ (floats) ⟶ Fixed-point representation
- multiplies & adds become integers mult/add.

• Converts our nice linear system into a non-linear system.
• Direct Form II generally no longer suitable because of quantization effects.

- Fixed point representation of numbers. In binary

$$x = (b_{-A}, b_{-A+1} \cdots b_{-1}, b_0, b_1, b_2 \cdots b_B)_2 \qquad b_i = \begin{cases} 0 \\ 1 \end{cases}$$

b_{-A} is most significant bit (MSB), b_B is least significant (LSB) $x = \sum_{i=-A}^{B} b_i 2^{-i}$

the location of the decimal point is implied an unimportant to hardware.

e.g. $x = (11.01)_2 = 1\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2} = (3.25)_{10}$

called a A.B format.

- unsigned N-bit integers can store positive integers $[0, 2^{N-1}]$

where $-A = N-1$, $B = 0$ i.e. 8 bit (uint8_t in C) $[0, 255]$

16 bit (uint16_t) $[0, 65535]$

- To represent positive and negative integers most ALUs use 2's complement.

where $x<0$ $(1\ \bar{b}_1\ \bar{b}_2 \cdots \bar{b}_B)$ because addition = subtraction and $\sum$ result that does not overflow can have terms that do.

$+ \quad 0\ 0\ 0 \cdots 1$

modulo 2 ignore carry.

- e.g. 8 bit signed (int8_t) $x \in [-128, 127]$

16 bit signed (int16_t) $x \in [-32768, 32767]$

to convert from float to fixed point: Let $Q = 1 << \#\text{fractional bits}$ and round floatvalue * Q.

- Quantization can be tricky to get correct, and introduces
  - possible instability
  - quantization noise + errors.
  - possibility of overflow/underflow
- Different Filter structures are less sensitive to these than Direct Form II.
  - Cascade of second order stages. (SOS)