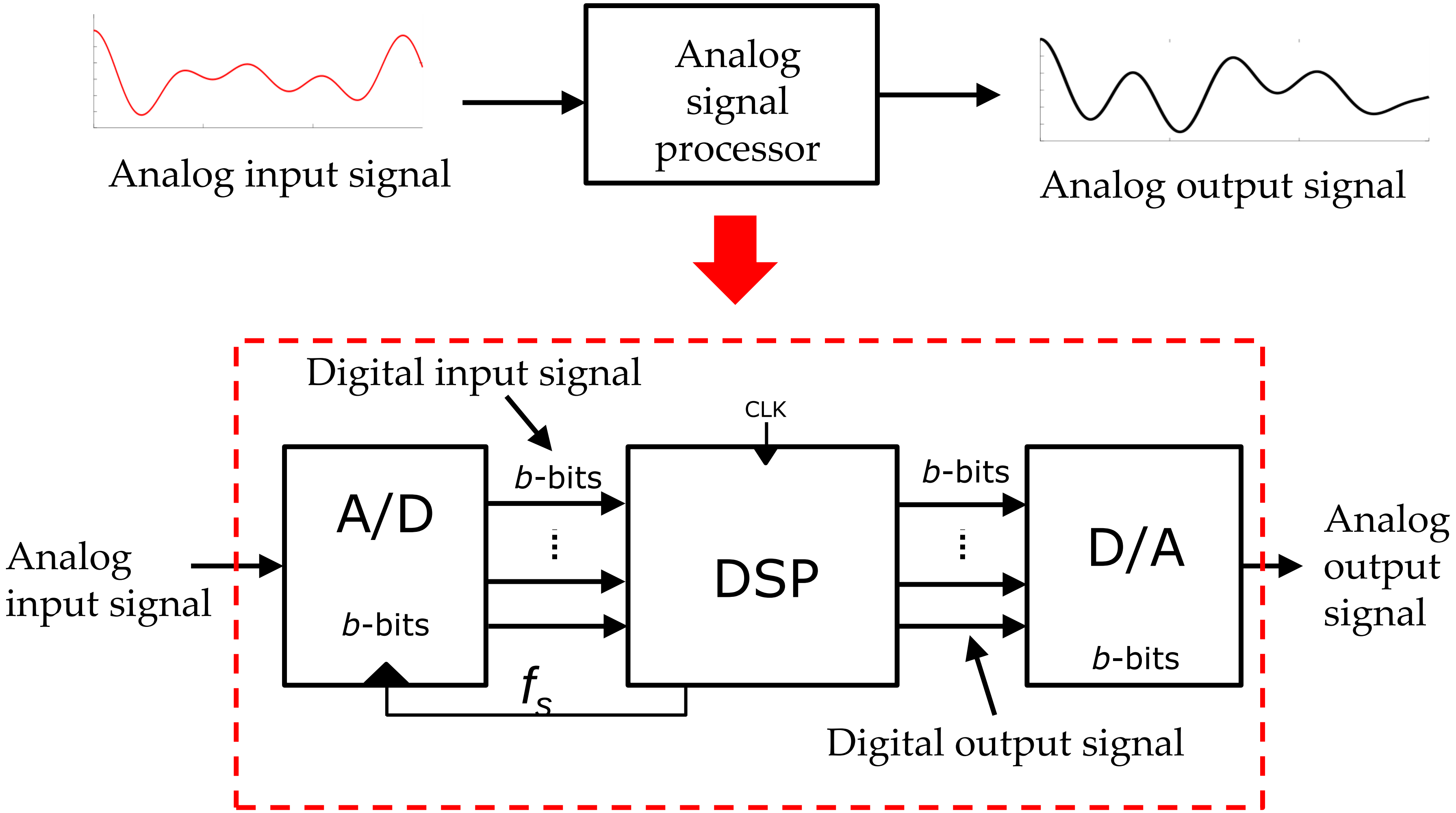
EE401 - Digital Signal Processing (Digital Filters and DFT)

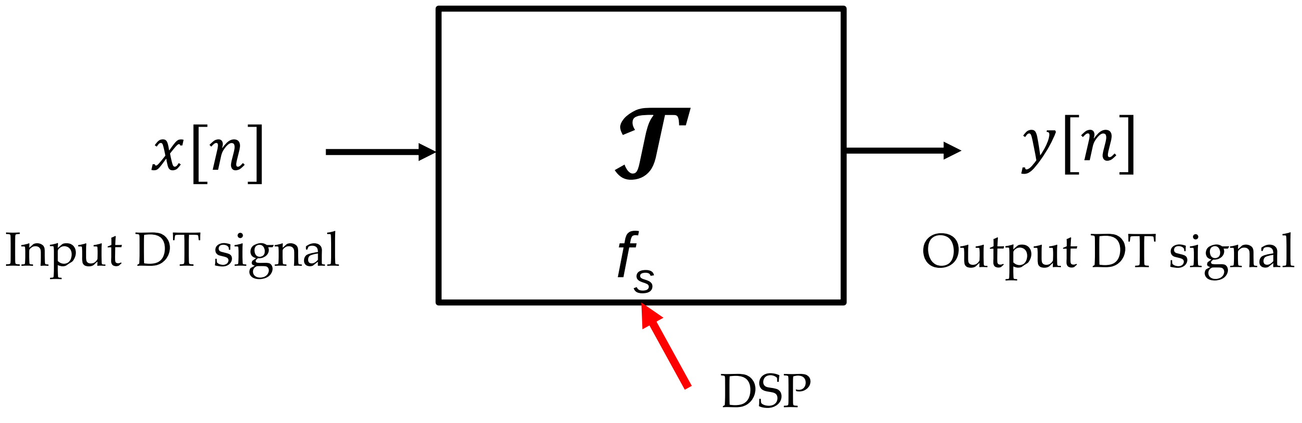
Acknowledgment
The notes are adapted from those given by
Dr. Dushyantha Basnayaka

DSP Systems



Analog input signal
Analog signal processor
Analog output signal
Digital input signal
CLK
A/D
b-bits
b-bits
DSP
b-bits
D/A
b-bits
f_s
Analog input signal
Analog output signal
Digital output signal

Simplified DSP Model



- This is the simplified model of a DSP system.
- The DSP denoted by $\mathcal{T}$ gets an input DT signal and produces an output DT signal.
- $y[n] = \mathcal{T}(x[n])$ or $x[n] \xrightarrow{\mathcal{T}} y[n]$

There is always a sampling frequency f_s associated with a DSP.

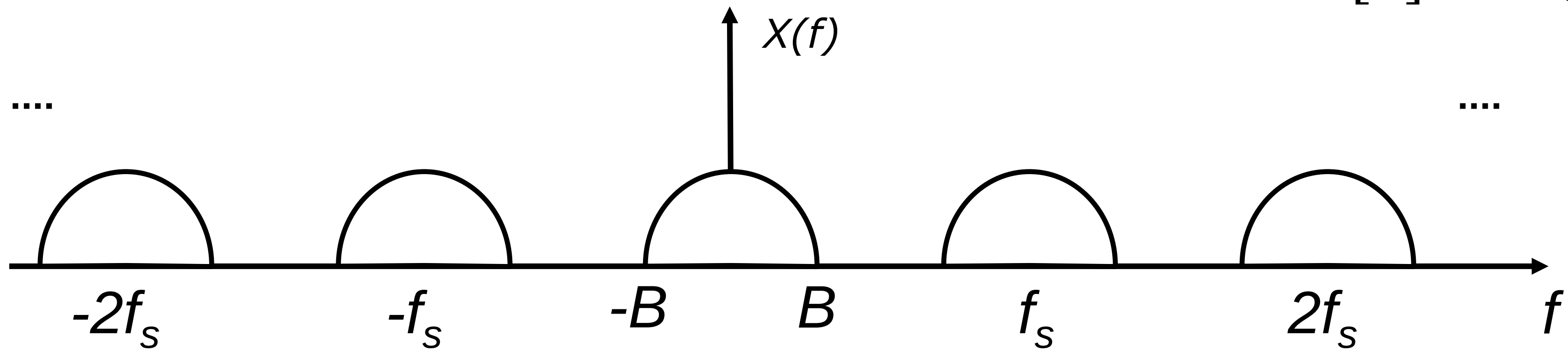
Discrete-Time Fourier Transform

The DTFT of a DT signal, $x[n] = x_a(nT_s)$, can be obtained as:

$$X(f) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X_a(f - kf_s) = \sum_{k=-\infty}^{\infty} x[n]e^{-j2\pi nT_s f}$$

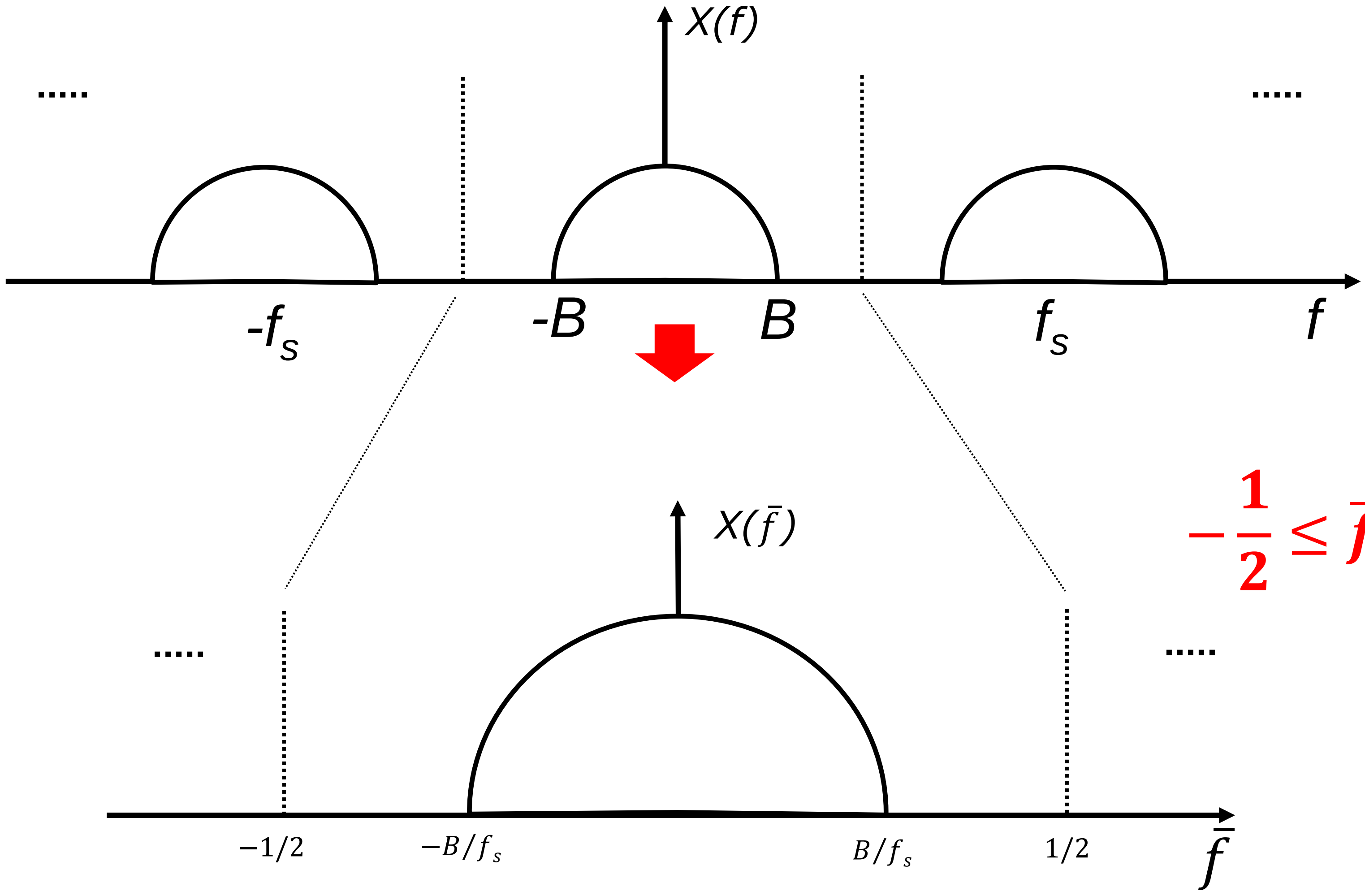
We also use this shorthand notation:

$$x[n] \overset{\mathcal{F}}{\Leftrightarrow} X(f)$$



Normalized Frequency

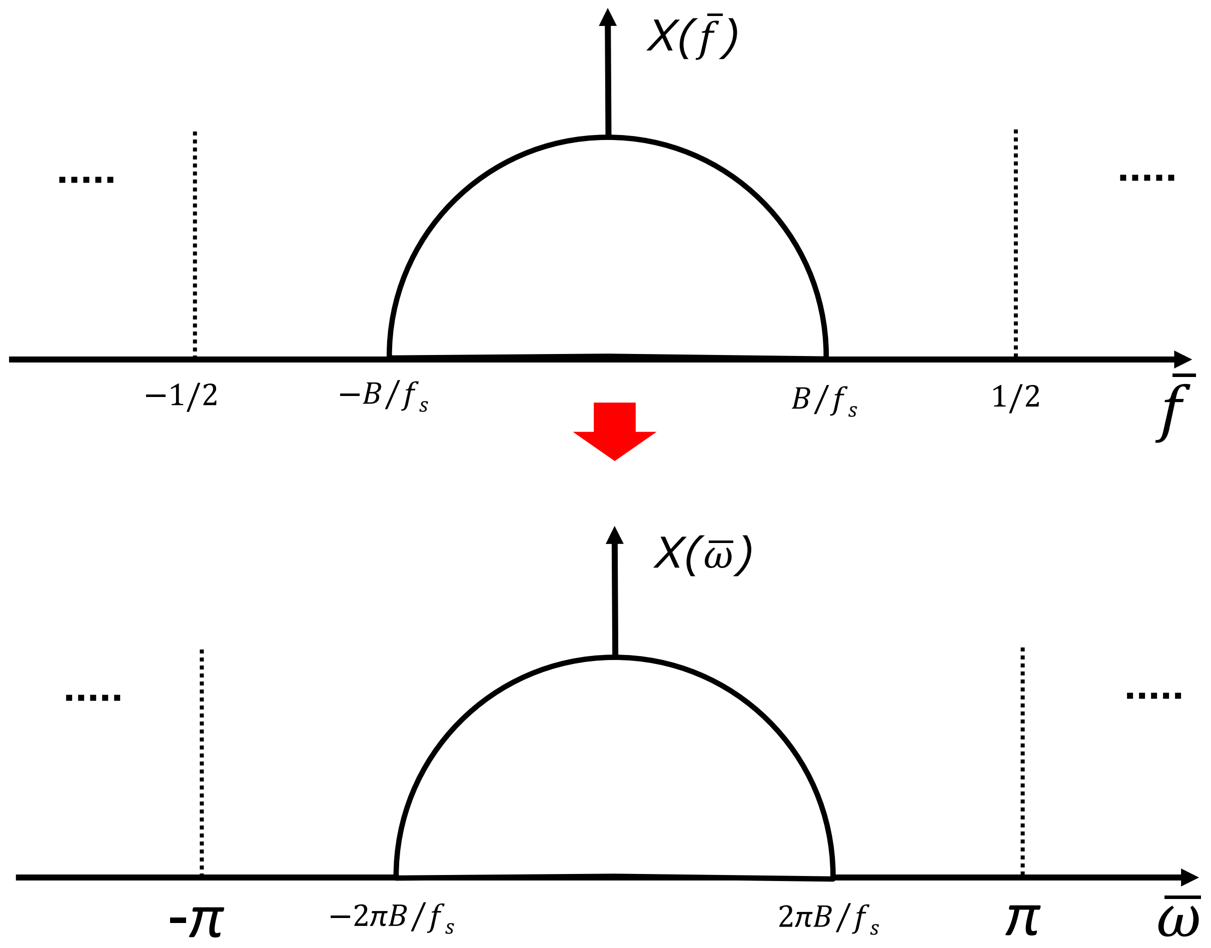
Normalised frequency $\bar{f} = \frac{f}{f_s}$



$$-\frac{1}{2} \leq \bar{f} \leq \frac{1}{2}$$

Normalized Frequency

or $\bar{\omega} = 2\pi\bar{f}$



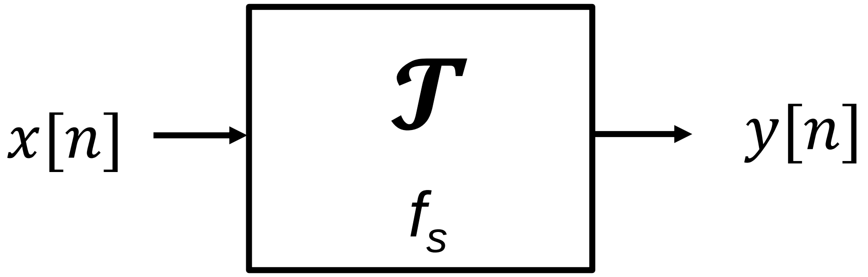
$$-\pi \leq \bar{\omega} \leq \pi$$

Note that the above identities are true because $f_s > 2f_{max} = 2B$

$$x[n] \overset{\mathcal{F}}{\Leftrightarrow} X(\omega)$$

If $\bar{\omega} = \bar{\omega}_0$, then the actual corresponding frequency in hertz is $f_0 = \frac{\bar{\omega}_0 f_s}{2\pi}$.

Digital Filters



- **Memory-less DSP systems:**

$$y[n] = x[n]^2$$

$$y[n] = ax[n] + b$$

- **DSP systems with memory:**

$$y[n] = 2x[n] + x[n-1] + 1.34x[n-2]$$

$$y[n] = y[n-1] + x[n] + x[n-1]$$

Digital Filters

In general, digital filters studied in this course have the following form:

$$y[n] = \underbrace{\sum_{k=1}^{N} a_k y[n-k]}_{\text{Feedback part}} + \underbrace{\sum_{k=0}^{M} b_k x[n-k]}_{\text{Feedforward part}}$$

- **Example**

$$y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

- This is known as a linear constant-coefficient difference equation.

Dublin City University Ollscoil Chathair Bhaile Átha Cliath

Digital Filters

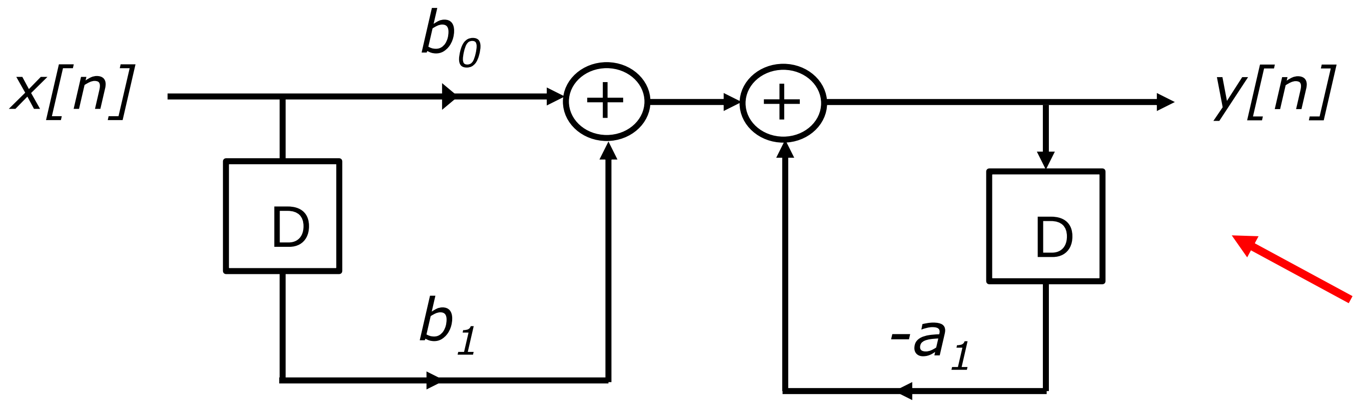
In general, digital filters studied in this course have the following form:

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

Feedback part

Feedforward part

$$y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

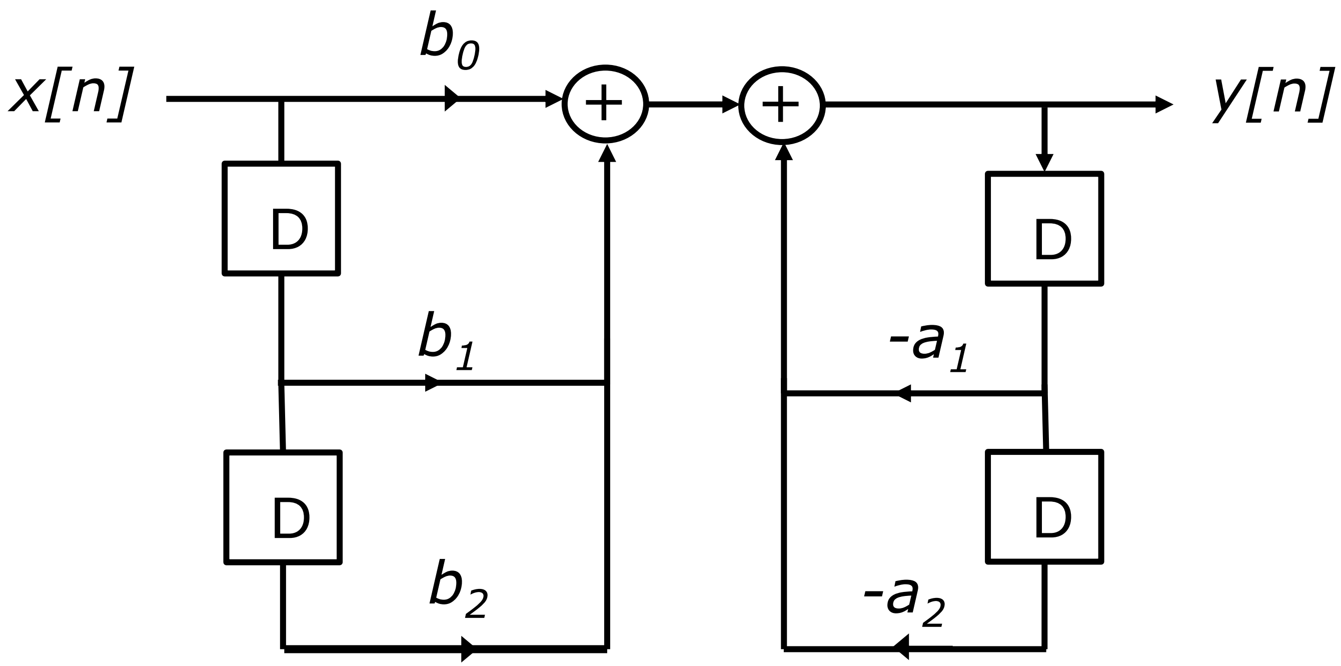


This is known as a block diagram realization of a DSP filter.

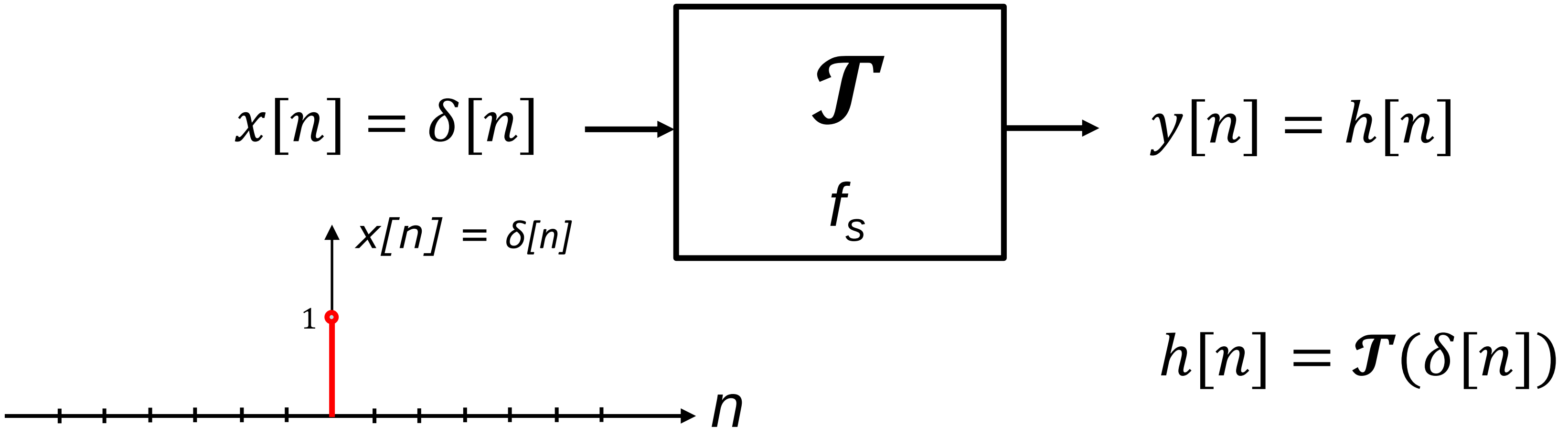
Digital Filters

Consider the following example

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2]$$



Unit Sample Response



Example

$$y[n] = x[n] + 0.5x[n-1] + 0.25x[n-2]$$

Find the unit sample response of the filter:

$$h(n) = \{1, 0.5, 0.25, 0, 0, 0, \ldots\}$$

LTI Systems

Time Invariant Systems: a system is classified as time invariant if and only if

$$y_o[n] = \mathcal{T}(x_o[n])$$

implies that

$$y_o[n-k] = \mathcal{T}(x_o[n-k])$$

E.g.: $y[n] = nx[n]$ is a time variant system.

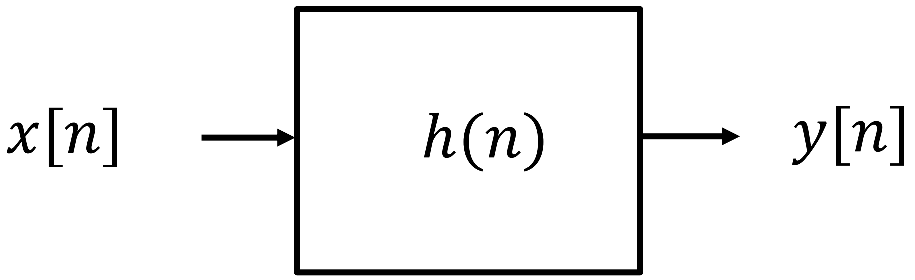
Linear Systems: a system is said to be linear if and only if:

$$\mathcal{T}(a_1 x_1[n] + a_2 x_2[n]) = a_1 \mathcal{T}(x_1[n]) + a_2 \mathcal{T}(x_2[n])$$

E.g.: $y[n] = x[n]^2$ is a non-linear system.

LTI Systems

How to find the output of an arbitrary LTI system to an arbitrary input?



The output, $y[n]$, of a linear time invariant (LTI) system can be obtained in terms of $x[n]$ and $h[n]$ as:

Convolution Sum: $\boldsymbol{y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = x[n] * h[n]}$

where $h[n] = \mathcal{T}(\delta[n])$. This equation is known as the convolution sum (or convolution) and it is valid only for LTI systems.

LTI Systems

Property of the convolution sum:

$$x[n] \longrightarrow \boxed{h[n]} \longrightarrow y[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \;----(A)$$

$$= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \;----(B)$$

Proof: By defining a new index $m = n - k$, one can get (B) from (A). You also need to change the dummy variable m back again to k to get (B) in the form above.

Digital Filter Types

There are two main types of digital filter:

a) FIR (finite-impulse-response) filters
b) IIR (infinite-impulse-response) filters

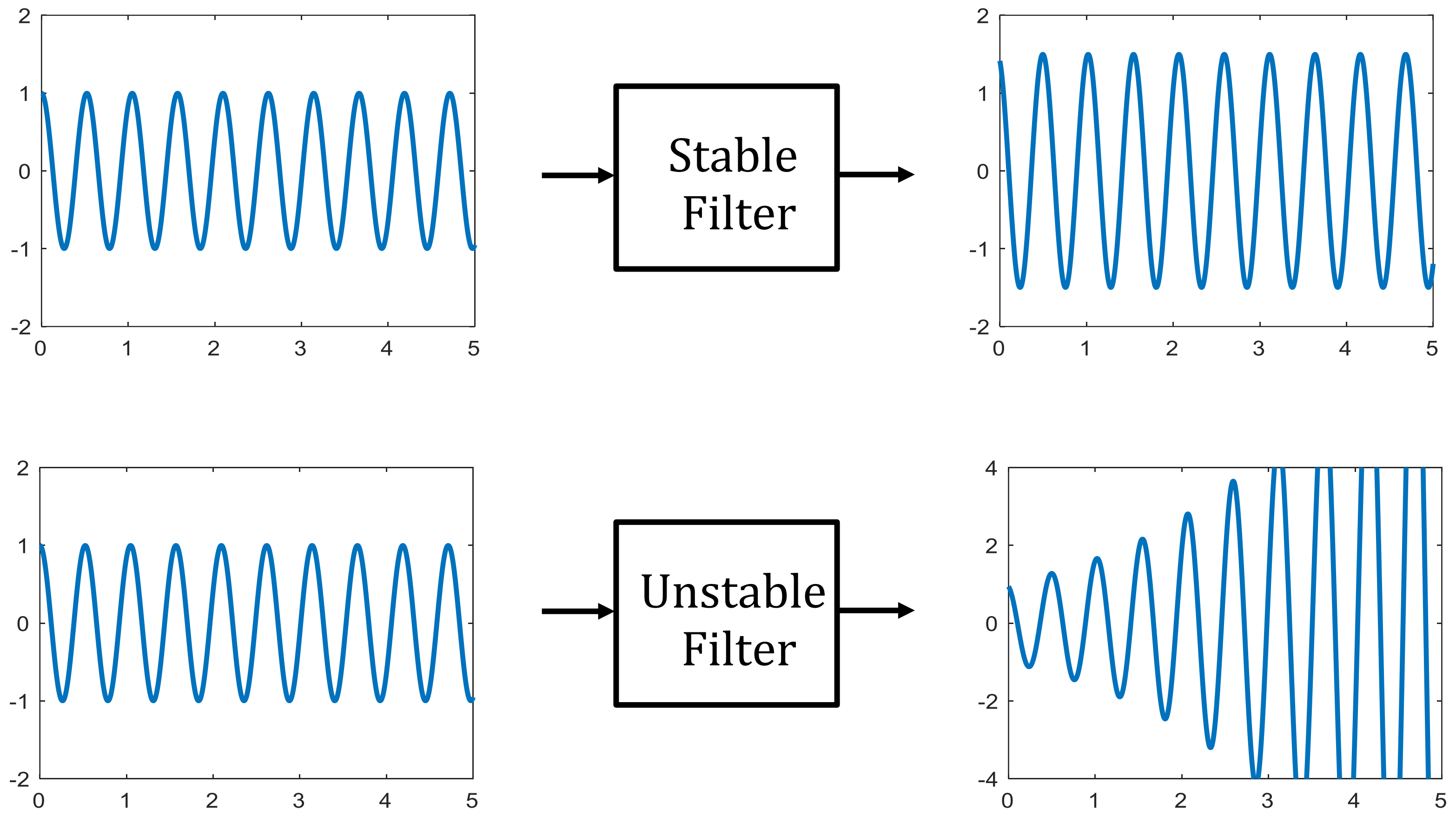
FIR Filters: The unit sample response (or impulse response) $h[n]$ has a finite duration. They do not have any feedback part. For instance:

$$\textbf{FIR:} \qquad y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

IIR Filters: The unit sample response (or impulse response) $h[n]$ has an infinite duration. They have a feedback path. For instance:

$$\textbf{IIR:} \quad y[n] = -a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$$

Stability of Digital Filters

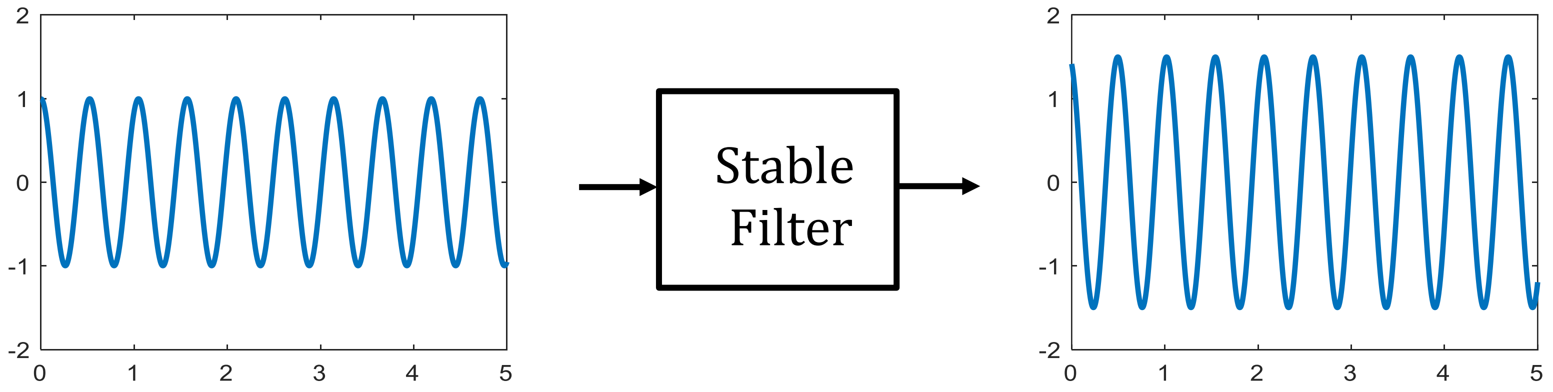


Stability of Digital Filters

A digital system is classified to be stable if and only if every bounded input produces a bounded output. Note that systems that produce an unbounded output for an unbounded input may or may not be unstable.

How to check mathematically if a sequence is bounded or not? For instance $x[n]$:

$$|x[n]| \leq M_x \qquad \text{for all} \quad n$$



Dublin City University Ollscoil Chathair Bhaile Átha Cliath

Stability of Digital Filters

How to check the stability of a digital filter at the design stage? There are many methods.

Impulse Response Method: Firstly, obtain the impulse response $h[n]$. If the impulse response is absolutely summable, the corresponding filter is considered to be stable.

$h[n]$ is absolutely summable if and only if:

$$\sum_{n=-\infty}^{\infty} |h[n]| \leq M_h < \infty$$

Since h[n] of FIR filters has a finite duration, **FIR filters are always stable**. The IIR filters are not always stable and should be carefully designed to be stable.

Digital Filters in MATLAB

How to implement the following first order recursive filter in MATLAB?

$$y[n] = 0.95y[n-1] + 0.23x[n]$$

where $y[-1] = c = 1$

The next page includes a program to simulate this digital filter, where it has been assumed that:

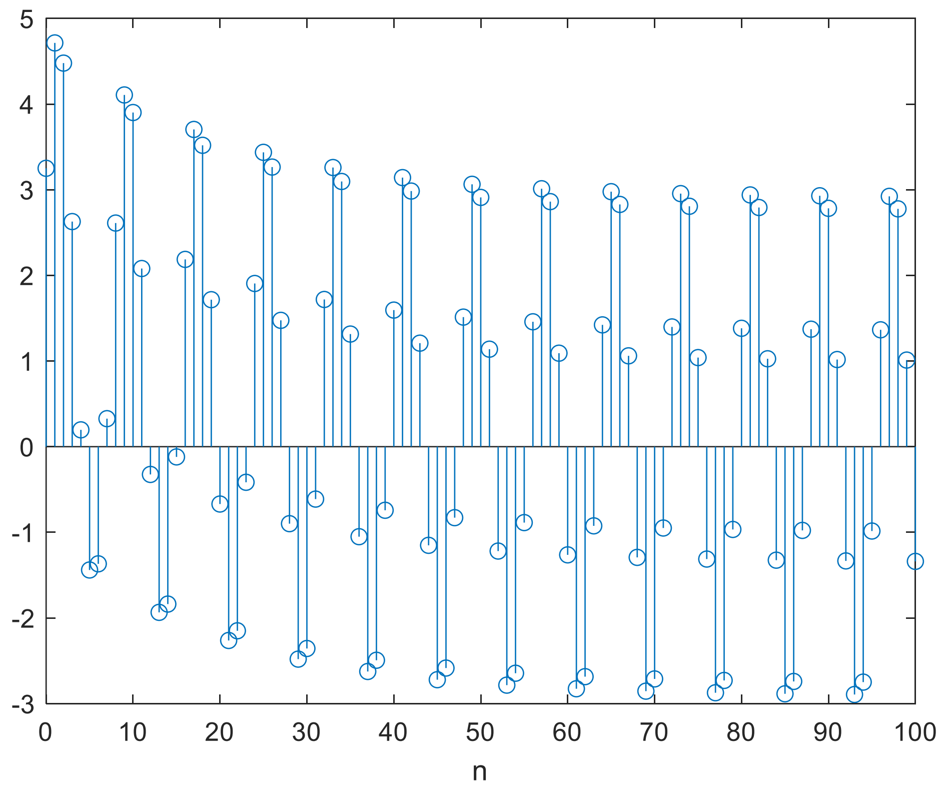
$$x[n] = 10\cos\left(\frac{\pi n}{4}\right)$$

and the filter is also not relaxed.
A relaxed filter is when the initial conditions are zero.

Digital Filters in MATLAB

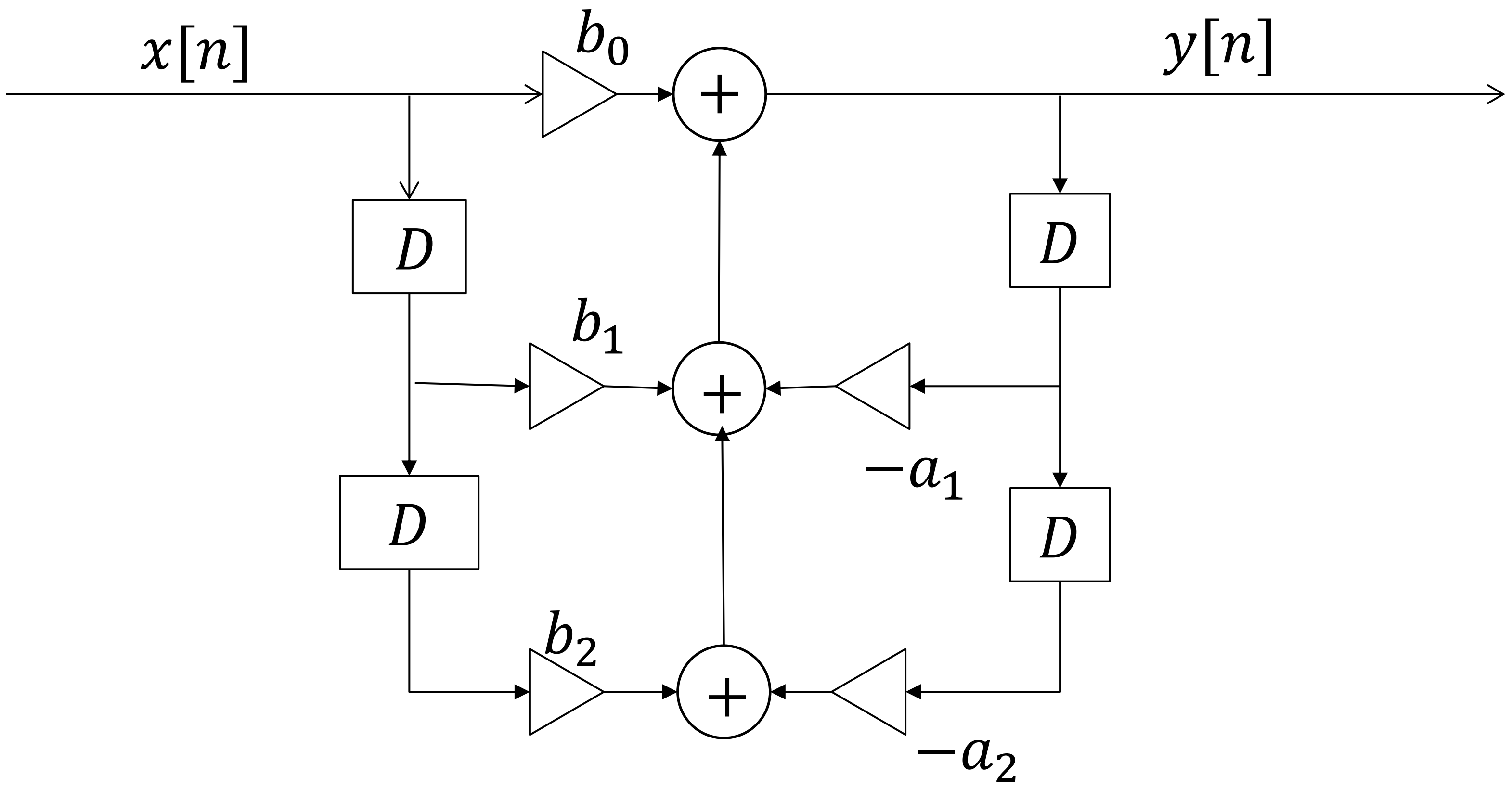
```
clear all
A=10;
c=1;  %initial output
a=0.95; %filter coefficient
b=0.23; %filter coefficient
n=[0:100];
xn=A*cos((pi/4*n));
y(1)=a*c+b*xn(1);  %y0  n=0;
y(2)=a*y(1)+b*xn(2); %y1 n=1
for ii=3:length(n)
    y(ii)=a*y(ii-1)+b*xn(ii);  %y2-yn
end

stem(n,y)
xlabel('n')
```

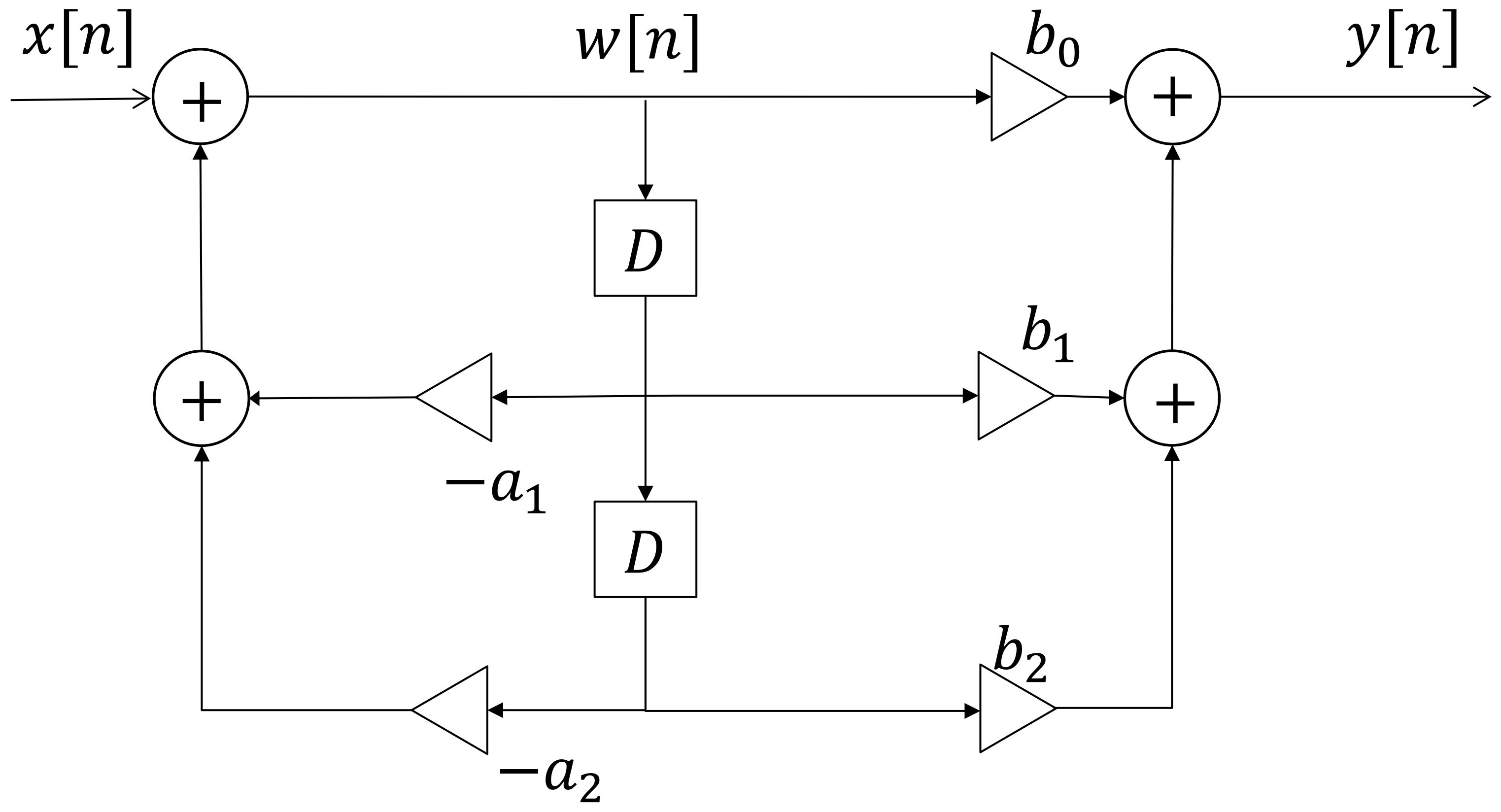

5
4
3
2
1
0
-1
-2
-3
0
10
20
30
40
50
60
70
80
90
100
n

Different implementations

Direct Form I filter



Direct Form II Filter



$$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2]$$

$$y[n] = b_0 w[n] + b_1 w[n-1] + b_2 w[n-2]$$

Direct form I has more delays but is advantageous for fixed-point arithmetic as there is no internal overflow. This is because there is only one summation point.

Fixed-point arithmetic

A fixed number of bits is used for the integer part and the fractional part.

For example, assume 8-bit format.
Reserve 1 bit for the sign and 3 bits for the integer.
4 bits are for the fraction.

S	Integer	Fraction

Limited range of numbers
Maximum is $(2^3 - 1) + (1 - 2^{-4})$
Minimum is 2^{-4}

Advantage is efficiency which is important for real-time operations.

Internal Overflow

Consider a two's complement number

For a negative number, set all 1s to 0s and all 0s to 1s
Then add 1. The first bit (in red) is the sign bit.

Start with $+5 = 2^0 + 2^2 = 101$

-5 1101→ 1010→1011

$$\begin{array}{rr}-5 & 1011 \\ +3 & 0011 \\ \hline -2 & 1110\end{array}$$

Valid

Carry bit into the sign bit is 0.
Overflow bit out of sign bit is 0.

$$\begin{array}{rr} -6 & 1010 \\ +5 & 0101 \\ \hline -1 & 1111 \end{array}$$

Result is valid

Carry bit into the sign bit is 0.
Overflow bit out of sign bit is 0.

If carry bit into the sign bit is the same as
the overflow bit out of the sign bit, the result is valid.

Sometimes when adding three numbers,
there may be an internal error but the final result is correct

6	0110	
+5	0101	
11	1011	Invalid
−4	1100	
+7	0111	Valid

In direct form I, all summation points together so there is no ‘intermediate’ summation and so overflow errors are avoided.