FIR Filters

Chapter 5

With this chapter we turn to systems as opposed to signals. The systems discussed in this chapter are finite impulse response (FIR) digital filters.

- The term digital filter arises because these filters operate on discrete-time signals
- The term finite impulse response arises because the filter output is computed as a weighted, finite term sum, of past, present, and perhaps future values of the filter input, i.e.,

$$y[n] = \sum_{k=-M_1}^{M_2} b_k x[n-k] \tag{5.1}$$

  where both M_1 and M_2 are finite

- One of the simplest FIR filters we may consider is a 3–term moving average filter of the form

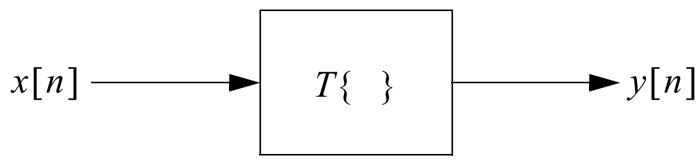
$$y[n] = \frac{1}{3}(x[n+1] + x[n] + x[n-1]) \tag{5.2}$$

- An FIR filter is based on a *feed-forward* difference equation as demonstrated by (5.2)
  - Feed-forward means that there is no feedback of past or future outputs to form the present output, just input related terms
- Continuous-time filters will be discussed in the circuits and systems courses

Discrete-Time Systems

- A discrete-time system transforms or maps an input sequence (signal) $x[n]$ into and output sequence (signal) $y[n]$ via a function or operation denoted as

$$y[n] = T\{x[n]\} \tag{5.3}$$



- Some examples are:

$$\begin{aligned} y[n] &= (x[n])^3 \\ &= \min\{x[n], x[n-1], x[n-2]\} \\ &= x[n] - x[n-1] \\ &= \frac{1}{2}(x[n] + x[n-1]) \end{aligned}$$

The Running (Moving) Average Filter

- A three-sample *causal* moving average filter is a special case of (5.1)

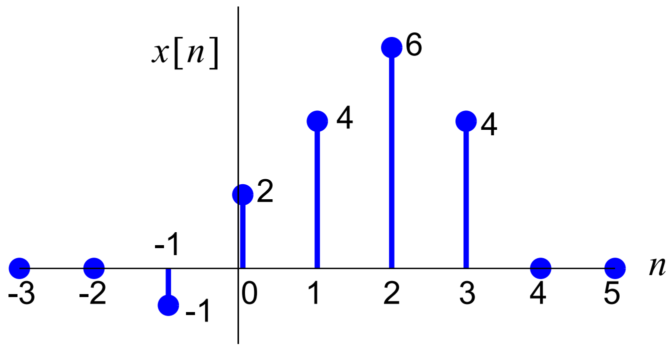
$$y[n] = \frac{1}{3}(x[n] + x[n-1] + x[n-2]), \tag{5.4}$$

which uses no future input values to compute the present output

  – <u>Note</u>: The term causal means that the present output, say

$y[n]$, utilizes only past and present signal values (no future values of the input)

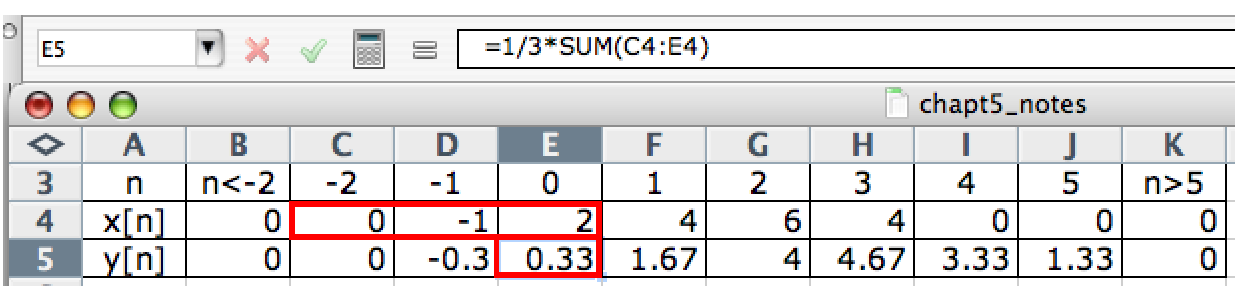
- Consider a *finite-length* input sequence having *support* (non-zero values) over the interval $-1 \le n \le 3$



- For $n = 0$ the 3–point causal moving average filter of (5.4) forms the output as

$$
\begin{aligned}
y[0] &= \frac{1}{3}(x[0] + x[-1] + x[-2]) \\
&= \frac{1}{3}(2 - 1 + 0) = \frac{1}{3} = 0.333 \qquad (5.5) \\
y[2] &= \frac{1}{3}(6 + 4 + 2) = \frac{12}{3} = 4
\end{aligned}
$$

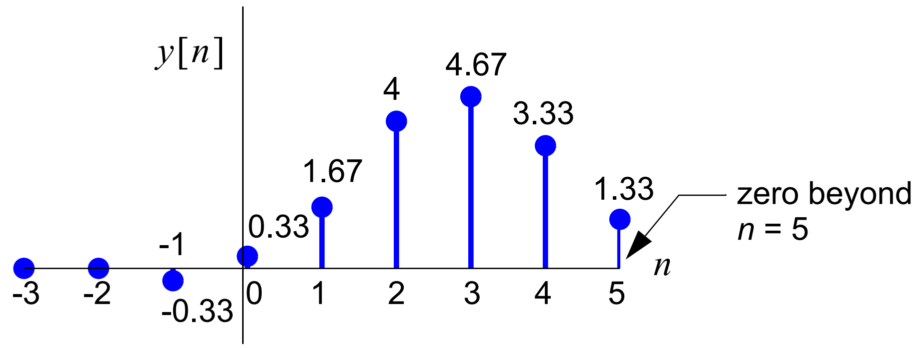
- If we put this into a spreadsheet we could write formulas into the cells that calculate the output sequence values as follows



	A	B	C	D	E	F	G	H	I	J	K
3	n	n<-2	-2	-1	0	1	2	3	4	5	n>5
4	x[n]	0	0	-1	2	4	6	4	0	0	0
5	y[n]	0	0	-0.3	0.33	1.67	4	4.67	3.33	1.33	0

- The output is $y[n]$ can be calculated using MATLAB functions as follows

```
>> n= -3:5;
>> x = [0 0 -1 2 4 6 4 0 0]
>> % We will learn about the filter function later
>> y = filter(1/3*[1 1 1],1,x);
>> stem(n,y,'filled')
```

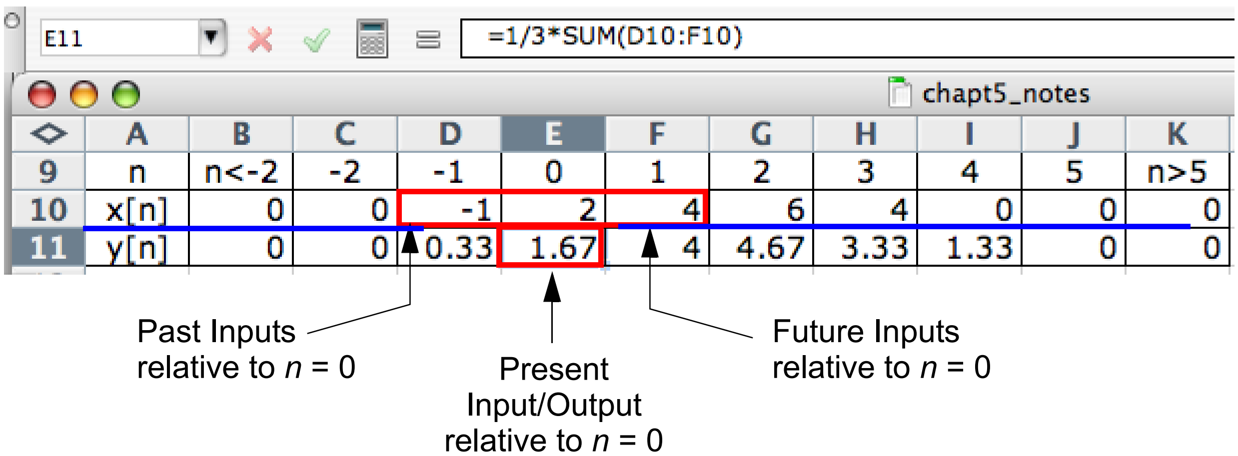


- The input/output relationship of (5.4) is known as a *difference equation*
- The action of the moving average filter has resulted in the output being *smoother* than the input
- Since only past and present values of the input are being used to calculate the present output, this filtering operation can operate in *real-time*
- A variation of the above 3-point averager is

$$y[n] = \frac{1}{3}(x[n+1] + x[n] + x[n-1]) \tag{5.6}$$

which can be termed a *centralized averager*, since the previous input, the present input, the next input are used to form the output

- This system is *noncausal* and cannot be computed in real-time, since the future input would not be available
- Inside the spreadsheet we have the following:



	A	B	C	D	E	F	G	H	I	J	K
9	n	n<-2	-2	-1	0	1	2	3	4	5	n>5
10	x[n]	0	0	-1	2	4	6	4	0	0	0
11	y[n]	0	0	0.33	1.67	4	4.67	3.33	1.33	0	0

- Notice that the output is the same as before, except it occurs one sample index earlier
  - To implement the causal filter we have to delay the output by one sample
- How we position and choose the width of the averaging window over the input signal are design choices
  - What happens if we make the window wider?

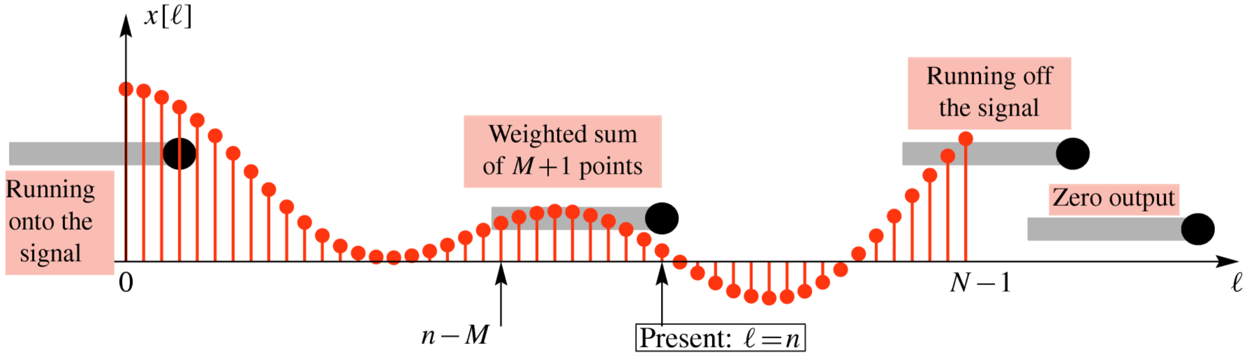
The General FIR Filter

- The class of causal FIR filters has difference equation of the form

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{5.7}$$

– Note: When $M = 2$ and $b_0 = b_1 = b_2 = 1/3$, we have the special case of the causal 3-point moving average filter of (5.4)

- The text has defined the following terms regarding filtering with an Mth order causal FIR filter



Example: An $M = 4$ FIR

- Suppose that $\{b_k\} = \{-2, 0, 1, 2, 3\}$ and $x[n]$ is as given in the following spreadsheet table

F17 =-2*F16+0*E16+1*D16+2*C16+3*B16

chapt5_notes

	A	B	C	D	E	F	G	H	I	J	K	L
15	n	n<0	0	1	2	3	4	5	6	7	8	n>8
16	x[n]	0	1	2	3	4	5	6	0	0	0	0
17	y[n]	0	-2	-4	-5	-4			22		27	[redacted]

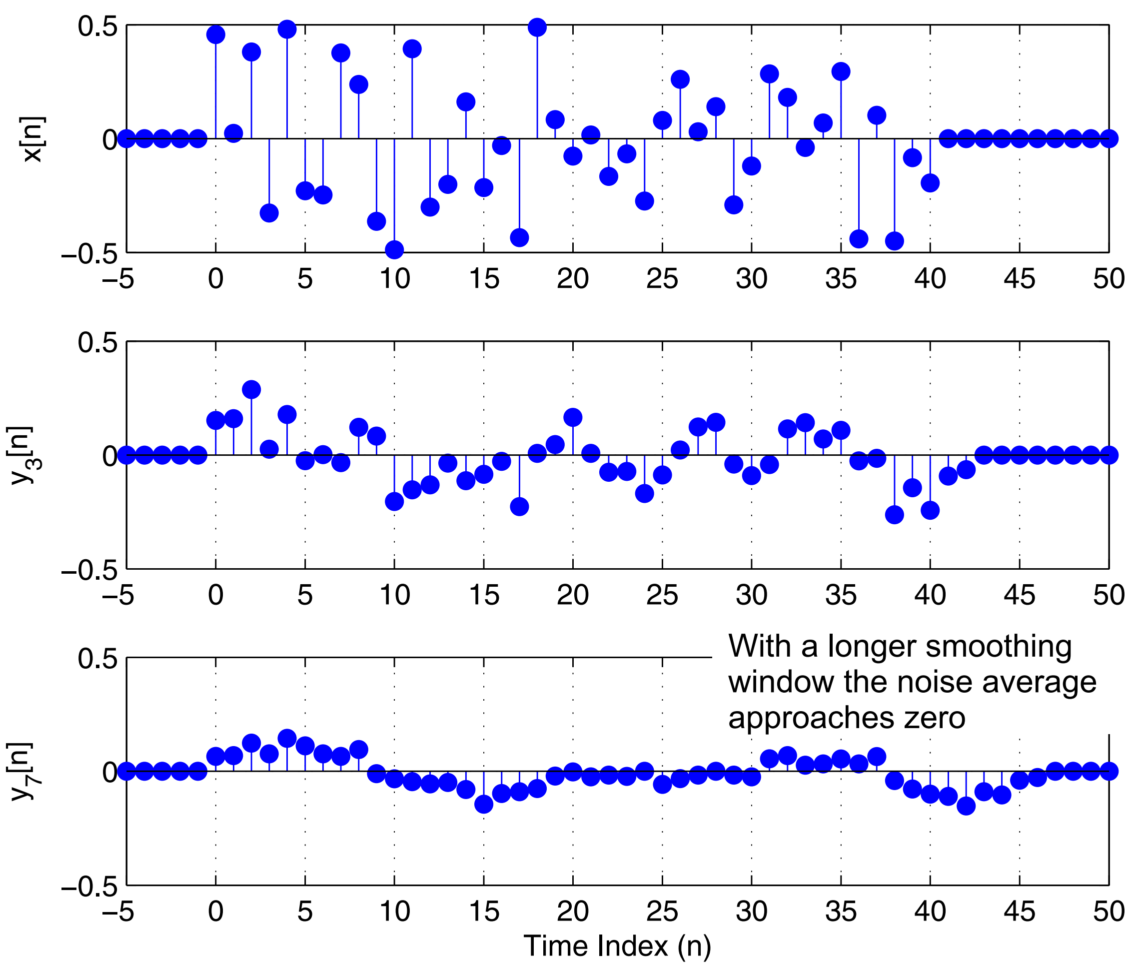
$$y[3] = -2 \times 4 + 0 \times 3 + 1 \times 2 + 2 \times 1 + 3 \times 0 = -4$$

- Fill-in the missing table entries

Example: Filtering A Windowed Noise Sequence

- In this example we create an input sequence composed of uniformly distributed random numbers $x \in [-1/2, 1/2]$ for $0 \le n \le 40$ and zero otherwise
- The filter coefficients $\{b_k\}$ represent both 3-point and 7-point moving average filters

```
>> n = -5:50;
>> x = [zeros(1,5), rand(1,41)-1/2, zeros(1,10)];
>> y3 = filter(ones(1,3)/3,1,x);
>> y7 = filter(ones(1,7)/7,1,x);
```



The Unit Impulse Response

Three interconnected concepts of this subsection are the *unit impulse sequence*, the *unit impulse response*, and the *convolution sum*.

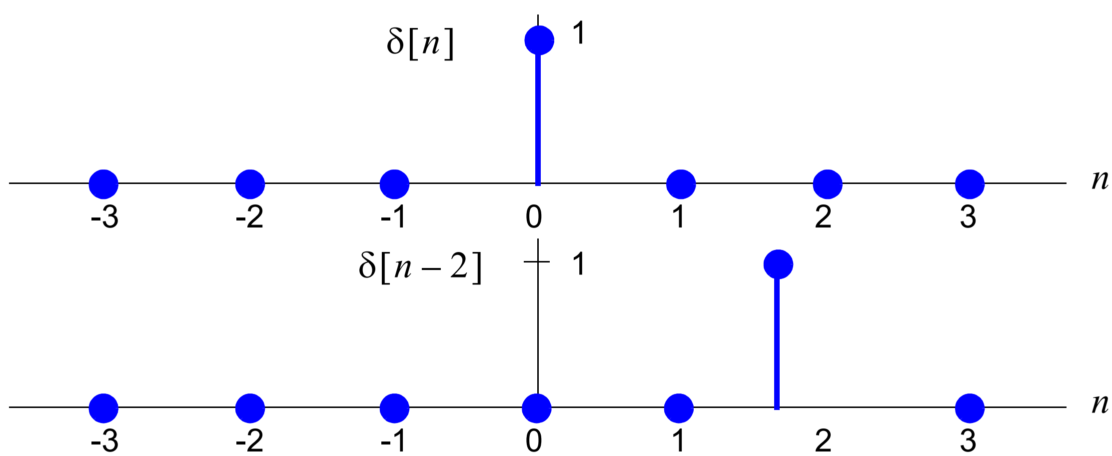
Unit Impulse Sequence: $\delta[n]$

- A sequence having a nonzero value of one only when its argument is equal to zero, i.e., $n = 0$

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \tag{5.8}$$

- The unit impulse sequence can be shifted right or left by integer n_0 by writing

$$\delta[n - n_0] = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases} \tag{5.9}$$



- We can both time shift and amplitude scale the impulse sequence, such that a linear combination of them can be used to form any sequence, e.g.,

$$\begin{aligned} x[n] &= 2\delta[n] + 4\delta[n-1] + 6\delta[n-2] \\ &\quad + 4\delta[n-3] + 2\delta[n-4] \end{aligned} \tag{5.10}$$

- Each of the five scaled and time shifted impulses forms a single nonzero sample value of the complete sequence

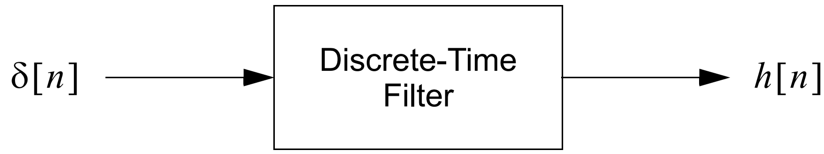
n	...	-2	-1	0	1	2	3	4	5	6	...
$2\delta[n]$	0	0	0	2	0	0	0	0	0	0	0
$4\delta[n-1]$	0	0	0	0	4	0	0	0	0	0	0
$6\delta[n-2]$	0	0	0	0	0	6	0	0	0	0	0
$4\delta[n-3]$	0	0	0	0	0	0	4	0	0	0	0
$2\delta[n-4]$	0	0	0	0	0	0	0	2	0	0	0
$x[n]$	0	0	0	2	4	6	4	2	0	0	0

- The generalization to the above is the ability to expand any sequence in this fashion, i.e.,

$$\begin{aligned} x[n] &= \sum_k x[k]\delta[n-k] \\ &= \cdots + x[-1]\delta[n+1] + x[0]\delta[n] \\ &\quad + x[1]\delta[n-1] + x[2]\delta[n-2] + \cdots \end{aligned} \tag{5.11}$$

Unit Impulse Response Sequence: $h[n]$

- When we input to an FIR filter the sequence $x[n] = \delta[n]$, the filter output (assuming the filter is initially at rest) is the unit impulse response, denoted $y[n] = h[n]$



- Note this definition holds for any discrete-time filter, not just FIR filters

---

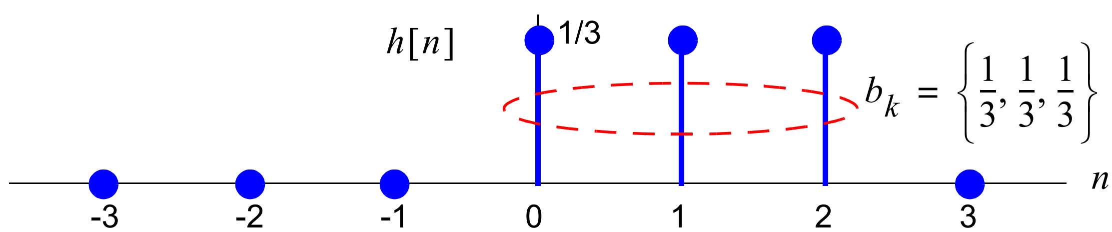
Example: 3-Point Moving Average Filter Impulse Response

- For this filter $b_k = \{1/3, 1/3, 1/3\}$
- Using (5.7)

$$h[n] = \sum_{k=0}^{2} b_k \delta[n-k]$$

$$= \frac{1}{3}\sum_{k=0}^{2} \delta[n-k] \qquad (5.12)$$

$$= \frac{1}{3}(\delta[n] + \delta[n-1] + \delta[n-2])$$



- For a general FIR filter of (5.7) we observe that

$$h[n] = \sum_{k=0}^{M} b_k \delta[n-k] \qquad (5.13)$$

– Note in particular that the impulse response is finite, that is it extends over $n \in [0, M]$, hence the term finite impulse response (FIR) system is justified

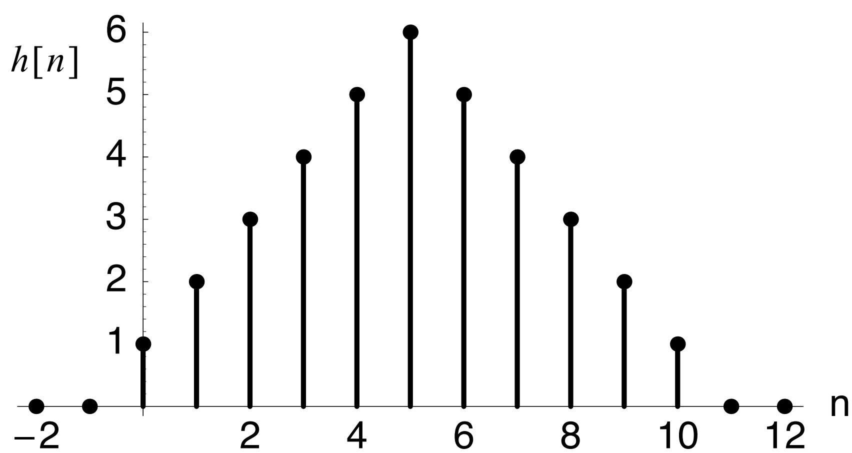
---

Example: $M = 10$ Triangle Impulse Response

$$h[n] = \sum_{k=0}^{10} (6 - |k-5|)\delta[n-k] \tag{5.14}$$

- Evaluate point-by-point and plot, e.g.,

$$(6 - |k-5|)\big|_{k=5} = 6 \qquad (6 - |k-5|)\big|_{k=4} = 5$$



The Unit-Delay System: $y[n] = x[n - n_0]$

- When $n_0 = 1$, $y[n] = x[n-1]$ corresponds to a system imparting a *unit delay*
- A unit delay system is a special FIR filter where

$$b_k = \{0, 1\} \tag{5.15}$$

and the filter order is $M = 1$

- The impulse response of a delay by n_0 system is

$$h[n] = \delta[n - n_0] \tag{5.16}$$

Convolution and FIR Filters

- It can be shown (more on this later) that a general expression of a filter's output can be expressed in terms of the impulse response and the input as

$$y[n] = \sum_{k=0}^{M} h[k]x[n-k] \tag{5.17}$$

- This formula has a special name: *convolution sum formula*
- We say that $y[n]$ is the convolution of $x[n]$ and $h[n]$

Example: Convolution Using the Text Table Method

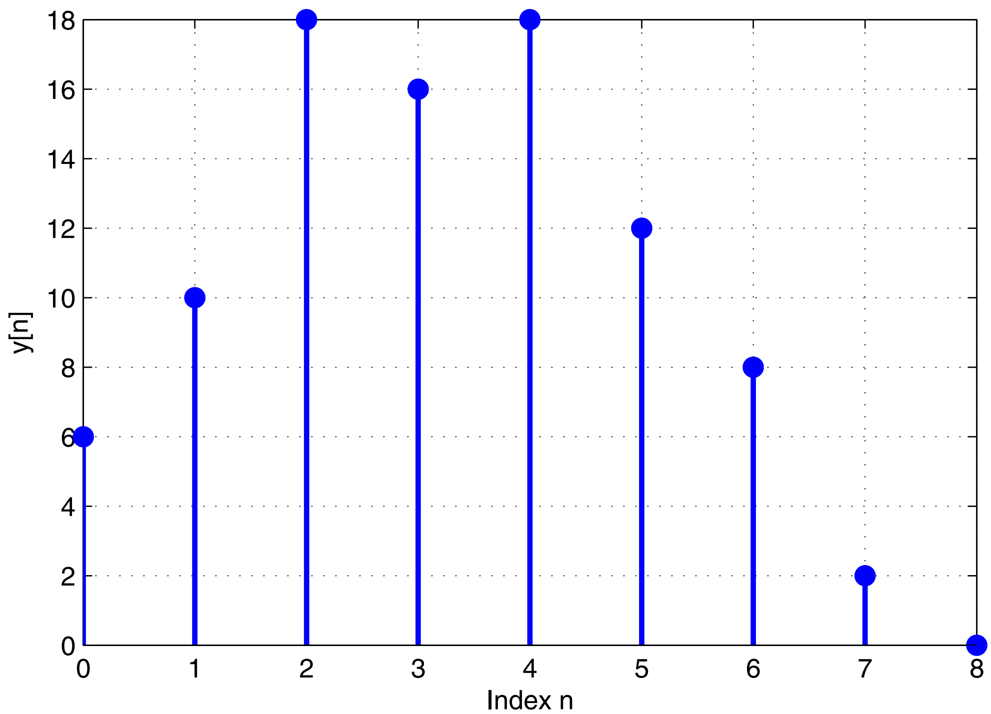
- Convolve $x[n] = \{2, 4, 6, 4, 2\}$ with $h[n] = \{3, -1, 2, 1\}$, where both sequence start at $n = 0$ and are nonzero for the values given

n	$n<0$	0	1	2	3	4	5	6	7	$n>7$
$x[n]$	0	2	4	6	4	2	0	0	0	0
$h[n]$	0	3	-1	2	1					
$h[0]x[n]$	0	6	12	18	12	6	0	0	0	0
$h[1]x[n-1]$	0	0	-2	-4	-6	-4	-2	0	0	0
$h[2]x[n-2]$	0	0	0	4	8	12	8	4	0	0
$h[3]x[n-3]$	0	0	0	0	2	4	6	4	2	0
$y[n]$	0	6	10	18	16	18	12	8	2	0

sum down

- We can check the answers using MATLAB's filter function

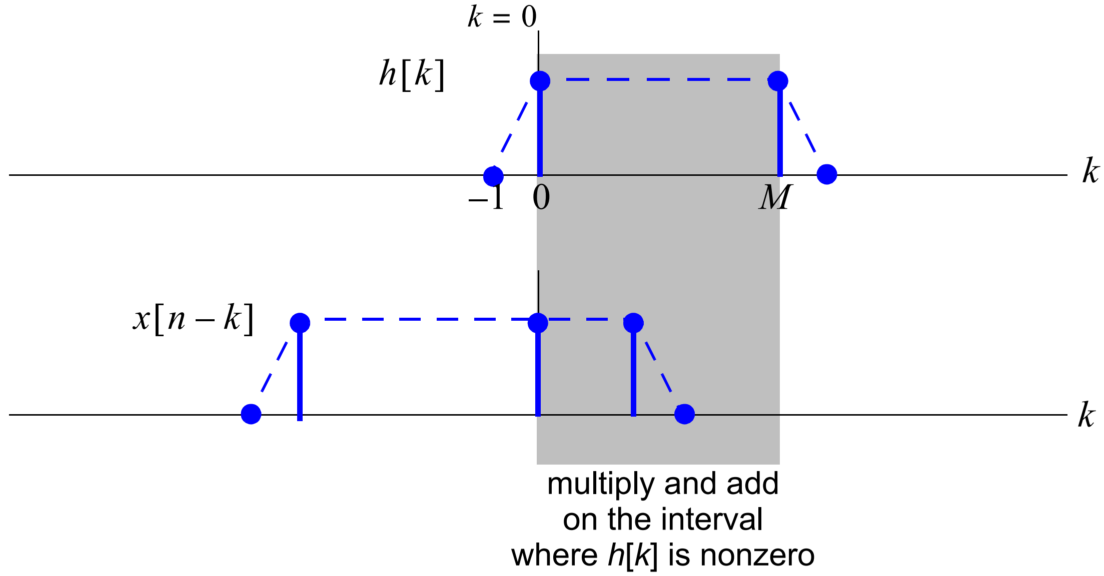
```
>> n = 0:8;
>> x = [2  4  6  4  2  0  0  0  0];
>> h = [3  -1  2  1];
>> y = filter(h,1,x);
>> y

y = 6     10     18     16     18     12      8      2      0
```



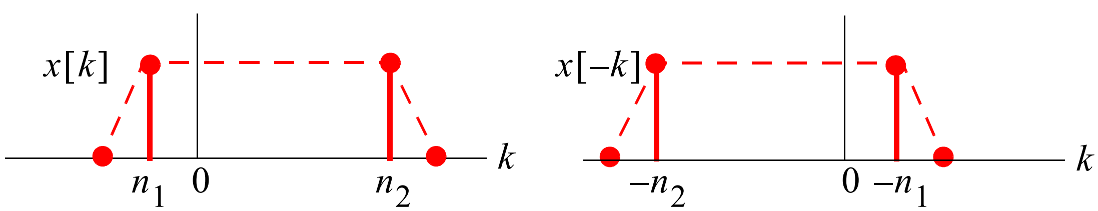
Example: Convolution Using an Alternate Table Method

- In this example we consider another approach to performing convolution of finite length sequences using a table
- Convolve $x[n] = \{0, 1, 2, 3\}$ and $h[n] = \{1, 1, 2, 3\}$
- In the convolution sum formula we need to multiply the sequence $h[k]$ by the sequence $x[n-k]$ for a fixed/given value of n

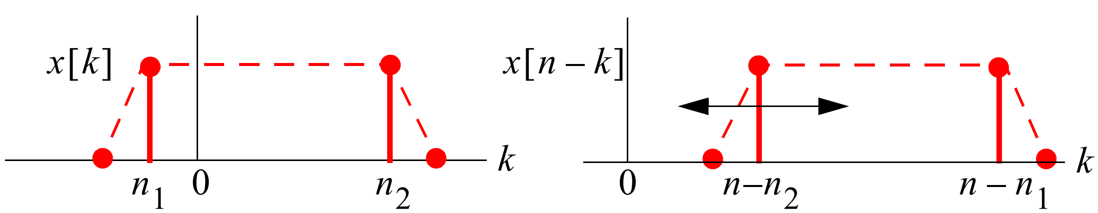
$$y[n] = \sum_{k=0}^{M} h[k]x[n-k]$$



- What is $x[n-k]$ relative to $x[k]$?
  - $x[-k]$ is $x[k]$ flipped from left to right about $k = 0$



  - $x[n-k]$ is $x[k]$ flipped and shifted by n



  - Check by plugging $n - n_1$ into $x[n-k]$

$$x[n-(n-n_1)] = x[n_1] \text{ OK!}$$

  - Also plug $n - n_2$ into $x[n-k]$

$$x[n-(n-n_2)] = x[n_2] \text{ OK!}$$

- Using the above view of the convolution sum, we obtain an alternate table form for the convolution sum

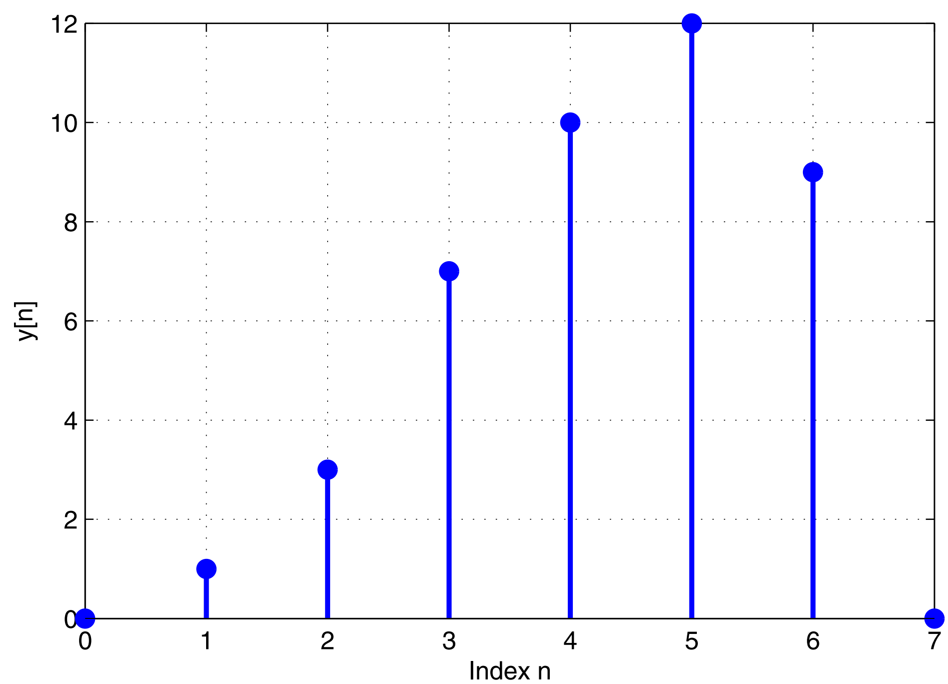
L3 `=$E$2*E3+$F$2*F3+$G$2*G3+$H$2*H3`

chapt5_notes

	A	B	C	D	E	F	G	H	I	J	K	L
1					h[k] vs k							
2	x[n-k]		0	0	1	1	2	3	0	0		
3	n=0	3	2	1	0	0	0	0	0	0	0	0
4	n=1	0	3	2	1	0	0	0	0	0	0	1
5	n=2	0	0	3	2	1	0	0	0	0	0	3
6	n=3	0	0	0	3	2	1	0	0	0	0	7
7	n=4	0	0	0	0	3	2	1	0	0	0	10
8	n=5	0	0	0	0	0	3	2	1	0	0	12
9	n=6	0	0	0	0	0	0	3	2	1	0	9
10					k=0	k=1	k=2	k=3				y[n]

- Check using MATLAB's filter function

```
>> n = 0:7;
>> x = [0 1 2 3 0 0 0 0];
>> h = [1 1 2 3];
>> y = filter(h,1,x);
>> y = 0      1      3      7      10      12      9      0
```



Using MATLAB's Filter Function

- We have been using the function `filter()` in the few examples, so how does it work?

```
>> help filter
 FILTER One-dimensional digital filter.
    Y = FILTER(B,A,X) filters the data in vector X with the
    filter described by vectors A and B to create the filtered
    data Y.  The filter is a "Direct Form II Transposed"
    implementation of the standard difference equation:

    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

    If a(1) is not equal to 1, FILTER normalizes the filter
    coefficients by a(1).

    [Y,Zf] = FILTER(B,A,X,Zi) gives access to initial and final
    conditions, Zi and Zf, of the delays.  Zi is a vector of length
    MAX(LENGTH(A),LENGTH(B))-1, or an array with the leading dimension
    of size MAX(LENGTH(A),LENGTH(B))-1 and with remaining dimensions
    matching those of X.
```

- From the help listing we see that it simply evaluates a general difference equation of the form

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \tag{5.18}$$

- At the present time (Chapter 5) we do not have any feedback terms in our difference equation, so $N = 0$ and we assume that $a_0 = 1$, resulting in (5.18) reducing to

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{5.19}$$

- When using `filter()` to implement an FIR filter, we set the vectors `a = [1]` and `b =` $[b_0, b_1, \ldots, b_M]$

Convolution in MATLAB

- We have seen how MATLAB's filter function can directly evaluate (number crunch) a difference equation for a given filter coefficient set $\{b_k\}$ and input sequence $x[n]$
- Convolution between any two finite support (duration) sequences can also be performed using the function `conv()`

```
>> help conv
 CONV Convolution and polynomial multiplication.
    C = CONV(A, B) convolves vectors A and B.  The resulting
    vector is length LENGTH(A)+LENGTH(B)-1.
    If A and B are vectors of polynomial coefficients, convolving
    them is equivalent to multiplying the two polynomials.
```

- Check this out using the previous example values for $h[n]$ and $x[n]$

```
>> h = [1 1 2 3];
>> x = [0 1 2 3];
>> y = conv(x,h)

y =   0     1     3     7    10    12     9
```

- We can also use the `conv()` function to perform polynomial multiplication, or at least obtain the coefficient set corresponding to polynomial multiplication, e.g.

$$\begin{aligned} P_1(x) &= a_0 + a_1 x + a_2 x^2 \\ P_2(x) &= b_0 + b_1 x + b_2 x^2 \end{aligned} \tag{5.20}$$

- Let $P_3(x) = P_1(x)P_2(x)$, then the coefficients are

```
[c0 c1 c2 ...] = conv([a0 a1 a2],[b0 b1 b2])
```

such that

$$P_3(x) = c_0 + c_1 x + c_2 x^2 + \cdots \tag{5.21}$$

- For example

```
>> conv([1 2 3],[1 -1 2])

ans =  1     1     3     1     6
```

- So we conclude that

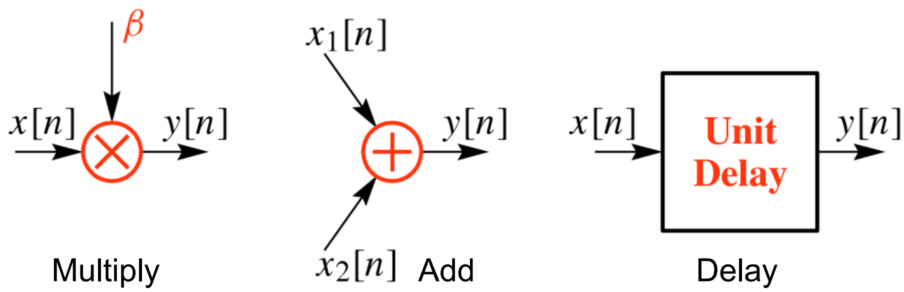
$$(1 + 2x + 3x^2)(1 - x + 2x^2) = 1 + x + 3x^2 + x^3 + 6x^4 \tag{5.22}$$

- Verify!

Implementation of FIR Filters

- The implementation of an FIR requires three basic building blocks
  - multiplication
  - addition
  - signal delay
- We also need to be able to store filter coefficients in memory

Building Blocks



Multiplier: $y[n] = \beta x[n]$

- In a DSP system the multiplier must be fast and must have sufficient precision (bit width; think logic circuits) to support the desired application
- A high quality filter will in general require more multiplications than one of lesser quality, so throughput suffers if the multiplier is not fast
  - There are classes of filters that do not require multiplies
- FIR filters having 50 coefficients or more are not that uncommon

Adder: $y[n] = x_1[n] + x_2[n]$

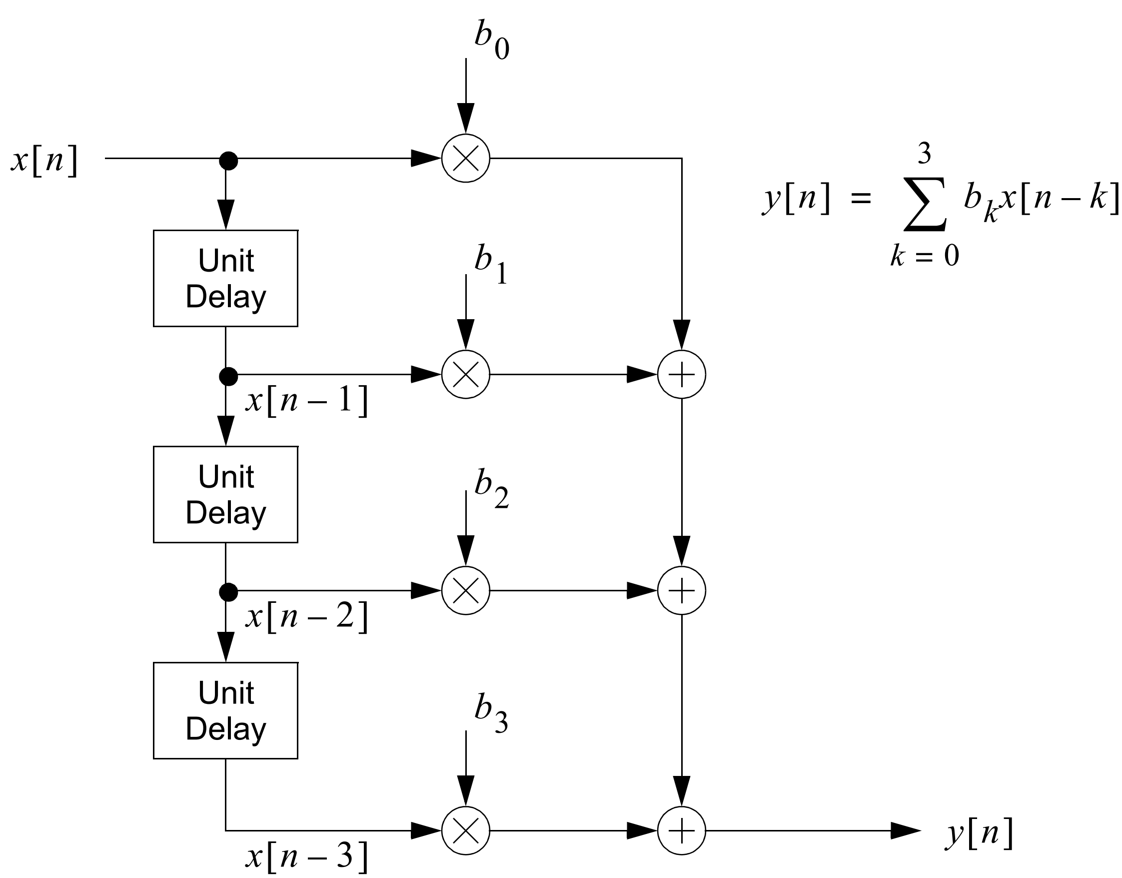
- Signal addition is a very basic DSP function
- In an FIR filter additions are required in combination with multiplications, hence DSP microprocessors feature multiply-accumulate (MAC) units
- Adders generally operate with just two inputs at a time

Unit Delay: $y[n] = x[n-1]$

- The unit delay provides a one sample signal delay
- A sample value is stored in a memory slot for one sample clock cycle, and then made available as an input to the next processing stage
- An M-unit delay requires M memory cells (note each memory cell must store say B-bits) configured as a shift register (B-bits wide)

Block Diagrams

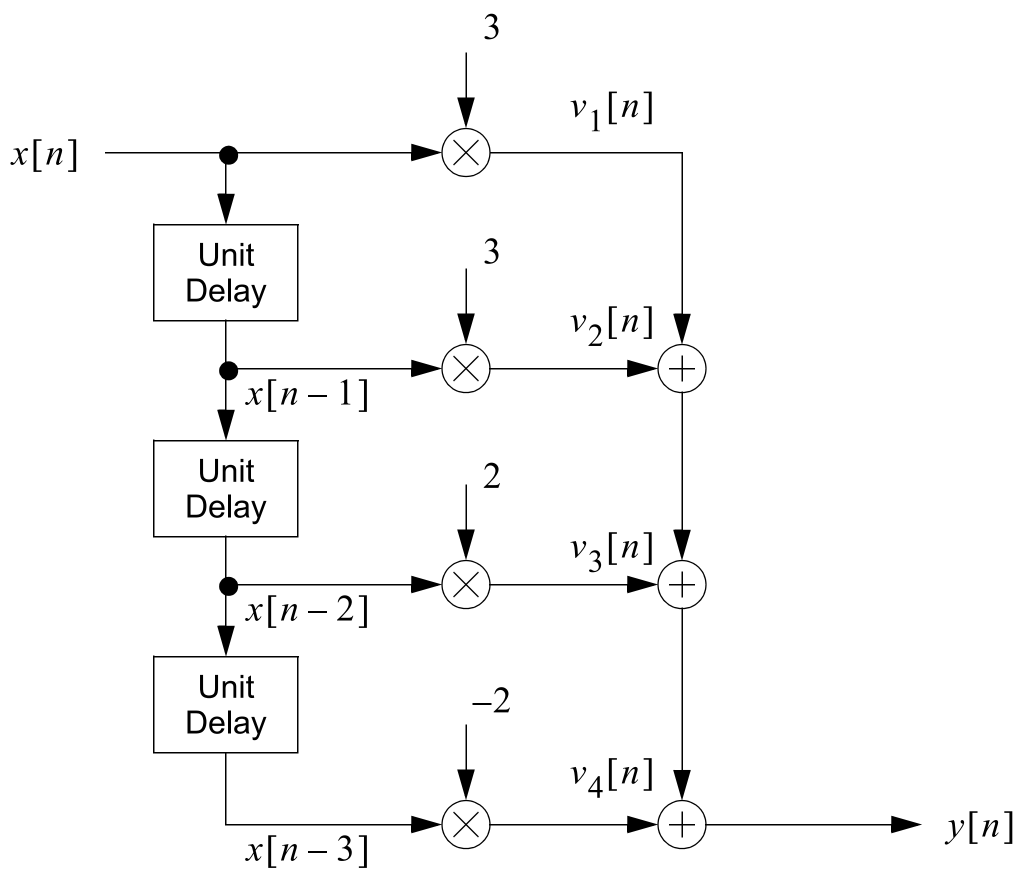
- Using the building blocks described above, we can construct a block diagram for say an $M = 3$ FIR filter



- Notice that the signal flow is strictly feed-forward, since all paths connecting the input to the output flow in the forward direction
- Feedback paths will be needed in the filters of Chapter 8
- The structure of this block diagram is called *direct form*
  - Other structures can be used to implement the same difference equation, and hence the same input/output relationship
  - Once we understand how the direct form structure works, we can easily write the equations or draw the block diagram

---

Example: $M = 3$ Block Diagram to Difference Equation



- By inspection we know that

$$y[n] = 3x[n] + 3x[n-1] + 2x[n-2] - 2x[n-3]$$

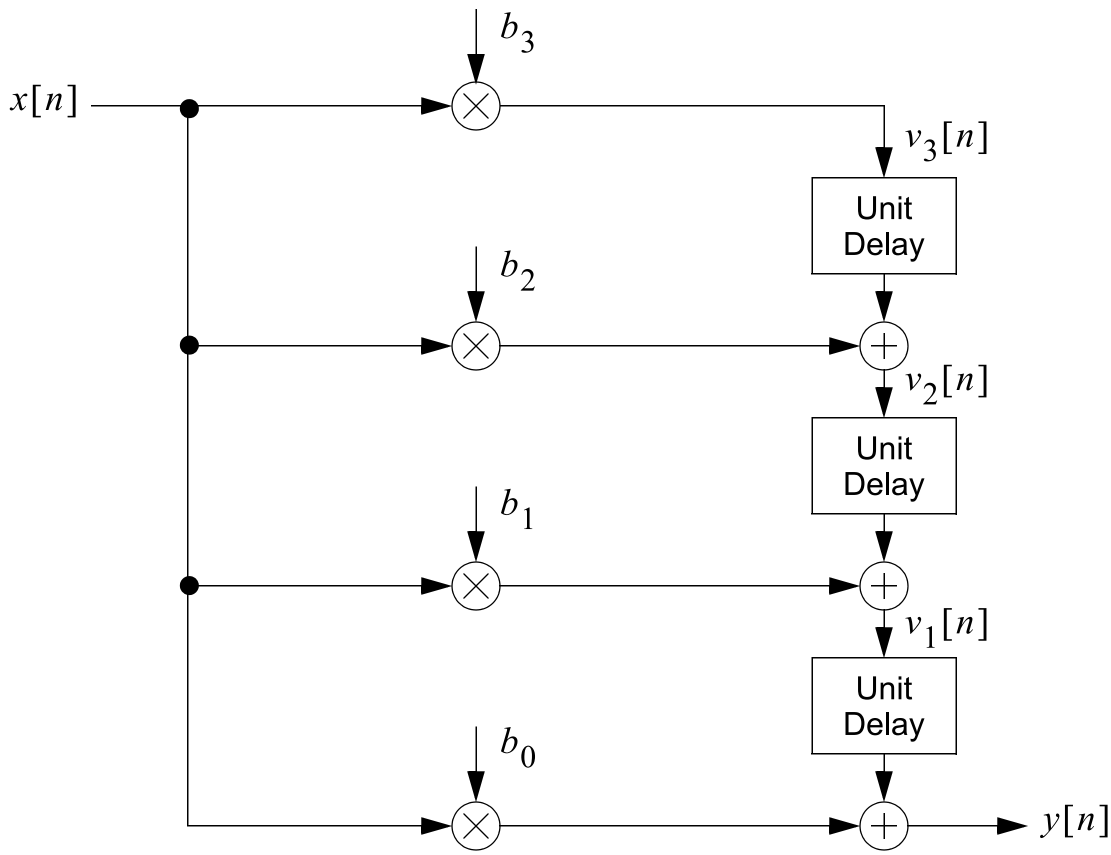
- If we were not familiar with this form, we can write an equation for the output by adding some additional labels, and then making substitutions, e.g.,

$$\begin{aligned} y[n] &= v_1[n] + v_2[n] + v_3[n] + v_4[n] \\ &= (b_0 x[n]) + (b_1 x[n-1]) \\ &\quad + (b_2 x[n-2]) + (b_3 x[n-3]) \end{aligned}$$

– Given a new unfamilar block diagram requires a procedure to derive the difference equation

Step	Description
(a)	Create a signal label for the input to each unit delay
(b)	Delay outputs can be written in terms of the delay inputs
(c)	At each summing node write an equation
(d)	Reduce the multiple equations you obtain via a series substitutions similar to solving a system of equations

Example: A Transposed Block Diagram



- To discover the difference equation we begin by writing equations at summer output

$$
\begin{aligned}
y[n] &= b_0 x[n] + v_1[n-1] \\
v_1[n] &= b_1 x[n] + v_2[n-1] \\
v_2[n] &= b_2 x[n] + v_3[n-1] \\
v_3[n] &= b_3 x[n]
\end{aligned}
$$

- Next make substitutions, beginning with $v_1[n]$

$$y[n] = b_0 x[n] + (b_1 x[n-1] + v_2[n-2])$$

- Next substitute in $v_2[n]$

$$y[n] = b_0 x[n] + b_1 x[n-1] + (b_2 x[n-2] + v_3[n-3])$$

- Finally substitute in $v_3[n]$

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + b_3 x[n-3]$$

- The block diagram studied in the last example implements a *transposed direct form*
  - Notice that the order of operation is different from the direct form, specifically the signal flow is in the opposite direction and the input and output are swapped
  - Changing the order of operations can be important in applications where hardware architectures come into play, e.g., VLSI designs or particular FPGA structures for high speed DSP
  - Finite word-length effects also come into play when choosing the block diagram topology

Linear Time-Invariant (LTI) Systems

- The FIR filter we have been considering has two important properties that need further discussion
  - *linearity*
  - *time invariance*
- In this section of the notes and text, a transformation of the input signal $x[n]$ to the output signal $y[n]$ is denoted via

$$x[n] \mapsto y[n] \text{ (text) or } x[n] \Rightarrow y[n] \text{ (notes)} \tag{5.23}$$

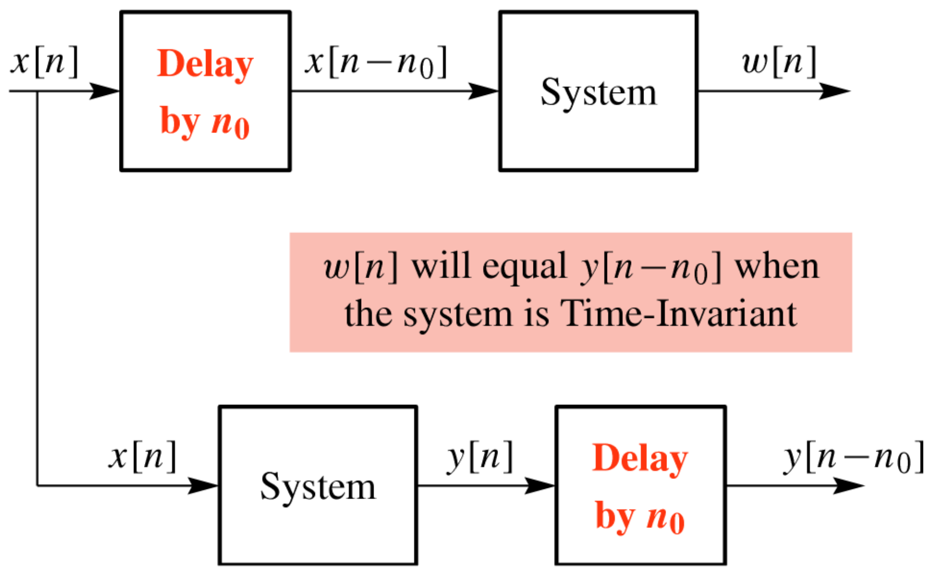
- This notation applies to difference equation systems as well as instantaneous systems such as $y[n] = x^2[n]$

Time Invariance

- A discrete-time system is time-invariant if for an input delayed/shifted by n_0, the output is also delayed by the same amount, i.e.,

$$x[n-n_0] \Rightarrow y[n-n_0] \tag{5.24}$$

given that $x[n] \Rightarrow y[n]$ and (5.24) holds for any n_0



<u>Example</u>: $y[n] = ax[n] + b$

- Using the above proof template, we let

$$w[n] = ax[n-n_0] + b,$$

that is we delay the input before it enters the system

- Now we compare $y[n-n_0]$ to $w[n]$

$$y[n-n_0] = \{ax[n]+b\}\big|_{n \to n-n_0} = ax[n-n_0]+b$$

$$\text{yes!} = w[n]$$

so the system is time-invariant

Example: $y[n] = nx[n]$

- Let

$$w[n] = nx[n-n_0]$$

  – Note the delay is applied before being input to the system

- Now compare the delayed output to $w[n]$

$$y[n-n_0] = (n-n_0)x[n-n_0] \neq nx[n-n_0] = w[n]$$

so the system **is not** time-invariant

Linearity

- For a system to be linear, *superposition* must hold, meaning that given $x_1[n] \Rightarrow y_1[n]$ and $x_2[n] \Rightarrow y_2[n]$, then it follows that

$$\begin{aligned} x[n] &= \alpha x_1[n] + \beta x_2[n] \\ \Rightarrow y[n] &= \alpha y_1[n] + \beta y_2[n] \end{aligned} \tag{5.25}$$

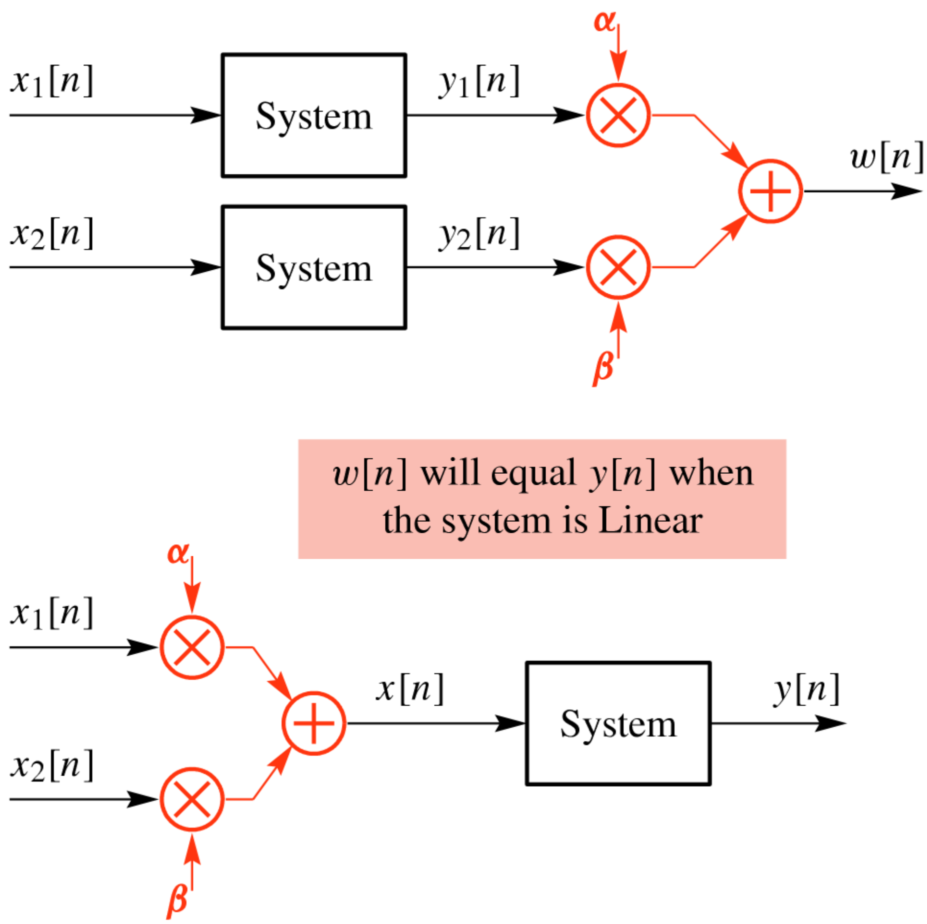
for all values of α and β

  – As a special case $\alpha = \beta = 1$ results in

$$x[n] = x_1[n] + x_2[n] \Rightarrow y_1[n] + y_2[n] = y[n] \tag{5.26}$$

– Also if $\beta = 0$ we have

$$x[n] = \alpha x_1[n] \Rightarrow \alpha y_1[n] = y[n] \tag{5.27}$$



Example: Revisit $y[n] = nx[n]$

- Using the above proof template,

$$w[n] = \alpha(nx_1[n]) + \beta(nx_2[n])$$

and

$$\begin{aligned} y[n] &= n(\alpha x_1[n] + \beta x_2[n]) \\ &= \alpha n x_1[n] + \beta n x_2[n] \end{aligned}$$

- Thus we see that $y[n] = w[n]$, so the system **is linear**

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

- Again using the template,

$$w[n] = \alpha(ax_1[n] + b) + \beta(ax_2[n] + b)$$
$$= a(\alpha x_1[n] + \beta x_2[n]) + b(\alpha + \beta)$$

and

$$y[n] = a(\alpha x_1[n] + \beta x_2[n]) + b$$
$$= a(\alpha x_1[n] + \beta x_2[n]) + b$$

- The last term of $w[n]$ does not agree with the last term of $y[n]$, thus the system is **not linear**
  – Note that if $b = 0$, then linearity holds

The FIR Case

- A special case of interest is the FIR filter system we have been studying

Time Invariance: Let $v[n] = x[n - n_0]$ initially, in forming $w[n]$

$$w[n] = \sum_{k=0}^{M} b_k v[n-k] = \sum_{k=0}^{M} b_k x[(n-k) - n_0]$$
$$= \sum_{k=0}^{M} b_k x[(n - n_0) - k]$$

Next,

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

$$\Rightarrow y[n-n_0] = \sum_{k=0}^{M} b_k x[(n-n_0)-k]$$

Which proves time invariance!

Linearity: Input $\alpha x_1[n] + \beta x_2[n]$

$$\begin{aligned} y[n] &= \sum_{k=0} b_k(\alpha x_1[n-k] + \beta x_2[n-k]) \\ &= \alpha \sum_{k=0}^{M} b_k x_1[n-k] + \beta \sum_{k=0}^{M} b_k x_2[n-k] \\ &= \alpha y_1[n] + \beta y_2[n] \end{aligned}$$

Which proves linearity!

- An FIR filter is an example of a *linear time-invariant* (LTI) system

Convolution and LTI Systems

Will will now establish that the impulse response characterizes any LTI system, and the convolution of the input with the impulse response produces the output.

Derivation of the Convolution Sum

- Step 1: For any signal we can write

$$x[n] = \sum_{l} x[l]\delta[n-l] \tag{5.28}$$

where in the most general case the sum runs from $-\infty$ to $+\infty$

- Step 2: Time invariance means that

$$\delta[n-n_0] \Rightarrow h[n-n_0]$$

$$\text{so } x[l]\delta[n-l] \Rightarrow x[l]h[n-l]$$

for any l

- Step 3: Apply (5.28) on both sides to the system and use linearity to write

$$\begin{aligned} x[n] &= \sum_{l} x[l]\delta[n-l] \\ &\Rightarrow \sum_{l} x[l]h[n-l] = y[n] \end{aligned} \tag{5.29}$$

- For the case of an input signal having support over the entire axis, n, the convolution sum formula becomes

$$\boxed{y[n] = \sum_{l=-\infty}^{\infty} x[l]h[n-l]} \tag{5.30}$$

for all LTI systems

Example: FIR Convolution

- For the special case of $h[n]$ nonzero over the interval $0 \le n \le M$, the sum limits of (5.30) are reduced since $h[n-l]$ is zero outside the interval $0 \le n - l \le M$
- Solving the inequality for l we have

$$(n-M) \le l \le n$$

so

$$y[n] = \sum_{l = n-M}^{n} x[l]h[n-l] \tag{5.31}$$

Example: Proof that (5.17) and (5.31) are Equivalent

- Starting from (5.31) we change variables in the sum formula by letting $k = n - l$
- With $k = n - l$ it follows that $l = n - k$
- With k the new summation variable, the lower sum limit of (5.31) becomes $l = n - M \to k = n - (n - M) = M$
- The upper sum limit becomes $l = n \to k = n - n = 0$, so

$$y[n] = \sum_{k = M}^{0} x[n-k]h[k] = \sum_{k=0}^{M} x[n-k]h[k] \tag{5.32}$$

- Where the last step follows from the fact that changing the order of the summation does not alter the sum

Some Properties of LTI Systems

- Convolution as an Operator: $y[n] = x[n]*h[n]$
  - When viewed as an operator we are saying that $y[n]$ is the result of convolving the sequence $x[n]$ with $h[n]$

Example: Convolving with $\delta[n-n_0]$

- When one of the two signals is an impulse sequence, we get a result known as the *sifting property*

$$\begin{aligned} x[n]*\delta[n-n_0] &= \sum_{k=-\infty}^{\infty} x[k]\delta[(n-n_0)-k] \\ &= x[n-n_0] \end{aligned} \tag{5.33}$$

- The impulse sequence inside the sum turns on only when $k = n - n_0$, so only one term of the sum is retained (sifted out)

- Commutative Property: $x[n]*h[n] = h[n]*x[n]$
  - In an earlier example we showed that the convolution sum for FIR filtering can be written in two ways; this established the commutative property as a special case
  - In general terms we can write

$$y[n] = x[n]*h[n] = \sum_{l=-\infty}^{\infty} x[l]h[n-l] \tag{5.34}$$

Now let $k = n - l$ in the sum to obtain

$$\begin{aligned} x[n]*h[n] &= \sum_{k=+\infty}^{-\infty} x[n-k]h[k] \\ &= \sum_{k=-\infty}^{\infty} x[n-k]h[k] = h[n]*x[n] \end{aligned} \qquad (5.35)$$

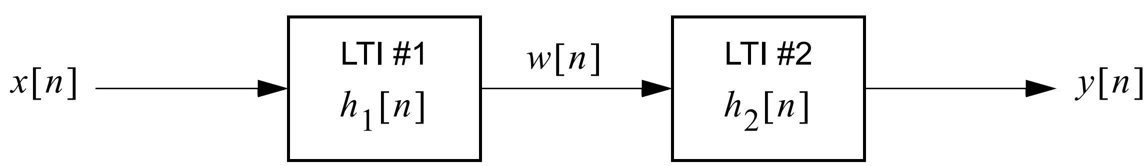
- Associative Property:

$$(x_1[n]*x_2[n])*x_3[n] = x_1[n]*(x_2[n]*x_3[n])$$

  - This property applies when three signals and two convolutions are involved
  - It says that either convolution operation may be done first
  - proof in text

Cascaded LTI Systems

- Cascaded LTI systems occur frequently in practice, e.g., a signal must go through two filtering operations before the final result is obtained



- The natural question is how is $y[n]$ related to $x[n]$, $h_1[n]$, and $h_2[n]$?
- To start with we know that $w[n] = x[n]*h_1[n]$
- Likewise we know that $y[n] = w[n]*h_2[n]$

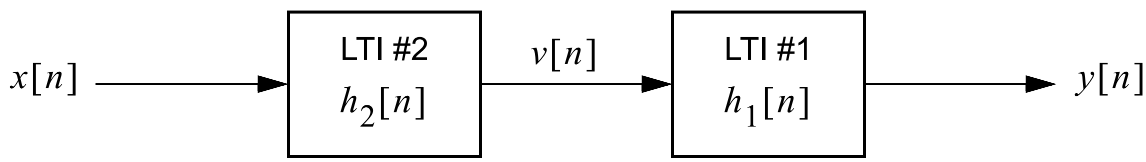
- Putting these two results together we obtain

$$y[n] = (x[n]*h_1[n])*h_2[n] \tag{5.36}$$

- From the commutative associative properties, we can also write that

$$\begin{aligned} y[n] &= x[n]*(h_1[n]*h_2[n]) \\ &= x[n]*(h_2[n]*h_1[n]) \\ &= (x[n]*h_2[n])*h_1[n] \end{aligned} \tag{5.37}$$

  which establishes that we may change the order of the LTI systems in the cascade



- Thus when we have a cascade of LTI systems the equivalent impulse response is

$$h[n] = h_1[n]*h_2[n] = h_2[n]*h_1[n] \tag{5.38}$$

<u>Example</u>: 4-point Moving Average and Backwards Difference

- Consider the cascade of a moving average system with impulse response

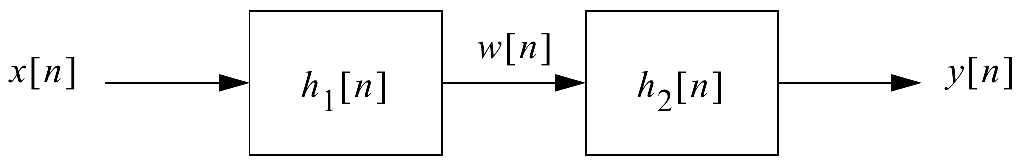
$$h_1[n] = \begin{cases} 1/4, & 0 \le n \le 3 \\ 0, & \text{otherwise} \end{cases}$$

and a first-order backwards difference system, which has difference equation

$$y[n] = w[n] - w[n-1]$$

which implies

$$h_2[n] = \delta[n] - \delta[n-1]$$



- We know that the impulse response of the cascade, $h_3[n]$, is given by

$$\begin{aligned} h_3[n] &= h_1[n] * h_2[n] \\ &= \sum_{k=-\infty}^{\infty} h_1[k] h_2[n-k] \end{aligned}$$

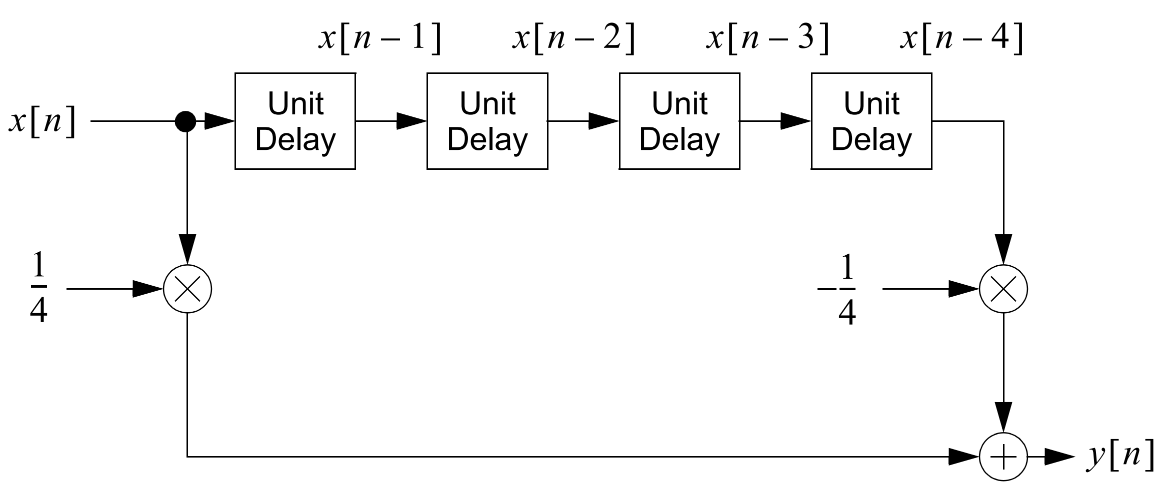
- We can again use the table method introduced in the text or the alternate table method introduced in the notes, to find $h_3[n]$
  - Since $h_1[n]$ has support on the interval [0, 3], the sum limits run from $k = 0$ to $k = 3$
  - The support interval for $h_3[n]$ runs [0+0, 3+1] = [0, 4]

			$h_1[k]$ vs. k						
$h_2[n-k]$			$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$			$h_3[n]$
$n = 0$	0	-1	1	0	0	0	0	0	1/4
$n = 1$	0	0	-1	1	0	0	0	0	0
$n = 2$	0	0	0	-1	1	0	0	0	0
$n = 3$	0	0	0	0	-1	1	0	0	0
$n = 4$	0	0	0	0	0	-1	1	0	-1/4

- We can check this using MATLAB's convolution function

```
>> conv([1 1 1 1]/4,[1 -1])

ans =  .25000   0   0   0  -.25000 % first point at n=0
```

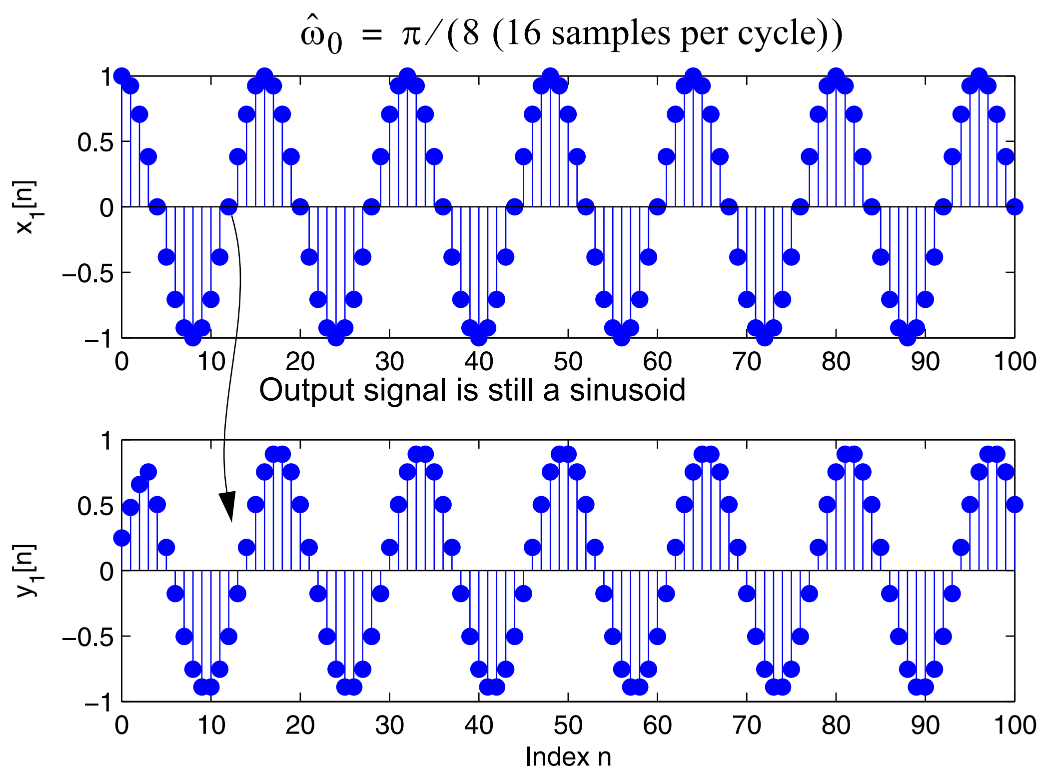
- We can now draw the direct form block diagram corresponding to $h_3[n]$



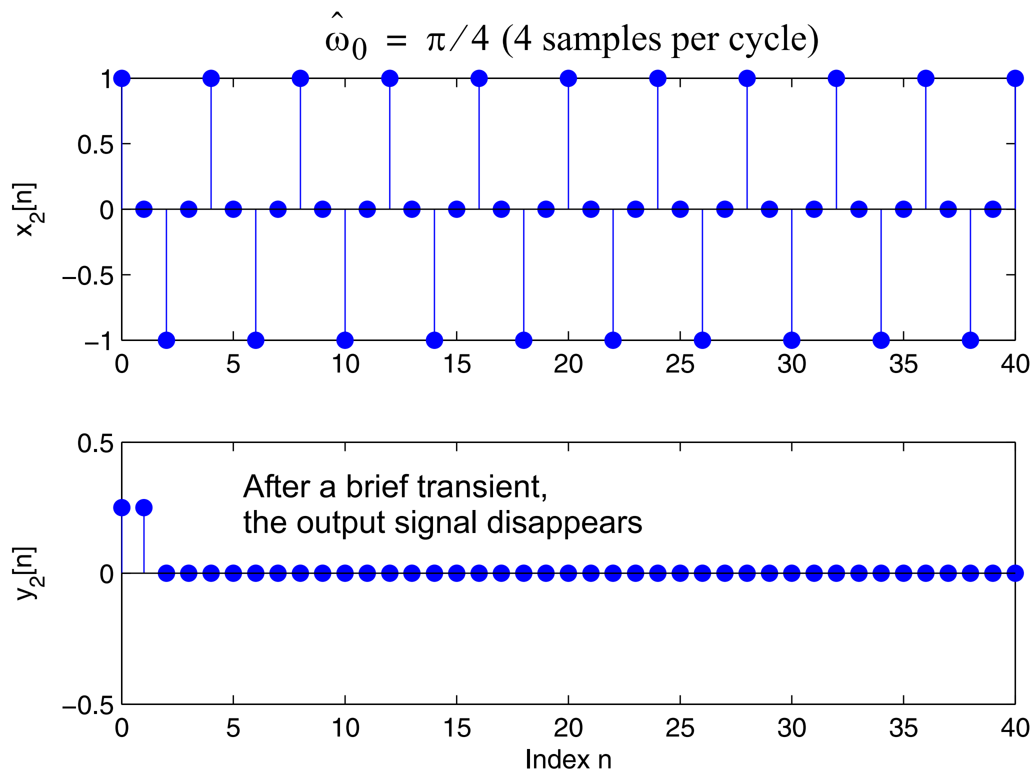
Filtering a Sinusoidal Sequence with a Moving Average Filter

- To provide motivation for studying the frequency response of FIR filters, in this section we will consider the output of a 4-tap ($M = 3$) moving average filter when $x[n] = \cos(\hat{\omega}_0 n)$
- Specifically we will consider $\hat{\omega}_0 = \pi/8$and $\pi/4$

```
>> n = 0:100;
>> h1 = [1 1 1 1]/4;
>> x1 = cos(pi/8*n);
>> x2 = cos(pi/2*n);
>> y1 = filter(h1,1,x1);
>> y2 = filter(h1,1,x2);
```



- The output signal at $\hat{\omega}_0 = \pi/8$ is still a sinusoid, but the amplitude is reduced and there is small amount of phase shift



- For the case of $\hat{\omega}_0 = \pi/4$ the output signal disappears
- This filter must have a special property that allows at least sinusoids of a particular frequency to be blocked
- In Chapter 6 we study the frequency response, and will discover how it is possible for the above to occur