ECE 410 **DIGITAL SIGNAL PROCESSING** **D. Munson**
University of Illinois **Chapter 14**

Fast Fourier Transform (FFT)

FFTs comprise a class of algorithms for quickly computing the DFT.

DFT:

$$X_p = \sum_{n=0}^{N-1} x_n \bullet W_N^{np} \qquad 0 \le p \le N-1$$

$$W_N \triangleq e^{-j\frac{2\pi}{N}}$$

A straightforward computation requires:

$N^2 \otimes, \quad N(N-1) \oplus$

where these multiplications and additions are generally complex.

There are many different FFTs. We will consider only radix-2 decimation-in-time and decimation-in-frequency algorithms.

Radix-2 FFTs, where the sequence length N is restricted to be a power of two, require only $0(N \log_2 N)$ computations.

Decimation-in-Time Radix-2 FFT

Suppose $N = 2^M$

Idea: Divide input sequence into two groups, those elements of $\{x_n\}$ with n even and those with n odd. Then combine the size N/2 DFTs of these two subsequences to calculate the first half of $\{X_m\}_{m=0}^{N-1}$ and the second half of $\{X_m\}_{m=0}^{N-1}$.

Let

$$\left.\begin{aligned} y_n &= x_{2n} \\ z_n &= x_{2n+1} \end{aligned}\right\} \quad 0 \le n \le \frac{N}{2} - 1$$

Show $\{X_p\}_{p=0}^{N-1}$ can be obtained from the $\frac{N}{2}$ point DFTs $\{Y_p\}_{p=0}^{\frac{N}{2}-1}$ and $\{Z_p\}_{p=0}^{\frac{N}{2}-1}$.

Splitting a size N problem into two size $\frac{N}{2}$ problems will reduce computation because

$$\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 = \frac{N^2}{2} < N^2$$

Our strategy will then be to divide each size $\frac{N}{2}$ problem into two size $\frac{N}{4}$ problems, etc.

Derivation Relating X_p to Y_p and Z_p:

$$X_p = \sum_{k=0}^{\frac{N}{2}-1} \left(x_{2k} W_N^{2kp} + x_{2k+1} W_N^{(2k+1)p} \right)$$

$$= \sum_{k=0}^{\frac{N}{2}-1} y_k \; W_{N/2}^{kp} + W_N^p \sum_{k=0}^{\frac{N}{2}-1} z_k \; W_{N/2}^{kp} \qquad (1)$$

since $W_N^{2kp} = e^{-j\frac{2\pi}{N}2kp} = e^{-j\frac{2\pi}{N/2}kp} = W_{N/2}^{kp}$

For p = 0, 1, … , $\frac{N}{2}$ – 1, the first sum in (1) is Y_p, and the second sum is $W_N^p \, Z_p$.

$$\Rightarrow \quad \boxed{X_p = Y_p + W_N^p \, Z_p \qquad 0 \le p \le \frac{N}{2} - 1} \qquad (2)$$

What about X_p for $p > \frac{N}{2} - 1$? We can get these by using (1) to write:

$$X_{p+\frac{N}{2}} = \sum_{k=0}^{\frac{N}{2}-1} y_k \; W_{N/2}^{k\left(p+\frac{N}{2}\right)} + W_N^{p+\frac{N}{2}} \sum_{k=0}^{\frac{N}{2}-1} z_k \; W_{N/2}^{k\left(p+\frac{N}{2}\right)}$$

Note that:

$$W_{N/2}^{k\left(p+\frac{N}{2}\right)} = W_{N/2}^{kp} \; W_{N/2}^{k\frac{N}{2}} = W_{N/2}^{kp} \bullet 1$$

and

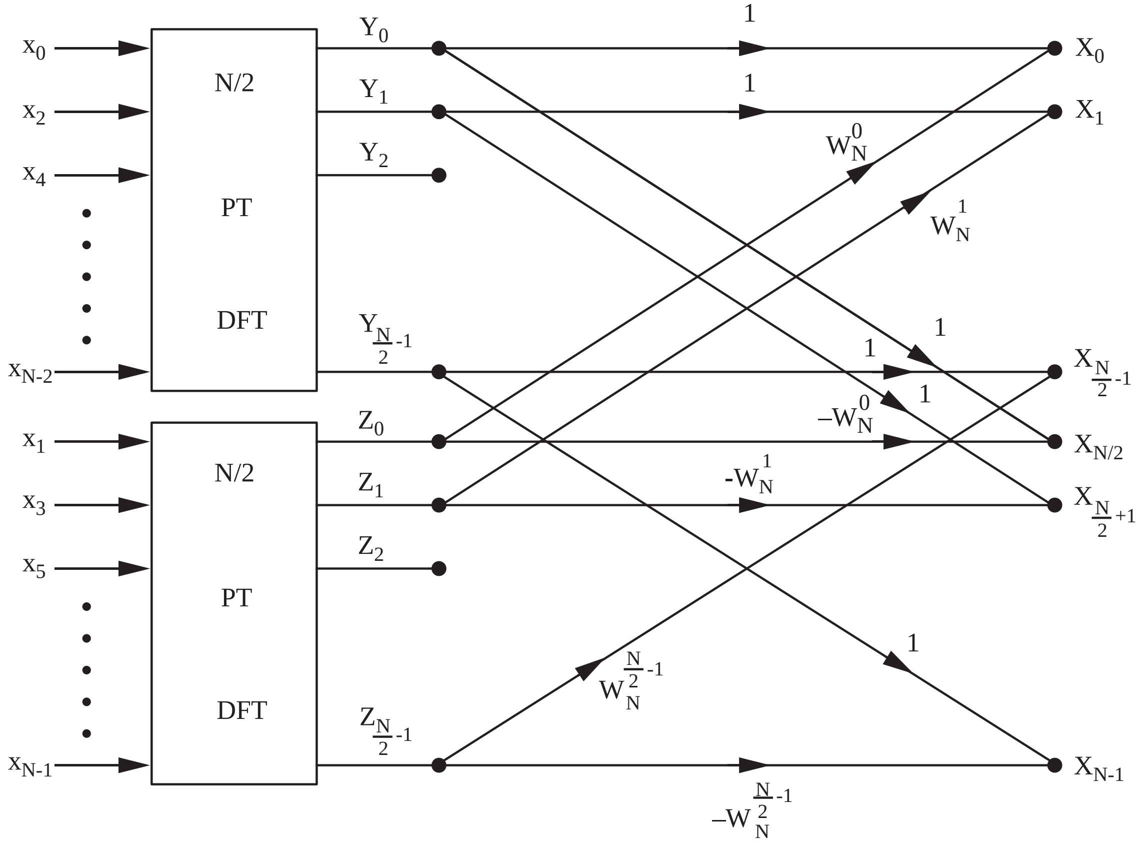
$$W_N^{p+\frac{N}{2}} = W_N^p \; e^{-j\frac{2\pi}{N}\frac{N}{2}} = -W_N^p$$

So:

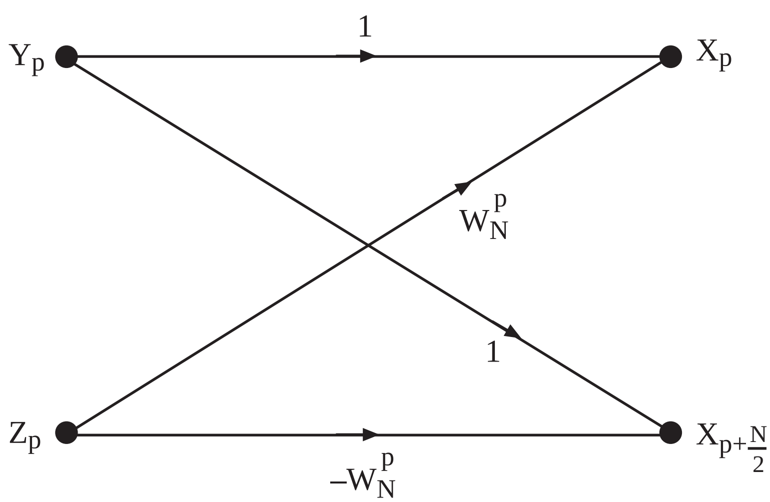
$$W_{p+\frac{N}{2}} = \sum_{k=0}^{\frac{N}{2}-1} y_k \; W_{N/2}^{kp} - W_N^p \sum_{k=0}^{\frac{N}{2}-1} z_k \; W_{N/2}^{kp}$$

$$\Rightarrow \quad \boxed{X_{p+\frac{N}{2}} = Y_p - W_N^p \, Z_p \qquad 0 \le p \le \frac{N}{2} - 1} \qquad (3)$$

(2) and (3) show how to compute an N point DFT using two $\frac{N}{2}$ point DFTs. These two equations are the essence of the FFT and describe the following flow graph:



The operation to combine the $\frac{N}{2}$ point DFT outputs Y_p and Z_p is called a <u>butterfly</u>:



$$0 \le p \le \frac{N}{2} - 1$$

This butterfly diagram summarizes (2) and (3).

Our overall strategy will be to:

Replace the N-point DFT by $\frac{N}{2}$ butterflies <u>preceded</u> by the $\frac{N}{2}$-point DFTs.

Replace each $\frac{N}{2}$-point DFT by $\frac{N}{4}$ butterflies <u>preceded</u> by two $\frac{N}{4}$-point DFTs.

•
•
•
•

Replace each 4-point DFT by two butterflies preceded by two 2-point DFTs.

Replace each 2-point DFT by a single butterfly preceded by two one-point DFTs. But, a one-point DFT is the identity operation, so a two-point DFT is just a single butterfly.

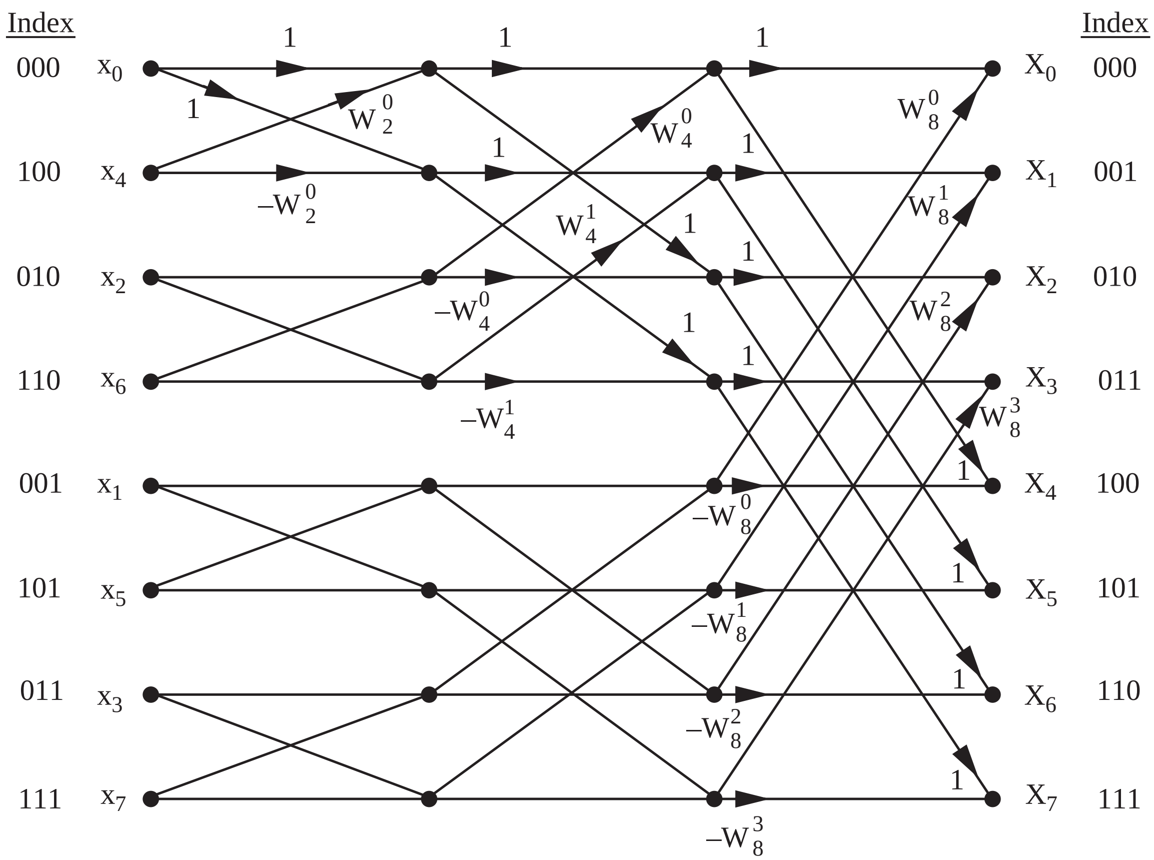
Since $N = 2^M$, this recursion leads to $M = \log_2 N$ stages of $\frac{N}{2}$ butterflies each.

Thus, for a DSP chip that can perform one multiplication and one addition (one multiply-accumulate) in each clock cycle, a radix-2 DIT FFT requires

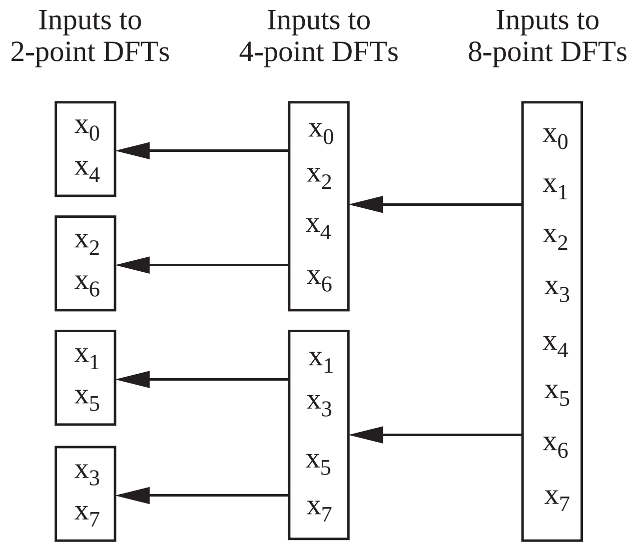
$$N \log_2 N multiply-accumulates$$

which can be far less than the N^2 multiply-accumulates required by a straightforward DFT.

Example (N = 8, DIT FFT)



The input x_n is required in "bit-reversed" order. Why? This follows since to compute an N-point DFT using two N/2 point DFTs, we break up the input into even and odd points. We do this successively as we work backward in the flow diagram:



Note: FFT computation can be performed "in place." We need only one length-N array in memory since the output of a butterfly can be written back into the input locations.

Example ~ computational comparison

Suppose $N = 2^{14} = 16{,}384$.

Compare the number of multiply-accumulates in straightforward and DIT FFT implementations of the DFT.

Straightforward: $N^2 = 268{,}435{,}456$ multiply-accumulates

FFT: $N \log_2 N = 2^{14}\,(14) = 229{,}376$ multiply-accumulates

$$\text{Savings factor} = \frac{268{,}435{,}456}{229{,}376} = 1170!$$

Suppose that in 1964 a state-of-the-art computer required 10 hours to compute a straightforward length 2^{14} DFT. Then, in 1965, after publication of the FFT, this same computation could be performed in about 30 seconds!

Decimation in Frequency Radix – 2 FFT

Idea: Essentially is backwards from DIT. Separate $\{x_n\}_{n=0}^{N-1}$ into first half and second half and then compute even and odd points in $\{X_p\}_{p=0}^{N-1}$ separately, using two $\frac{N}{2}$-point DFTs.

<u>Derivation of algorithm</u>:

$$X_p = \sum_{n=0}^{N-1} x_n W_N^{np}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x_m W_N^{mp} + \sum_{m=0}^{\frac{N}{2}-1} x_{m+N/2} W_N^{(m+N/2)p}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} \left(x_m + x_{m+N/2} W_N^{(N/2)p}\right) W_N^{mp} \qquad (10)$$

Look at even and odd points in X_p separately.

Evens:

(10) $\Rightarrow$

$$X_{2q} = \sum_{m=0}^{\frac{N}{2}-1} \left(x_m + x_{m+N/2} \bullet 1\right) W_{N/2}^{mq}$$

$$\Rightarrow \left[\underbrace{\{X_{2q}\}_{q=0}^{N/2-1}}_{\uparrow} = \underbrace{\text{DFT}}_{\uparrow} \left[\{x_m + x_{m+N/2}\}_{m=0}^{N/2-1} \right] \right] \qquad (11)$$

even points in desired length-N DFT — N/2 point DFT

Odds:

(10) $\Rightarrow$

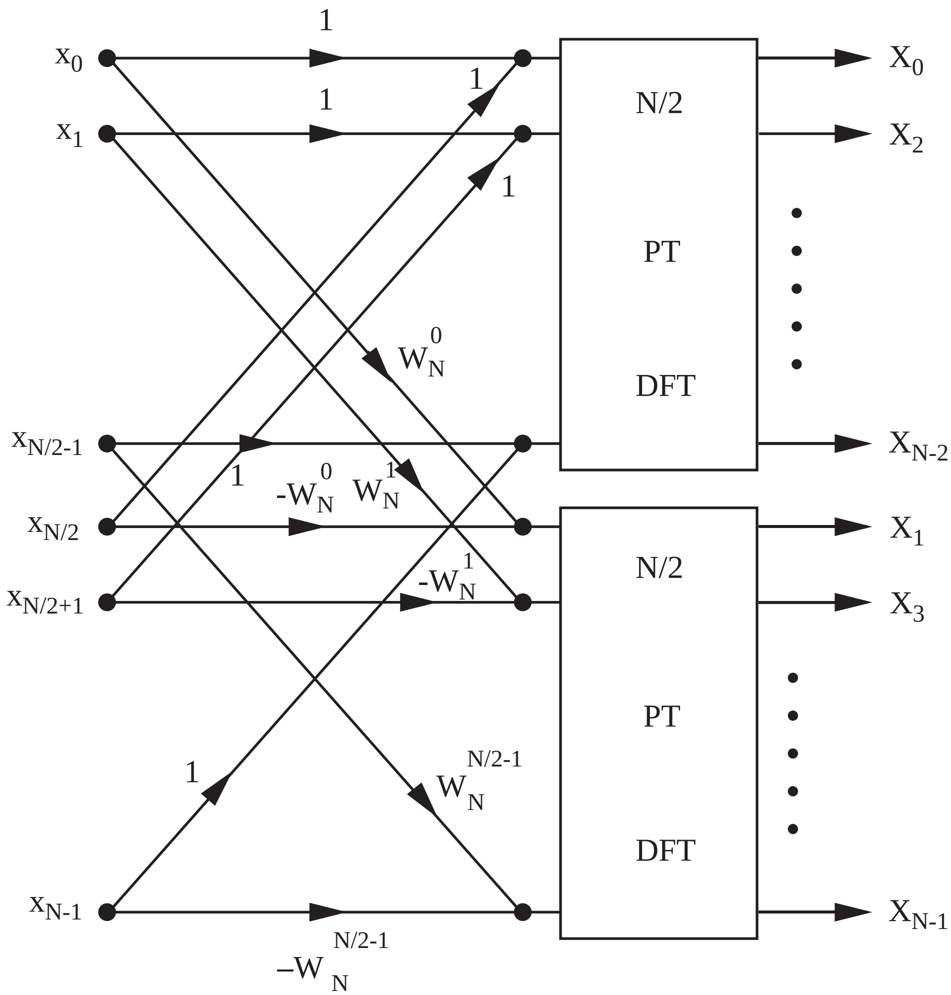
$$X_{2q+1} = \sum_{m=0}^{\frac{N}{2}-1} \left(x_m + x_{m+N/2} W_N^{(N/2)(2q+1)}\right) W_{N/2}^{mq} W_N^m$$

$$= \sum_{m=0}^{\frac{N}{2}-1} \left[\left(x_m - x_{m+N/2}\right) W_N^m\right] W_{N/2}^{mq}$$

$$\Rightarrow \left[\left\{ X_{2q+1} \right\}_{q=0}^{N/2-1} = \mathrm{DFT}\left[\left\{ (x_m - x_{m+N/2}) W_N^m \right\}_{m=0}^{N/2-1} \right] \right] \qquad (12)$$

↑ odd points in desired length-N DFT

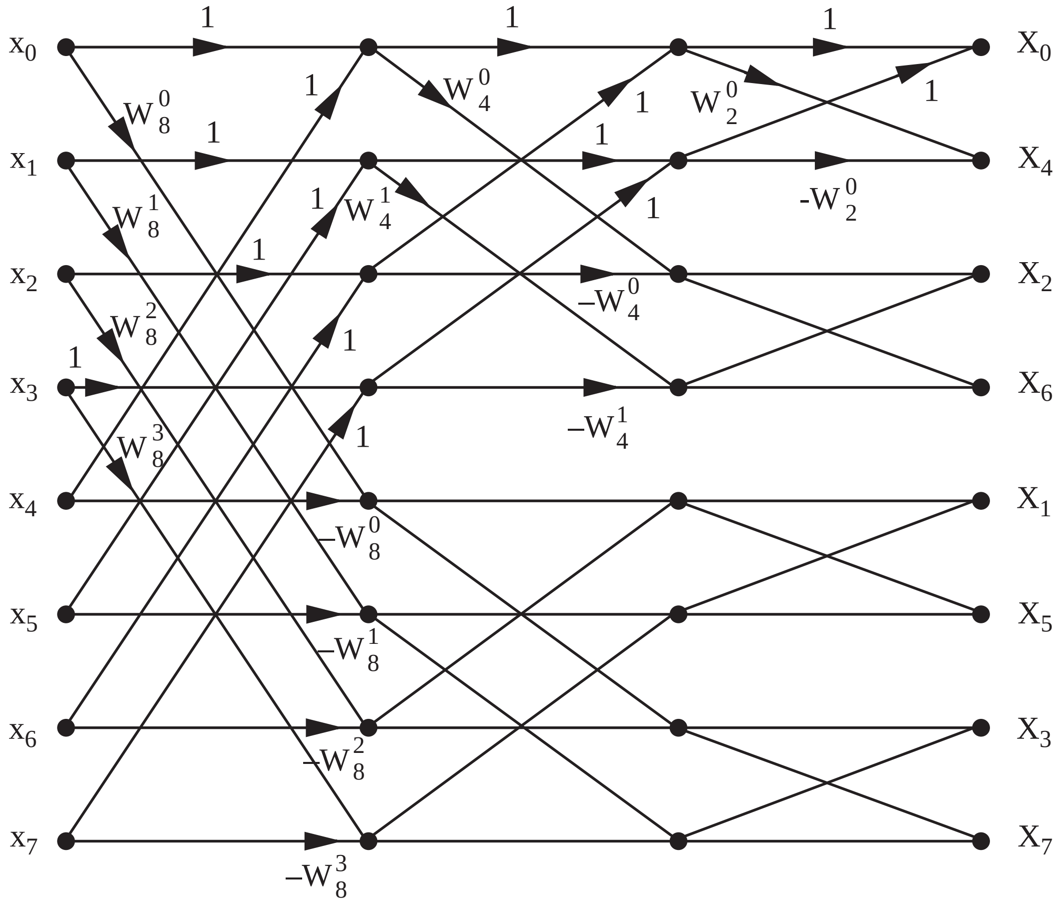
(11) and (12) give:



The complete DIF algorithm computes each $\frac{N}{2}$-point DFT using two $\frac{N}{4}$-point DFTs, etc. As in the DIT algorithm, we get $\log_2 N$ stages of $\frac{N}{2}$ butterflies each, but now the <u>output</u> appears in bit-reversed order.

Example (N = 8, DIF FFT)

14.8



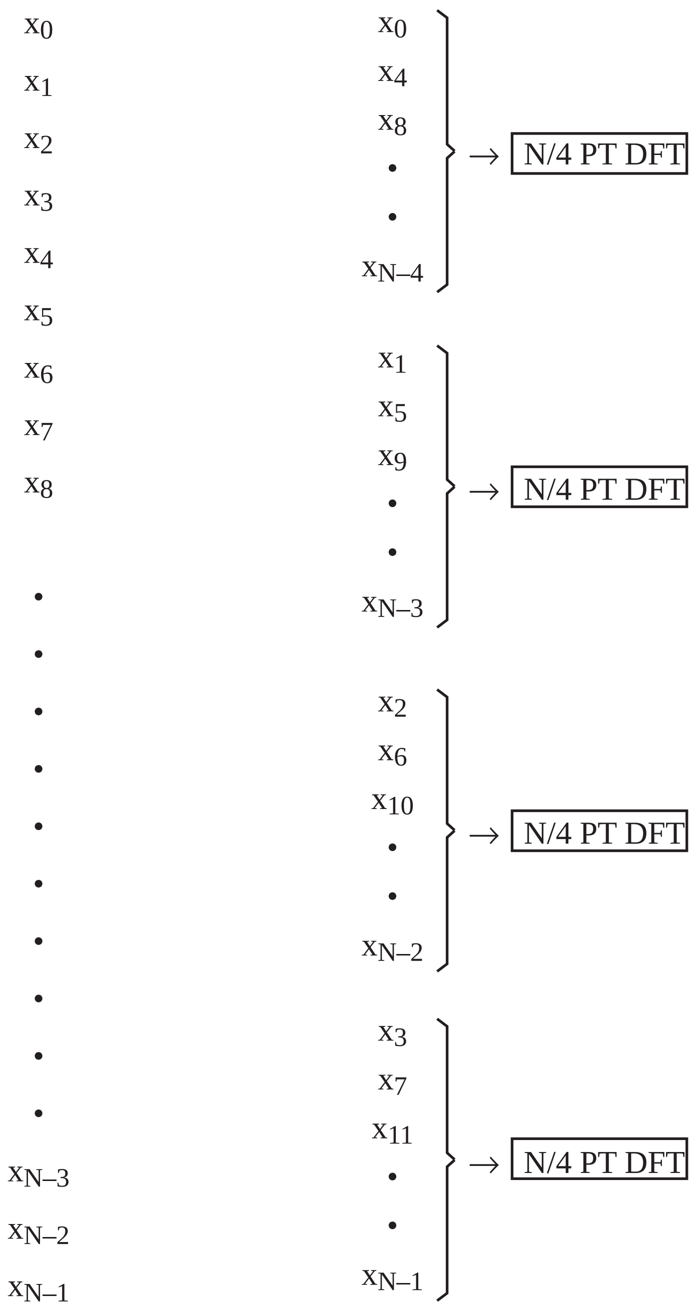
The branch weights are found by using (11) and (12).

Note: As mentioned above, the <u>output</u> appears in bit-reversed order.

Comment: The DIF flow diagram is simply the transpose of the DIT diagram (switch input and output, and reverse all flows).

<u>Other Comments:</u>

1) FFT computer algorithms incorporate the reordering (“bit reversal”) of input or output. You don’t have to do this yourself.
2) Can generalize Radix-2 approach to Radix-3, Radix-4, etc. with $N = 3^M$, $N = 4^M$, etc. For a Radix-4 DIT algorithm, break input up into four groups.

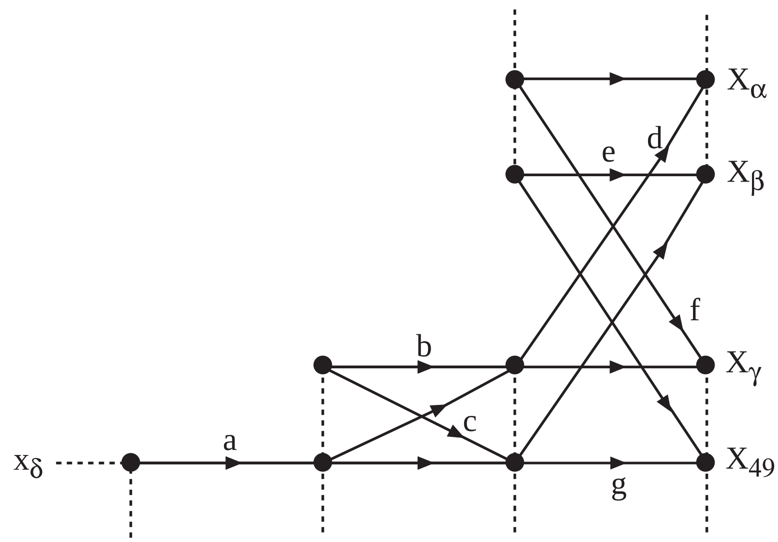


The outputs of the N/4-point DFTs can then be combined, using modified butterflies with 4 inputs and 4 outputs each, to calculate $\{X_m\}_{m=0}^{N-1}$.

Example

Shown below is part of a radix-2, 64-point DIT FFT. Determine the indices α–δ and the coefficients a–g.

14.10



Solution: Use Eqs. (2) and (3) from p. 47.2 in course notes:

$$X_p = Y_p + W_N^p Z_p \qquad 0 \le p \le \frac{N}{2} - 1$$

$$X_{p+\frac{N}{2}} = Y_p - W_N^p Z_p \qquad 0 \le p \le \frac{N}{2} - 1$$

$N = 64,\ \beta + \frac{N}{2} = 49 \ \Rightarrow\ \beta = \underline{17}$

$\gamma = 49 - \frac{N}{4} = \underline{33}$

$\alpha = 33 - \frac{N}{2} = \underline{1}$

δ is bit reversal of $49 = (110001)_2 \ \Rightarrow\ \delta = (100011)_2 = \underline{35}$

$d = W_{64}^1 = e^{-j\frac{2\pi}{64}}$ $\qquad g = -W_{64}^{17} = -e^{-j\frac{34\pi}{64}}$

$e = 1$ $\qquad b = 1$

$f = 1$ $\qquad c = 1$ $\qquad a = 1$ $\qquad \leftarrow$ since this is a top branch in butterfly of 16 pt DFT

Fast Linear Convolution

Recall the cyclic convolution property of the DFT:

$$y_n = \sum_{m=0}^{N-1} h_m \, x_{<n-m>_N} \quad \text{iff} \quad Y_m = H_m X_m \quad 0 \le m \le N-1$$

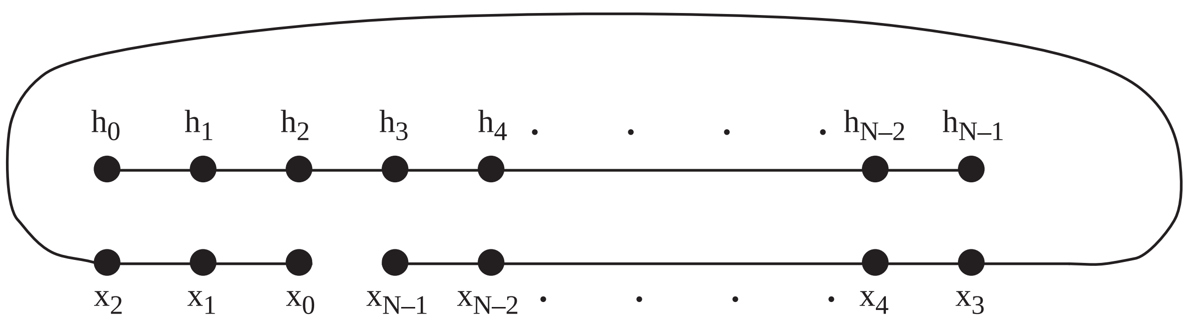
So, we can implement cyclic convolution via

$$\{y_n\} = \text{DFT}^{-1}\Big[\text{DFT}\big[\{h_n\}\big] \bullet \text{DFT}\big[\{x_n\}\big]\Big] \qquad (\Delta\Delta)$$

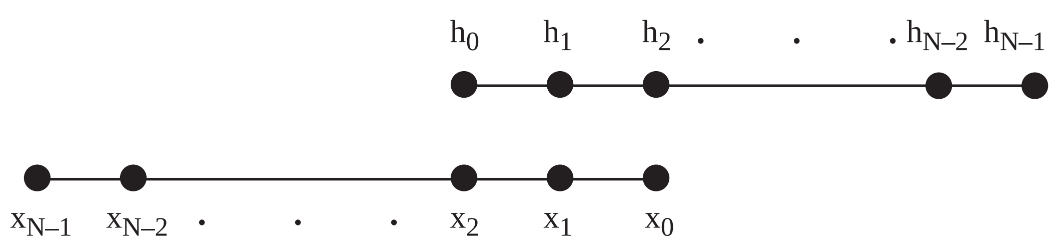
This can be done quickly for long sequence lengths using the FFT.

But, what is cyclic convolution?

To compute y_2:



We would rather implement a linear (regular) convolution:



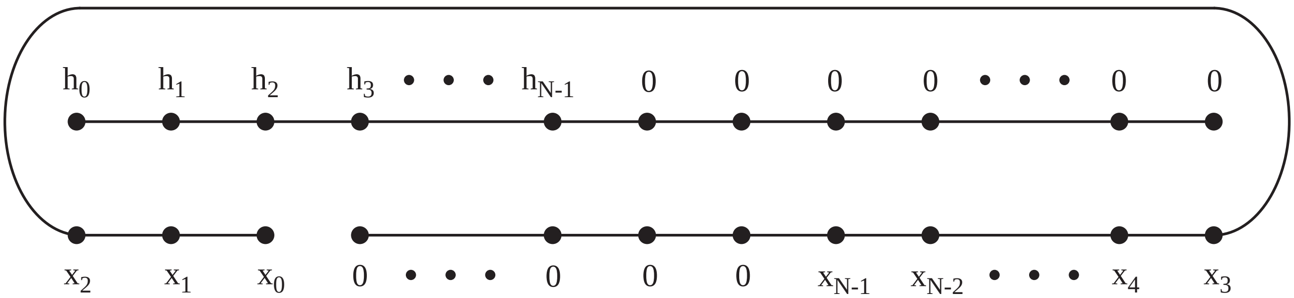
To compute a linear convolution via a cyclic convolution, we must eliminate the wrap-around of nonzero terms in the cyclic convolution. Use zero-padding with N – 1 zeros, i.e., let:

$$\hat{h}_n = \begin{cases} h_n & 0 \le n \le N-1 \\ 0 & N \le n \le 2N-2 \end{cases}$$

$$\hat{x}_n = \begin{cases} x_n & 0 \le n \le N-1 \\ 0 & N \le n \le 2N-2 \end{cases}$$

Now, cyclically convolve the zero-padded sequences.

The result is that $\{\hat{y}_n\}_{n=0}^{2N-2}$ will be a linear convolution of $\{h_n\}_{n=0}^{N-1}$ with $\{x_n\}_{n=0}^{N-1}$. For example, in computing $\hat{y}_2$, we will have:



Obviously, the zero-padding eliminates the wrap-around problem. Using an FFT with (ΔΔ), and zero-padded sequences, provides a fast means of performing linear convolution.

What if $\{h_n\}$ and $\{x_n\}$ are not of the same length?

If $\{h_n\}$ is of length M and $\{x_n\}$ is of length N, then pad each sequence to length N + M – 1 (or nearest larger power of 2 <u>if</u> you are using a radix-2 FFT).

Let's check and see that (ΔΔ), with zero padding, works for a specific example.

Example

$$h_n = \{\underset{\uparrow}{1}, 1, 1\}, \quad x_n = \{\underset{\uparrow}{1}, -1, 1\}$$

To produce a linear convolution via (ΔΔ), first pad each sequence with N – 1 = 2 zeros:

$$\hat{h}_n = \{1, 1, 1, 0, 0\}$$

$$\hat{x}_n = \{1, -1, 1, 0, 0\}$$

Now,

$$\hat{H}_m = \sum_{n=0}^{4} \hat{h}_n \, e^{-j\frac{2\pi}{5}nm}$$

$$= 1 + e^{-j\frac{2\pi}{5}m} + e^{-j\frac{4\pi}{5}m}$$

Likewise,

$$\hat{X}_m = 1 - e^{-j\frac{2\pi}{5}m} + e^{-j\frac{4\pi}{5}m}$$

So,

$$\begin{aligned}\hat{Y}_m = \hat{H}_m \hat{X}_m &= 1 + \cancel{e^{-j\frac{2\pi}{5}m}} + \cancel{e^{-j\frac{4\pi}{5}m}} \\ &\quad - \cancel{e^{-j\frac{2\pi}{5}m}} - \cancel{e^{-j\frac{4\pi}{5}m}} - \cancel{e^{-j\frac{6\pi}{5}m}} \\ &\quad + e^{-j\frac{4\pi}{5}m} + \cancel{e^{-j\frac{6\pi}{5}m}} + e^{-j\frac{8\pi}{5}m} \\ &= 1 + e^{-j\frac{4\pi}{5}m} + e^{-j\frac{8\pi}{5}m} \\ &= 1 + e^{-j\frac{2\pi}{5}2m} + e^{-j\frac{2\pi}{5}4m}\end{aligned}$$

Since

$$\hat{Y}_m = \sum_{n=0}^{4} \hat{y}_n \, e^{-j\frac{2\pi}{5}m}$$

we see that

$$\hat{y}_n = \{1, 0, 1, 0, 1\}$$

It is easy to see that this is the correct <u>linear</u> convolution:

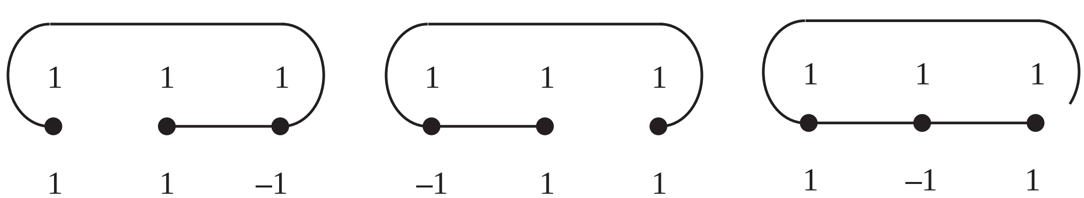
```
          1   1   1
1  –1     1
```

Performing the usual shift and add operations gives the sequence {1, 0, 1, 0, 1}. ✔

Now, what if we had not zero padded?
Then (ΔΔ) would have produced a <u>cyclic</u> convolution.
The cyclic convolution formula is

$$y_n = \sum_{m=0}^{2} h_m \, x_{<n-m>_3}$$

which is computed pictorially as



$y_0 = 1$ $y_1 = 1$ $y_2 = 1$

Let's check that (ΔΔ) without zero-padding gives this same result.

$$H_m = \sum_{n=0}^{2} h_n \, e^{-j\frac{2\pi}{3}nm}$$

$$= 1 + e^{-j\frac{2\pi}{3}m} + e^{-j\frac{4\pi}{3}m}$$

$$X_m = 1 - e^{-j\frac{2\pi}{3}m} + e^{-j\frac{4\pi}{3}m}$$

$$Y_m = H_m X_m$$

$$= 1 + \cancel{e^{-j\frac{2\pi}{3}m}} + \cancel{e^{-j\frac{4\pi}{3}m}}$$

$$- \cancel{e^{-j\frac{2\pi}{3}m}} - \cancel{e^{-j\frac{4\pi}{3}m}} - \cancel{e^{-j\frac{6\pi}{3}m}}$$

$$+ e^{-j\frac{4\pi}{3}m} + \cancel{e^{-j\frac{6\pi}{3}m}} + e^{-j\frac{8\pi}{3}m}$$

Interchanging the latter two terms and using 2π periodicity of the complex exponential gives

$$Y_m = 1 + e^{-j\frac{2\pi}{3}m} + e^{-j\frac{2\pi}{3}2m}$$

Matching up terms with

$$Y_m = \sum_{n=0}^{2} y_n \, e^{-j\frac{2\pi}{3}nm} = y_0 + y_1 \, e^{-j\frac{2\pi}{3}m} + y_2 \, e^{-j\frac{2\pi}{3}2m}$$

gives

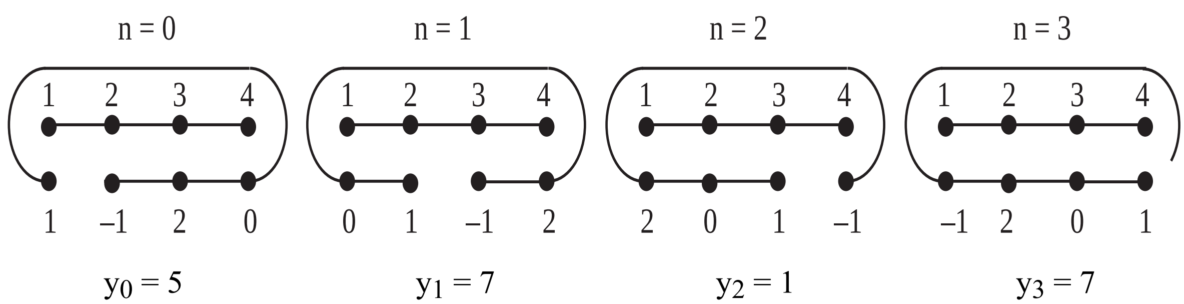
$$\{y_n\} = \{\underset{\uparrow}{1}, 1, 1\} \qquad ✔$$

Note: We worked through this example to show that (ΔΔ) can give a linear convolution or a cyclic convolution, depending on whether we first zero pad. In practice, if N is large the DFTs and inverse DFT would be computed using FFTs. If N is small, then it is faster to perform the convolution in the sequence domain.

For practice at computing cyclic convolution in the sequence domain, consider the following example.

Example

Find $y_n = h_n \circledast x_n$ where $\{h_n\}_{n=0}^{3} = \{1, 2, 3, 4\}$ and $\{x_n\}_{n=0}^{3} = \{1, 0, 2, -1\}$.



Example (Convolution via FFT)

Suppose that a sequence $\{x_n\}_{n=0}^{7000}$ is to be filtered with an FIR filter having coefficients $\{h_n\}_{n=0}^{1100}$.

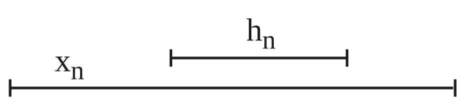
a) Ignoring possible savings from coefficient symmetry, what is the total number of multiplications required to compute the output $\{y_n\}_{n=0}^{8100}$ by implementing the usual convolution formula with a direct-form filter structure?

b) Using the FFT method (with a radix-2 FFT and zero-padding to length 8192), how many complex multiply-accumulates (MAs) are required to compute $\{y_n\}_{n=0}^{8100}$? How many real MAs are required? (For simplicity, count <u>all</u> “multiplications” in an FFT, even those by ±1, ±j, as complex multiplications.)

Solution

a) Output of regular convolution is composed of 3 parts:

\# of MAs is $1 + 2 + 3 + \ldots + 1100 = \dfrac{(1100)(1101)}{2}$

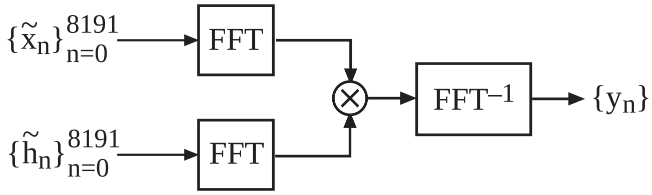


\# of MAs is 1101 (7001 – 1100)

\# of MAs is $1100 + 1099 + \ldots + 1 = \dfrac{(1100)(1101)}{2}$

Total # MAs = 1100(1101) + 1101(5901) = $\boxed{7{,}708{,}101}$

b) FFT method is



where $\{\tilde{x}_n\}$ and $\{\tilde{h}_n\}$ are zero-padded versions of $\{x_n\}$ and $\{h_n\}$.

complex MAs = $3\ (N \log_2 N) + N = 3\ (8192 \bullet 13) + 8192 =$ $\boxed{327,680}$

The number of real MAs required to implement a complex MA is generally 4. To see this, write out the detailed calculation (a) (b) + c where a, b, and c are all complex. Assuming this factor of 4 overhead, we have

real MAs = 4 (327, 680) = $\boxed{1,310,720}$

Thus, in this example the FFT approach requires fewer than 20% of the MAs required by a straightforward convolution.

<u>Block Convolution</u>

Given $\{x_n\}_{n=0}^{N-1}$ and $\{h_n\}_{n=0}^{M-1}$, we have developed an approach for efficiently computing $y_n = h_n * x_n$ using zero-padding and FFTs. But, what if N >> M? If N, the length of the input, is really large, we are faced with two problems:

1)Very long FFTs will be required, which will lead to computational inefficiency.

2)There will be a very long delay in computing $\{y_n\}$ since our scheme requires that all of $\{x_n\}$ be acquired before any element in the output sequence can be computed.

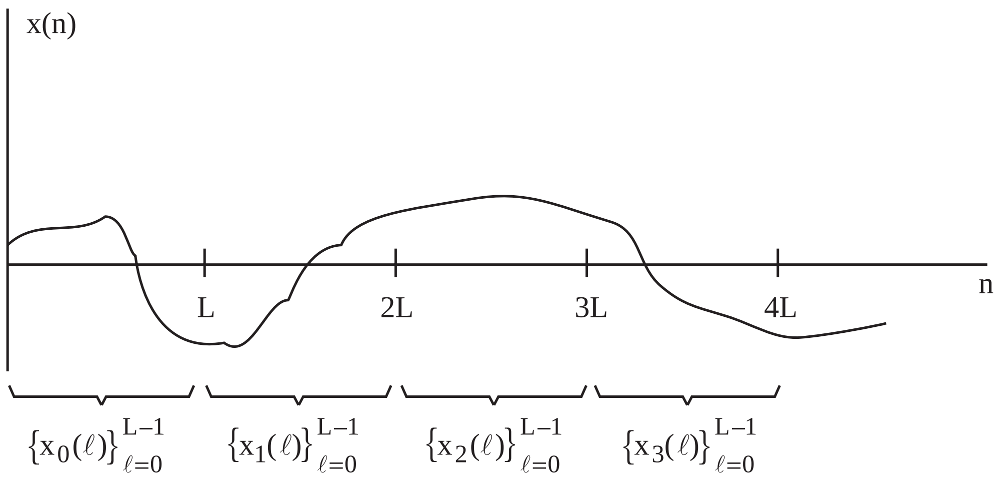
What to do? Answer: Segment the long input $\{x_n\}_{n=0}^{N-1}$ into shorter pieces, convolve the individual pieces with $\{h_n\}_{n=0}^{M-1}$ and then stitch together the results of the shorter convolutions to form $\{y_n\}$. There are two popular ways of doing this.

Method 1: Overlap and Add

Here, we divide up the input into nonoverlapping sections of length L. Let

$$x_k(\ell) = x(kL + \ell) \qquad 0 \le \ell \le L-1, \quad k = 0, 1, 2, \ldots$$

Picture:



We have

$$x(n) = \sum_k x_k(n - kL) \qquad 0 \le n \le N - 1.$$

Now, convolution is a linear operation, so

$$y(n) = h(n) * x(n) = h(n) * \left[\sum_k x_k(n - kL)\right] = \sum_k h(n) * x_k(n - kL)$$

Let $y_k(n) = h(n) * x_k(n)$. Then by shift-invariance,

$$y(n) = \sum_k y_k(n - kL) \ . \qquad (1)$$

We compute each $\{y_k(n)\}$ via the FFT as in the previous lecture. For simplicity, assume $M + L - 1$ is a sequence length for which we have an FFT algorithm. Then

1) Pad $\{x_k(n)\}_{n=0}^{L-1}$ with M–1 zeros to give $\{\tilde{x}_k(n)\}_{n=0}^{L+M-2}$.

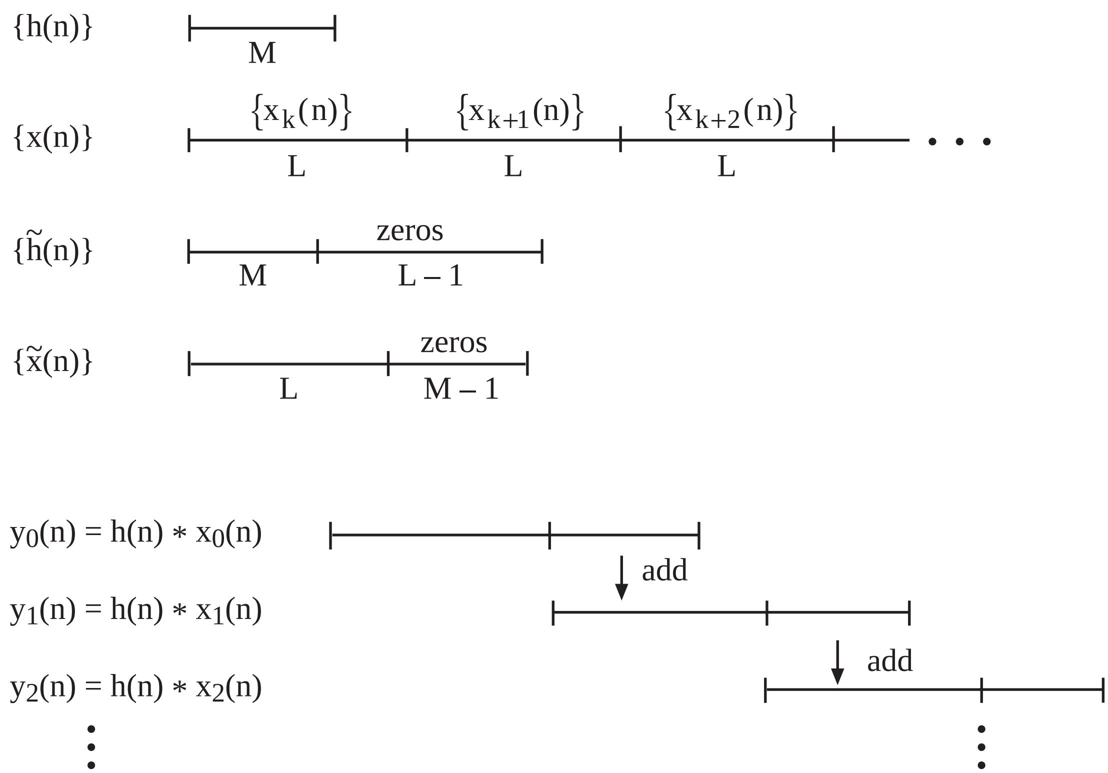
   Pad $\{h(n)\}_{n=0}^{M-1}$ with L–1 zeros to give $\{\tilde{h}(n)\}_{n=0}^{L+M-2}$.

2) Calculate the FFTs of $\{\tilde{x}_k(n)\}_{n=0}^{L+M-2}$ and $\{\tilde{h}(n)\}_{n=0}^{L+M-2}$.

3) Multiply FFTs together and take FFT^{-1} to give $\{y_k(n)\}_{n=0}^{L+M-2}$.

Finally, calculate $\{y(n)\}$ via (1) by adding together the appropriately shifted $\{y_k(n)\}$.

Pictorially:



Sum of the shifted (by kL) $y_k(n)$ gives $\{y(n)\}$.

Example

Given $\{h(n)\}_{n=0}^{249}$ and $\{x(n)\}_{n=0}^{\infty}$ we wish to compute $\{y(n)\} = \{h(n)\} * \{x(n)\}$ using the FFT method. What is the best block length L, using the Overlap and Save method with radix-2 FFTs?

We have $M = 250$. Let K = FFT length. Then since $K = L + M - 1$, the block length will be $L = K - 249$. Each length-K FFT and inverse FFT requires $K \log_2 K$ MAs. Multiplication of FFTs requires K MAs. We shall assume that the FFT of $\{\tilde{h}(n)\}$ is precomputed once and stored. Thus, the amount of computation for each input block will be

$$2\,K \log_2 K + K = K\,(2\log_2 (K) + 1) \qquad \text{MAs.}$$

This amount of computation is needed to compute each $\{y_k(n)\}$ $k = 0, 1, 2, \ldots$ from each input block $\{x_k(n)\}$ of length $L = K - 249$. Thus, the computation per input sample (or per output sample), ignoring the few additions needed to sum the overlapping $\{y_k(n)\}$ blocks, is

$$\frac{K\left[2\log_2 K+1\right]}{K-249} \qquad (2)$$

Trying some different values for the FFT length K, we find:

K	L	Complex MAs Per output
256	7	621.7
512	263	37.0
1024	775	27.7
2048	1799	26.2
4096	3847	26.6

K = FFT length

L = input block length

MAs given by (2)

For larger K, (2) approaches ($2\log_2 K$) + 1, which grows with K.

Even allowing for the required complex arithmetic (4 real MAs per complex MA), the FFT approach offers considerable savings over a direct filter implementation, which would require 250 MAs per output.

<u>Notes</u>:

1)Based on the above table, and if we are at all concerned about delay, we would select an FFT block length of either 512 or 1024.

2)If $\{x_n\}$ and $\{h_n\}$ were complex-valued, then the direct filter implementation would require roughly 1000 MAs per filter output.

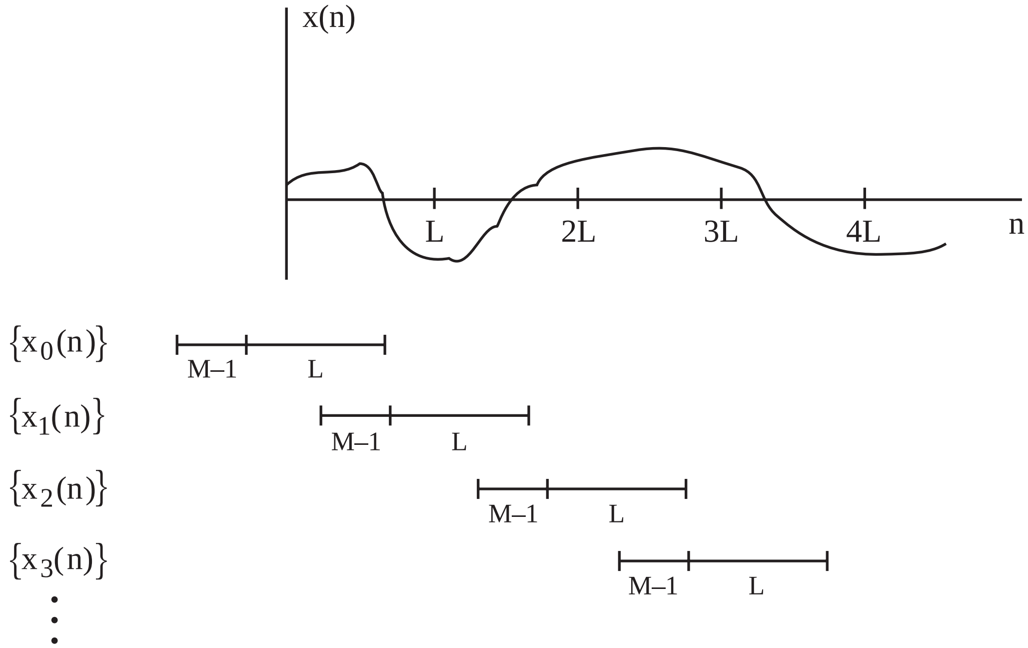
3)If a sequence is real, there are tricks that can be used to speed up computation (by a factor of approximately two) of its DFT. If both {x(n)} and {h(n)} are real, in which case {y(n)} is real, these tricks can be used to reduce the number of MAs in the FFT approach by nearly a factor of two over the entries shown in the above table.

Method 2: Overlap and Save

Could just as easily be called Overlap and Discard.

Here, we define the $\{x_k(n)\}$ to be overlapping as shown below.

14.20



The first M–1 entries of $\{x_0(n)\}$ are filled with zeros. All other entries of $\{x_0(n)\}$ and all entries of all other subsequences $\{x_k(n)\}$ are filled with the values of $\{x(n)\}$ directly above. In general, each subsequence overlaps with its two neighboring subsequences. The algorithm to calculate $\{y(n)\}$ is then:

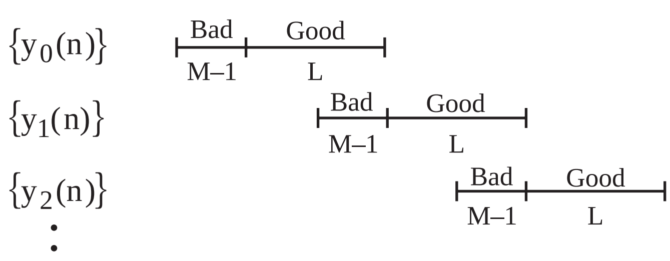
1)Zero-pad $\{h(n)\}_{n=0}^{M-1}$ with L–1 zeros to produce $\{\tilde{h}(n)\}_{n=0}^{M+L-2}$.

2)Cyclically convolve (via FFT) $\{\tilde{h}(n)\}_{n=0}^{M+L-2}$ with each $\{x_k(n)\}_{n=0}^{M+L-2}$ to give

$$y_k(n) = \tilde{h}(n) \circledast x_k(n). \qquad 0 \le n \le M + L - 2$$

The result is that the first M–1 samples of each $\{y_k(n)\}$ will be useless, but the last L samples will be samples of $\{y(n)\}$.

3)Assemble $\{y(n)\}$ as shown:



The "bad" samples are discarded and the "good" samples are concatenated to form $\{y(n)\}$.