

DSP Hardware Laboratory

Class 6

Introduction to FFT

Cross-Correlation

- If we want to determine if the frequency of a sampled continuous-wave signal (i.e., we know the signal is a sine wave) matches a frequency of interest, a technique we could use is a simplified form of a discrete cross-correlation:
 - $$r = \sum_{n=0}^{N-1} x[n]h[n]$$
 - $x[n]$ is the sampled waveform from $[0:N-1]$
 - $h[n]$ is a sine-wave of the frequency we are interested in, of length $[0:N-1]$ (since our “matched filter” is the length of our time domain sequence, we don’t convolve like standard cross-correlation)

Cross-Correlation Example

MATLAB Experiment #1

```
fs = 8000;  
fc = 1;  
fOther = 2;  
t = linspace(0, 1-1/fs, fs);
```

```
sig_in = cos(2*pi*t*fc);  
sig_out = cos(2*pi*t*fc + pi/2);  
h = cos(2*pi*t*fc);
```

$x[n]$

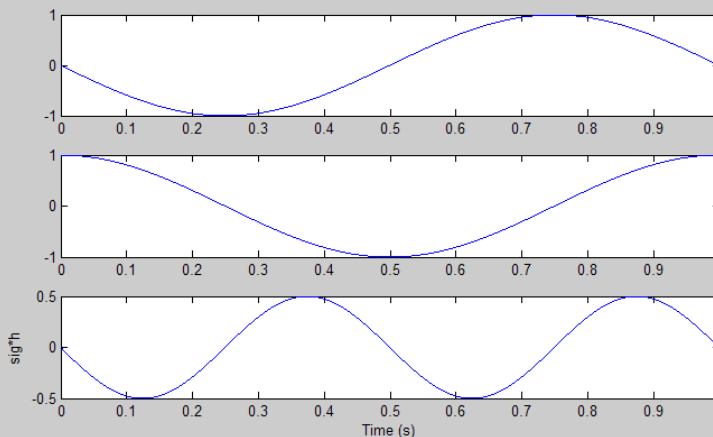
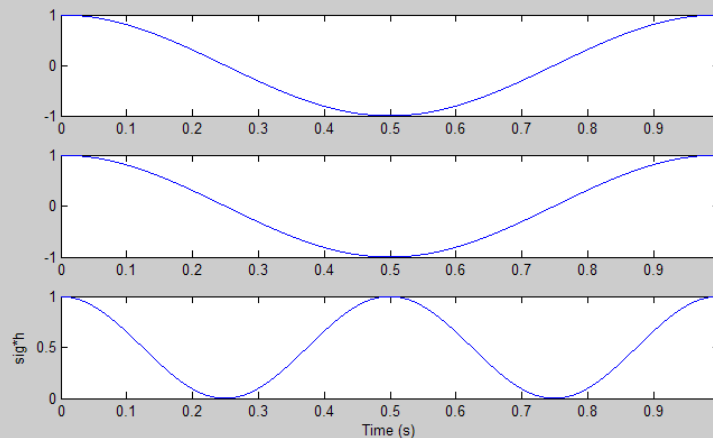
$h[n]$

$x[n]*h[n]$

$x[n]$

$h[n]$

$x[n]*h[n]$

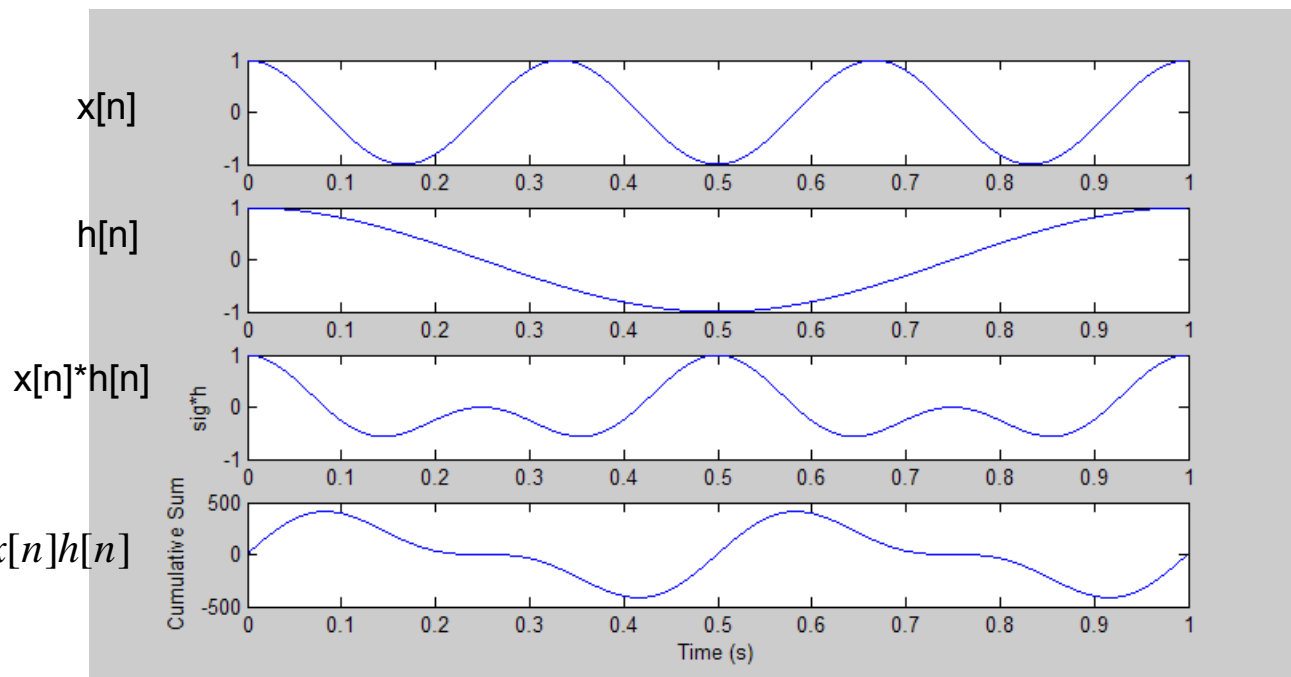


Cross-Correlation Example (2)

MATLAB Experiment #2

```
fs = 8000;  
fc = 1;  
fOther = 2;  
t = linspace(0,1-1/fs,fs);  
  
h = cos(2*pi*t*fc);  
sig = cos(2*pi*t*(fc+fOther));
```

$$r = \sum_{n=0}^{N-1} x[n]h[n]$$



Cross-Correlation, Cosine/Sine

MATLAB Experiment #3

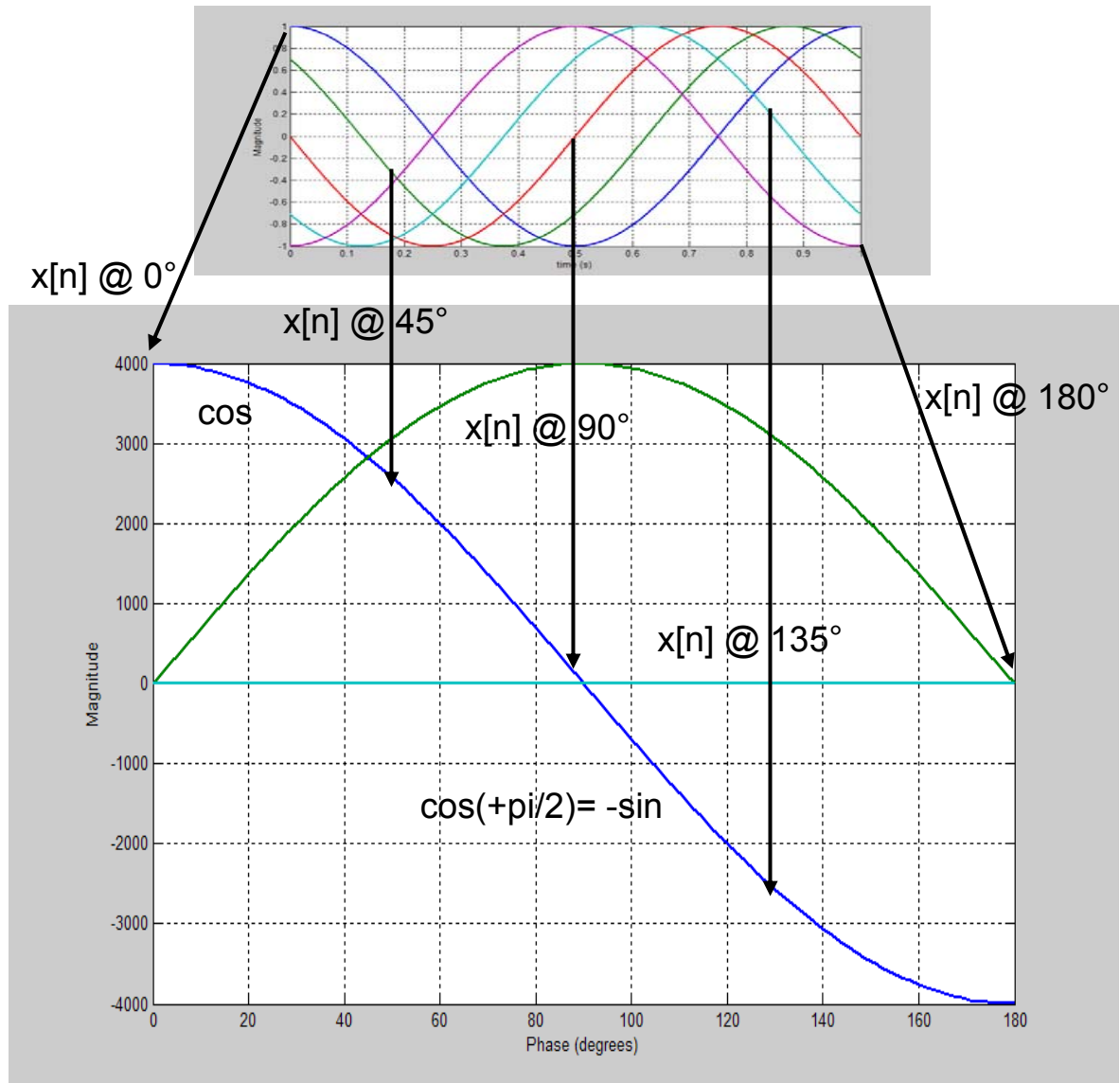
```
h_i = cos(2*pi*t*fc);
h_q = cos(2*pi*t*fc+pi/2);
```

```
for ph = phases
```

```
    sig = cos(2*pi*t*fc+ph);
    mags_1(ph==phases,1) = sum(sig.*h_i);
    mags_1(ph==phases,2) = sum(sig.*h_q);
```

```
    sig = cos(2*pi*t*(fc+fOther)+ph);
    mags_2(ph==phases,1) = sum(sig.*h_i);
    mags_2(ph==phases,2) = sum(sig.*h_q);
```

```
end
```



Discrete Fourier Transform (2)

From the previous discussion, we can then understand intuitively why the DFT is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

where the W 's (also known as “twiddle factors”) are defined as:

$$W_N^{nk} \equiv e^{-j2\pi nk/N} = \cos(2\pi nk/N) - i \sin(2\pi nk/N)$$

Just to make sure we are clear so far, each k (i.e., bin of the DFT), requires the following calculation:

$$X(k) = x(0)W_N^{0k} + x(1)W_N^{1k} + x(2)W_N^{2k} + \dots + x(n)W_N^{(N-1)k}$$

Note that for each k we need N complex multiplies (i.e., $a+jb * c+jd = ac + jad + jbc - bd$), and $N-1$ complex additions. This means that computing the DFT is order is $O(N^2)$ – calculating the DFT will quickly become prohibitive as N increases!

Decimation in Frequency (DIF) FFT

- DFTs can be decomposed using DFTs of even and odd points, which is called a **Decimation-In-Time (DIT)** FFT, or they can be decomposed using a first-half/second-half approach, which is called a **Decimation-In-Frequency (DIF)** FFT. Generally, the user does not need to worry which type is being used.

Decimation in Frequency (DIF) FFT

- **How does the FFT work?**

- By making use of periodicities in the sine-waves that are multiplied with the input sequence to perform the transforms, the FFT greatly reduces the amount of calculation required. Here's a little overview:
- Functionally, the FFT decomposes the set of data to be transformed into a series of smaller data sets to be transformed.
- Then, it decomposes *those* smaller sets into even *smaller* sets. At each stage of processing, the results of the previous stage are combined in special way.
- Finally, it calculates the DFT of each small data set.
 - For example, an FFT of size 32 is broken into 2 FFT's of size 16, which are broken into 4 FFT's of size 8, which are broken into 8 FFT's of size 4, which are broken into 16 FFT's of size 2. Calculating a DFT of size 2 is trivial.

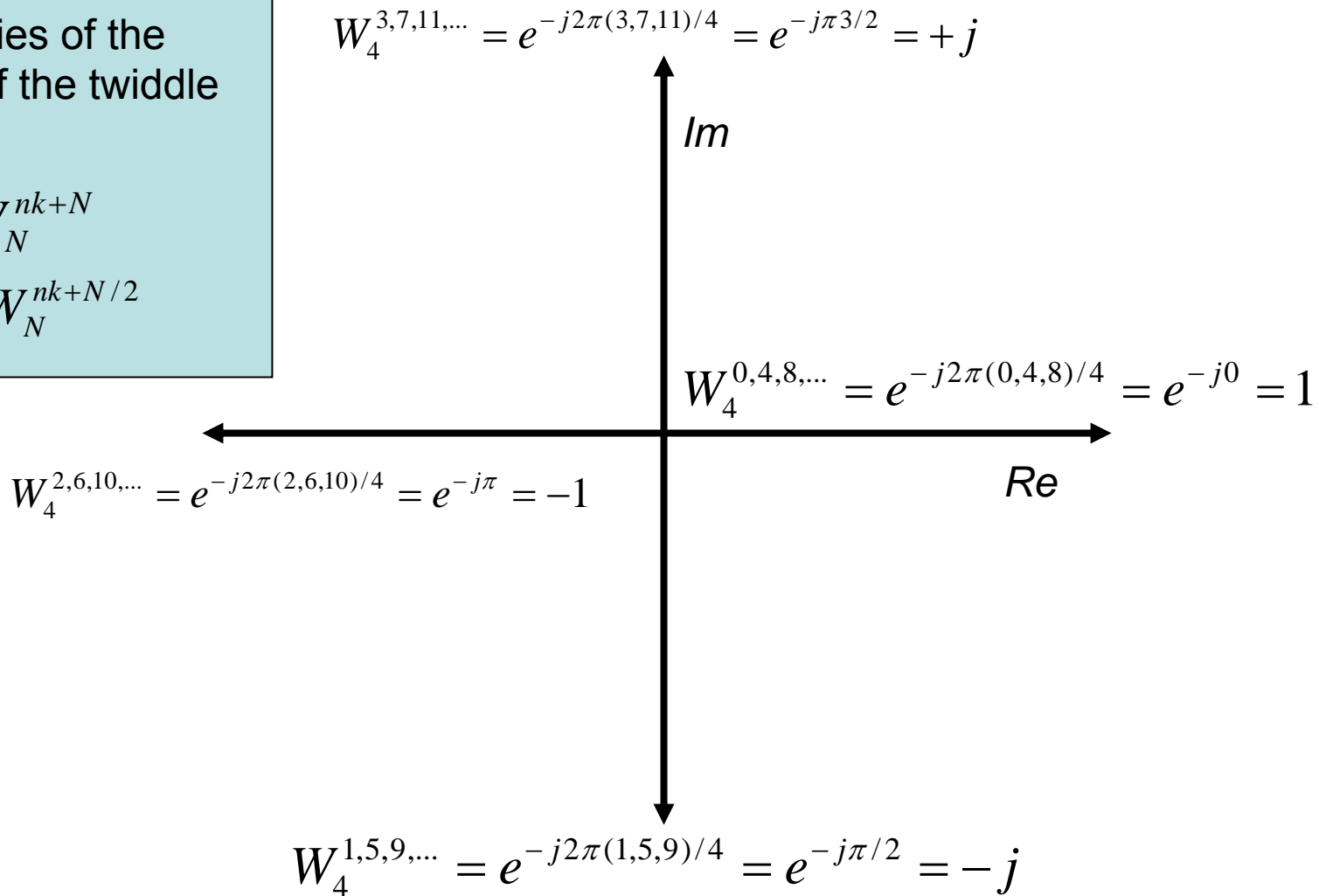
Some text from: <http://www.dspguru.com/info/faqs/fftfaq.htm>

Periodicity of Twiddle Factors

Two properties of the periodicity of the twiddle factors:

$$W_N^{nk} = W_N^{nk+N}$$

$$W_N^{nk} = -W_N^{nk+N/2}$$



FFT Speed

- As opposed to an $O(N^2)$ operation like the DFT, the FFT is an $O(N \cdot \log(N))$ operation:
 - 8192 point DFT: around 67,100,000 multiplies
 - 8192 point FFT: around 32,000 multiplies

Decimation in Frequency FFT (1)

Start with the definition of the DFT, which is:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

Consider the sequence of samples:

$$x(0:N-1) = x(0), \dots, x(N-1)$$

Divide this sequence into 0:N/2-1 and N/2:N-1:

$$x(0:N) = x(0), x(1), x(2), \dots, x(N-1)$$

$$x(N/2:N) = x(N/2-1), \dots, x(N-1)$$

From this we can write the DFT as two summations:

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n)W_N^{nk}$$

Decimation in Frequency FFT (2)

From the DFT as two summations:

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n)W_N^{nk}$$

In the second summation, let:

$$n = n + N/2$$

Working through, we get:

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n + N/2)W_N^{(n+N/2)k}$$

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2-N/2}^{N-1-N/2} x(n + N/2)W_N^{(kn+kN/2)}$$

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + W_N^{(N/2)k} \sum_{n=0}^{N/2-1} x(n + N/2)W_N^{nk}$$

Consider the twiddle factors that are not a function of n :

$$W_N^{(N/2)k} = e^{-j2\pi(N/2)k/N} = e^{-j\pi k}$$

$$e^{-j\pi k} = (\cos(\pi) - j\sin(\pi))^k = \underline{-1^k = 1, -1, 1, -1, \dots}$$

Thus, we can rewrite the summation above as:

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + (-1)^k \sum_{n=0}^{N/2-1} x(n + N/2)W_N^{nk}$$

Decimation in Frequency FFT (3)

Alright, we have written our original summation as the sum of two shorter summations...let's take advantage of it!

$$X(k) = \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + -1^k \sum_{n=0}^{N/2-1} x(n + N/2)W_N^{nk}$$

Combining summations:

$$X(k) = \sum_{n=0}^{N/2-1} [x(n) + -1^k x(n + N/2)]W_N^{nk}$$

Because of the periodicity of W ("hidden" here), we now have rewritten the summation to take $N/2$ complex multiplies for each k ...we are starting to see some benefit!

Let's separate this summation into two summations, one for even values of k , and one for odd values of k :

$$X(k) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)]W_N^{nk} \quad \text{for } k=0,2,4,\dots,N-1$$

$$X(k) = \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)]W_N^{nk} \quad \text{for } k=1,3,5,\dots,N-1$$

Note that we have not yet accomplished our real goal, of decomposing the original DFT into two $N/2$ DFT. The two summations above are not quite right, since $n=[0,N/2-1]$, but our twiddle factors are W_N , not $W_{N/2}$.

Decimation in Frequency FFT (4)

The goal is to form two smaller DFTs (so that we can reapply this technique). By replacing k with $2k$ for the even and k with $2k+1$ for the odd, we can accomplish this:

$$X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)] W_N^{n2k} \quad \text{for } k=0,1,2,\dots,N/2-1$$

$$X(2k+1) = \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)] W_N^{n(2k+1)} \quad \text{for } k=0,1,2,\dots,N/2-1$$

After this replacement, we can rewrite the twiddle factors into the appropriate form for a $N/2$ DFT:

$$W_N^{n2k} = e^{-j2\pi n2k/N} = e^{-j2\pi nk/(N/2)} = W_{N/2}^{nk}$$

$$g(n) = x(n) + x(n + N/2)$$

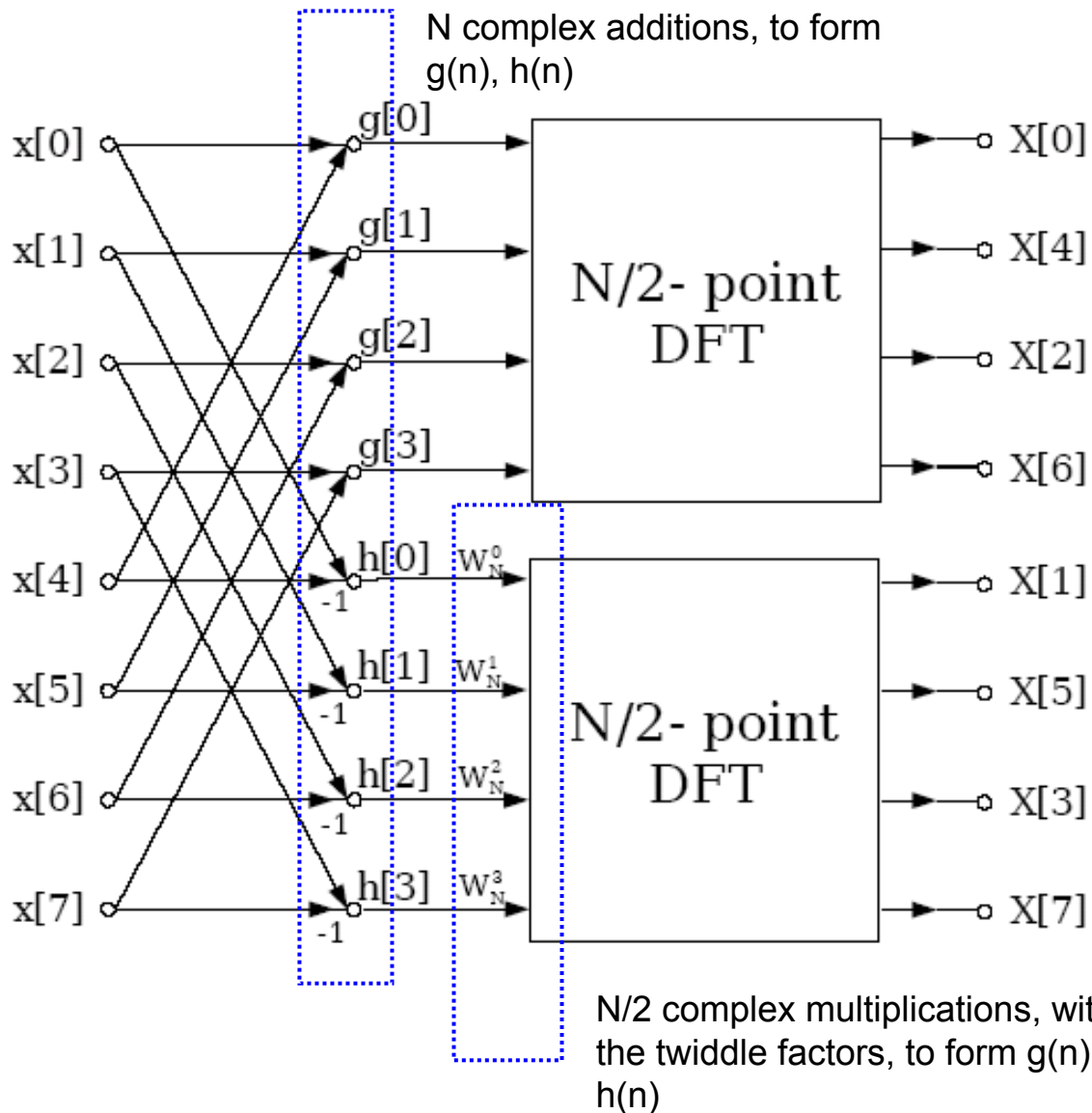
$$h(n) = x(n) - x(n + N/2)$$

$$X(2k) = \sum_{n=0}^{N/2-1} g(n) W_{N/2}^{nk} \quad \text{for } k=0,1,2,\dots,N/2-1$$

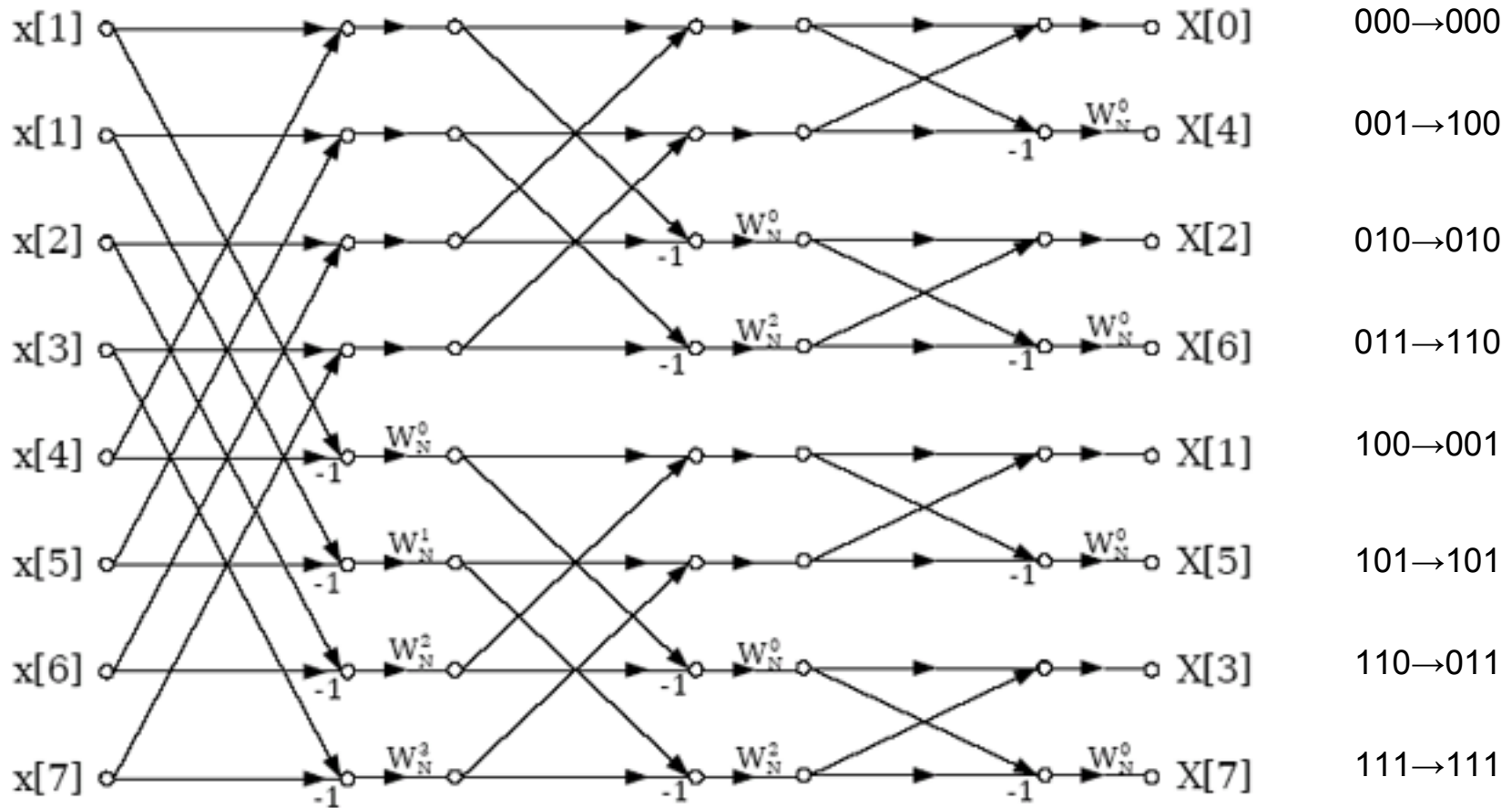
$$X(2k+1) = W_N^n \sum_{n=0}^{N/2-1} h(n) W_{N/2}^{nk} \quad \text{for } k=0,1,2,\dots,N/2-1$$

We have now decomposed our N point FFT into: N complex additions (to form $a(n)$ and $b(n)$), $N/2$ complex multiplications, and two $N/2$ DFTs. Real gains can be had if we apply this derivation to the two $N/2$ DFTs.

Decimation in Frequency FFT (5)



Decimation in Frequency FFT (6)



Some FFT Facts (1)

- We can view the FFT as a bank of filters (or correlators) at the frequencies between $[0, f_s)$
- The number of filters is equal to the number of points in the FFT (L)
- DFT computation is $O(N^2)$
- FFT algorithm reduces to $O(N \log N)$
 - Actually computationally possible
- N -point FFT result is N complex numbers
 - FFT of real data is symmetric: first half represents frequencies from 0 - $F_s/2$, second half is complex conjugate of first half.

Some FFT Facts (2)

- Frequency Resolution (bin size) is F_s/L
- Only important factor is length of time over which FFT is taken
 - FFT on 1 second of data = 1 Hz bin size regardless of sampling rate
 - Zero Padding can give longer FFT with less data (better frequency accuracy, but not better resolution)
- FFT is naturally computed in bit reversed order

FFT From DSPLIB

DSPF_sp_cfftr2_dit *Single-precision floating-point radix-2 FFT with complex input*

Function void DSPF_sp_cfftr2_dit (float * x, float * w, short n)

Arguments

x	Pointer to complex data input.
w	Pointer to complex twiddle factor in bit-reverse order.
n	Length of FFT in complex samples, power of 2 such that $n \geq 32$ and $n \leq 32K$.

Description

This routine performs the decimation-in-time (DIT) radix-2 FFT of the input array x. x has N complex floating-point numbers arranged as successive real and imaginary number pairs. Input array x contains N complex points ($N \cdot 2$ elements). The coefficients for the FFT are passed to the function in array w which contains $N/2$ complex numbers (N elements) as successive real and imaginary number pairs. The FFT coefficients w are in $N/2$ bit-reversed order. The elements of input array x are in normal order. The assembly routine performs 4 output samples (2 real and 2 imaginary) for a pass through inner loop.

Bit Reverse (1)

DSPF_sp_bitrev_cplx *Bit reversal for single-precision complex numbers*

Function void DSPF_sp_bitrev_cplx (double *x, short *index, int nx)

Arguments

x	Complex input array to be bit reversed. Contains 2*n _x floats.
index	Array of size ~sqrt(n _x) created by the routine bitrev_index to allow the fast implementation of the bit reversal.
n _x	Number of elements in array x[]. Must be power of 2.

Bit Reverse (2)

Special Requirements

- nx must be a power of 2.
- The table from `bitrev_index` is already created.
- The array x is actually an array of $2*nx$ floats. It is assumed to be double-word aligned.

Implementation Notes

- LDDW is used to load in one complex number at a time (both the real and the imaginary parts).
- There are 12 stores in 10 cycles but all of them are to locations already loaded. No use of the write buffer is made.
- If $nx \leq 4K$ one can use the char (8-bit) data type for the index variable. This would require changing the LDH when loading index values in the assembly routine to LDB. This would further reduce the size of the index table by half its size.
- Endianness:** Little endian configuration used.
- Interruptibility:** This code is interrupt-tolerant, but not interruptible.