ENE/EIE 208 Electrical Engineering Mathematics

Fast Fourier Transformation (FFT) Algorithm

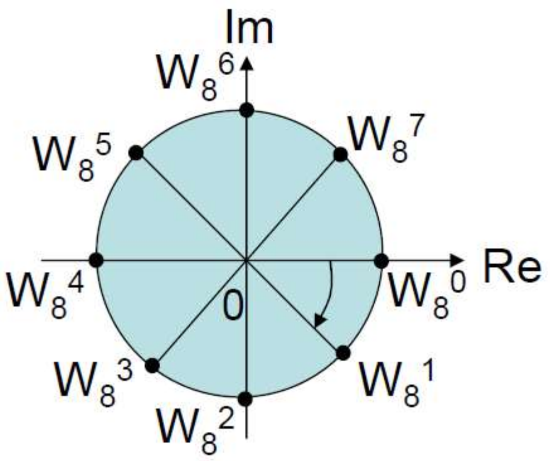
The Fast Fourier Transform

- The time taken to evaluate a DFT on a digital computer depends mainly on the number of multiplication involved, since these are the slowest operations.
- With DFT, this number is directly related to N^2 (matrix multiplication of a vector), where N is the length of the transform.
- N is chosen to be at least 256 to get a reasonable approximation of the spectrum of the sequence. Hence $256^2 = 65536$! And computational speeds becomes major consideration.
- Highly efficient computer algorithms for estimating DFT have been developed since the mid-60's by two mathematicians named Cooley and Tukey. These are known as Cooley-Tukey Fast Fourier Transform (CT-FFT) algorithms. They rely on the fact that the standard DFT involves a lot of redundant calculations.

The Fast Fourier Transform

Re-writing: $F[n] = \sum_{k=0}^{N-1} f[k] e^{-j\frac{2\pi}{N}nk}$ as $\boxed{F[n] = \sum_{k=0}^{N-1} f[k] W_N^{nk}}$

- The same values of W_N^{nk} (called a twiddle factor) are calculated many times as the computational proceeds.
- First, the integer products nk repeats for different combinations of k and n.
- Secondly, W_N^{nk} is a periodic function with only N distinct values.



N-Rotation operator,

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

$= \cos(2\pi/N) - j\sin(2\pi/N)$

$W_8 = \cos(\pi/4) - j\sin(\pi/4)$

Therefore $W_N^{nk} = (e^{-j2\pi/N})^{nk}$ and $W_8^{nk} = (e^{-j\pi/4})^{nk}$,

The Fast Fourier Transform

For example, consider $N = 8$

$$W_8^1 = e^{-j\frac{2\pi}{8}} = e^{-j45^o} = \frac{1-j}{\sqrt{2}} = a, \text{ say.}$$

Then $a^2 = -j$ $\quad a^3 = -ja = -a^*$ $\quad a^4 = -1$

$a^5 = -a$ $\quad a^6 = j$ $\quad a^7 = ja = a^*$ $\quad a^8 = 1$

From the above, it can be seen that:

$$W_8^4 = -W_8^0$$
$$W_8^5 = -W_8^1$$
$$W_8^6 = -W_8^2$$
$$W_8^7 = -W_8^3$$

Also, if nk falls outside the range 0-7,

we still get one of the above values:

eg. if $n = 5$ and $k = 7$, $\quad W_8^{35} = a^{35} = (a^8)^4 \cdot a^3 = a^3$

The Fast Fourier Transform

Decimation-in-time algorithm …

Let us begin by splitting the single summation over N samples into 2 summations, each with N/2 samples, one for k even and the other for k odd.

Substitute m = k/2 for k even and m = (k-1)/2 for k odd and write:

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_N^{(2m+1)n}$$

Note that $W_N^{2mn} = e^{-j\frac{2\pi}{N}(2mn)} = e^{-j\frac{2\pi}{N/2}mn} = W_{N/2}^{mn}$

Therefore $F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_{N/2}^{mn} + W_N^n \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_{N/2}^{mn}$

ie. $F[n] = G[n] + W_N^n H[n]$

The Fast Fourier Transform

$$F[n] = G[n] + W_N^n H[n]$$

Thus, the N-point DFT F[n] can be obtained from two N/2-point transforms, one on even input data, G[n] and the other one on odd input data, H[n].

Although the frequency index n ranges over N values, only N/2 values of G[n] and H[n] need to be computed since G[n] and H[n] are periodic in n with period N/2.

For example: N = 8:

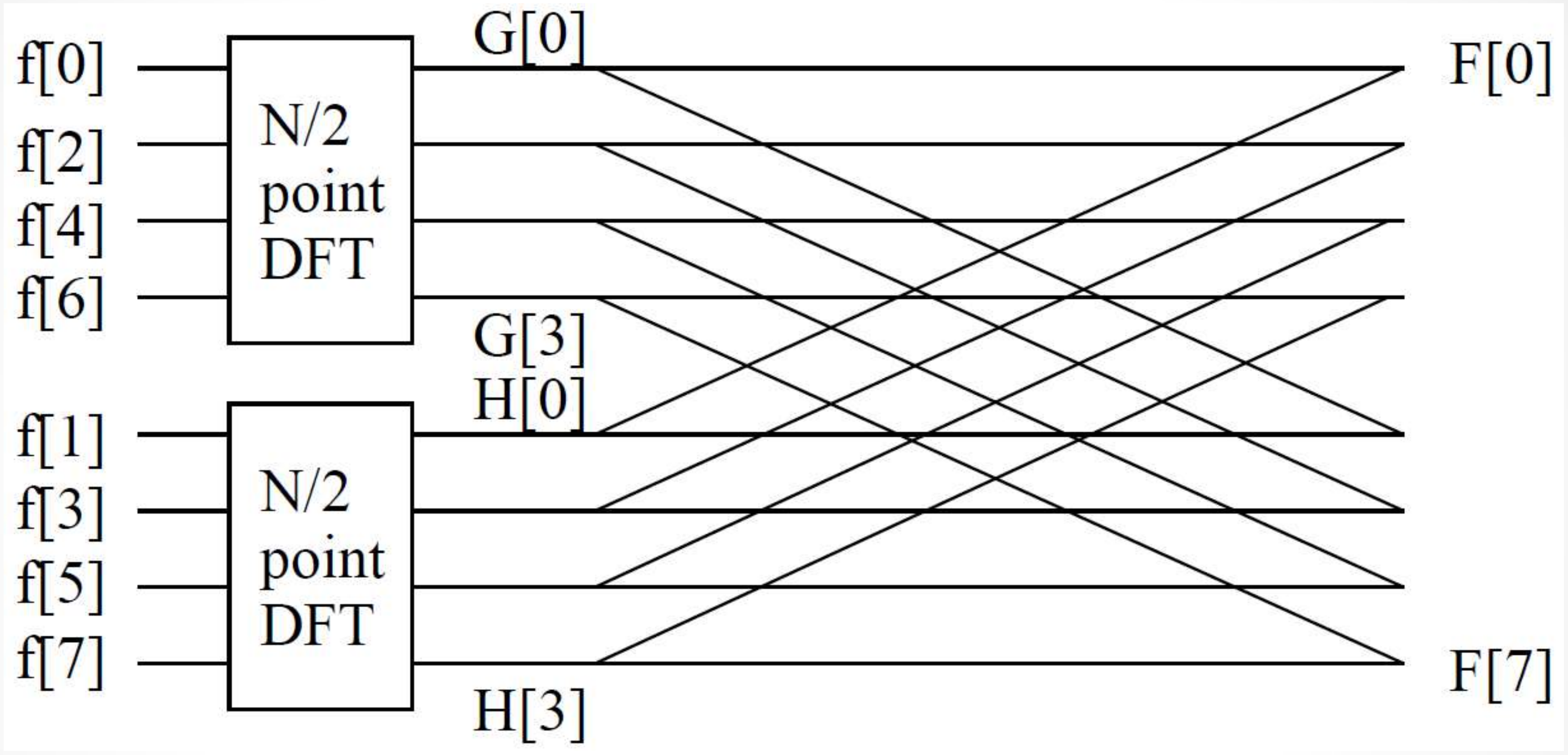
- Even input data: f[0], f[2], f[4], f[6]
- Odd input data: f[1], f[3], f[5], f[7]

From $F[n] = G[n] + W_N^n H[n]$

We'll get

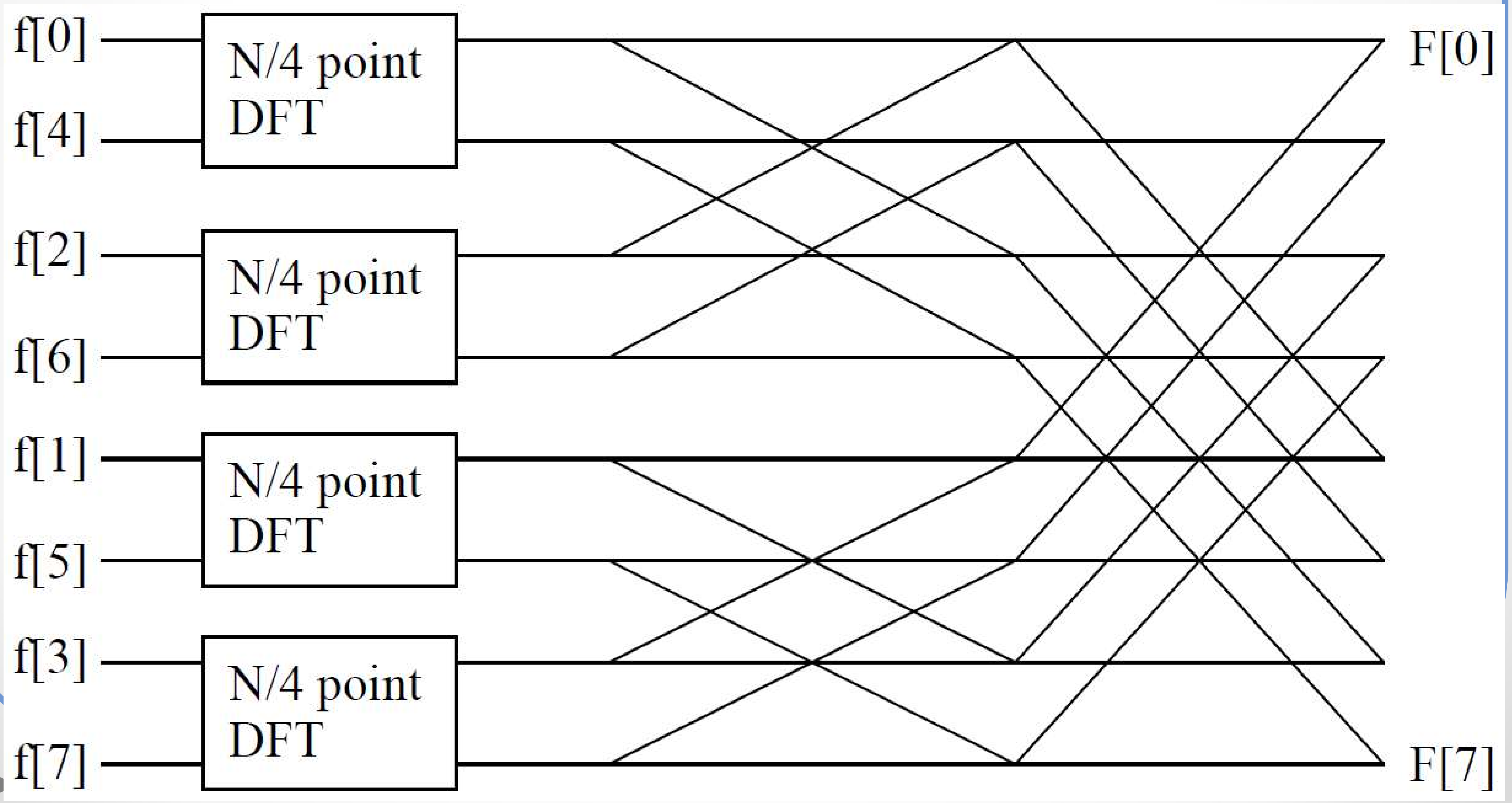
$$F[0] = G[0] + W_8^0 H[0]$$
$$F[1] = G[1] + W_8^1 H[1]$$
$$F[2] = G[2] + W_8^2 H[2]$$
$$F[3] = G[3] + W_8^3 H[3]$$
$$F[4] = G[0] + W_8^4 H[0] = G[0] - W_8^0 H[0]$$
$$F[5] = G[1] + W_8^5 H[1] = G[1] - W_8^1 H[1]$$
$$F[6] = G[2] + W_8^6 H[2] = G[2] - W_8^2 H[2]$$
$$F[7] = G[3] + W_8^7 H[3] = G[3] - W_8^3 H[3]$$

This is shown graphically on the FFT signal flow graph below:



Assuming that N is a power of 2, we can repeat the above process on the two N/2-point transforms, breaking them down to N/4-point transforms, etc…, until we come down to 2-pont transforms.

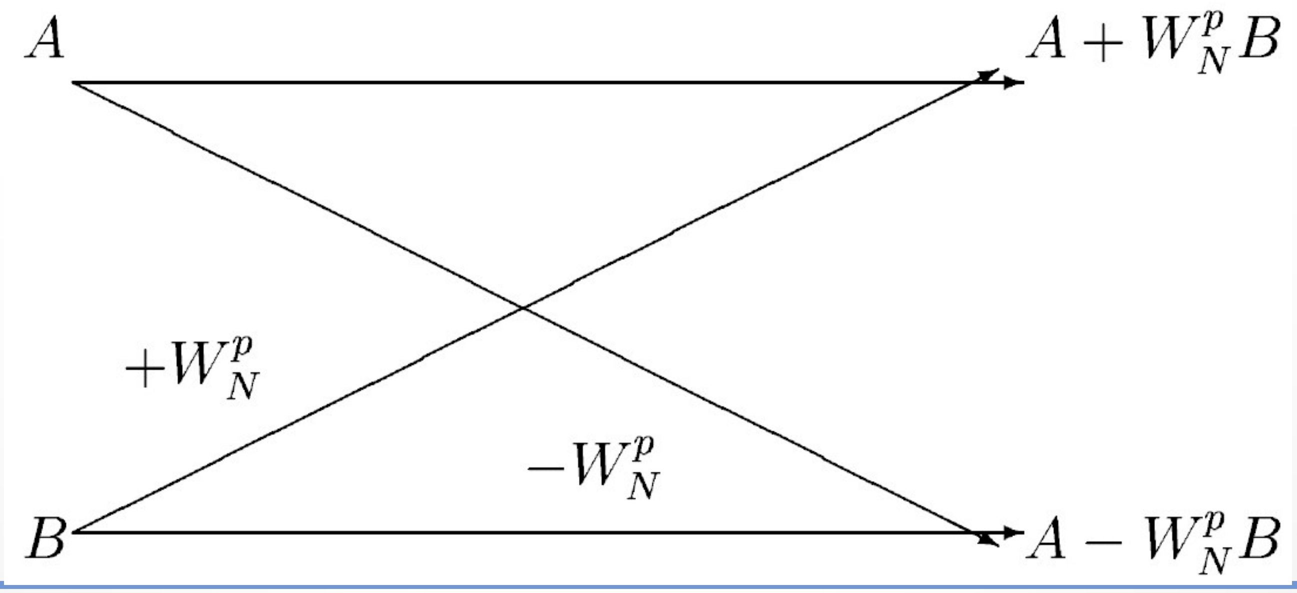
For N=8, only one further stage is needed (i.e. there are γ stages, where N = 2^{γ}), as shown below.

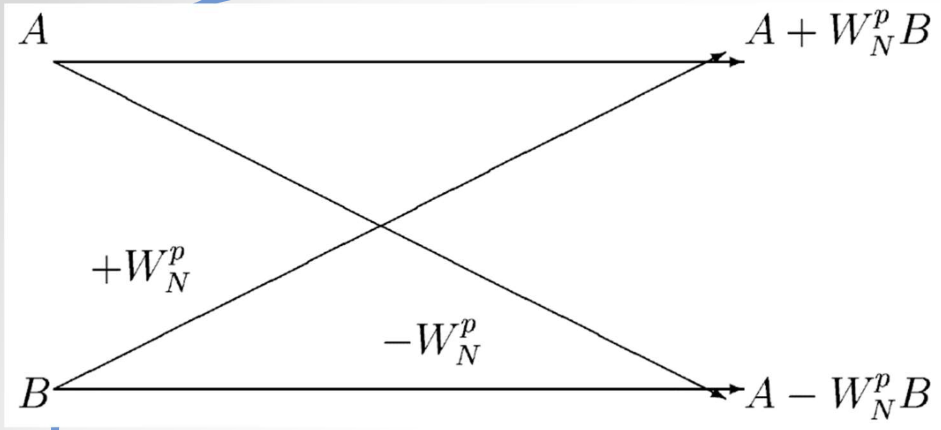


Thus, the FFT is computed by dividing up, or decimating, the sample sequence f[k] into sub-sequences until only 2-point DFT's remain.

Since it is the input, or time, samples divided up, this algorithm is known as the decimation-in-time (DIT) algorithm. (An equivalent algorithm exists for which the output, or frequency, points are sub-divided --- the decimation-in-frequency algorithm.)

The basic computation at the heart of the FFT is known as the butterfly because of its crisscross appearance. For the DIT FFT algorithm, the butterfly computation is of the form shown:





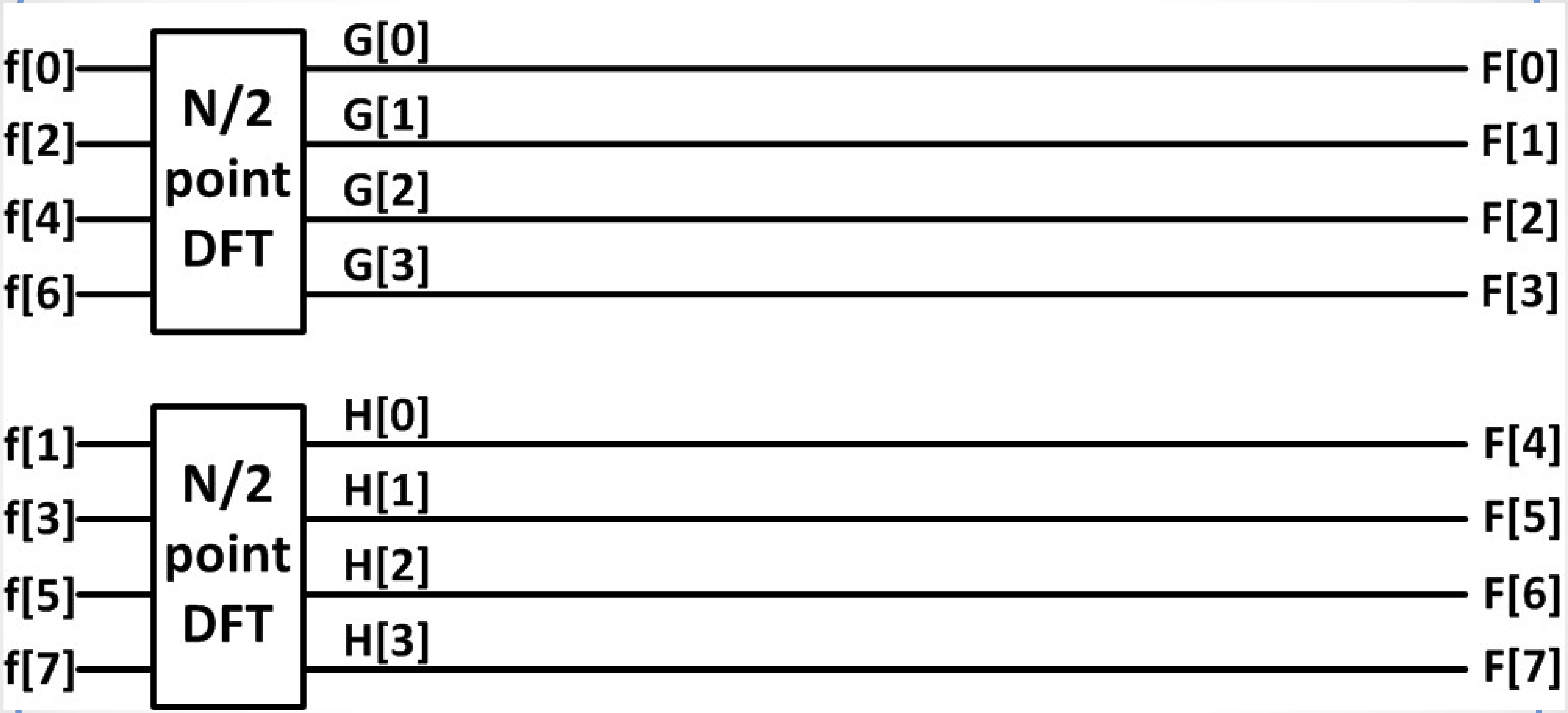
where A and B are complex numbers. Thus, a butterfly computation requires 1 complex multiplication and 2 complex additions.

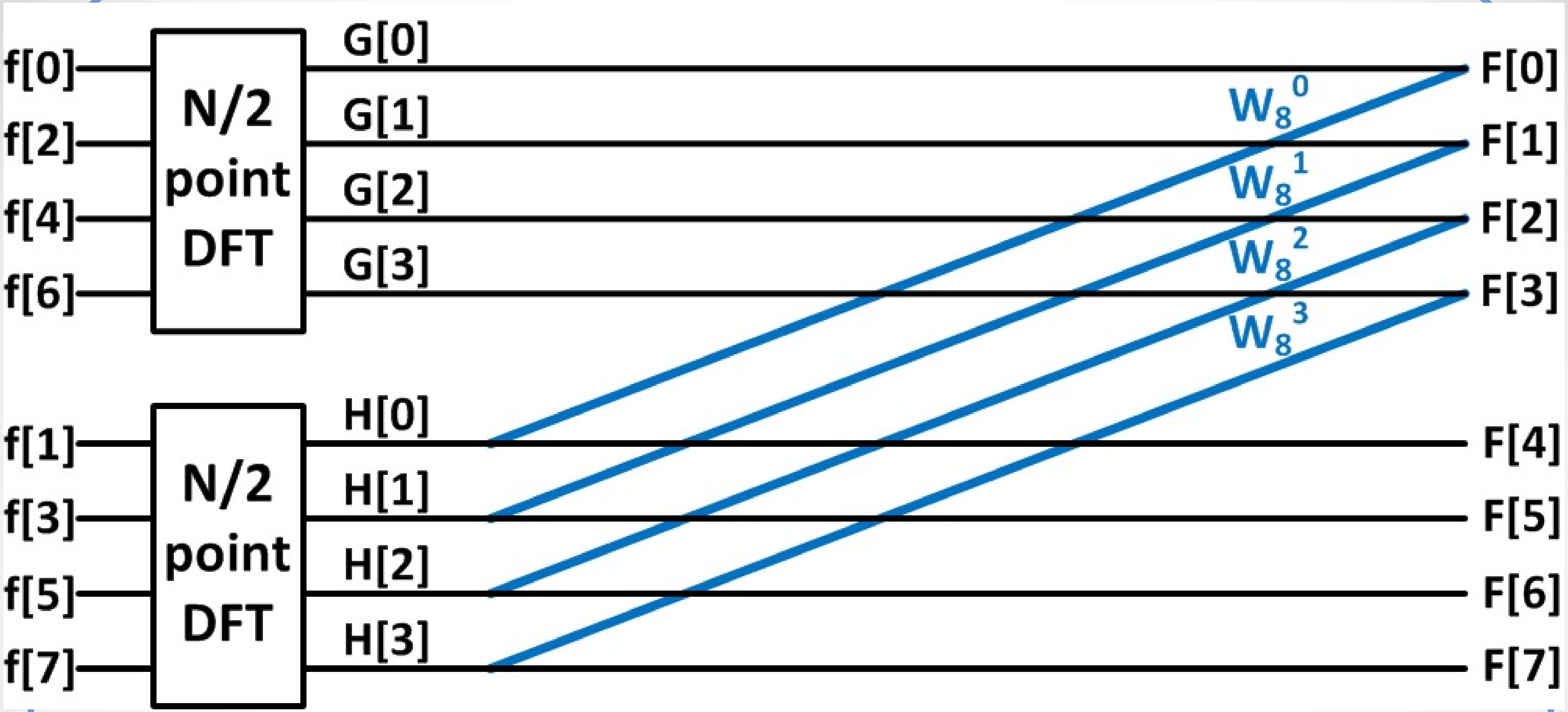
Note also, that the input samples are "bit-reversed" (see table) because at each stage of decimation the sequence input samples is separated into even- and odd-indexed samples.

Index (k)	Binary representation	Bit-reversed Binary	Bit-reversed index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

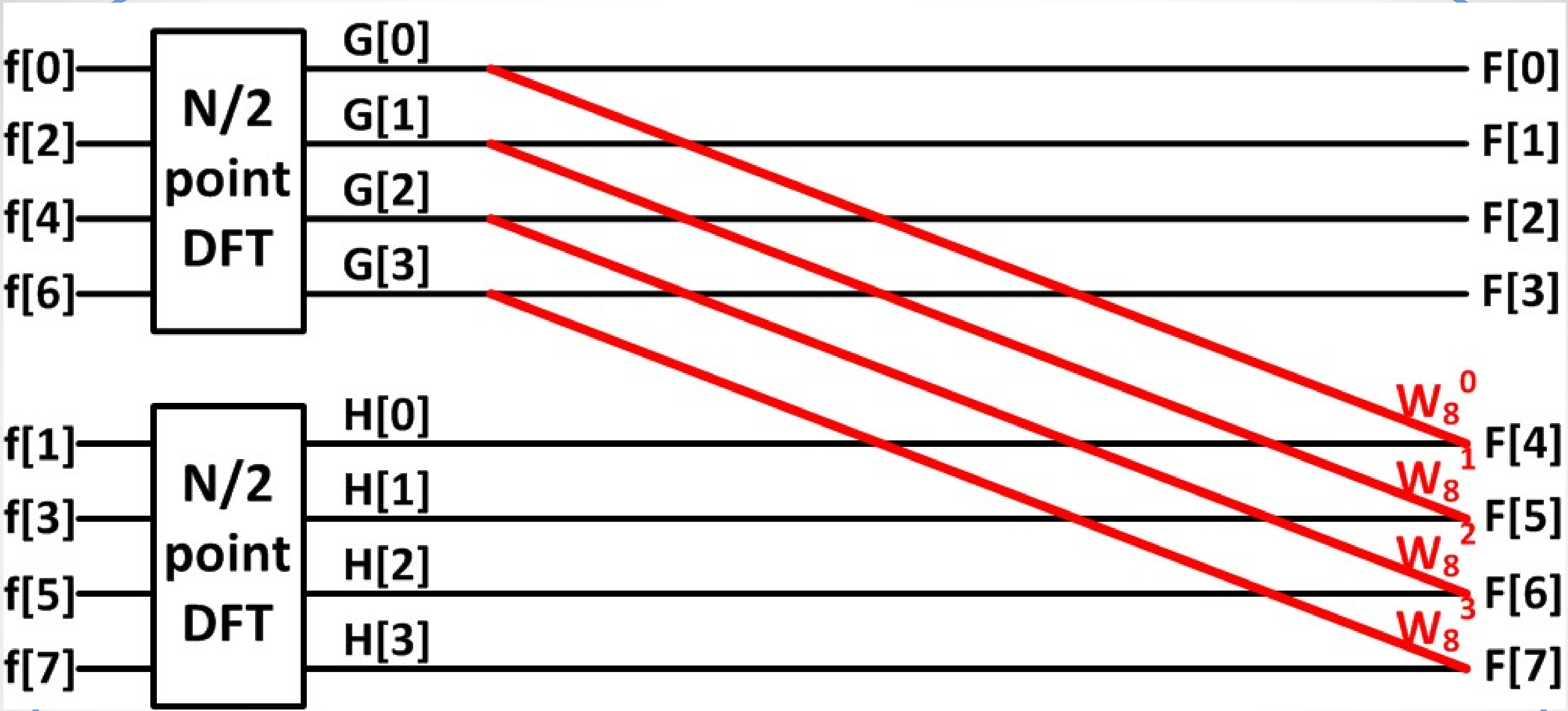
(The bit-reversal algorithm only applies if N is an integral power of 2.)

A step-by-step FFT signal flow graph generation





$$F[0] = G[0] + W_8^0 H[0]$$
$$F[1] = G[1] + W_8^1 H[1]$$
$$F[2] = G[2] + W_8^2 H[2]$$
$$F[3] = G[3] + W_8^3 H[3]$$



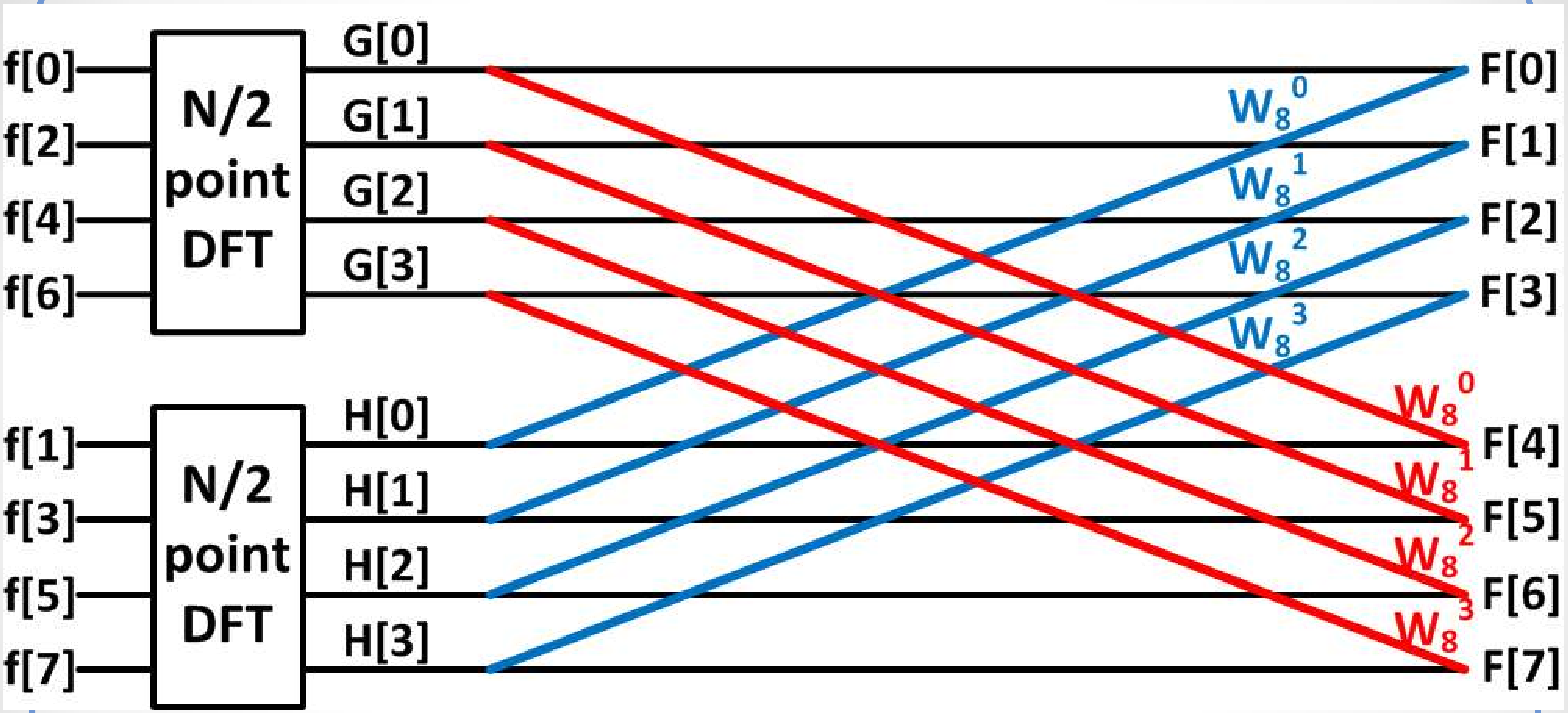
$$F[4] = G[0] + W_8^4 H[0] = G[0] - W_8^0 H[0]$$

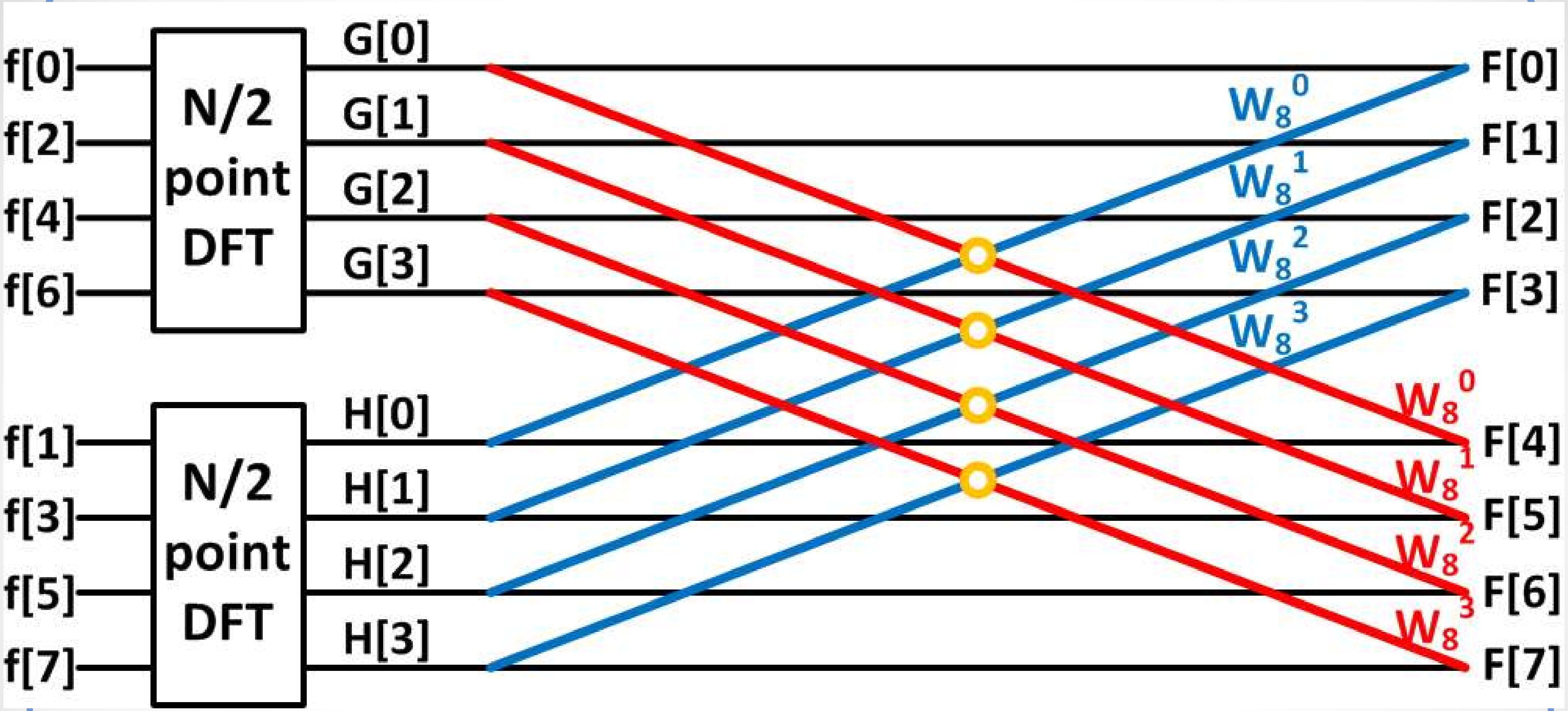
$$F[5] = G[1] + W_8^5 H[1] = G[1] - W_8^1 H[1]$$

$$F[6] = G[2] + W_8^6 H[2] = G[2] - W_8^2 H[2]$$

$$F[7] = G[3] + W_8^7 H[3] = G[3] - W_8^3 H[3]$$

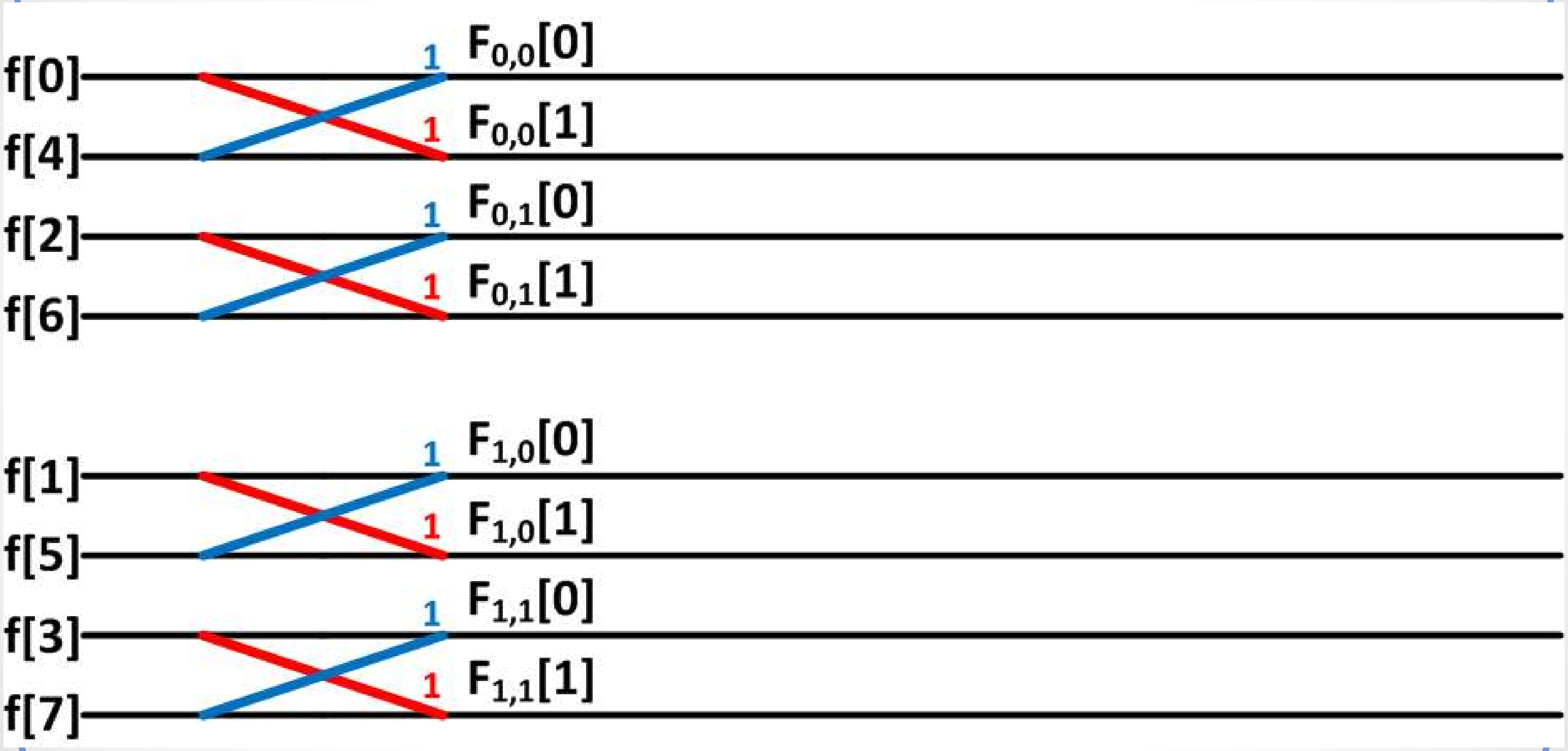
Note: the W_N^{nk} is the negative of the W_N^{nk}!


f[0]
f[2]
f[4]
f[6]
N/2 point DFT
G[0]
G[1]
G[2]
G[3]
f[1]
f[3]
f[5]
f[7]
N/2 point DFT
H[0]
H[1]
H[2]
H[3]
W_8^0
W_8^1
W_8^2
W_8^3
W_8^0
W_8^1
W_8^2
W_8^3
F[0]
F[1]
F[2]
F[3]
F[4]
F[5]
F[6]
F[7]



The circles are where the FFT butterflies are.

An 8-point FFT can be decimated into two 4-point DFTs and four 2-point DFTs accordingly, as shown.



Consider a 2-point DFT which has f[0] and f[2] as inputs, from def.

$$F[n] = \sum_{k=0}^{N-1} f[k] W_N^{nk} \quad \text{so} \quad F[0] = \sum_{k=0}^{1} f[k] W_2^{0k} = \boxed{f[0] + f[1]}$$

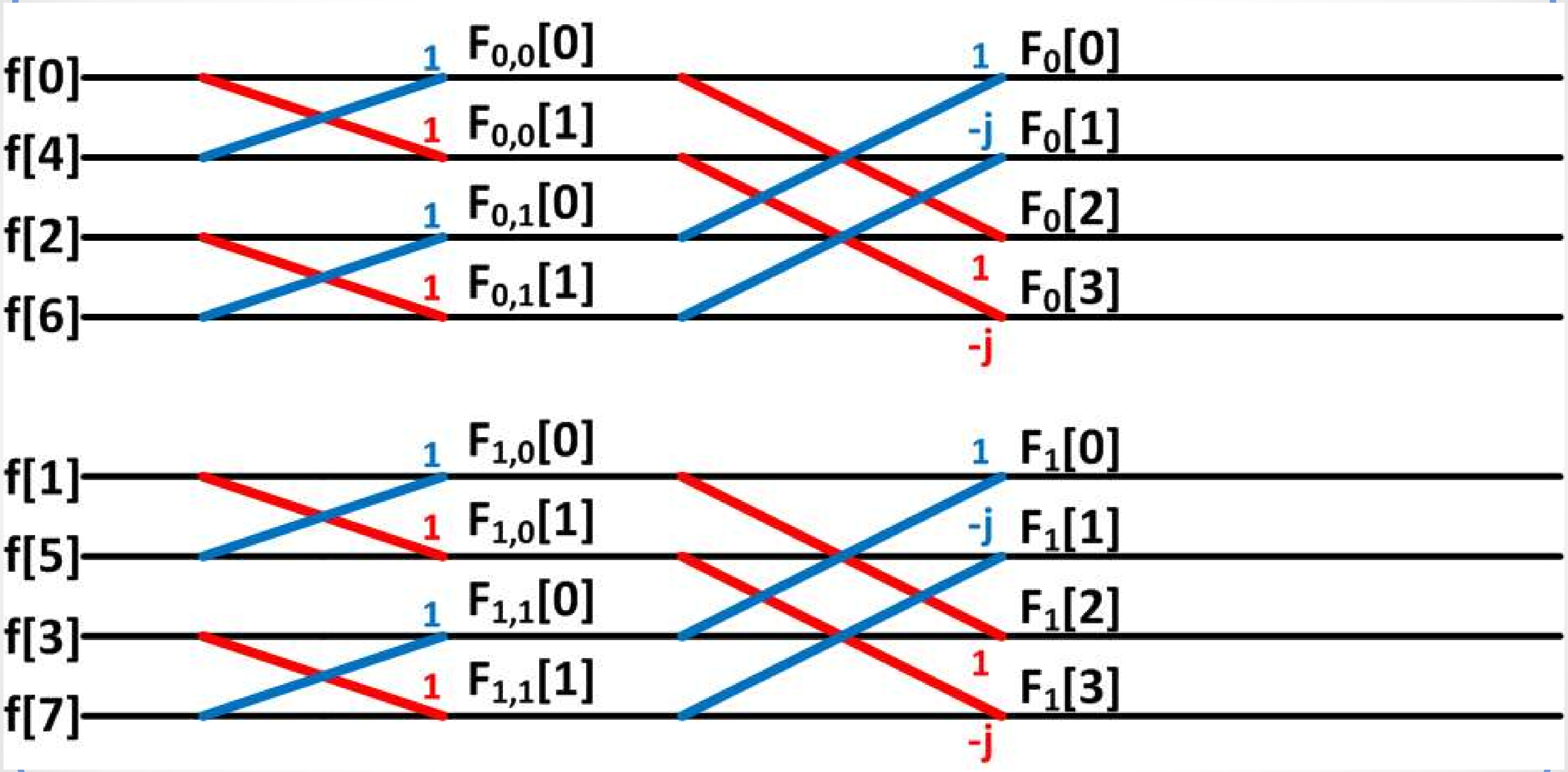
$$F[1] = \sum_{k=0}^{1} f[k] W_2^{k} = f[0] + f[1] W_2^{1} = \boxed{f[0] - f[1]}$$

$\boxed{\text{The twiddle factor } W_2^{nk} \text{ has 2 values: +1 and -1}}$

Therefore,

$$F_{0,0}[0] = f[0] + f[4]$$
$$F_{0,0}[1] = f[0] - f[4]$$
$$F_{0,1}[0] = f[2] + f[6]$$
$$F_{0,1}[1] = f[2] - f[6]$$
$$F_{1,0}[0] = f[1] + f[5]$$
$$F_{1,0}[1] = f[1] - f[5]$$
$$F_{1,1}[0] = f[3] + f[7]$$
$$F_{1,1}[1] = f[3] - f[7]$$

These will be the inputs to the next stage.


f[0]
f[4]
f[2]
f[6]
f[1]
f[5]
f[3]
f[7]
$F_{0,0}[0]$
$F_{0,0}[1]$
$F_{0,1}[0]$
$F_{0,1}[1]$
$F_{1,0}[0]$
$F_{1,0}[1]$
$F_{1,1}[0]$
$F_{1,1}[1]$
$F_0[0]$
$F_0[1]$
$F_0[2]$
$F_0[3]$
$F_1[0]$
$F_1[1]$
$F_1[2]$
$F_1[3]$
1
-j

The next stage would be 4-point DFTs:

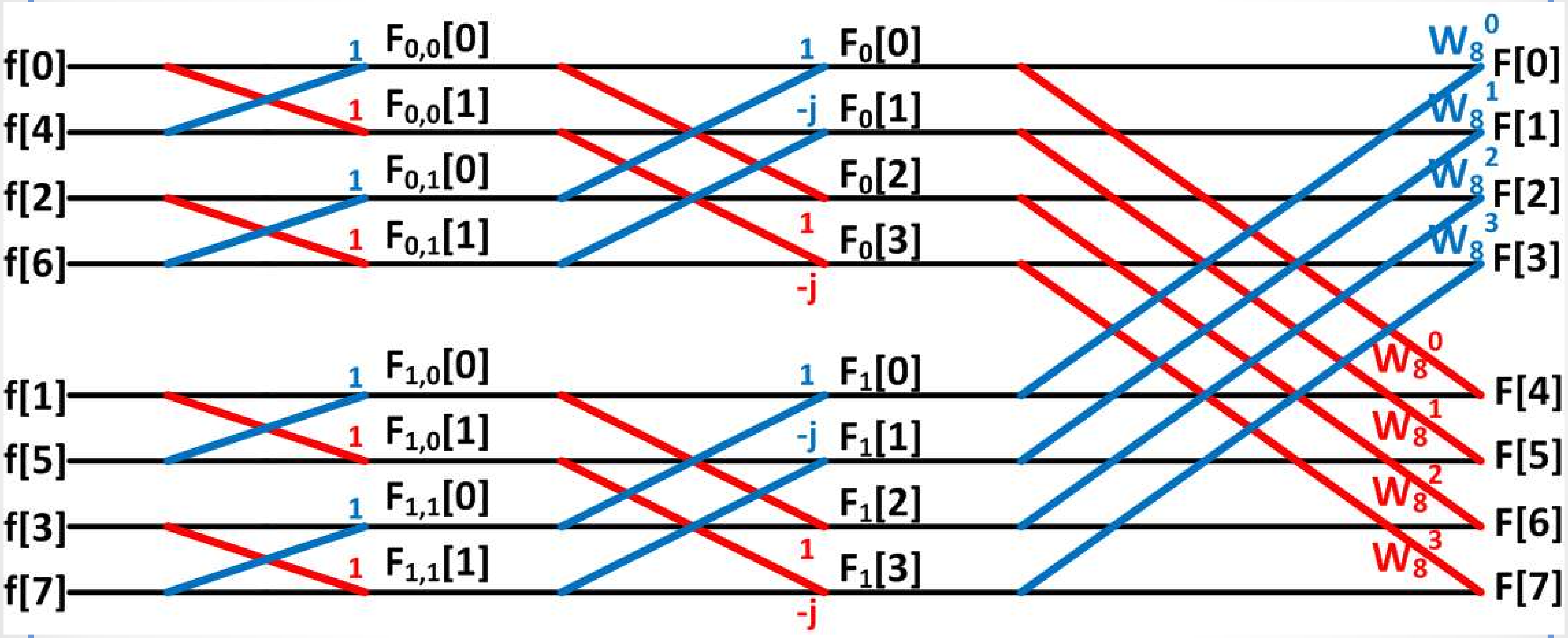
The twiddle factor W_4^{nk} has 4 values: +1, -j, -1 and +j

From the diagram,

$$
\begin{aligned}
F_0[0] &= F_{0,0}[0] + W_4^0 F_{0,1}[0] = F_{0,0}[0] + F_{0,1}[0] \\
F_0[2] &= F_{0,0}[0] + W_4^2 F_{0,1}[0] = F_{0,0}[0] - F_{0,1}[0] \\
F_0[1] &= F_{0,0}[1] + W_4^1 F_{0,1}[1] = F_{0,0}[1] - jF_{0,1}[1] \\
F_0[3] &= F_{0,0}[1] + W_4^3 F_{0,1}[1] = F_{0,0}[1] + jF_{0,1}[1]
\end{aligned}
$$

$$
\begin{aligned}
F_1[0] &= F_{1,0}[0] + W_4^0 F_{1,1}[0] = F_{1,0}[0] + F_{1,1}[0] \\
F_1[2] &= F_{1,0}[0] + W_4^2 F_{1,1}[0] = F_{1,0}[0] - F_{1,1}[0] \\
F_1[1] &= F_{1,0}[1] + W_4^1 F_{1,1}[1] = F_{1,0}[1] - jF_{1,1}[1] \\
F_1[3] &= F_{1,0}[1] + W_4^3 F_{1,1}[1] = F_{1,0}[1] + jF_{1,1}[1]
\end{aligned}
$$

These will be the inputs to the next stage which is the final stage, which we have already done before.

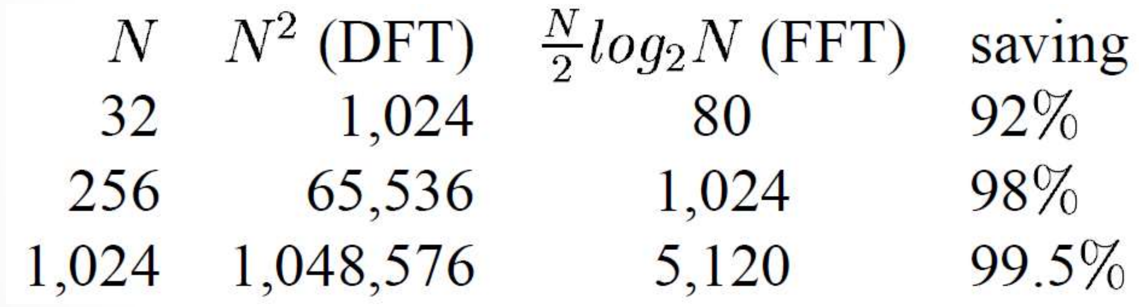


Can you identify where the FFT butterflies are?

Computational Speed of FFT

The DFT requires N^2 complex multiplications (and $(N-1)^2$ complex additions).

For FFT, each stage (i.e. each halving), N/2 complex multiplications are required to combine the results of the previous stage. Since there are ($\log_2 N$) stages, the number of complex multiplications required to evaluate an N-point DFT with the FFT is approximately $N/2 \times \log_2 N$ (and $N \log_2 N$ complex additions).



N	N^2 (DFT)	$\frac{N}{2} log_2 N$ (FFT)	saving
32	1,024	80	92%
256	65,536	1,024	98%
1,024	1,048,576	5,120	99.5%

Practical Considerations

If N is a power of 2 (or $N = 2^m$), it is easy to compute N-point DFT via the algorithm highlighted earlier. Such algorithm sometimes is appropriately called the
"Radix-2 Decimation-in-Time Fast-Fourier Transformation Algorithm" or
"R-2 DIT FFT".

If N is not a power of 2, there are 2 strategies available to complete an N-point FFT.

1. Take advantage of such factors as N possesses. For example, if N is divisible by 3 (e.g. N = 48), the final decimation stage would include a 3-point transform.

2. Pack the data with zeroes; e.g. include 16 zeroes with 48 data points (for N = 48) and compute a 64-point FFT. (Or a better approach might be to pack data with more realistic "dummy values".)

Example 12.6

Using the DFT, estimate the frequency characteristics of the decaying exponential signal $g(t) = \exp(-0.5t)u(t)$. Plot the magnitude and phase spectra.

Solution

Step 1: Impulse-train sampling Based on Table 5.1, the CTFT of the decaying exponential is given by

$$g(t) = \mathrm{e}^{-0.5t}u(t) \xleftrightarrow{\text{CTFT}} G(\omega) = \frac{1}{0.5 + \mathrm{j}\omega}.$$

This CTFT pair implies that the bandwidth of $g(t)$ is infinite. Ideally speaking, the Nyquist sampling theorem can never be satisfied for the decaying exponential signal. However, we exploit the fact that the magnitude $|G(\omega)|$ of the CTFT decreases monotonically with higher frequencies and we neglect any frequency components at which the magnitude falls below a certain threshold η. Selecting the value of $\eta = 0.01 \times |G(\omega)|_{\max}$, the threshold frequency B is given by

$$\left|\frac{1}{0.5 + \mathrm{j}2\pi B}\right| \le 0.01 \times |G(\omega)|_{\max}.$$

Since the maximum value of the magnitude $|G(\omega)|$ is 2 at $\omega = 0$, the above expression reduces to

$$\sqrt{0.25 + (2\pi B)^2} \geq 50,$$

or $B \geq 7.95$ Hz. The Nyquist sampling rate f_1 is therefore given by

$$f_1 \geq 2 \times 7.95 = 15.90 \text{ samples/s}.$$

Selecting a sampling rate of $f_1 = 20$ samples/s, or a sampling interval $T_1 = 1/20 = 0.05$ s, the DT approximation of the decaying exponential is given by

$$g[k] = g(kT_1) = e^{-0.5kT_1}u[k] = e^{-0.025k}u[k].$$

Since there is a discontinuity at $k = 0$, we set $g[0] = 0.5$.

Step 2: Time-limitation To truncate the length of $g[k]$, we apply a rectangular window of length $N = 203$ samples. The truncated sequence is given by

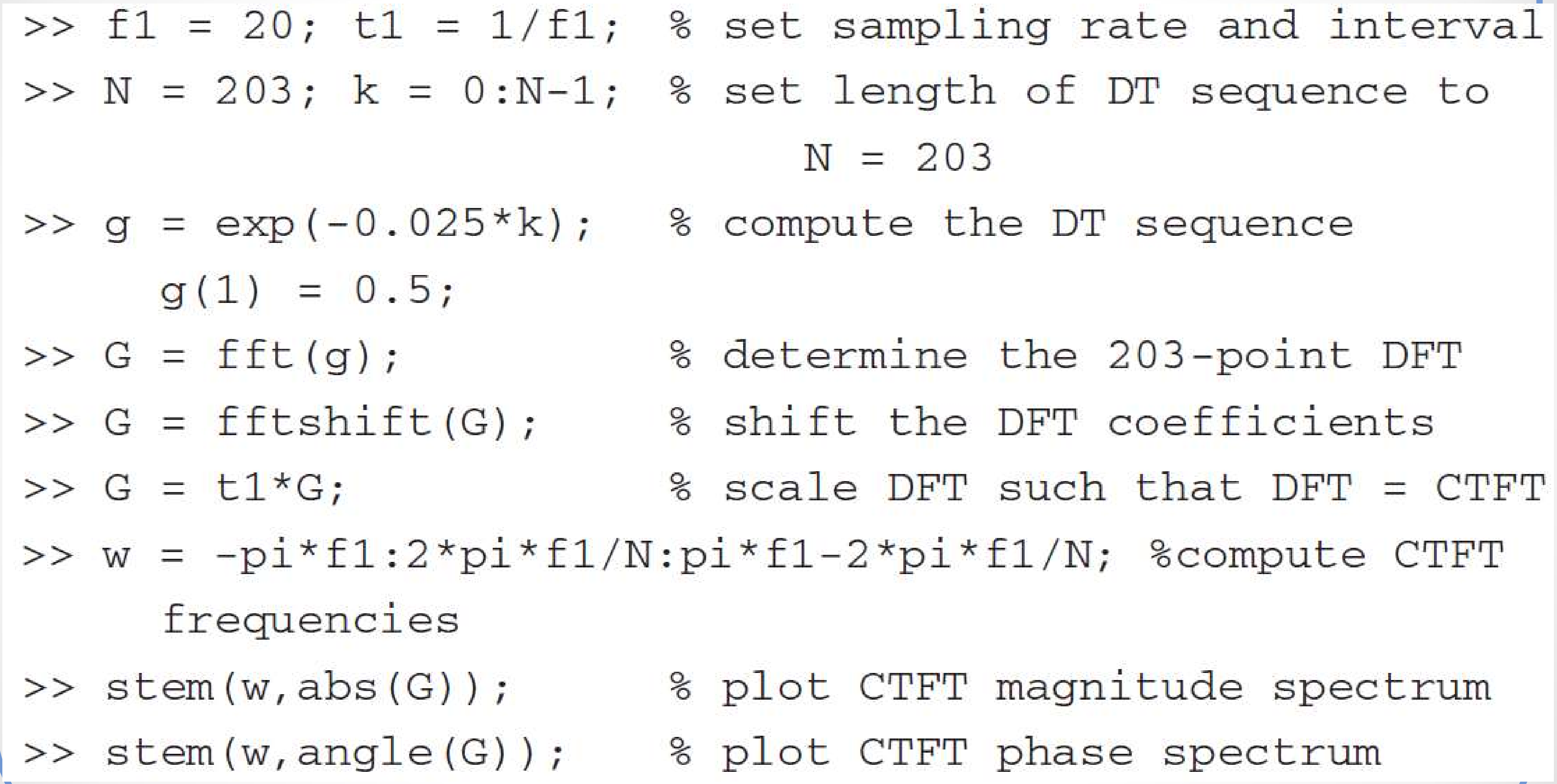
$$g_w[k] = e^{-0.025k}(u[k] - u[k-199]) = \begin{cases} e^{-0.025k} & 0 \le k \le 202 \\ 0 & \text{elsewhere.} \end{cases}$$

Step 3: DFT computation The DFT of the truncated DT sequence $g_w[k]$ can now be computed directly from Eq. (12.16). MATLAB provides a built-in function `fft`, which has the calling syntax of

```
>> G = fft(g);
```

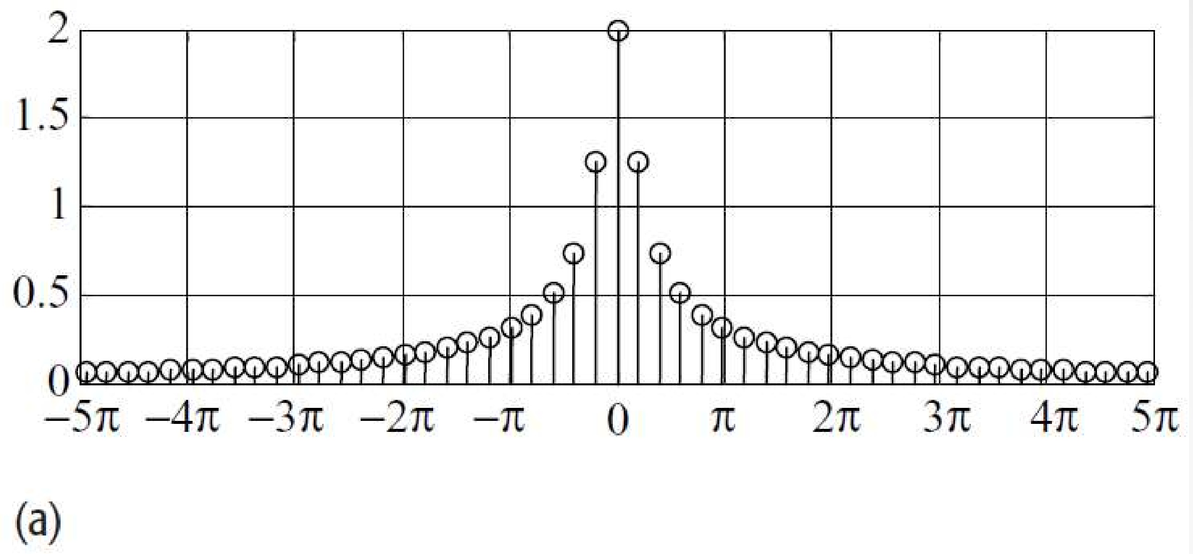
where `g` is the signal vector containing the values of the DT sequence $g_w[k]$ and `G` is the computed DFT. Both `g` and `G` have a length of N, implying that an N-point DFT is being taken. The built-in function `fft` computes the DFT within the frequency range $0 \le r \le (N-1)$. Since the DFT is periodic, we can obtain the DFT within the frequency range $-(N-1)/2 \le r \le (N-1)/2$ by a circular shift of the DFT coefficients. In MATLAB, this is accomplished by the `fftshift` function.

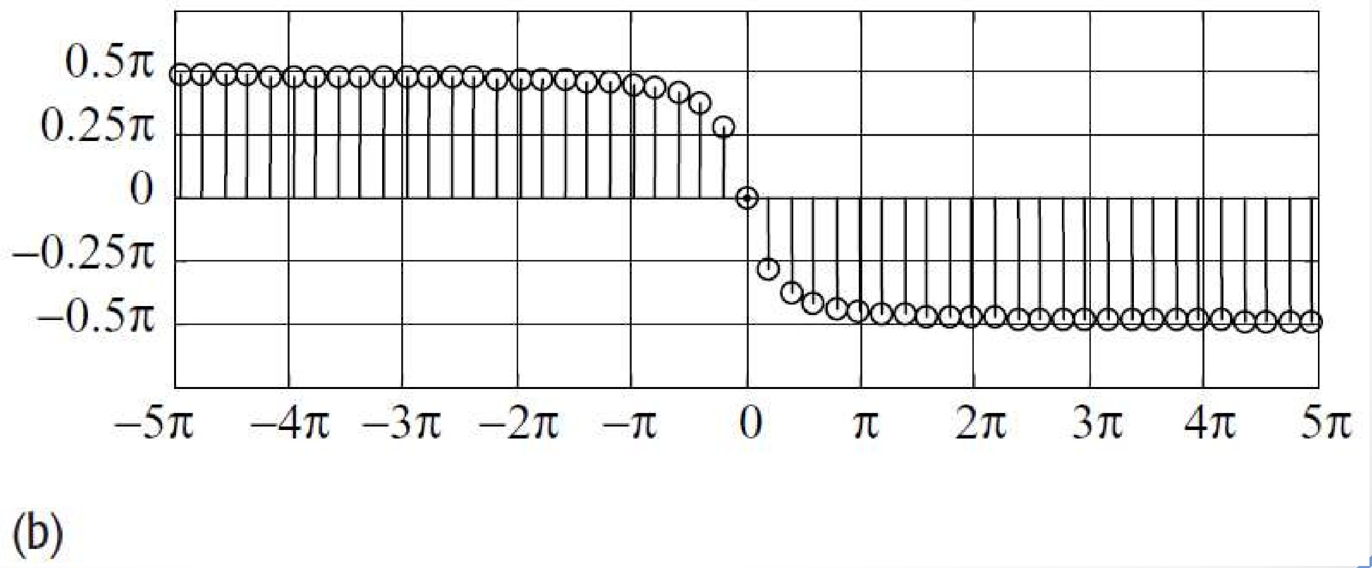
MATLAB Code



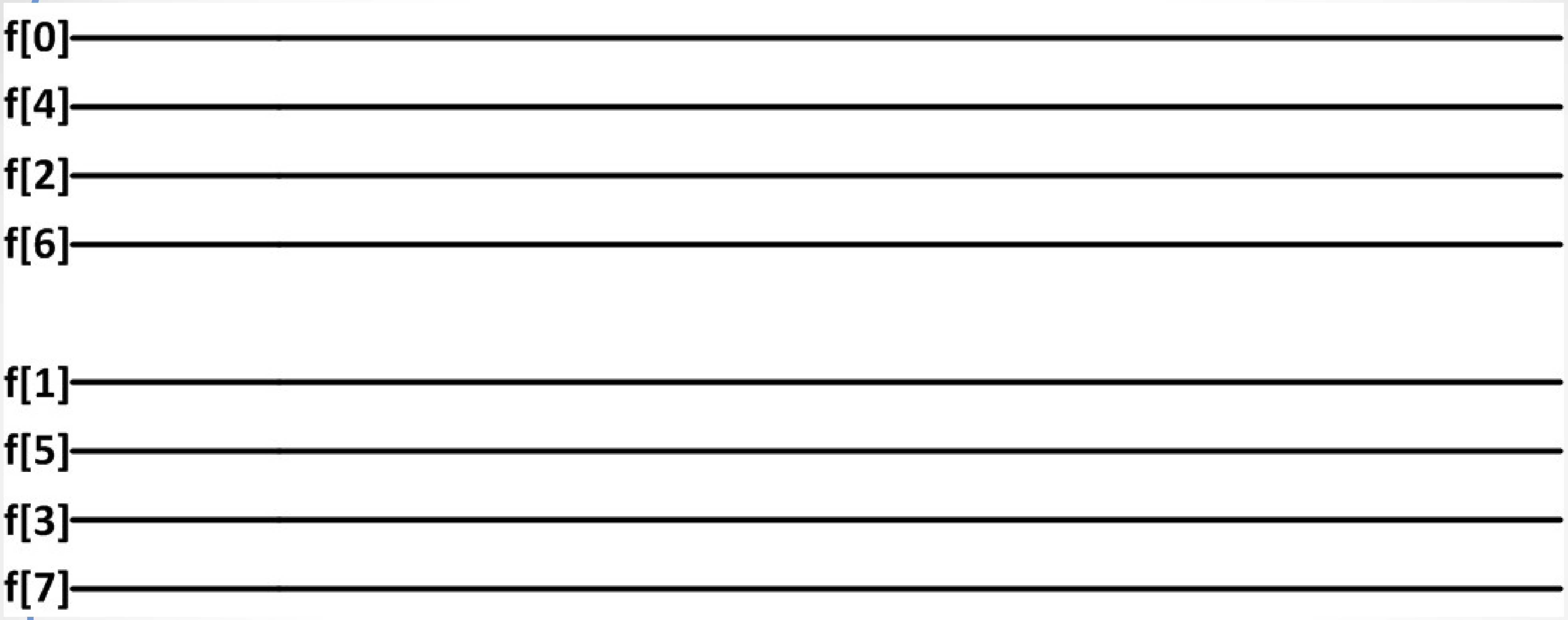
```
>> f1 = 20; t1 = 1/f1;  % set sampling rate and interval
>> N = 203; k = 0:N-1;  % set length of DT sequence to
                             N = 203
>> g = exp(-0.025*k);   % compute the DT sequence
     g(1) = 0.5;
>> G = fft(g);          % determine the 203-point DFT
>> G = fftshift(G);     % shift the DFT coefficients
>> G = t1*G;            % scale DFT such that DFT = CTFT
>> w = -pi*f1:2*pi*f1/N:pi*f1-2*pi*f1/N; %compute CTFT
     frequencies
>> stem(w,abs(G));      % plot CTFT magnitude spectrum
>> stem(w,angle(G));    % plot CTFT phase spectrum
```

Fig. 12.5. Spectral estimation of decaying exponential signal $g(t) = \exp(-0.5t)u(t)$ using the DFT in Example 12.6. (a) Estimated magnitude spectrum; (b) estimated phase spectrum.





For your own practice, 8-point R-2 DIT FFT


f[0]
f[4]
f[2]
f[6]
f[1]
f[5]
f[3]
f[7]