RADIX-4 FFT ALGORITHMS[*]

Douglas L. Jones

Abstract

The decimation-in-time (DIT) radix-4 FFT recursively partitions a DFT into four quarter-length DFTs of groups of every fourth time sample. The outputs of these shorter FFTs are reused to compute many outputs, thus greatly reducing the total computational cost. The radix-4 decimation-in-frequency FFT groups every fourth output sample into shorter-length DFTs to save computations. The radix-4 FFTs require only 75% as many complex multiplies as the radix-2 FFTs.

The radix-4 decimation-in-time[1] and decimation-in-frequency[2]fast Fourier transforms (FFTs)[3] gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs. The radix-4 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation[4] into four parts: sums over all groups of every fourth discrete-time index $n = [0, 4, 8, \dots, N-4]$, $n = [1, 5, 9, \dots, N-3]$, $n = [2, 6, 10, \dots, N-2]$ and $n = [3, 7, 11, \dots, N-1]$ as in (1). (This works out only when the FFT length is a multiple of four.) Just as in the radix-2 decimation-in-time FFT[5], further mathematical manipulation shows that the length-N DFT can be computed as the sum of the outputs of four length-$\frac{N}{4}$ DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called **twiddle factors**$W_N^k = e^{-\left(i\frac{2\pi k}{N}\right)}$, W_N^{2k}, and W_N^{3k}.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-\left(i\frac{2\pi nk}{N}\right)} = \sum_{n=0}^{\frac{N}{4}-1} x(4n)\, e^{-\left(i\frac{2\pi\times(4n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)\, e^{-\left(i\frac{2\pi(4n+1)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+2)\, e^{-\left(i\frac{2\pi(4n+2)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)\, e^{-\left(i\frac{2\pi(4n+3)k}{N}\right)} = \mathrm{DFT}_{\frac{N}{4}}[x(4n)] + W_N^k \mathrm{DFT}_{\frac{N}{4}}[x(4n+1)] + W_N^{2k}\mathrm{DFT}_{\frac{N}{4}}[x(4n+2)] + W_N^{3k}\mathrm{DFT}_{\frac{N}{4}}[x(4n+3)] \quad (1)$$

This is called a **decimation in time** because the time samples are rearranged in alternating groups, and a **radix-4** algorithm because there are four groups. Figure 1 (Radix-4 DIT structure) graphically illustrates this form of the DFT computation.

[*]Version 1.4: Sep 18, 2006 9:51 pm -0500
[†]http://creativecommons.org/licenses/by/1.0
[1]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>
[2]"Decimation-in-Frequency (DIF) Radix-2 FFT" <http://cnx.org/content/m12018/latest/>
[3]"Overview of Fast Fourier Transform (FFT) Algorithms" <http://cnx.org/content/m12026/latest/>
[4]"DFT Definition and Properties" <http://cnx.org/content/m12019/latest/>
[5]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>

Radix-4 DIT structure

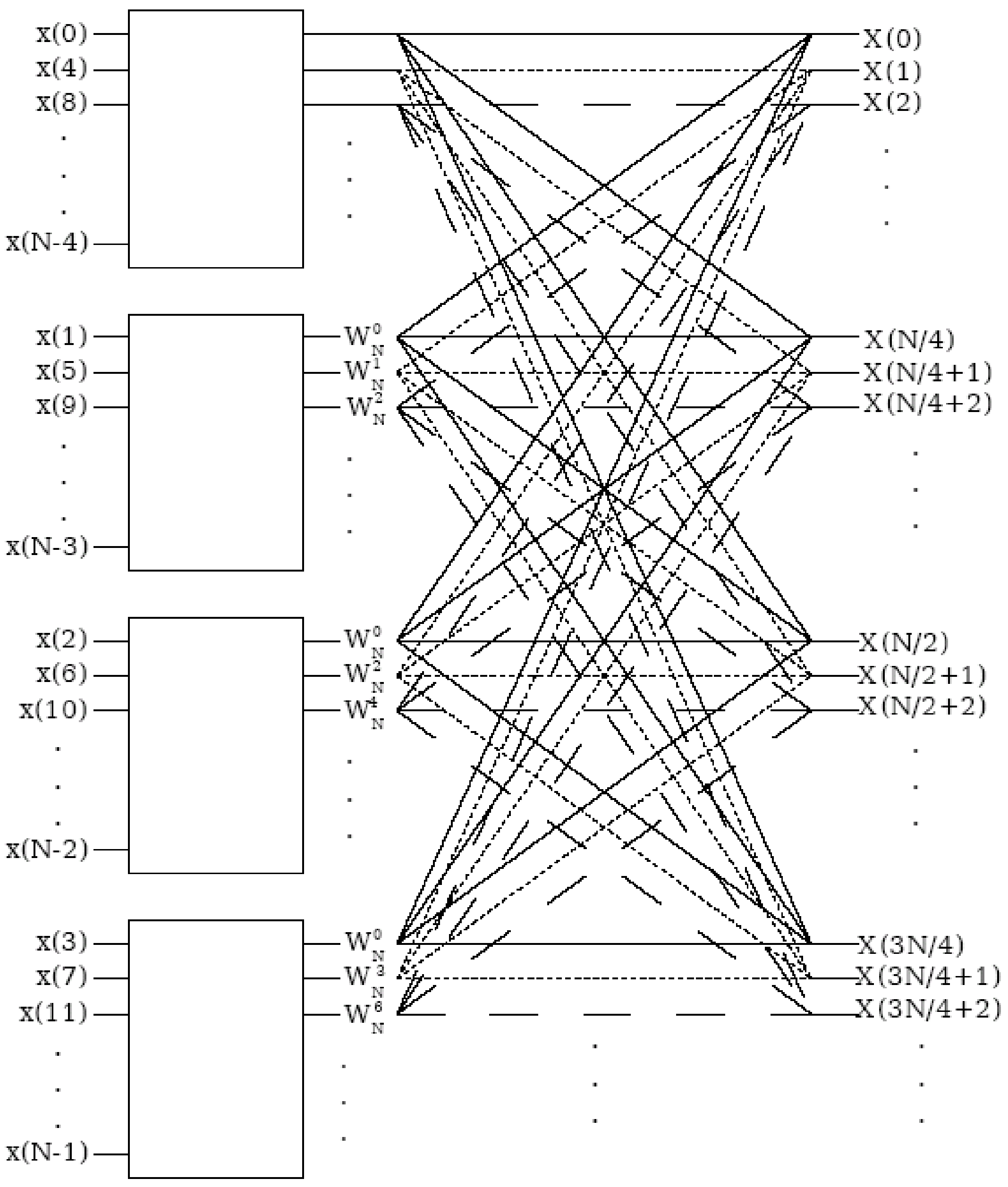
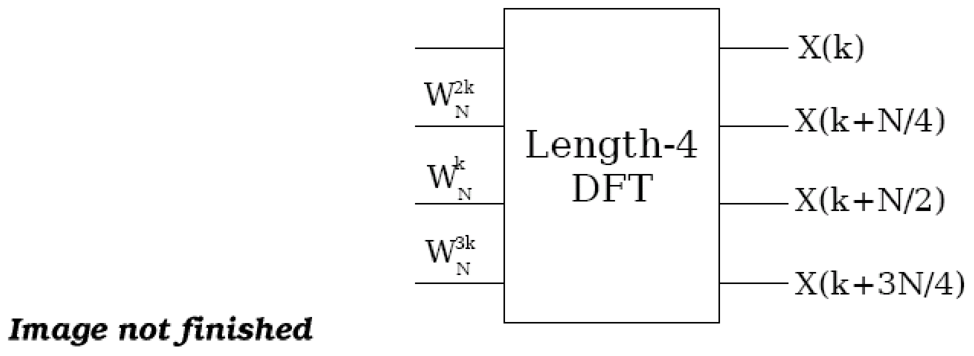


Figure 1: Decimation in time of a length-N DFT into four length-$\frac{N}{4}$ DFTs followed by a combining stage.

Due to the periodicity with $\frac{N}{4}$ of the short-length DFTs, their outputs for frequency-sample k are reused to compute $X(k)$, $X\left(k+\frac{N}{4}\right)$, $X\left(k+\frac{N}{2}\right)$, and $X\left(k+\frac{3N}{4}\right)$. It is this reuse that gives the radix-4 FFT its efficiency. The computations involved with each group of four frequency samples constitute the **radix-4 butterfly**, which is shown in Figure 2. Through further rearrangement, it can be shown that this computation can be simplified to three twiddle-factor multiplies and a length-4 DFT! The theory of multi-dimensional index maps[6] shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFTs with twiddle-factor multiplications in between. The length-4 DFT requires no multiplies and only eight complex additions (this efficient computation can be derived using a radix-2 FFT[7]).



(a) (b)

Figure 2: The radix-4 DIT butterfly can be simplified to a length-4 DFT preceded by three twiddle-factor multiplies.

If the FFT length $N=4^M$, the shorter-length DFTs can be further decomposed recursively in the same manner to produce the full **radix-4 decimation-in-time FFT**. As in the radix-2 decimation-in-time FFT[8], each stage of decomposition creates additional savings in computation. To determine the total computational cost of the radix-4 FFT, note that there are $M=\log_4 N=\frac{\log_2 N}{2}$ stages, each with $\frac{N}{4}$ butterflies per stage. Each radix-4 butterfly requires 3 complex multiplies and 8 complex additions. The total cost is then

Radix-4 FFT Operation Counts

- $3\frac{N}{4}\frac{\log_2 N}{2}=\frac{3}{8}N\log_2 N$ complex multiplies (75% of a radix-2 FFT)
- $8\frac{N}{4}\frac{\log_2 N}{2}=N\log_2 N$ complex adds (same as a radix-2 FFT)

The radix-4 FFT requires only 75% as many complex multiplies as the radix-2[9] FFTs, although it uses the same number of complex additions. These additional savings make it a widely-used FFT algorithm.

The decimation-in-time operation regroups the input samples at each successive stage of decomposition, resulting in a "digit-reversed" input order. That is, if the time-sample index n is written as a base-4 number, the order is that base-4 number reversed. The digit-reversal process is illustrated for a length-$N=64$ example below.

[6]"Multidimensional Index Maps" <http://cnx.org/content/m12025/latest/>
[7]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>
[8]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>
[9]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>

Example 1: N = 64 = 4^3

Original Number	**Original Digit Order**	**Reversed Digit Order**	**Digit-Reversed Number**
0	000	000	0
1	001	100	16
2	002	200	32
3	003	300	48
4	010	010	4
5	011	110	20
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1

It is important to note that, if the input signal data are placed in digit-reversed order before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an **in-place algorithm** that requires no extra memory to perform the FFT. Most FFT implementations are in-place, and overwrite the input data with the intermediate values and finally the output. A slight rearrangement within the radix-4 FFT introduced by *Burrus*[1] allows the inputs to be arranged in bit-reversed[10] rather than digit-reversed order.

A radix-4 decimation-in-frequency[11] FFT can be derived similarly to the radix-2 DIF FFT[12], by separately computing all four groups of every fourth **output** frequency sample. The DIF radix-4 FFT is a flow-graph reversal of the DIT radix-4 FFT, with the same operation counts and twiddle factors in the reversed order. The output ends up in digit-reversed order for an in-place DIF algorithm.

Exercise 1 ***(Solution on p. 5.)***

How do we derive a radix-4 algorithm when $N = 4^M 2$?

[10]"Decimation-in-time (DIT) Radix-2 FFT" <http://cnx.org/content/m12016/latest/>
[11]"Decimation-in-Frequency (DIF) Radix-2 FFT" <http://cnx.org/content/m12018/latest/>
[12]"Decimation-in-Frequency (DIF) Radix-2 FFT" <http://cnx.org/content/m12018/latest/>

Solutions to Exercises in this Module

Solution to Exercise (p. 4)
Perform a radix-2 decomposition for one stage, then radix-4 decompositions of all subsequent shorter-length DFTs.

References

[1] C.S. Burrus. Unscrambling for fast dft algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-36(7):1086–1089, July 1988.