

SPLIT-RADIX FFT ALGORITHMS*

Douglas L. Jones

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0[†]

Abstract

The split-radix FFT mixes radix-2 and radix-4 decompositions, yielding an algorithm with about one-third fewer multiplies than the radix-2 FFT. The split-radix FFT has lower complexity than the radix-4 or any higher-radix power-of-two FFT.

The split-radix algorithm, first clearly described and named by *Duhamel and Hollman*[2] in 1984, required fewer total multiply and add operations than any previous power-of-two algorithm. (*Yavne*[5] first derived essentially the same algorithm in 1968, but the description was so atypical that the work was largely neglected.) For a time many FFT experts thought it to be optimal in terms of total complexity, but even more efficient variations have more recently been discovered by *Johnson and Frigo*[4].

The split-radix algorithm can be derived by careful examination of the radix-2¹ and radix-4² flowgraphs as in Figure 1 below. While in most places the radix-4³ algorithm has fewer nontrivial twiddle factors, in some places the radix-2⁴ actually lacks twiddle factors present in the radix-4⁵ structure or those twiddle factors simplify to multiplication by $-i$, which actually requires only additions. By mixing radix-2⁶ and radix-4⁷ computations appropriately, an algorithm of lower complexity than either can be derived.

*Version 1.5: Nov 2, 2006 10:05 pm -0600

[†]<http://creativecommons.org/licenses/by/1.0>

¹"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

²"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

³"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

⁴"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

⁵"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

⁶"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

⁷"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

Motivation for split-radix algorithm

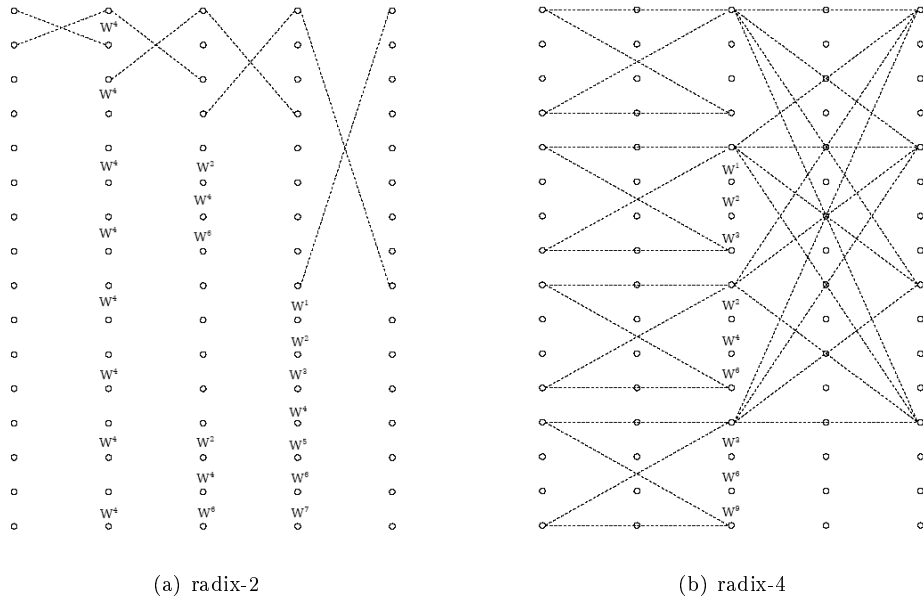


Figure 1: See Decimation-in-Time (DIT) Radix-2 FFT⁸ and Radix-4 FFT Algorithms⁹ for more information on these algorithms.

An alternative derivation notes that radix-2 butterflies of the form shown in Figure 2 can merge twiddle factors from two successive stages to eliminate one-third of them; hence, the split-radix algorithm requires only about two-thirds as many multiplications as a radix-2 FFT.

⁸"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

⁹"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

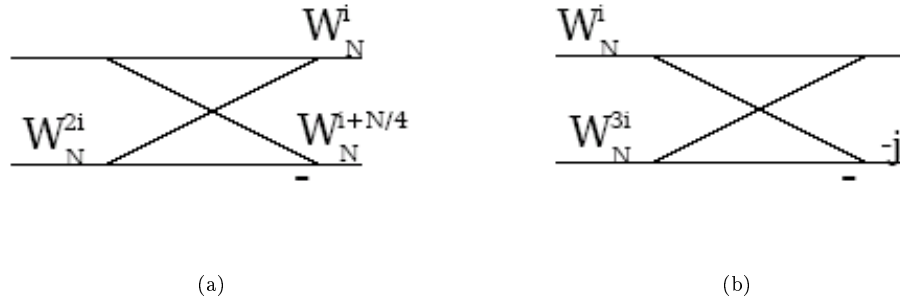


Figure 2: Note that these two butterflies are equivalent

The split-radix algorithm can also be derived by mixing the radix-2¹⁰ and radix-4¹¹ decompositions.

DIT Split-radix derivation

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i \frac{2\pi \times (2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) e^{-\left(i \frac{2\pi(4n+1)k}{N}\right)} + \\
 &\sum_{n=0}^{\frac{N}{4}-1} x(4n+3) e^{-\left(i \frac{2\pi(4n+3)k}{N}\right)} = \text{DFT}_{\frac{N}{2}}[x(2n)] + W_N^k \text{DFT}_{\frac{N}{4}}(x(4n+1)) + \\
 &W_N^{3k} \text{DFT}_{\frac{N}{4}}(x(4n+3))
 \end{aligned} \tag{1}$$

Figure 3 illustrates the resulting split-radix butterfly.

¹⁰"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

¹¹"Radix-4 FFT Algorithms" <<http://cnx.org/content/m12027/latest/>>

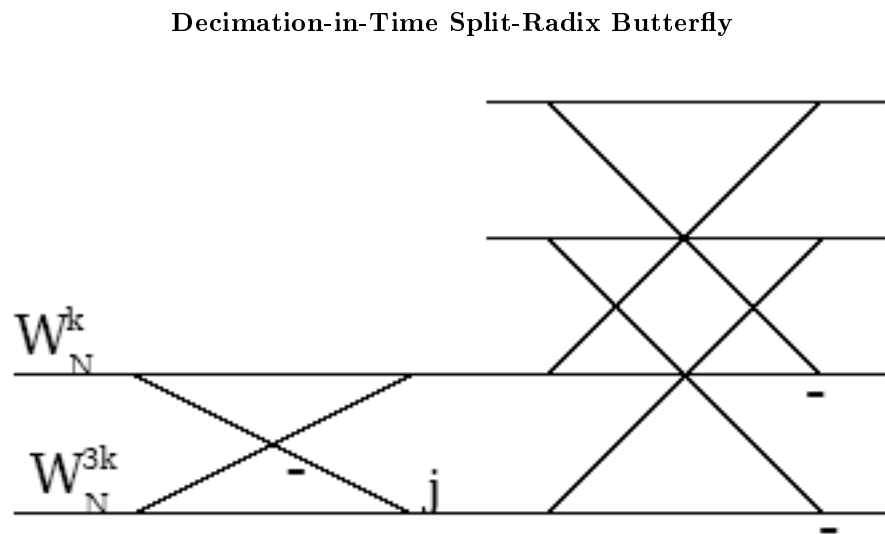


Figure 3: The split-radix butterfly mixes radix-2 and radix-4 decompositions and is L-shaped

Further decomposition of the half- and quarter-length DFTs yields the full split-radix algorithm. The mix of different-length FFTs in different parts of the flowgraph results in a somewhat irregular algorithm; *Sorensen et al.*[3] show how to adjust the computation such that the data retains the simpler radix-2 bit-reverse order. A decimation-in-frequency split-radix FFT can be derived analogously.

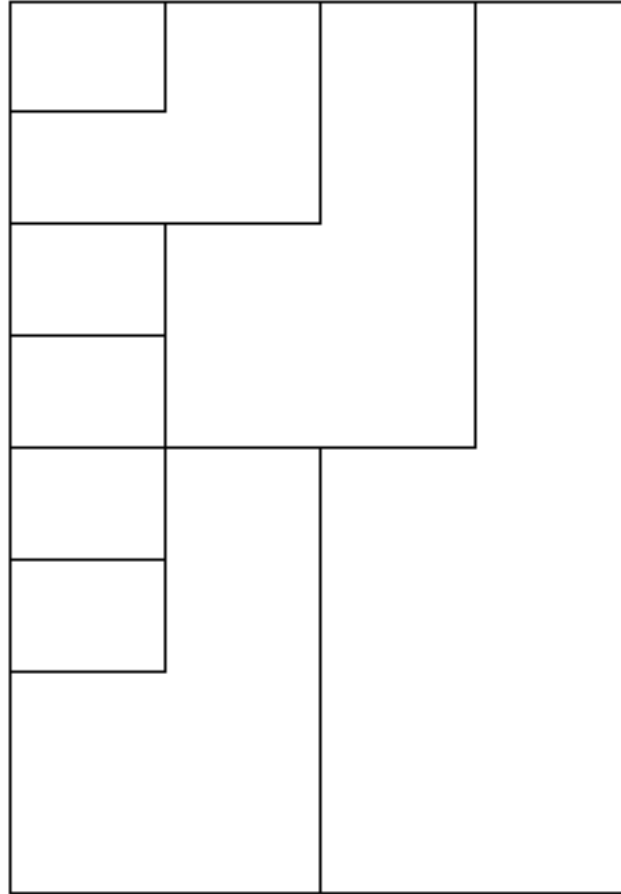


Figure 4: The split-radix transform has L-shaped butterflies

The multiplicative complexity of the split-radix algorithm is only about two-thirds that of the radix-2 FFT, and is better than the radix-4 FFT or any higher power-of-two radix as well. The additions within the complex twiddle-factor multiplies are similarly reduced, but since the underlying butterfly tree remains the same in all power-of-two algorithms, the butterfly additions remain the same and the overall reduction in additions is much less.

Operation Counts

	Complex M/As	Real M/As (4/2)	Real M/As (3/3)
Multiplies	$O\left[\frac{N}{3}\log_2 N\right]$	$\frac{4}{3}N\log_2 N - \frac{38}{9}N + 6 + \frac{2}{9}(-1)^M$	$N\log_2 N - 3N + 4$
Additions	$O[N\log_2 N]$	$\frac{8}{3}N\log_2 N - \frac{16}{9}N + 2 + \frac{2}{9}(-1)^M$	$3N\log_2 N - 3N + 4$

Table 1**Comments**

- The split-radix algorithm has a somewhat irregular structure. Successful programs have been written (Sorensen[3]) for uni-processor machines, but it may be difficult to efficiently code the split-radix algorithm for vector or multi-processor machines.
- *G. Bruun's algorithm*[1] requires only $N - 2$ more operations than the split-radix algorithm and has a regular structure, so it might be better for multi-processor or special-purpose hardware.
- The execution time of FFT programs generally depends more on compiler- or hardware-friendly software design than on the exact computational complexity. See Efficient FFT Algorithm and Programming Tricks¹² for further pointers and links to good code.

References

- [1] G. Bruun. Z-transform dft filters and ffts. *IEEE Transactions on Signal Processing*, 26:56–63, February 1978.
- [2] P. Duhamel and H. Hollman. Split-radix fft algorithms. *Electronics Letters*, 20:14–16, Jan 5 1984.
- [3] M.T. Heideman H.V. Sorensen and C.S. Burrus. On computing the split-radix fft. *IEEE Transactions on Signal Processing*, 34(1):152–156, 1986.
- [4] S.G Johnson and M. Frigo. A modified split-radix fft with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 54, 2006.
- [5] R. Yavne. An economical method for calculating the discrete fourier transform. *Proc. AFIPS Fall Joint Computer Conf.*, 33:115–125, 1968.

¹²"Efficient FFT Algorithm and Programming Tricks" <<http://cnx.org/content/m12021/latest/>>