

## Chapter 2

# Fast Solutions to Real-Data Discrete Fourier Transform

**Abstract** This chapter discusses the two approaches conventionally adopted for dealing with the real-data DFT problem. The first approach involves the design of specialized fast algorithms, such as those due to Bergland and Bruun, which are geared specifically to addressing real-data applications and therefore able to exploit, in a direct way, the real-valued nature of the data – which is known to result in a Hermitian-symmetric frequency spectrum. The second approach, which is the most commonly adopted, particularly for applications requiring a hardware solution, involves re-structuring the data so as to use an existing complex-data FFT algorithm, possibly coupled with pre-FFT and/or post-FFT stages, to produce the DFT of either one or two (produced simultaneously) real-valued data sets – such solutions thus said to be obtained via a “real-from-complex” strategy. A discussion is finally provided relating to the results obtained in the chapter.

### 2.1 Introduction

Since the original developments of spectrum analysis in the eighteenth century, the vast majority of real-world applications have been concerned with the processing of real-valued data, where the data generally corresponds to amplitude measurements of some signal of interest. As a result, there has always been a genuine practical need for fast solutions to the real-data DFT with two quite distinct approaches evolving over this period to address the problem. The first and more intellectually challenging approach involves trying to design specialized algorithms which are geared specifically to real-data applications and therefore able to exploit, in a direct way, the real-valued nature of the data which is known to result in a Hermitian-symmetric frequency spectrum, whereby for the case of an  $N$ -point transform

$$\operatorname{Re} \left( X^{(F)}[k] \right) = \operatorname{Re} \left( X^{(F)}[N - k] \right) \quad (2.1)$$

and

$$\operatorname{Im} \left( X^{(F)}[k] \right) = -\operatorname{Im} \left( X^{(F)}[N - k] \right), \quad (2.2)$$

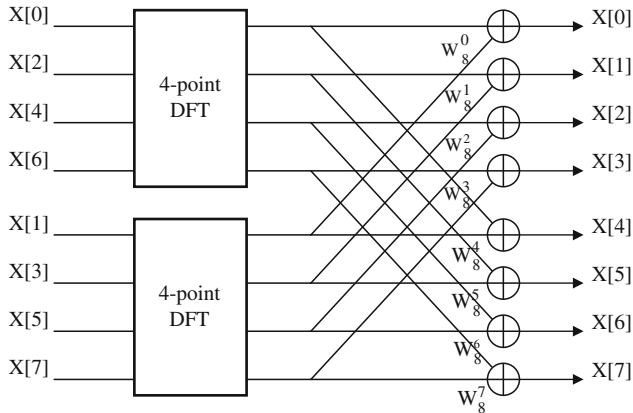
so that one half of the DFT outputs are in fact redundant. Such solutions, as typified by the Bergland [1] and Bruun [3, 13] algorithms, only need therefore to produce one half of the DFT outputs. The second and less demanding approach – but also the most commonly adopted, particularly for applications requiring a hardware solution – involves re-structuring the data so as to use an existing complex-data FFT algorithm, possibly coupled with pre-FFT and/or post-FFT stages, to produce the DFT of either one or two (produced simultaneously) real-valued data sets – such solutions thus said to be obtained via a *real-from-complex* strategy [17]. Both of these approaches are now discussed in some detail prior to a summary of their relative merits and drawbacks.

## 2.2 Real-Data FFT Algorithms

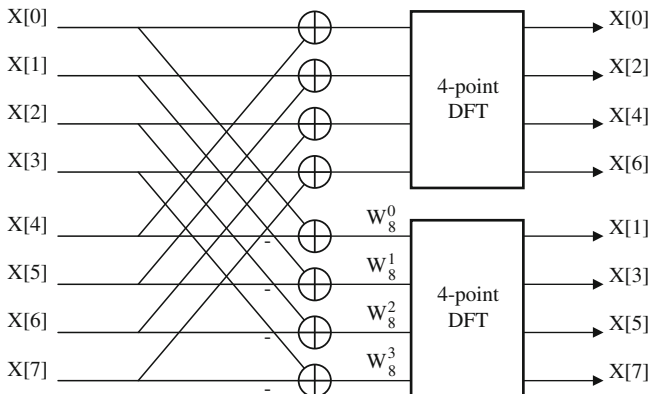
Since the re-emergence of computationally-efficient FFT algorithms, as initiated by the published work of James Cooley and John Tukey in the mid-1960s [5], a number of attempts have been made [1, 3, 6, 7, 9, 10, 12, 18, 19] at producing fast algorithms that are able to directly exploit the spectral symmetry that arises from the processing of real-valued data. Two such algorithms are those due to Glenn Bergland (1968) and Georg Bruun (1978) and these are now briefly discussed so as to give a flavour of the type of algorithmic structures that can result from pursuing such an approach. The Bergland algorithm effectively modifies the DIT version of the familiar Cooley–Tukey radix-2 algorithm [2] to account for the fact that only one half of the DFT outputs need to be computed, whilst the Bruun algorithm adopts an unusual recursive polynomial-factorization approach – note that the DIF version of the Cooley–Tukey fixed-radix algorithm, referred to as the Sande–Tukey algorithm [2], may also be expressed in such a form – which involves only real-valued polynomial coefficients until the last stage of the computation, making it thus particularly suited to the computation of the real-data DFT. Examples of the signal flow graphs (SFGs) for both DIT and DIF versions of the radix-2 FFT algorithm are as given below in Figs. 2.1 and 2.2, respectively.

### 2.2.1 The Bergland Algorithm

The Bergland algorithm is a real-data FFT algorithm based upon the observation that the frequency spectrum arising from the processing of real-valued data is Hermitian-symmetric, so that only one half of the DFT outputs needs to be computed. Starting with the DIT version of the familiar complex-data Cooley–Tukey radix-2 FFT algorithm, if the input data is real valued, then at each of the  $\log_2 N$  temporal stages of the algorithm the computation involves the repeated combination of two transforms to yield one longer double-length transform. From this, Bergland observed that the property of Hermitian symmetry may actually be exploited at each of the



**Fig. 2.1** Signal flow graph for DIT decomposition of eight-point DFT



**Fig. 2.2** Signal flow graph for DIF decomposition of eight-point DFT

$\log_2 N$  stages of the algorithm. Thus, as all the odd-addressed output samples for each such double-length transform form the second half of the frequency spectrum, which can in turn be straightforwardly obtained from the property of spectral symmetry, Bergland's algorithm instead uses those memory locations for storing the imaginary components of the data.

Thus, with Bergland's algorithm, given that the input data sequence is real valued, all the intermediate results may be stored in just  $N$  memory locations – each location thus corresponding to just one word of memory. The computation can also be carried out in an *in-place* fashion – whereby the outputs from each butterfly are stored in the same set of memory locations as used by the inputs – although the indices of the set of butterfly outputs are not in bit-reversed order, as they are with the Cooley–Tukey algorithm, being instead ordered according to the *Bergland*

*ordering* [1], as also are the indices of the twiddle factors or trigonometric coefficients. However, the *normal ordering* of the twiddle factors may, with due care, be converted to the Bergland ordering and the Bergland ordering of the FFT outputs subsequently converted to the normal ordering, as required for an efficient *in-place* solution [1, 17].

Thus, the result of the above modifications is an FFT algorithm with an arithmetic complexity of  $O(N \log_2 N)$  which yields a reduction of two saving, compared to the conventional zero-padded complex-data FFT solution – to be described in Section 2.3.1 – in terms of both arithmetic complexity and memory requirement.

### 2.2.2 The Brunn Algorithm

The Bruun algorithm is a real-data FFT algorithm based upon an unusual recursive polynomial-factorization approach, proposed initially for the case of  $N$  input samples where  $N$  is a power of two, but subsequently generalized to arbitrary even-number transform sizes by Hideo Murakami in 1996 [12].

Recall firstly, from Chapter 1, that the  $N$ -point DFT can be written in normalized form as

$$X^{(F)}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk} \quad k = 0, 1, \dots, N-1 \quad (2.3)$$

where the transform kernel is derived from the term

$$W_N = \exp(-i2\pi/N), \quad (2.4)$$

the primitive  $N$ th complex root of unity. Then, by defining the polynomial  $x(z)$  whose coefficients are those elements of the sequence  $\{x[n]\}$ , such that

$$x(z) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] \cdot z^n, \quad (2.5)$$

it is possible to view the DFT as a reduction of this polynomial [11], so that

$$X^{(F)}[k] = x(W_N^k) = x(z) \bmod (z - W_N^k) \quad (2.6)$$

where “mod” stands for the modulo operation [11] which denotes the polynomial remainder upon division of  $x(z)$  by  $(z - W_N^k)$  [11]. The key to fast execution of the Bruun algorithm stems from being able to perform this set of  $N$  polynomial remainder operations in a recursive fashion.

To compute the DFT involves evaluating the remainder of  $x(z)$  modulo some polynomial of degree one, more commonly referred to as a *monomial*, a total of  $N$  times, as suggested by Equations 2.5 and 2.6. To do this efficiently, one can combine the remainders recursively in the following way. Suppose it is required to evaluate

$x(z)$  modulo  $U(z)$  as well as  $x(z)$  modulo  $V(z)$ . Then, by first evaluating  $x(z)$  modulo the polynomial product,  $U(z).V(z)$ , the degree of the polynomial  $x(z)$  is reduced, thereby making subsequent modulo operations less computationally expensive.

Now the product of all of the monomials,  $(z - W_N^k)$ , for values of  $k$  from 0 up to  $N - 1$ , is simply  $(z^N - 1)$ , whose roots are clearly the  $N$  complex roots of unity. A recursive factorization of  $(z^N - 1)$  is therefore required which breaks it down into polynomials of smaller and smaller degree with each possessing as few non-zero coefficients as possible. To compute the DFT, one takes  $x(z)$  modulo each level of this factorization in turn, recursively, until one arrives at the monomials and the final result. If each level of the factorization splits every polynomial into an  $O(1)$  number of smaller polynomials, each with an  $O(1)$  number of non-zero coefficients, then the modulo operations for that level will take  $O(N)$  arithmetic operations, thus leading to a total arithmetic complexity, for all  $\log_2 N$  levels, of  $O(N.\log_2 N)$ , as obtained with the standard Cooley–Tukey radix-2 algorithm.

Note that for  $N$  a power of two, the Bruun algorithm factorizes the polynomial  $(z^N - 1)$  recursively via the rules

$$z^{2M} - 1 = (z^M - 1)(z^M + 1) \quad (2.7)$$

and

$$z^{4M} + a.z^{2M} + 1 = (z^{2M} + \sqrt{2-a}.z^M + 1)(z^{2M} - \sqrt{2-a}.z^M + 1), \quad (2.8)$$

where “ $a$ ” is a constant such that  $|a| \leq 2$ . On completion of the recursion, when  $M = 1$ , there remains polynomials of degree two that can each be evaluated modulo two roots of the form  $(z - W_N^k)$  for each polynomial. Thus, at each recursive stage, all of the polynomials may be factorized into two parts, each of half the degree and possessing at most three non-zero coefficients, leading to an FFT algorithm with an  $O(N.\log_2 N)$  arithmetic complexity. Moreover, since all the polynomials have purely real coefficients, at least until the last stage, they quite naturally exploit the special case where the input data is real valued, thereby yielding a reduction of two saving, compared to the conventional zero-padded complex-data FFT solution to be discussed in Section 2.3.1, in terms of both arithmetic complexity and memory requirement.

## 2.3 Real-From-Complex Strategies

By far the most common approach to solving the real-data DFT problem is that based upon the use of an existing complex-data FFT algorithm as it simplifies the problem, at worst, to one of designing pre-FFT and/or post-FFT stages for the packing of the real-valued data into the correct format required for input to the FFT algorithm and for the subsequent unpacking of the FFT output data to obtain the spectrum (or spectra) of the original real-valued data set (or sets). Note that any fast

algorithm may be used for carrying out the complex-data FFT, so that both DIT and DIF versions of fixed-radix FFTs, as already discussed, as well as more sophisticated FFT designs such as those corresponding to the mixed-radix, split-radix, prime factor, prime-length and Winograd's nested algorithms [8, 11], for example, might be used.

### ***2.3.1 Computing One Real-Data DFT via One Full-Length Complex-Data FFT***

The most straightforward approach to the problem involves first packing the real-valued data into the real component of a complex-valued data sequence, padding the imaginary component with zeros – this action more commonly referred to as *zero padding* – and then feeding the resulting complex-valued data set into a complex-data FFT. The arithmetic complexity of such an approach is clearly identical to that obtained when a standard complex-data FFT is applied to genuine complex-valued data, so that no computational benefits stemming from the nature of the data are achieved with such an approach. On the contrary, computational resources are wasted with such an approach, as excessive arithmetic operations are performed for the computation of the required outputs and twice the required amount of memory used for the storage of the input/output data.

### ***2.3.2 Computing Two Real-Data DFTs via One Full-Length Complex-Data FFT***

The next approach to the problem involves computing two N-point real-data DFTs, simultaneously, by means of one N-point complex-data FFT. This is achieved by packing the first real-valued data sequence into the real component of a complex-valued data sequence and the second real-valued data sequence into its imaginary component. Thus, given two real-valued data sequences,  $\{g[n]\}$  and  $\{h[n]\}$ , a complex-valued data sequence,  $\{x[n]\}$ , may be simply obtained by setting

$$x[n] = g[n] + i \cdot h[n], \quad (2.9)$$

with the  $k$ th DFT output of the resulting data sequence being written in normalized form, in terms of the DFTs of  $\{g[n]\}$  and  $\{h[n]\}$ , as

$$X^{(F)}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk}$$

$$\begin{aligned}
&= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} g[n] \cdot W_N^{nk} + i \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} h[n] \cdot W_N^{nk} \\
&= G[k] + iH[k] \\
&= (G_R[k] - H_I[k]) + i(G_I[k] + H_R[k]), \tag{2.10}
\end{aligned}$$

where  $G_R[k]$  and  $G_I[k]$  are the real and imaginary components, respectively, of  $G[k]$  – the same applying to  $H_R[k]$  and  $H_I[k]$  with respect to  $H[k]$ . Similarly, the  $(N-k)$ th DFT output may be written in normalized form as

$$\begin{aligned}
X^{(F)}[N-k] &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] \cdot W_N^{n(N-k)} \\
&= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} g[n] \cdot W_N^{-nk} + i \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} h[n] \cdot W_N^{-nk} \\
&= G^*[k] + iH^*[k] \\
&= (G_R[k] + H_I[k]) + i(-G_I[k] + H_R[k]) \tag{2.11}
\end{aligned}$$

where the superscript “\*” stands for the operation of complex conjugation, so that upon combining the expressions of Equations 2.10 and 2.11, the DFT outputs  $G[k]$  and  $H[k]$  may be written, in terms of the DFT outputs  $X^{(F)}[k]$  and  $X^{(F)}[N-k]$ , as

$$\begin{aligned}
G[k] &= G_R[k] + iG_I[k] \\
&= \frac{1}{2} \left( \text{Re} \left[ X^{(F)}[k] + X^{(F)}[N-k] \right] + i \text{Im} \left[ X^{(F)}[k] - X^{(F)}[N-k] \right] \right) \tag{2.12}
\end{aligned}$$

and

$$\begin{aligned}
H[k] &= H_R[k] + iH_I[k] \\
&= \frac{1}{2} \left( \text{Im} \left[ X^{(F)}[k] + X^{(F)}[N-k] \right] - i \text{Re} \left[ X^{(F)}[k] - X^{(F)}[N-k] \right] \right) \tag{2.13}
\end{aligned}$$

where the terms  $\text{Re}(X^{(F)}[k])$  and  $\text{Im}(X^{(F)}[k])$  denote the real and imaginary components, respectively, of  $X^{(F)}[k]$ .

Thus, it is evident that the DFT of the two real-valued data sequences,  $\{g[n]\}$  and  $\{h[n]\}$ , may be computed simultaneously, via one full-length complex-data FFT algorithm, with the DFT of the sequence  $\{g[n]\}$  being as given by Equation 2.12 and that of the sequence  $\{h[n]\}$  by Equation 2.13. The pre-FFT data packing stage is quite straightforward in that it simply involves the assignment of one real-valued data sequence to the real component of the complex-valued data array and one real-valued data sequence to its imaginary component. The post-FFT data unpacking stage simply involves separating out the two spectra from the complex-valued FFT output data, this involving two real additions/subtractions for each real-data DFT output together with two scaling operations each by a factor of 2 (which in fixed-point hardware reduces to that of a simple right shift operation).

### 2.3.3 Computing One Real-Data DFT via One Half-Length Complex-Data FFT

Finally, the last approach to the problem involves showing how an  $N$ -point complex-data FFT may be used to carry out the computation of one  $2N$ -point real-data DFT. The  $k$ th DFT output of the  $2N$ -point real-valued data sequence  $\{x[n]\}$  may be written in normalized form as

$$\begin{aligned} X^{(F)}[k] &= \frac{1}{\sqrt{2N}} \sum_{n=0}^{2N-1} x[n] \cdot W_{2N}^{nk} \quad k = 0, 1, \dots, N-1 \\ &= \frac{1}{\sqrt{2N}} \sum_{n=0}^{N-1} x[2n] \cdot W_N^{nk} + W_{2N}^k \frac{1}{\sqrt{2N}} \sum_{n=0}^{N-1} x[2n+1] \cdot W_N^{nk} \end{aligned} \quad (2.14)$$

which, upon setting  $g[n] = x[2n]$  and  $h[n] = x[2n+1]$ , becomes

$$\begin{aligned} X^{(F)}[k] &= \frac{1}{\sqrt{2N}} \sum_{n=0}^{N-1} g[n] \cdot W_N^{nk} + W_{2N}^k \frac{1}{\sqrt{2N}} \sum_{n=0}^{N-1} h[n] \cdot W_N^{nk} \quad k = 0, 1, \dots, N-1 \\ &= G[k] + W_{2N}^k H[k]. \end{aligned} \quad (2.15)$$

Therefore, by setting  $y[n] = g[n] + i \cdot h[n]$  and exploiting the combined expressions of Equations 2.10 and 2.11, the DFT output  $Y[k]$  may be written as

$$Y[k] = (G_R[k] - H_I[k]) + i (G_I[k] + H_R[k]) \quad (2.16)$$

and that for  $Y[N-k]$  as

$$Y[N-k] = (G_R[k] + H_I[k]) + i (-G_I[k] + H_R[k]). \quad (2.17)$$

Then, by combining the expressions of Equations 2.15–2.17, the real component of  $X^{(F)}[k]$  may be written as

$$\begin{aligned} X_R^{(F)}[k] &= \frac{1}{2} \operatorname{Re} (Y[k] + Y[N-k]) \\ &\quad + \frac{1}{2} \cos(k\pi/N) \cdot \operatorname{Im} (Y[k] + Y[N-k]) \\ &\quad - \frac{1}{2} \sin(k\pi/N) \cdot \operatorname{Re} (Y[k] - Y[N-k]) \end{aligned} \quad (2.18)$$

and the imaginary component as

$$X_I^{(F)}[k] = \frac{1}{2} \operatorname{Im} (Y[k] - Y[N-k])$$



$$\begin{aligned}
& + \frac{1}{2} \sin(k\pi/N) \cdot \text{Im}(Y[k] + Y[N - k]) \\
& - \frac{1}{2} \cos(k\pi/N) \cdot \text{Re}(Y[k] - Y[N - k]) .
\end{aligned} \tag{2.19}$$

Thus, it is evident that the DFT of one real-valued data sequence,  $\{x[n]\}$ , of length  $2N$ , may be computed via one  $N$ -point complex-data FFT algorithm, with the real component of the DFT output being as given by Equation 2.18 and the imaginary component of the DFT output as given by Equation 2.19. The pre-FFT data packing stage is conceptually simple, but nonetheless burdensome, in that it involves the assignment of the even-addressed samples of the real-valued data sequence to the real component of the complex-valued data sequence and the odd-addressed samples to its imaginary component. The post-FFT data unpacking stage is considerably more complex than that required for the approach of Section 2.3.2, requiring the application of eight real additions/subtractions for each DFT output, together with two scaling operations, each by a factor of 2, and four real multiplications by pre-computed trigonometric coefficients.

## 2.4 Data Re-ordering

All of the fixed-radix formulations of the FFT, at least for the case where the transform length is a power of two, require that either the inputs to or the outputs from the transform be permuted according to the *digit reversal* mapping [4]. In fact, it is possible to place the data re-ordering either before or after the transform for both the DIT and DIF formulations [4]. For the case of a radix-2 algorithm the data re-ordering is more commonly known as the *bit reversal* mapping, being based upon the exchanging of single bits, whilst for the radix-4 case it is known as the *di-bit reversal* mapping, being based instead upon the exchanging of pairs of bits.

Such data re-ordering, when mapped onto a uni-processor sequential computing device, may be carried out via the use of either:

1. An LUT, at the expense of additional memory; or
2. A fast algorithm using just shifts, additions/subtractions and memory exchanges; or
3. A fast algorithm that also makes use of a small LUT – containing the reflected bursts of ones that change on the lower end with incrementing – to try and optimize the speed at the cost of a slight increase in memory

with the optimum choice being dependent upon the available resources and the time constraints of the application.

Alternatively, when the digit-reversal mapping is appropriately parallelized, it may be mapped onto a multi-processor parallel computing device, such as an FPGA, possessing multiple banks of fast memory, thus enabling the time-complexity to be greatly reduced – see the recent work of Ren et al. [14] and Seguel and Bollman [16].

The optimum approach to digit-reversal is dictated very much by the operation of the FFT, namely whether the FFT is of *burst* or *streaming* type, as discussed in Chapter 6.

## 2.5 Discussion

The aim of this chapter has been to highlight both the advantages and the disadvantages of the conventional approaches to the solution of the real-data DFT problem. As is evident from the examples discussed in Section 2.2, namely the Bergland and Bruun algorithms, the adoption of specialized real-data FFT algorithms may well yield solutions possessing attractive performance metrics in terms of arithmetic complexity and memory requirement, but generally this is only achieved at the expense of a more complex algorithmic structure when compared to those of the highly-regular fixed-radix designs. As a result, such algorithms would not seem to lend themselves particularly well to being mapped onto parallel computing equipment.

Similarly, from the examples of Section 2.3, namely the real-from-complex strategies, the regularity of the conventional fixed-radix designs may only be exploited at the expense of introducing additional processing modules, namely the pre-FFT and/or post-FFT stages for the packing of the real-valued data into the correct format required for input to the FFT algorithm and for the subsequent unpacking of the FFT output data to obtain the spectrum (or spectra) of the original real-valued data set (or sets). An additional set of problems associated with the real-from-complex strategies, at least when compared to the more direct approach of a real-data FFT, relate to the need for increased memory and increased processing delay to allow for the possible acquisition/processing of pairs of data sets.

It is worth noting that an alternative DSP-based approach to those discussed above is to first convert the real-valued data to complex-valued data by means of a wideband DDC process, this followed by the application of a conventional complex-data FFT. Such an approach, however, introduces an additional function to be performed – typically an FIR filter with length dependent upon the performance requirements of the application – which also introduces an additional processing delay prior to the execution of the FFT. Drawing upon a philosophical analogy, namely the maxim of the fourteenth century Franciscan scholar, William of Occam, commonly known as “Occam’s razor” [15], why use two functions when just one will suffice. A related and potentially serious problem arises when there is limited information available on the signal under analysis as such information might well be compromised via the filtering operation, particularly when the duration of the signal is short relative to that of the transient response of the filter – as might be encountered, for example, with problems relating to the detection of extremely short duration dual-tone multi-frequency (DTMF) signals.

Thus, there are clear drawbacks to all such approaches, particularly when the application requires a solution in hardware using parallel computing equipment, so

that the investment of searching for alternative solutions to the fast computation of the real-data DFT is still well merited. More specifically, solutions are required possessing both highly regular designs that lend themselves naturally to mapping onto parallel computing equipment and attractive performance metrics, in terms of both arithmetic complexity and memory requirement, but without requiring excessive packing/unpacking requirements and without incurring the latency problems (as arising from the increased processing delay) associated with the adoption of certain of the real-from-complex strategies.

## References

1. G.D. Bergland, A fast Fourier transform algorithm for real-valued series. *Comm. ACM*. **11**(10) (1968)
2. E.O. Brigham, *The Fast Fourier Transform and Its Applications*. (Prentice Hall, Englewood Cliffs, NJ, 1988)
3. G. Bruun, *Z-Transform DFT Filters and FFTs*. *IEEE Trans. ASSP*. **26**(1) (1978)
4. E. Chu, A. George, *Inside the FFT Black Box* (CRC Press, Boca Raton, FL, 2000)
5. J.W. Cooley, J.W. Tukey, An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **19**(4), 297–301 (1965)
6. P. Duhamel, Implementations of split-radix FFT algorithms for complex, real and real-symmetric data. *IEEE Trans. ASSP* **34**(2), 285–295 (1986)
7. P. Duhamel, M. Vetterli, Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data. *IEEE Trans. ASSP* **35**(6), 818–824 (1987)
8. P. Duhamel, M. Vetterli, Fast Fourier transforms: A tutorial review and a state of the art. *Signal Process.* **19**, 259–299 (1990)
9. O. Ersoy, Real discrete Fourier transform. *IEEE Trans. ASSP* **33**(4) (1985)
10. J.B. Marten, Discrete Fourier transform algorithms for real valued sequences. *IEEE Trans. ASSP*. **32**(2) (1984)
11. J.H. McClellan, C.M. Rader, *Number Theory in Digital Signal Processing* (Prentice Hall, Englewood Cliffs, NJ, 1979)
12. H. Murakami, Real-valued fast discrete Fourier transform and cyclic convolution algorithms of highly composite even length. *Proc. ICASSP* **3**. 1311–1314 (1996)
13. H.J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*. (Springer, Berlin, 1981)
14. G. Ren, P. Wu, D. Padua, *Optimizing Data Permutations for SIMD Devices* (PDL'06, Ottawa, Ontario, Canada, 2006)
15. B. Russell, *History of Western Philosophy* (George Allen & Unwin, London, 1961)
16. J. Seguel, D. Bollman, A framework for the design and implementation of FFT permutation algorithms. *IEEE Trans. Parallel Distrib. Syst.* **11**(7), 625–634 (2000)
17. G.R.L. Sohie, W. Chen, *Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors*. Downloadable document from website: [www.Motorola.com](http://www.Motorola.com)
18. H.V. Sorensen, D.L. Jones, M.T. Heideman, C.S. Burrus, Real-valued fast Fourier transform algorithms. *IEEE Trans. ASSP* **35**(6), 849–863 (1987)
19. P.R. Uniyal, Transforming real-valued sequences: Fast Fourier versus fast Hartley transform algorithms. *IEEE Signal Process.* **42**(11) (1994)

The Regularized Fast Hartley Transform  
Optimal Formulation of Real-Data Fast Fourier  
Transform for Silicon-Based Implementation in  
Resource-Constrained Environments

Jones, K.J.

2010, XVII, 200 p. With online files/update., Hardcover

ISBN: 978-90-481-3916-3