Decimation-in-time (DIT) Radix-2 FFT

Discrete Fourier Transform (DFT) is commonly used to analyse any signal or data. Fast Fourier Transform (FFT) is the widely used algorithm to implement Discrete Fourier Transform (DFT). The simplest and easiest algorithm for FFT is the decimation-in-time (DIT) radix-2 FFT. In this design exercise, a C program will be developed to perform DIT radix-2 FFT.

I. DECIMATION IN TIME

The radix-2 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into two parts: a sum over the even-numbered discrete-time indices $n = [0, 2, 4, \cdots, N-2]$ and a sum over the odd-numbered indices $n = [1, 3, 5, \cdots, N-1]$ as follows

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \exp\left(\frac{-i2\pi nk}{N}\right), \\ &= \sum_{n=0}^{N/2-1} x(2n) \exp\left(\frac{-i2\pi(2n)k}{N}\right) + \sum_{n=0}^{N/2-1} x(2n+1) \exp\left(\frac{-i2\pi(2n+1)k}{N}\right), \\ &= \sum_{n=0}^{N/2-1} x(2n) \exp\left(\frac{-i2\pi(n)k}{N/2}\right) + \exp\left(\frac{-i2\pi k}{N}\right) \sum_{n=0}^{N/2-1} x(2n+1) \exp\left(\frac{-i2\pi(2n+1)k}{N/2}\right), \\ &= \mathrm{DFT}_{\frac{N}{2}}\left[[x(0), x(2), x(4), \cdots, x(N-2)]\right] + W_N^k \mathrm{DFT}_{\frac{N}{2}}\left[[x(1), x(3), x(5), \cdots, x(N-1)]\right]. \end{aligned} \tag{1}$$

The mathematical simplifications in (1) reveal that all DFT frequency outputs $X(k)$ can be computed as the sum of the outputs of two length$-\frac{N}{2}$ DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where the odd-indexed short DFT is multiplied by a so-called **twiddle factor** term $W_N^k = \exp(\frac{-i2\pi k}{N})$. This is called a decimation in time because the time samples are rearranged in alternating groups, and a radix-2 algorithm because there are two groups. Figure 1 graphically illustrates this form of the DFT computation, where for convenience the frequency outputs of the length$-\frac{N}{2}$ DFT of the even- indexed time samples are denoted $G(k)$ and those of the odd-indexed samples as $H(k)$. Because of the periodicity with $\frac{N}{2}$ frequency samples of these length$-\frac{N}{2}$ DFTs, $G(k)$ and $H(k)$ can be used to compute two of the length$-N$ DFT frequencies, namely $X(k)$ and $X(k + N/2)$, but with a different twiddle factor. This reuse of these short-length DFT outputs gives the FFT its computational savings.

Whereas direct computation of all N DFT frequencies according to the DFT equation would require N^2 complex multiplies and $N^2 - N$ complex additions (for complex-valued data), by reusing the results of the two short-length

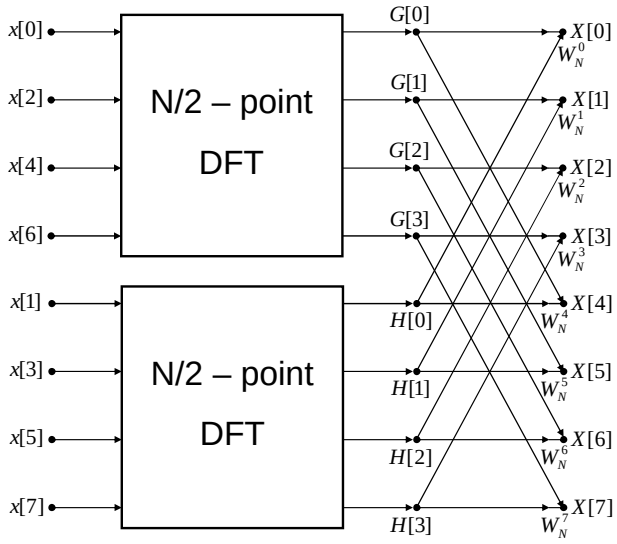

FIG. 1: Decimation in time of a length$-N$ DFT into two length$-\frac{N}{2}$ DFTs followed by a combining stage.

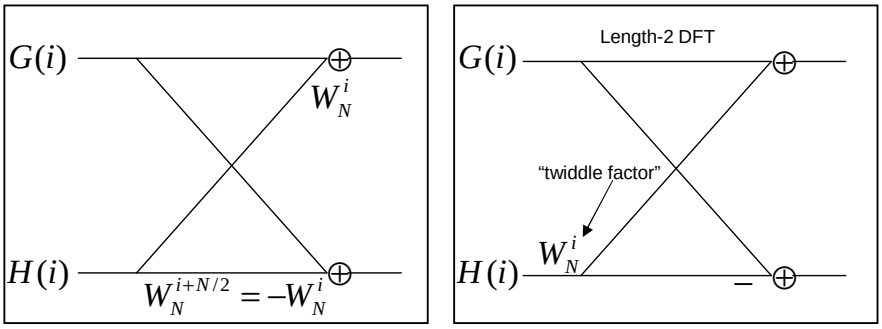

FIG. 2: Radix-2 DIT butterfly simplification: both operations produce the same outputs.

DFTs as illustrated in Figure 1, the computational cost is now
New Operation Counts

- $2(\frac{N}{2})^2 + N = \frac{N^2}{2} + N$ complex multiplies
- $2\frac{N}{2}(\frac{N}{2} - 1) + N = \frac{N^2}{2}$ complex additions

This simple reorganization and reuse has reduced the total computation by almost a factor of two over direct DFT computation!

II. ADDITIONAL SIMPLIFICATION

A basic butterfly operation is shown in Figure 2, which requires only $\frac{N}{2}$**twiddle-factor** multiplies per **stage**. It is worthwhile to note that, after merging the twiddle factors to a single term on the lower branch, the remaining butterly is actually a length-2 DFT! The theory of multi-dimensional index maps 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.

III. RADIX-2 DECIMATION-IN-TIME FFT

The same radix-2 decimation in time can be applied recursively to the two length$-\frac{N}{2}$ DFTs to save computa- tion. When successively applied until the shorter and shorter DFTs reach length-2, the result is the radix-2 DIT FFT algorithm (Figure 3).

The full radix-2 decimation-in-time decomposition illustrated in Figure 3 using the simplified butterflies (Figure 2) involves $M = \log_2 N$ stages, each with $N2$ butterflies per stage. Each butterfly requires 1 complex multiply and 2 adds per butterfly. The total cost of the algorithm is thus
Computational cost of radix-2 DIT FFT

- $\frac{N}{2}\log_2 N$ complex multiplies
- $N \log_2 N$ complex adds

This is a remarkable savings over direct computation of the DFT. For example, a length-1024 DFT would require 1048576 complex multiplications and 1047552 complex additions with direct computation, but only 5120 complex multiplications and 10240 complex additions using the radix-2 FFT, a savings by a factor of 100 or more. The relative savings increase with longer FFT lengths, and are less for shorter lengths.

Modest additional reductions in computation can be achieved by noting that certain twiddle factors, namely Using special butterflies for W_N^0, $W_N^{\frac{N}{2}}$, $W_N^{\frac{N}{4}}$, $W_N^{\frac{N}{8}}$, $W_N^{\frac{3N}{8}}$, require no multiplications, or fewer real multiplies than other ones. By implementing special butterflies for these twiddle factors as discussed in FFT algorithm and programming tricks, the computational cost of the radix-2 decimation-in-time FFT can be reduced to

- $2N \log_2 N - 7N + 12$ complex multiplies
- $3N \log_2 N - 3N + 4$ complex adds

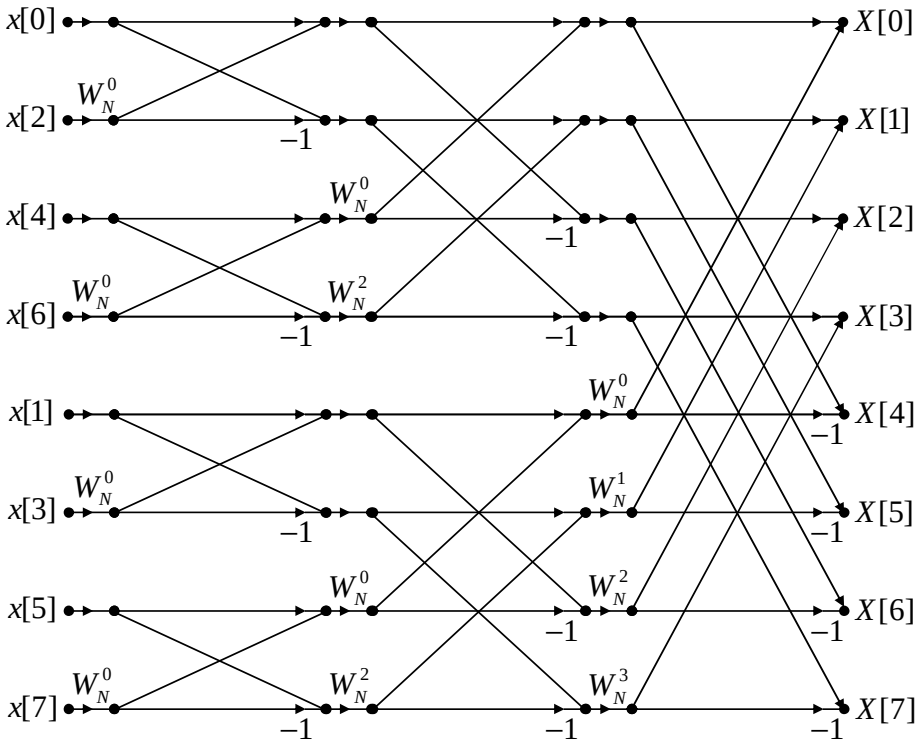

FIG. 3: Radix-2 Decimation-in-Time FFT algorithm for a length-8 signal.

In-order index	In-order index in binary	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

TABLE I: Bit Reversal

NOTE: In a decimation-in-time radix-2 FFT as illustrated in Figure 3, the input is in bit-reversed order (hence "decimation-in-time"). That is, if the time-sample index n is written as a binary number, the order is that binary number reversed. The bit-reversal process is illustrated for a length$-N = 8$ example below.

Example 1: $N = 8$
It is important to note that, if the input signal data are placed in bit-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.

IV. INVERSE DFT (IFFT)

$$x(n) \;=\; \frac{1}{N}\sum_{k=0}^{N-1} X(k)\exp\left(\frac{i2\pi nk}{N}\right). \tag{2}$$