Overview

- *The computation of the Discrete Fourier Transform*
  - *motivation*
  - *the direct computation of the DFT*
  - *the efficient computation of the DFT (FFT)*
  - *the decimation-in-time FFT algorithm (DIT)*
    - *structure*
    - *conclusions relative to the FFT-DIT algorithm*
    - *programming of the DIT algorithm*
  - *the decimation-in-frequency FFT algorithm (DIF)*
    - structure
    - *conclusions relative to the FFT-DIF algorithm*
  - important realization aspects
    - *the necessity of "bit reversal"*
    - *the efficient computation of the W_N^r coefficients*
    - *case where N is not a power of two*
    - *the computation of the IDFT using the DFT*
    - *the DFT of real signals*

The computation of the Discrete Fourier Transform

- Motivation
  - the properties of the DFT make it particularly suitable to analyze and design systems in the Fourier domain,
  - since techniques generally known as Fast Fourier Transform (FFT) exist to realize a fast and efficient computation of the DFT, the DFT represents in practice a common and very important tool in many digital signal processing applications,
    - **NOTE 1**: the maximum efficiency in the fast computation of the DFT is achieved when all N samples uniformly distributed on the unit circumference of the Z plane are computed,
    - **NOTE 2**: when samples of the Fourier transform need to be computed on a fraction only of the unit circumference (*i.e.,* less than 2π), other flexible computation techniques exist (although less efficient than the FFT) such as the Goertzel algorithm or the "chirp" Z transform algorithm,
    - **NOTE 3**: the computational efficiency of the FFT is so considerable that frequently it is preferable to replace the conventional computation of the linear convolution between two discrete sequences, by the DFTs of both sequences, their multiplication in the discrete-frequency domain and the inverse DFT of the resulting product sequence,
    - **NOTE 4**: a common way to evaluate the complexity of a fast DFT computation algorithm involves analyzing the total number of addition and multiplication operations involved.

The computation of the Discrete Fourier Transform

- direct computation of the DFT

equation for the direct DFT (analysis): $$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn},\quad \mathrm{k}=0,1,\cdots,\mathrm{N}-1$$

equation for the inverse DFT (synthesis): $$x[n]=\frac{1}{N}\sum_{k=0}^{N-1}X[k]W_N^{-kn},\quad \mathrm{n}=0,1,\cdots,\mathrm{N}-1$$

where $W_N=e^{-j2\pi/N}$ and x[n] and X[k] are, in general, complex-valued.

- NOTE: since the analysis and synthesis DFT equations are quite similar, it can be anticipated that except for small adaptations, it is possible to compute the inverse DFT using an algorithm implementing the direct DFT and *vice-versa*.

Admitting that both x[n] and X[k] are complex-valued, it can be easily concluded that the total number of arithmetic operations involved in the computation of the direct DFT (recalling that all N values of X[k] need to be computed and recalling the real operations involved in the product of two complex numbers: (a+jb)(c+jd)=(ac-bd)+j(bc+ad)) is given by:

$\rightarrow$ complex multiplications: $N^2 \equiv 4N^2$ real multiplications + $2N^2$ real additions

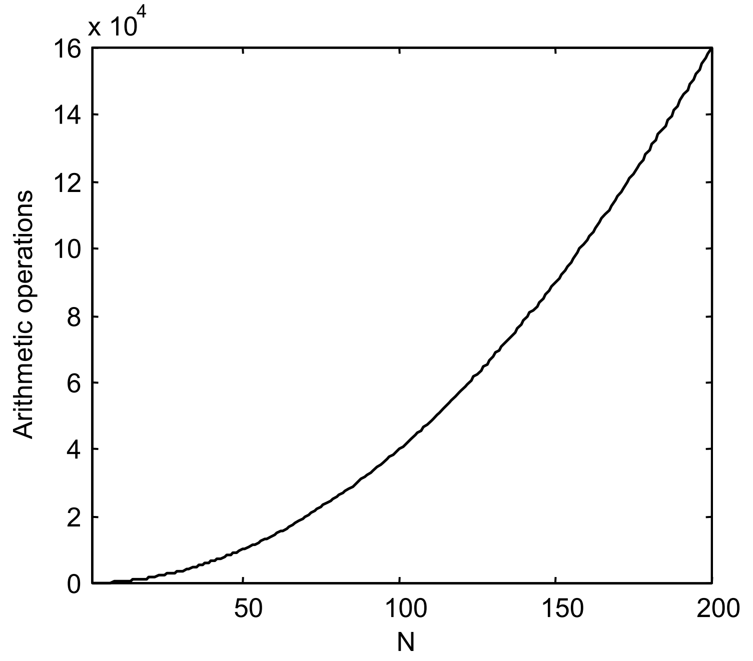
$\rightarrow$ complex additions: $N(N-1) \equiv 2N(N-1)$ real additions

The computation of the Discrete Fourier Transform

hence, in total we have:

$\rightarrow$ real multiplications: $4N^2$ $\rightarrow$ real additions: $N(4N-2)$

These results reveal that the computational cost involved in the direct computation of the DFT is proportional to N^2, which in practical terms may become prohibitive if N is “large”:



On the other hand, it is also required that in the realization of the DFT, besides the input x[n], all coefficients W_N^{kn} are stored and are accessible.

The computation of the Discrete Fourier Transform

- The efficient computation of the DFT

  – the techniques of efficient computation of the DFT not only reduce the computational cost of the direct DFT computation but also make it proportional to N at a smaller rate than that of the direct computation (N^2),

  – most DFT computation techniques take advantage of the properties of symmetry and periodicity of the W_N^{kn} coefficients:

    - $W_N^{-kn} = W_N^{k(N-n)} = (W_N^{kn})^*$ (complex conjugate symmetry)
    - $W_N^{kn} = W_N^{k(N+n)} = W_N^{(N+k)n}$ (periodicity in n and k)

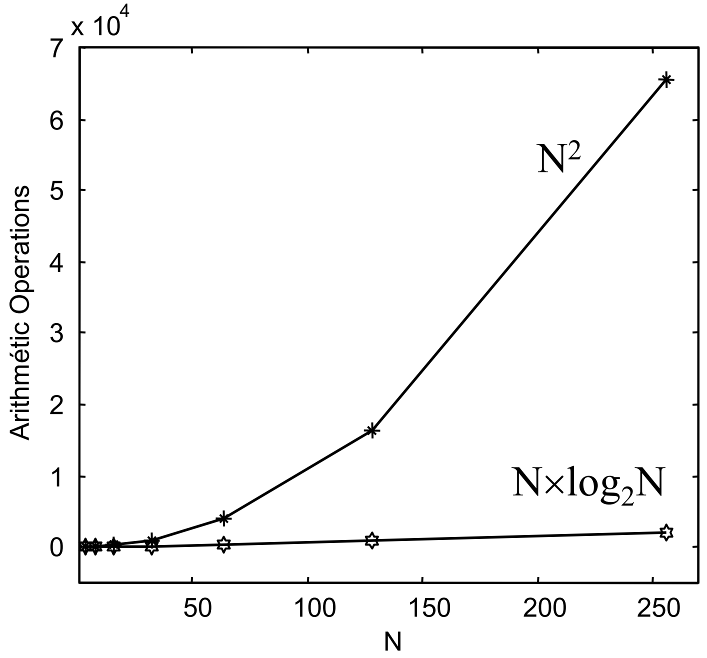
  – the most relevant techniques for the efficient computation of the DFT are generally know as Fast Fourier Transform (FFT) and, despite the fact that their origins date back to 1805 with works of Gauss, it was only after 1965, with the publication of an algorithm by Cooley and Tukey for the computation of the DFT, that those techniques had a clear development and practical impact.

The computation of the Discrete Fourier Transform

– the basic principle of the FFT is the computation of a DFT of a given length using shorter DFTs, this is achieved by means of a <u>decomposition</u> of the <u>input sequence,</u> or of the <u>output sequence,</u> of the DFT, in successively smaller sequences by taking advantage of the properties of symmetry and periodicity of the W_N^{kn} coefficients

– if the decomposition is relative to the input sequence, the corresponding FFT algorithm is called <u>decimation in time</u> (DIT), if the decomposition is relative to the output sequence, the corresponding algorithm is called <u>decimation in frequency</u> (DIF),

– as we shall see, in case N is a power of two number, the computational cost of the DIT or DIF algorithm is proportional to $N \times \log_2 N$ which, when compared to the direct computation ($\propto N^2$), represents a computational gain of $N / \log_2 N$, as illustrated in the following figure:



– in the literature [e.g., Oppenheim, section 9.2], it is usual to describe a recursive DFT computation technique known as the Goertzel algorithm; despite the fact that we don't review it here, we highlight some of its characteristics that are thoroughly described in standard textbooks [Oppenheim, Mitra]:

- the Goertzel algorithm is a recursive algorithm that (contrarily to the FFT) avoids the need to store the W_N^{kn} coefficients,
- the computational cost of the Goertzel algorithm is less than that of the direct computation but is still proportional to N^2,
- while the FFT algorithms are particularly suitable when all N DFT coefficients X[k] are needed, the Goertzel algorithm is suitable when a number M of DFT coefficients need to be computed for any $M \leq N$; in fact, the Goertzel algorithm or even the direct DFT computation may be preferable from the point of view of the computational load when $M < \log_2 N$.

The computation of the Discrete Fourier Transform

- The decimation-in-time FFT algorithm (DIT)

  – structure

    - admitting to simplify that N is a power of two number (*i.e.,* $N=2^v$, with v integer) this algorithm splits the computation of the N-length DFT in shorter DFTs, by successively dividing the input sequence in two sub-sequences with half the length, one of them regarding the samples having even index, and the other regarding the samples having odd index; thus, for the first stage of decomposition we have:

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn}=\sum_{n=0}^{N/2-1}x[2n]W_N^{k2n}+\sum_{n=0}^{N/2-1}x[2n+1]W_N^{k(2n+1)}=\sum_{n=0}^{N/2-1}x[2n]\left(W_N^2\right)^{kn}+W_N^k\sum_{n=0}^{N/2-1}x[2n+1]\left(W_N^2\right)^{kn},\quad \mathrm{k}=0,1,\cdots,\mathrm{N}-1$$

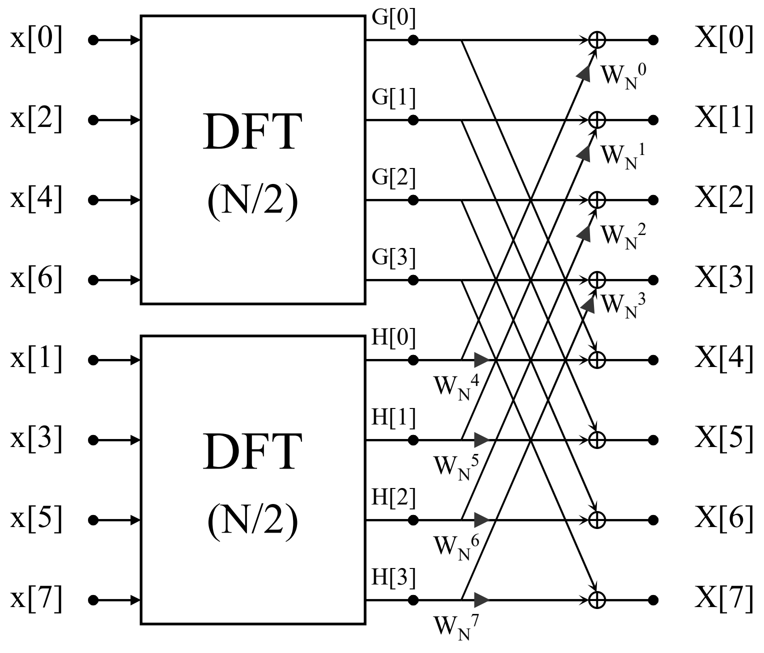
by noting, however, that $W_N^2=e^{-j2\pi/(N/2)}=W_{N/2}$, we have:

$$X[k]=\sum_{n=0}^{N/2-1}x[2n]W_{N/2}^{kn}+W_N^k\sum_{n=0}^{N/2-1}x[2n+1]W_{N/2}^{kn}=G[k]+W_N^kH[k],\quad \mathrm{k}=0,1,\cdots,\mathrm{N}-1$$

where G[k] and H[k] are two N/2-length DFTs, the former involves the even-indexed input samples, and the latter involves the odd-indexed input samples.

The computation of the Discrete Fourier Transform

Considering N=8, and recalling that G[k] and H[k] are periodic with period N/2 (which for N=8 leads to H[4]=H[0] and G[4]=G[0]), the previous decomposition may be illustrated as follows:



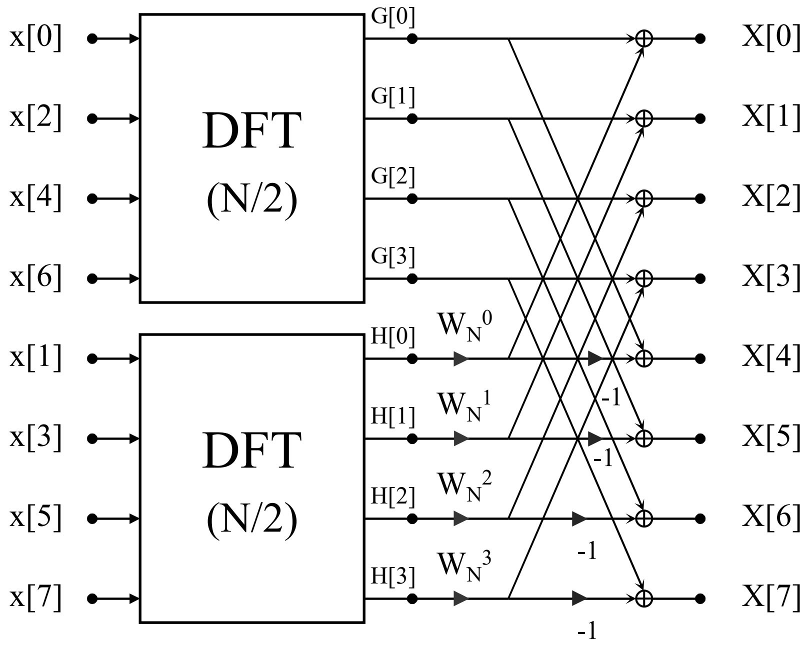
→ From a computational point of view, this decomposition represents already a significant gain since it replaces a structure whose computational cost is proportional to N^2 (in terms of multiplications or additions), by another one whose computational cost is proportional to $2\times(N/2)^2+N = N^2/2+N$, which is less than N^2.

The computation of the Discrete Fourier Transform

→ the computation of each coefficient X[k], when combining G[k] and H[k], requires a complex multiplication using W_N^k, where $0 \le k \le N-1$. Taking advantage of the symmetry of these coefficients, which is clear from:

$$\boxed{W_N^{(\ell+N/2)} = W_N^{\ell} W_N^{N/2} = -W_N^{\ell}}$$

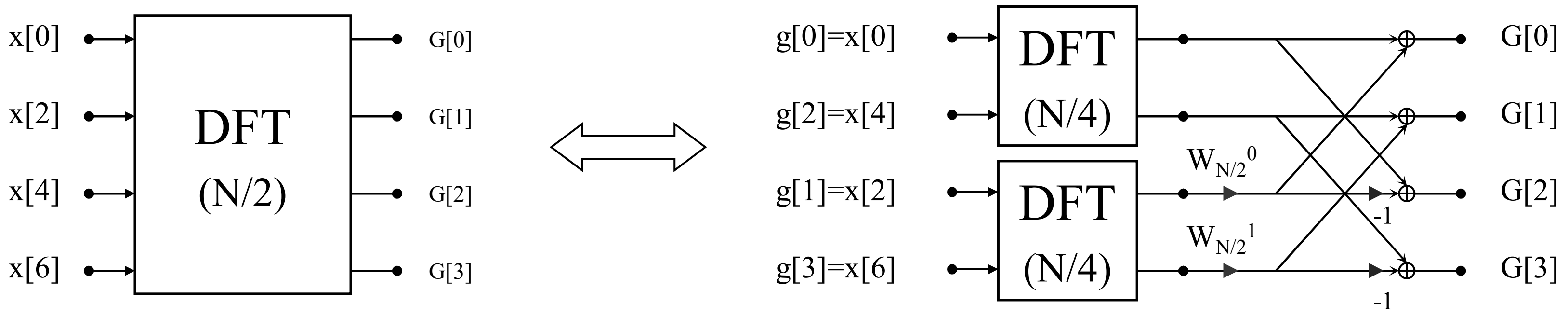
the number of multiplications by W_N^k may be reduced by 50% by reutilizing in the second half of the computation of X[k], the multiplications involving the coefficients W_N^k and already performed when computing the first half of X[k]. For example, for N=8, in particular we have $X[3]=G[3]+ W_8^3 H[3]$ and also $X[7]=G[3]+ W_8^7 H[3]=G[3]- W_8^3 H[3]$, which allows to redesign the previous structure to a simplified form:



The computation of the Discrete Fourier Transform

The computational cost of this structure (in terms of number of multiplications) is thus proportional to $N^2/2+N/2$, and the number of additions (a subtraction is equivalent to an addition) is proportional to $N^2/2+N$.

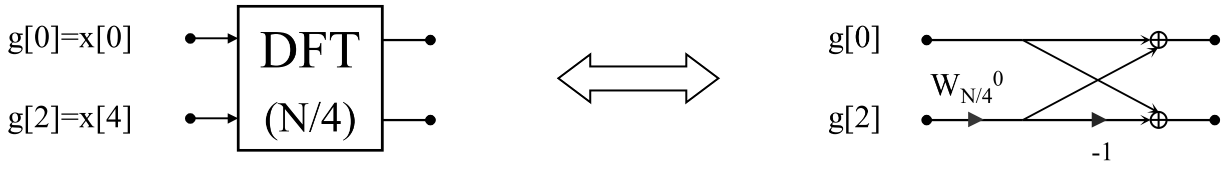
- The previous procedure may again be repeated regarding each DFT of length N/2. For example, the N/2-length DFT G[k] (N/2=4 in our example) may be replaced by the following sub-structure:



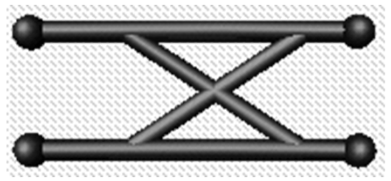
$\rightarrow$ its computational cost (multiplications) is proportional to $2(N/4)^2+N/4 = N^2/8+N/4$, making that if this structure is used in the computation of G[k] and H[k], the total computational cost (multiplications) involved in the computation of X[k] is proportional to $2\times N^2/8+2N/4+N/2 = N^2/4+N$.

The computation of the Discrete Fourier Transform

In our example, where N=8, one more optimization possibility exists that concerns the computation of the DFTs whose length is N/4=2. It may be simplified as illustrated for the first DFT of the previous structure:



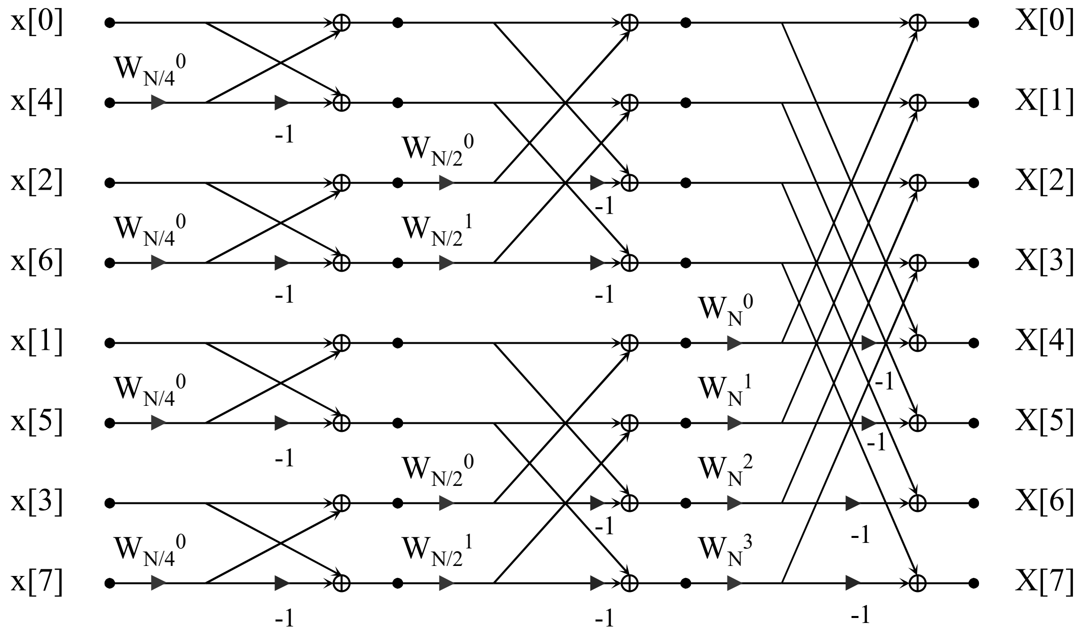
This basic computational structure is known in the literature as butterfly due to its shape, and has a computational cost corresponding to a single complex multiplication and two complex additions.



If this structure is replaced in that of the previous slide and the result is replaced in the initial structure, we obtain the complete structure represented in the next slide (N=8).

The computation of the Discrete Fourier Transform

- Structure of the FFT-DIT algorithm for N=8:



→ The total computational cost of this structure and concerning multiplications, is thus equal to $1\times N/2+N/4+N/4+N/2$ = #stages $\times N/2 = N/2 \times \log_2 N$.

→ as it is easily recognized from the illustration, the total computational cost of this structure and concerning additions is $N \times \log_2 N$.

The computation of the Discrete Fourier Transform

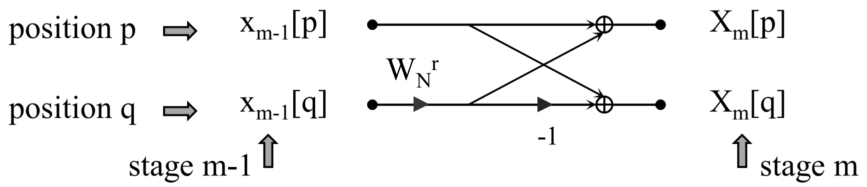
– Conclusions regarding the FFT-DIT algorithm

$\rightarrow$ The computational structure of the DIT algorithm, when N is a power of two, is based on an organization in $\log_2 N$ stages of several groups of butterflies.

$\rightarrow$ If the number of inputs or outputs of each butterfly is two (as we have considered so far, although there are other possibilities, for example four) then the algorithm is said to be radix-2, under this assumption:

- the number of butterflies per stage is constant and equal to N/2,
- from stage to stage, the number of groups varies by a factor of two, and the number of butterflies in each stage varies inversely by the same factor.

$\rightarrow$ The basic computational structure which is the butterfly, generates outputs which replace the input values without affecting other data positions in the same stage of the structure, which not only facilitates the parallel processing but also allows the in-place processing; that is, from stage to stage, the input data are replaced by the output data, which avoids the need for additional memory.



The computation of the Discrete Fourier Transform

→ Since the DIT results from the successive decomposition (from the N-length DFT, till the smallest DFT of length two) of the input data sequence of the DFT into two sub-sequences, one corresponding to the even-indexed samples and the other corresponding to the odd-indexed samples, the order of the data at the input of the DIT algorithm must reflect the order resulting from the successive decomposition; this order has the interesting property that it may be expressed by the reversing the bits of the binary number representing the natural order of the input ("bit-reversal").

- Example for N=8:

decimal | binary		binary | decimal
$0 \equiv 000$		$000 \equiv 0$
$1 \equiv 001$		$100 \equiv 4$
$2 \equiv 010$		$010 \equiv 2$
$3 \equiv 011$	bit	$110 \equiv 6$
$4 \equiv 100$	$\xrightarrow{}$ reversed	$001 \equiv 1$
$5 \equiv 101$		$101 \equiv 5$
$6 \equiv 110$		$011 \equiv 3$
$7 \equiv 111$		$111 \equiv 7$

The computation of the Discrete Fourier Transform

- **NOTE 1**: several Digital Signal Processors have a special addressing mode that is suited to realize “bit-reversed” addressing,
  - Note: Matlab offers two specialized functions: `bitrevorder(.)` and `digitrevorder(.)`

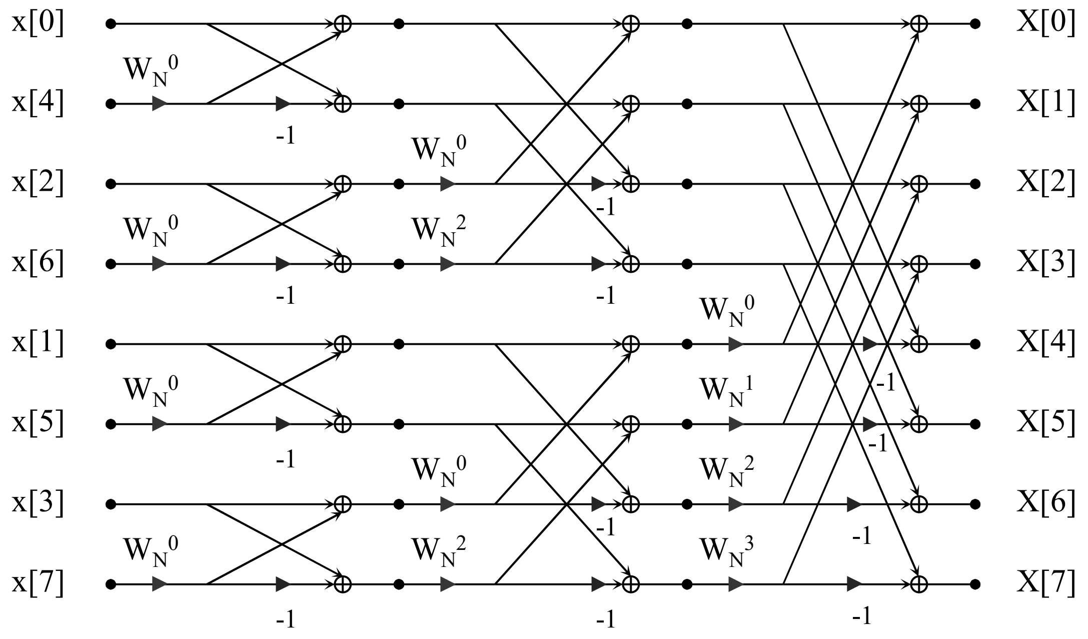
- **NOTE 2**: some of the indexes are equal to their own “bit-reversed” version,

- **NOTE 3**: given the nature of the DIT algorithm, a correction of the order of the input data to its “bit-reversed” order <u>is necessary</u> so that the output data is organized according to its natural sequential order.

The computation of the Discrete Fourier Transform

– Programming of the FFT-DIT algorithm

$\rightarrow$ The programming of a computational procedure concerning the FFT-DIT results from the analysis of its butterfly structure, as repeated below for N=8, taking that of slide 13 and including a slight modification of the $W_{N/b}^{a}$ complex coefficients that are now expressed as W_N^{ab}, which facilitates the programming since the butterfly is a basic computational structure that is iteratively computed.



The computation of the Discrete Fourier Transform

→ The computational procedure includes two phases:

1- first, implement the bit-reversal of the input data,

2- then the computation is iterated for all N/2 × $\log_2 N$ butterflies (N/2 butterflies per stage × $\log_2 N$ stages), insuring that for each butterfly, addresses are correctly generated that index:

a) the input/output data,

b) the W_N^r complex coefficient

→ Using a "pseudo-C" programming language, the sketch of a possible code implementing the FFT-DIT is presented in the next slide.

The computation of the Discrete Fourier Transform

- FFT-DIT

```
/* inicialization */
estg='number of stages'; N=2^estg; grup=N; butf=1;
/* bit reversal */
bitreversal(x, N);
/*
 * iterates over all butterflies using
 * a stage, group and butterfly counter
 */
for (i=0; i<estg; i++)
{
   grup=grup/2;
   for (j=0; j<grup; j++)
   {
      for (k=0; k<butf; k++)
      {
         ind1=j*2*butf+k;
         ind2=ind1+butf;
         arg=grup*k;
         temp=x[ind2]*W_N^arg;
         x[ind2]=x[ind1]-temp;
         x[ind1]=x[ind1]+temp;
      }
   }
   butf=butf*2;
}
```

The computation of the Discrete Fourier Transform

- The decimation in frequency FFT algorithm (DIF)
  - structure
    - Instead of structuring the DFT computation on the basis of a successive decomposition of the x[n] input sequence into shorter sequences, the FFT-DIF algorithm implements a successive decomposition of the output X[k] sequence into shorter sequences, such that these are computed using shorter DFTs. Admitting to simplify that N is a power of two number, we have for the first level of decomposition and considering even-indexed ouputs:

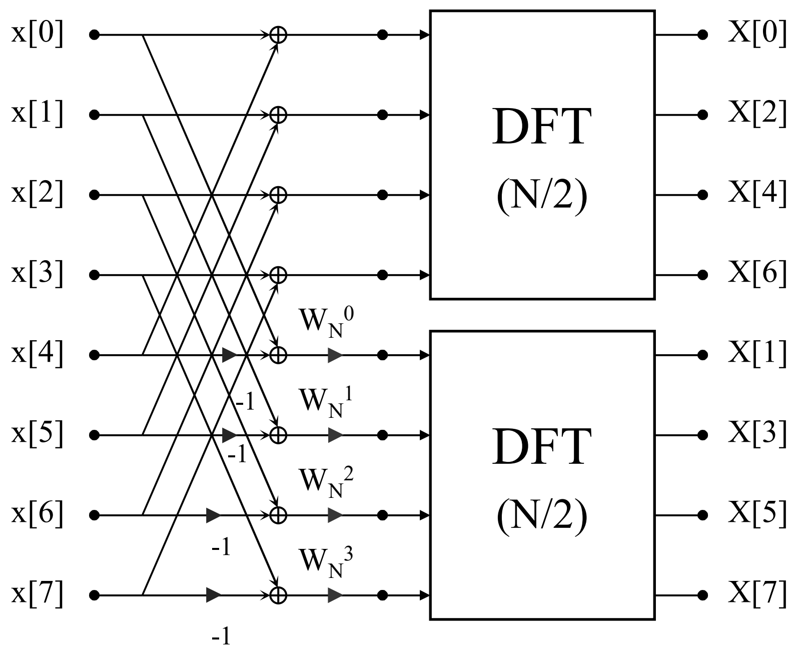
$$X[2\ell]=\sum_{n=0}^{N-1}x[n]W_N^{2\ell n}=\sum_{n=0}^{N/2-1}x[n]W_N^{2\ell n}+\sum_{n=0}^{N/2-1}x[n+N/2]W_N^{2\ell(n+N/2)}=\sum_{n=0}^{N/2-1}x[n]W_{N/2}^{\ell n}+\sum_{n=0}^{N/2-1}x[n+N/2]W_{N/2}^{\ell n}=\sum_{n=0}^{N/2-1}\left(x[n]+x[n+N/2]\right)W_{N/2}^{\ell n}$$

and, also, for the odd-indexed outputs:

$$X[2\ell+1]=\sum_{n=0}^{N-1}x[n]W_N^{(2\ell+1)n}=\sum_{n=0}^{N/2-1}x[n]W_N^{(2\ell+1)n}+\sum_{n=0}^{N/2-1}x[n+N/2]W_N^{(2\ell+1)(n+N/2)}=\sum_{n=0}^{N/2-1}\left(x[n]-x[n+N/2]\right)W_N^{n}W_{N/2}^{\ell n}$$

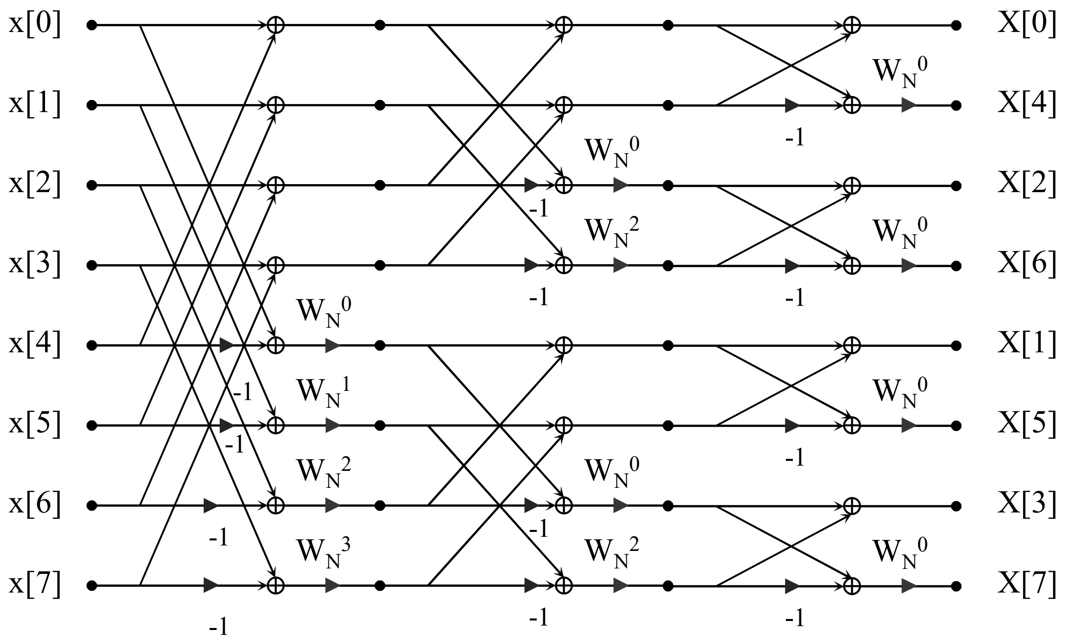
The computation of the Discrete Fourier Transform

Considering N=8, the previous equations give rise to the following structure:



The computation of the Discrete Fourier Transform

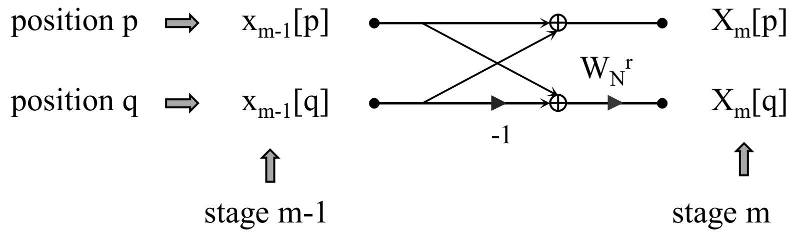
By iterating this procedure of splitting the DFT computation into shorter DFTs, we get to the following structure, considering N=8 (where the $W_{N/b}^{a}$ coefficients have already been replaced by W_N^{ab}):



The computation of the Discrete Fourier Transform

– Conclusions relative to the FFT-DIF algorithm

→ by comparing the structure of the previous slide with that of slide 17 that is relative to the FFT-DIT, it can be concluded that the structure of the FFT-DIF algorithm results from a transposition of that of the FFT-DIT algorithm, consisting of a reversal of the signal flow in all branches, an exchange between derivation nodes and summation nodes, and an exchange between inputs and outputs. In particular, the DIF butterfly results from the transposition of that of the DIT butterfly:



→ It results from the previous that:

- as it happens with the DIT algorithm, it is also true with the DIF algorithm that the number of butterflies per stage is N/2 and that the total number of arithmetic operations is **N/2×log$_2$N multiplications** and **N×log$_2$N** complex **additions**,
- being a transposition of the DIF algorithm, the DIF algorithm also allows the "in-place" computation,
- contrarily to the DIT algorithm, the inputs of the DIF algorithm are sequentially ordered while the outputs are "bit-reversed" ordered which requires a reordering to the natural sequential order that can be done using the same "bit-reversal" algorithm already seen for the FFT-DIT.

The computation of the Discrete Fourier Transform

- Implementation particularities
  - on the necessity of “bit-reversal”
    - in addition to the implementation structures discussed in the previous slides, other alternatives exist [see Oppenheim, sections 9.3 and 9.4] including a modified FFT-DIT algorithm that considers sequentially ordered inputs and “bit-reversed” ordered outputs, or a modified FFT-DIF algorithm that considers “bit-reversed” ordered inputs and sequentially ordered outputs. These alternatives may be combined in such a way that there is no need for any bit-reversal reordering in a system including signal analysis and synthesis. For example, in the implementation of the circular convolution, the direct DFTs of the two inputs may be realized using the FFT-DIF algorithm (the corresponding outputs will be bit-reversed ordered) and may be subsequently multiplied. The result keeps the bit-reversed order and may be presented at the input of an inverse DFT that is implemented using an FFT-DIT algorithm. The end result will then be sequentially ordered, as desired.

The computation of the Discrete Fourier Transform

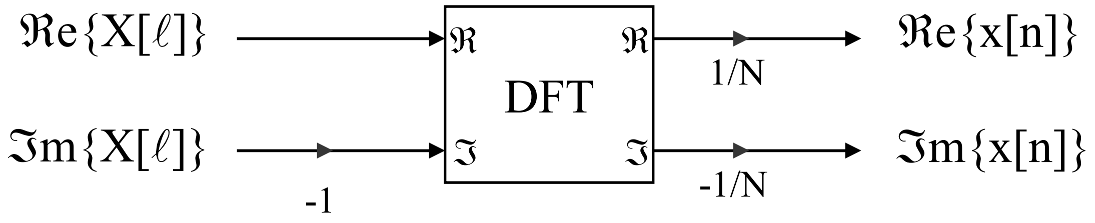
- Computation of the W_N^r coefficients
  - According to the assessment of the most critical issues in each application in terms of processing power or memory, we may prefer to store all necessary W_N^r coefficients, or to compute them “on-the-fly” when they are needed. In the first case tables of the values of `cos()` e `sin()` functions are needed that may, however, be optimized to avoid redundancies. In the second case, it is possible to compute the coefficients recursively: $W_N^r = W_N^{r-1} W_N$ but care must be taken in order to avoid the propagation of representation or rounding errors (*i.e.,* due to the finite resolution) that are particularly critical in the case of fixed-point processing.
- Case when N is not a power-of-two number
  - We have described the radix-2 decomposition but other possibilities exist such as radix-4 or a combination of *radix-2* and *radix-4*, that is, mixed radix, that offer other computational advantages when N is a power of 4 or a combined power of 2 or 4. Computational gains are also possible when N is a composite integer number, i.e., it can be expressed as a product of two integer numbers. In these cases it is also possible to develop fast computational algorithms although their structure and indexing scheme are more complicated than those described here.

The computation of the Discrete Fourier Transform

– the IDFT computation using the DFT

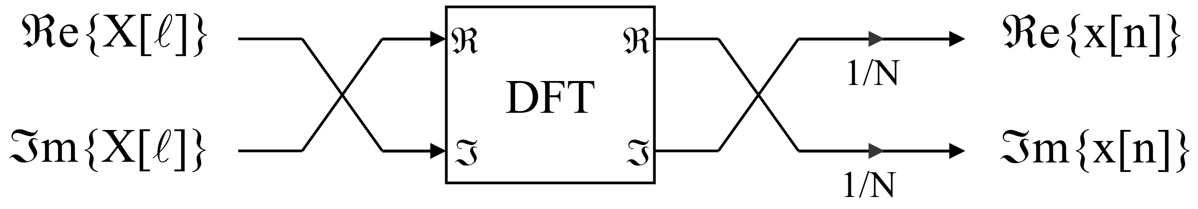
- the development of the DIT and DIF algorithms has been illustrated using the particular case of the DFT implementation. Although it is easy to adapt the previous discussion to the case of the IDFT implementation, there is a practical interest understanding how the utilization of an FFT algorithm implementing the direct DFT transform may be used to compute the inverse DFT. With this purpose in mind, and because the substantive differences between the DFT and IDFT are the sign of the complex exponential and the multiplicative 1/N factor, we describe three alternatives:

1- as $\boxed{x[n]=\frac{1}{N}\left\{\sum_{\ell=0}^{N-1}X^*[\ell]W_N^{\ell n}\right\}^*}$ we have the following possible approach:

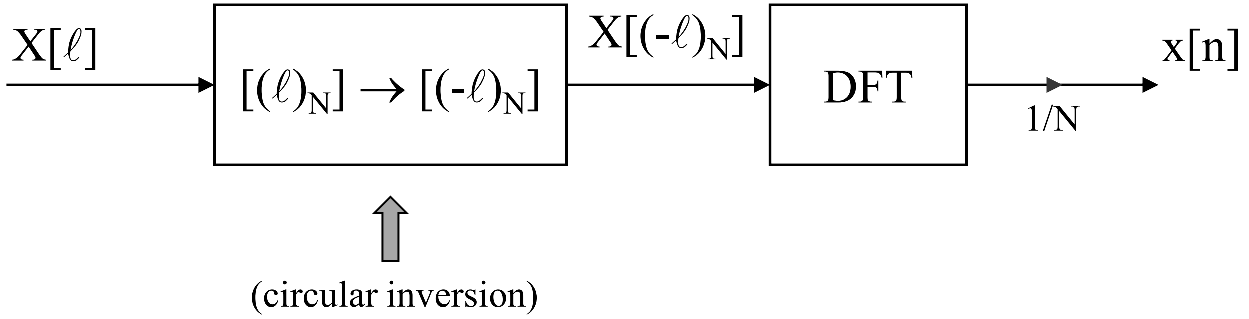


The computation of the Discrete Fourier Transform

2- as $x[n] = j\frac{1}{N}\left\{\sum_{\ell=0}^{N-1} jX^*[\ell]W_N^{\ell n}\right\}^*$ we have the following possible approach:



3- as $x[n] = \frac{1}{N}\left\{\sum_{\ell=0}^{N-1} X\left[(-\ell)_N\right]W_N^{-((-\ell)_N)n}\right\}$ we have the following possible approach:



$X[\ell] \rightleftarrows X[N-\ell],\ \ell=0,1,\ \ldots,\ N/2-1$

NOTE: naturally, one may also write: $x\left[(-n)_N\right] = \frac{1}{N}\left\{\sum_{\ell=0}^{N-1} X[\ell]W_N^{-((-n)_N)\ell}\right\}$

The computation of the Discrete Fourier Transform

– the DFT of real signals

- In general, the DFT and IDFT equations presume complex inputs and outputs. However, it occurs frequently that the data at the input of a DFT is real, or that the data at the output of an IDFT is real, despite the fact that signals are processed in the complex frequency domain for example. In these cases, the utilization of a complex DFT or IDFT represents a computational inefficiency since the imaginary part of the data vector is zero. Two particularly interesting cases that occur frequently in practice are addressed that eliminate that inefficiency:

  1- the computation of the **DFTs of two real signals**, whose length is N, **using a single complex DFT** of length **N**, and

  2- the computation of the **DFT of a real signal** whose length is N **using a single complex DFT** of length **N/2**.

The computation of the Discrete Fourier Transform

1- DFTs of two real-valued signals of length N

→ If $x_1[n]$ and $x_2[n]$ are two real-valued signals of length N, their DFTs comply with the following symmetry:

$$\begin{cases} X_1[k] = X_1^*\left[(-k)_N\right] = X_1^*[N-k] \\ X_2[k] = X_2^*\left[(-k)_N\right] = X_2^*[N-k] \end{cases}$$

by constructing the signal $y[n]= x_1[n] + j\, x_2[n]$, we have:

$$y[n] = x_1[n] + jx_2[n] \overset{\mathrm{F}}{\longleftrightarrow} Y[k] = X_1[k] + jX_2[k]$$

and also: $Y[N-k] = X_1[N-k] + jX_2[N-k]$

and conjugating both sides of this equation, leads to:

$$Y^*[N-k] = X_1^*[N-k] - jX_2^*[N-k] = X_1[k] - jX_2[k]$$

which, when combined with the first equation, allows to recover $X_1[k]$ and $X_2[k]$:

$$X_1[k] = \frac{Y[k]+Y^*[N-k]}{2} = \underbrace{\frac{Y_{\Re e}[k]+Y_{\Re e}[N-k]}{2}}_{\text{even part of the real component of Y[k]}} + j\underbrace{\frac{Y_{\Im m}[k]-Y_{\Im m}[N-k]}{2}}_{\text{odd part of the imaginary component of Y[k]}}$$

$$X_2[k] = \frac{Y[k]-Y^*[N-k]}{2j} = \underbrace{\frac{Y_{\Im m}[k]+Y_{\Im m}[N-k]}{2}}_{\text{even part of the imaginary component of Y[k]}} - j\underbrace{\frac{Y_{\Re e}[k]-Y_{\Re e}[N-k]}{2}}_{\text{odd part of the real component of Y[k]}}$$

The computation of the Discrete Fourier Transform

2- DFT of a real-valued signal whose length is N

$\rightarrow$ is x[n] is a real-vaçued signal whose length is N, and as discussed in the presentation of the “decimation-in-time” algorithm, it is possible to compute its DFT using two DFTs of length N/2:

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn}=\sum_{n=0}^{N/2-1}x[2n]W_{N/2}^{kn}+W_N^k\sum_{n=0}^{N/2-1}x[2n+1]W_{N/2}^{kn}=G[k]+W_N^kH[k] \quad \Leftrightarrow \quad \begin{cases} X[k]=G[k]+W_N^kH[k], & 0\le k\le N/2-1 \\ X[N/2+k]=G[k]-W_N^kH[k], & 0\le k\le N/2-1 \end{cases}$$

This way, it is sufficient to compute the DFTS G[k] and H[k] , whose length (i.e., periodicity) is N/2, and then combine them as performed for the last stage of an FFT-DIT algorithm. As G[k] and H[k] are the Fourier transforms of two real-valued sequences of length N/2, it is possible, as seen in the previous slide, to compute them using a single complex FFT of length N/2 and whose input is:

$$y[n]=g[n]+jh[n]=x[2n]+jx[2n+1] \ , \quad 0\le n\le N/2-1.$$