CHAPTER 6 Efficient Computation of the DFT: Fast Fourier Transform Algorithms

DFT plays an important role in many applications of digital signal processing, including ***linear filtering***, ***correlation analysis***, and ***spectrum analysis***.

Two different approaches are described :

One is a ***divide-and-conquer approach :*** a DFT of size *N* is reduced to the computation of smaller DFTs from which the larger DFT is computed.

The *second* approach is based on the *formulation of the DFT* as a ***linear filtering operation*** on the data.

6.1 Efficient Computation of the DFT: FFT Algorithms

The ***computational problem*** for the **DFT** is to compute the sequence *{ X(k) }* of *N* complex-valued numbers given another sequence of data *{ x(n) }* of the length *N*.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1 \qquad (6.1.1)$$

Where $W_N = e^{-j2\pi/N}$ (6.1.2)

The **IDFT** becomes

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \qquad 0 \le n \le N-1 \qquad (6.1.3)$$

6.1 Effcient Computation of the DFT: FFT Algorithms

The ***efficient computational algorithms*** for the ***DFT*** applies as well to the efficient computation of the ***IDFT***.

For each value of k, direct computation of $X(k)$ involves ***N complex multiplication*** ($4N$ real multiplications). and ***(N – 1) complex additions*** ($4N – 2$ real additions).

To compute all N values of the DFT requires ***N^2 complex multiplications*** and ***$N^2 – N$ complex additions***.

6.1 Effcient Computation of the DFT: FFT Algorithms

Direct computation of the DFT is basically inefficient primarily because it does ***not exploit*** the ***symmetry*** and ***periodicity properties*** of the *phase factor* W_N.

Symmetry property :

$$W_N^{k+N/2} = -W_N^k \qquad (6.1.4)$$

Periodicity property :

$$W_N^{k+N} = W_N^k \qquad (6.1.5)$$

6.1.1 Direct computation of the DFT.

For a complex-and-valued sequence *x(n)* of *N* points, the DFT may be expressed as.

$$X_R(k) = \sum_{n=0}^{N-1} \left[x_R(n) \cos \frac{2\pi kn}{N} + x_I(n) \sin \frac{2\pi kn}{N} \right] \qquad (6.1.6)$$

$$X_I(k) = -\sum_{n=0}^{N-1} \left[x_R(n) \sin \frac{2\pi kn}{N} - x_I(n) \cos \frac{2\pi kn}{N} \right] \qquad (6.1.7)$$

6.1.1 Direct computation of the DFT.

The ***direct computation*** of (6.1.6) and (6.1.7) requires:

1. $2N^2$ evaluations of trigonometric functions.
2. $4N^2$ real multiplications.
3. $4N(N-1)$ real additions.
4. A number of indexing and addressing operations.

6.1.2 Divide-and-conquer approach to computation of the DFT.

This approach is based on the ***decomposition*** of an ***N-point DFT*** into successively ***smaller DFTs***.

Let us consider the computation of an *N*-point DFT, where *N* can be factored as a product of two integers.

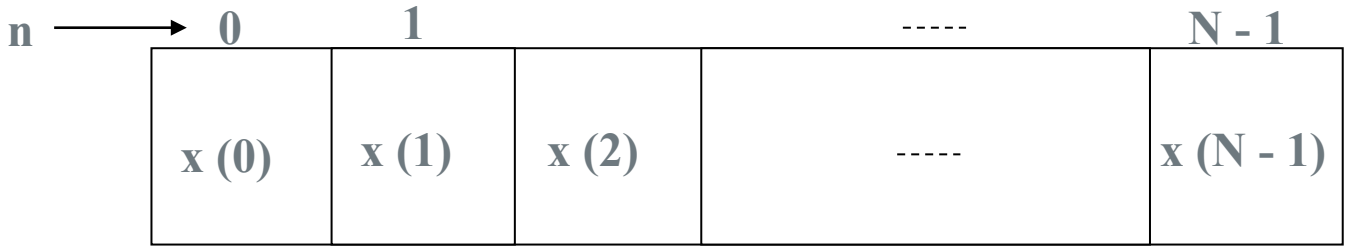
$$N = LM \qquad (6.1.8)$$

The sequence $x(n)$, $0 \le n \le N-1$, can be stored in either in a two-dimensional array indexed by *l* and *m*,

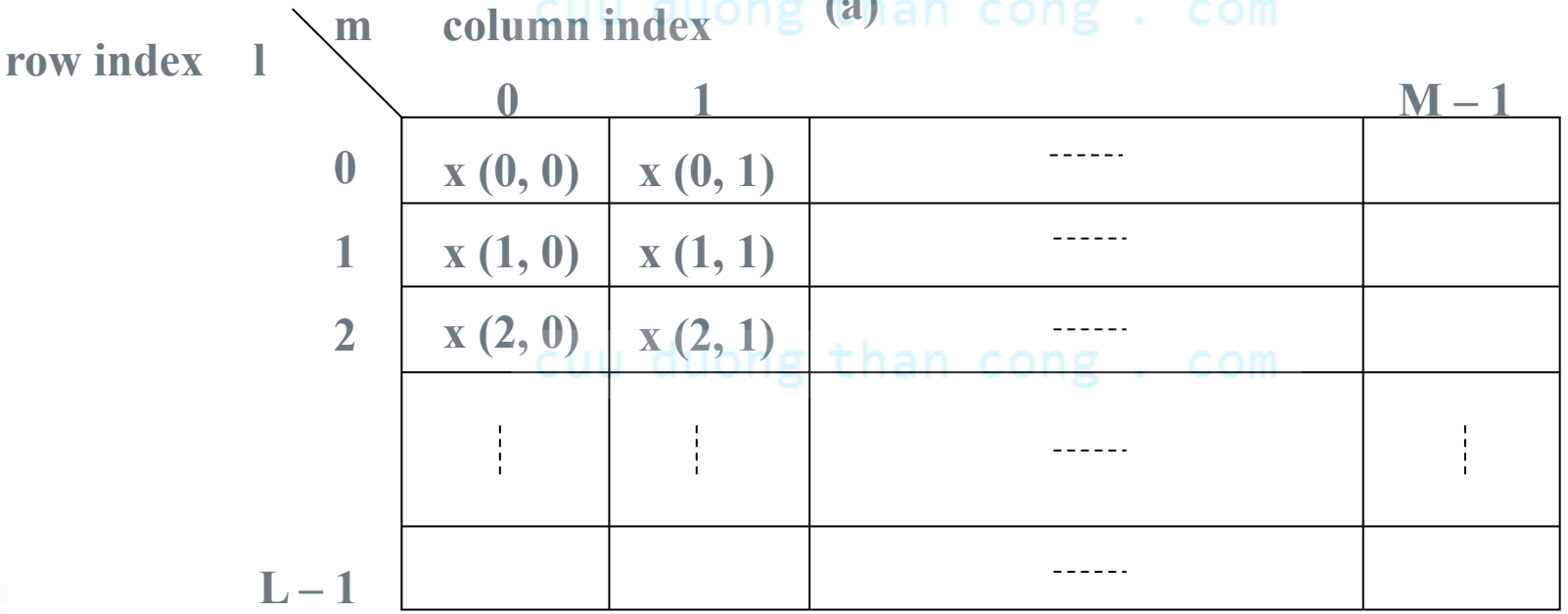
Where $0 \le l \le L-1$ (***row index***) and $0 \le m \le M-1$ (***column index***) as illustrated in Fig 6.1. We select mapping : $n = Ml + m$ (6.1.9) $n = l + mL$ (6.1.10)

6.1.2 Divide-and-conquer approach to computation of the DFT.

Figure 6.1 Two dimensional data array for storing the sequence $x(n),\ 0 \le n \le N-1$.



(a)



(b)

6.1.2 Divide-and-conquer approach to computation of the DFT.

A ***similar arrangement*** can be used to store the computed DFT values.

Where $0 \le p \le L - 1$ and $0 \le p \le M - 1$

If we select the mapping

$$k = Mp + q \qquad (6.1.11)$$

The DFT stored on a ***row-wise basis***.

The mapping

$$k = qL + p \qquad (6.1.12)$$

results in a ***column-wise*** storage of $X(k)$.

6.1.2 Divide-and-conquer approach to computation of the DFT.

Let us adopt a ***column-wise*** mapping for ***x(n)*** and the ***row-wise*** mapping for the ***DFT***, then

$$X(p,q)=\sum_{m=0}^{M-1}\sum_{l=0}^{L-1}x(l,m)W_N^{(Mp+q)(mL+l)} \qquad (6.1.13)$$

With these simplications,

$$X(p,q)=\sum_{l=0}^{L-1}\left\{W_N^{lq}\left[\sum_{m=0}^{M-1}x(l,m)W_M^{mq}\right]\right\}W_L^{lp} \qquad (6.1.15)$$

The computation of DFTs of length *M* and length *L*

6.1.2 Divide-and-conquer approach to computation of the DFT.

Let us subdivide the computation into three steps:

1. *First,* we compute the ***M-point DFTs***

$$F(l,q) \equiv \sum_{m=0}^{M-1} x(l,m) W_M^{mq}, \qquad 0 \le q \le M-1 \qquad (6.1.16)$$

for each the rows $l = 0, 1, \ldots, L-1$

2. *Second*, we compute a new ***rectangular array*** *G(l,q)* defined as

$$G(l,q) = W_N^{lq}(l,q), \qquad 0 \le l \le L-1; \quad 0 \le q \le M-1 \qquad (6.1.17)$$

6.1.2 Divide-and-conquer approach to computation of the DFT.

3. *Finally*, we compute ***L-point DFTs***

$$X(p,q) = \sum_{l=0}^{L-1} G(l,q) W_L^{lp} \qquad (6.1.18)$$

for each column $q = 0, 1, \ldots, M-1$, of the array $G(l, q)$

Let us evaluate the ***computational complexity*** of (6.1.15)

The first step requires ***LM^2 Complex multiplications*** and ***$LM(M-1)$ Complex additions***.

The second step requires ***LM complex multiplications***.

Finally, the third step requires ***LM^2 complex multiplications*** and ***$ML(L-1)$ complex additions***

6.1.2 Divide-and-conquer approach to computation of the DFT.

Therefore, the ***computational complexity*** is :

Complex multiplications :

$$N(M + L + 1)$$

Complex additions:

$$N(M + L - 2) \qquad (6.1.19)$$

Where $N = ML$

Thus, the number of multiplications has been reduced from, N^2 to $N(M + L + 1)$, and the number of additions has been reduced from $N(N - 1)$ to $N(M + L - 2)$.

6.1.2 Divide-and-conquer approach to computation of the DFT.

When N is a highly composite number, N can be factored into a product of prime numbers of the form.

$$N = r_1 r_2 \dots r_v \qquad (6.1.20)$$

then the decomposition above can be repeated $(v - 1)$ more times.

Algorithm 1. (**Algorithm 2.** self-study)

1. Store the signal column-wise.
2. Compute the M-point DFT of each row.
3. Multiply the resulting array by the phase factors W_N^{lq}
4. Compute the L-point DFT of each column.
5. Read the resulting array row-wise.

6.1.3 Radix-2 FFT Algorithm.

The approach is very efficient where N is highly composite, that is, when N can be factored as

$$N = r_1 r_2 r_3 \ldots r_v .$$

Where $\{ r_j \}$ are prime

In the case $r_1 = r_2 = \ldots = r_v \equiv r$, so that $\boldsymbol{N = r^v}$, the DFTs are of ***size r***. The number ***r*** is called the ***radix of the FFT algorithm***.

We described ***radix-2 algorithms***: the computations of the $\boldsymbol{N = 2^v}$ point DFT by the ***divide-and-conquer*** approach specified by (6.1.16) through (6.1.18).

6.1.3 Radix-2 FFT Algorithm.

We select $M = N/2$ and $L = 2$, thus

$$f_1(n) = x(2n)$$

$$f_2(n) = x(2n + 1) \qquad (6.1.23)$$

where $n = 0, 1, \ldots, N/2 + 1$

$f_1(n)$ and ***$f_2(n)$*** are obtained by ***decimating*** $x(n)$ by a factor of 2, and hence the resulting FFT algorithm is called a ***decimation-in-time algorithm***.

6.1.3 Radix-2 FFT Algorithm.

Now the ***N-point DFT*** can be expressed

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$= \sum_{n\ even} x(n) W_N^{kn} + \sum_{n\ odd} x(n) W_N^{kn} \qquad (6.1.24)$$

$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)}$$

But $W_N^2 = W_{N/2}$, can be expressed

6.1.3 Radix-2 FFT Algorithm.

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(2m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km} \qquad (6.1.25)$$

$$= F_1(k) + W_N^k F_2(k), \qquad k = 0, 1, \ldots, N-1$$

where $F_1(k)$ and $F_2(k)$ are the $N/2$-point DFTs of the sequence $f_1(m)$ and $f_2(m)$, *respectively*.

6.1.3 Radix-2 FFT Algorithm.

Since $F_1(k)$ and $F_2(k)$ are periodic ($N/2$), thus

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \qquad (6.1.26)$$

$$X\left(k + \frac{N}{2}\right) = F_1(k) - W_N^k F_2(k), \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \qquad (6.1.27)$$

Hence the computation of $X(k)$ requires :

$N^2/2 + N/2$ complex multiplications.

We may define

$$G_1(k) = F_1(k) \qquad k = 0, 1, \ldots, N/2 - 1$$

$$G_2(k) = W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

6.1.3 Radix-2 FFT Algorithm.

Thus DFT *X(k)* may be expressed as

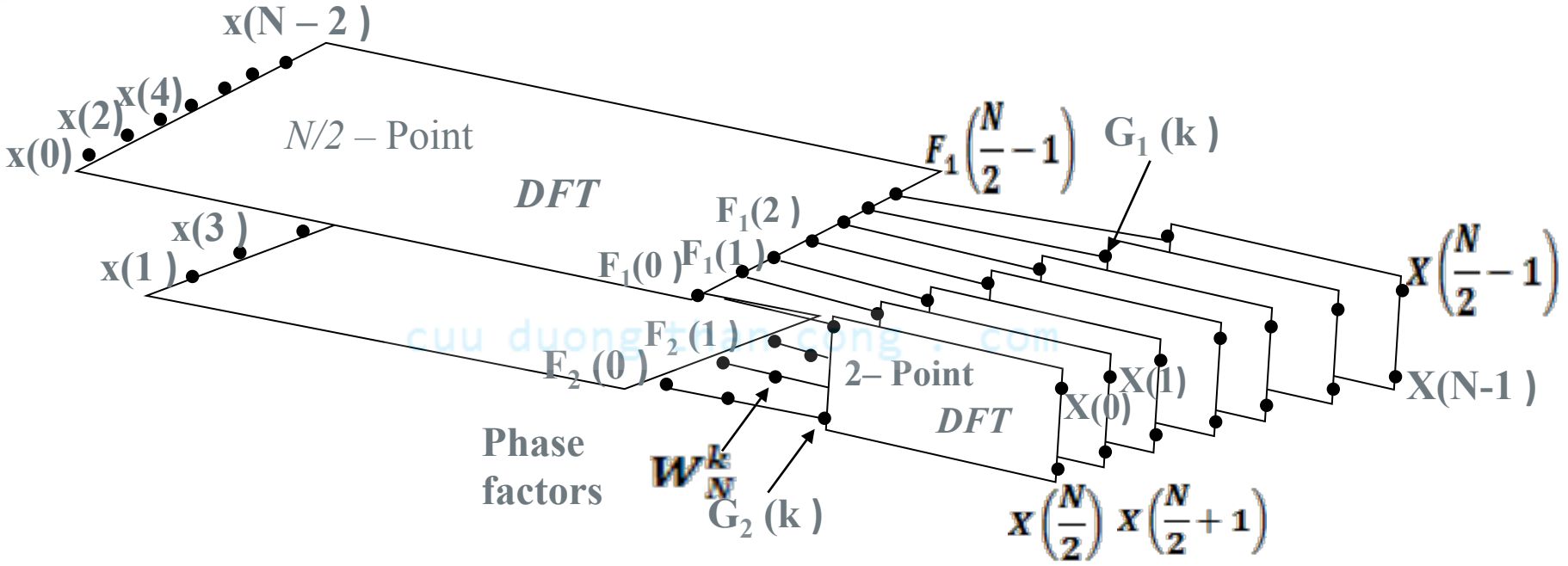
$$X(k) = G_1(k) + G_2(k) \qquad k = 0, 1, \ldots, N/2 - 1$$

$$X\left(k+\frac{N}{2}\right) = G_1(k) - G_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \quad (6.1.28)$$

This computation is illustrated in Fig. 6.4

6.1.3 Radix-2 FFT Algorithm.

Figure 6.4 First step in the decimation-in-time algorithm.



For $N = 2^v$, this decimation can be performed $v = log_2 N$ times.

Thus, the total number of ***complex multiplications*** is reduced to $(N/2)\ log_2 N$.

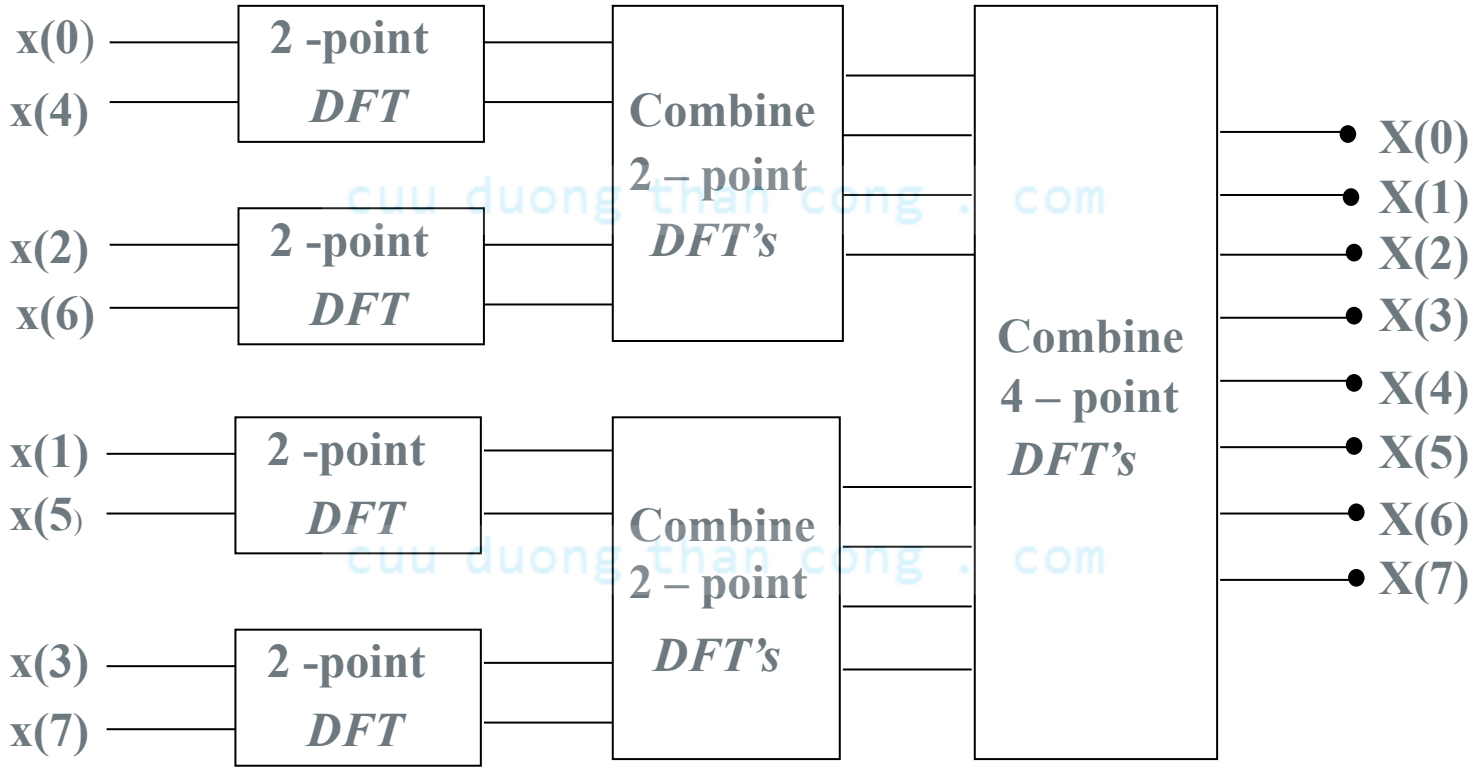
6.1.3 Radix-2 FFT Algorithm.

Table 6.1 Comparison of Computational Complexity for the Direct Computation of the DFT Versus the FFT Algorithm

Number of Points, N	Complex Multiplications in Direct Computation, N^2	Complex Multiplications in FFT Algorithm, $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,53 6	1,024	64,0
512	262,144	2,304	113,8
1,024	1,048,576	5,120	204,8

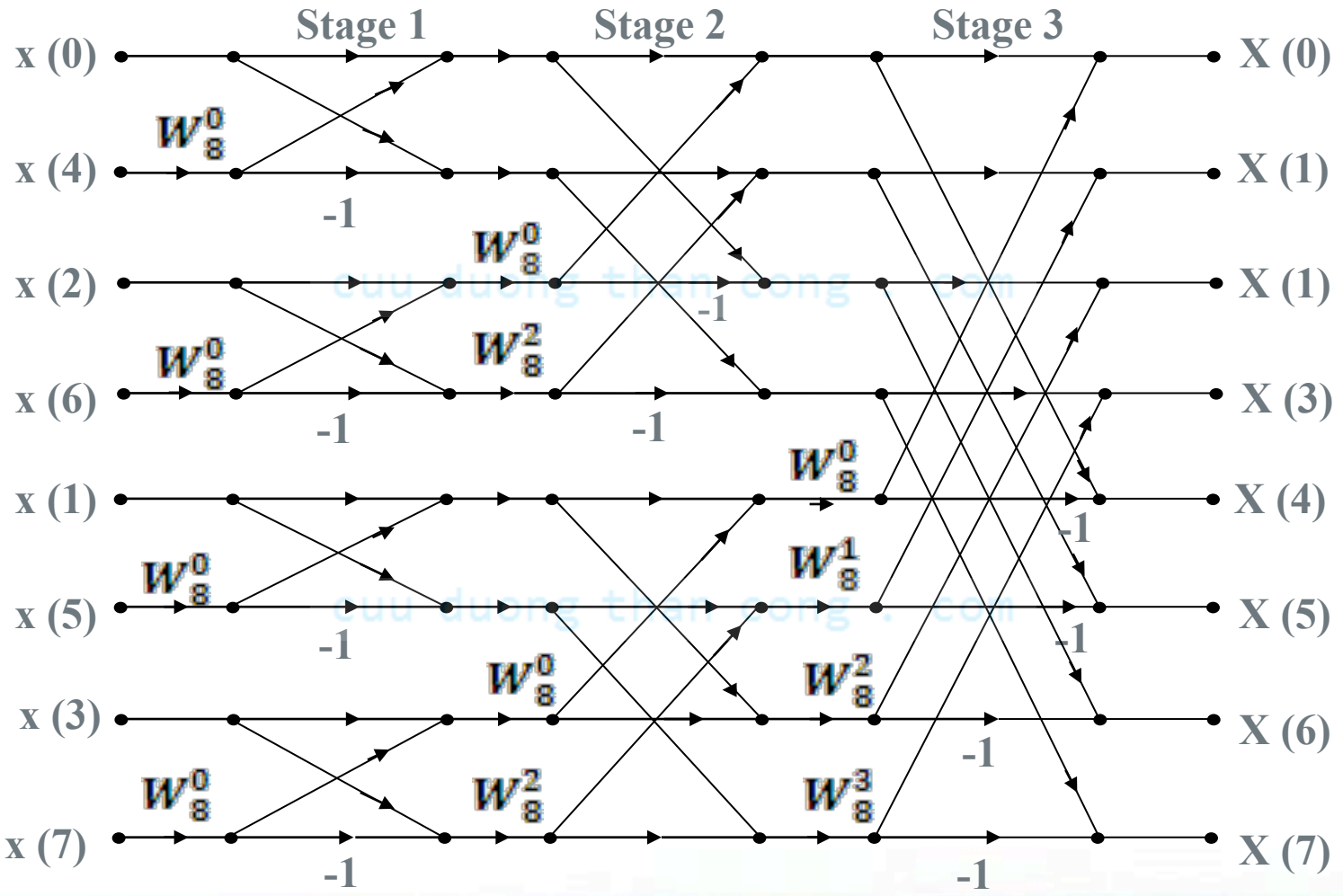
6.1.3 Radix-2 FFT Algorithm.

Figure 6.5 Three stages in the computation of an *N* = *8*-point DFT.



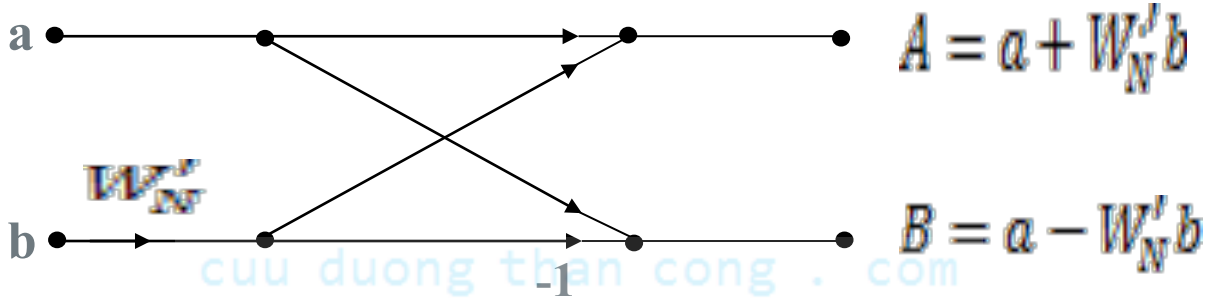
6.1.3 Radix-2 FFT Algorithm.

Figure 6.6 Eight-point decimation-in-time FFT algorithm.



6.1.3 Radix-2 FFT Algorithm.

Figure 6.7 Basic butterfly computation in the decimation-in-time FFT algorithm.



In general, for $N = 2^v$, there are ***N/2 butterfly per stage*** of the computation process and ***$\log_2 N$*** stages.

We can store the result (*A, B*) in the same locations as (*a, b*). We require a fixed amount to store the results (***N complex numbers***) of the computations at each stage.

6.1.3 Radix-2 FFT Algorithm.

Since the same *2N* storage locations are used throughout the computation of the *N*-point DFT we say that the ***computations are done in place***.

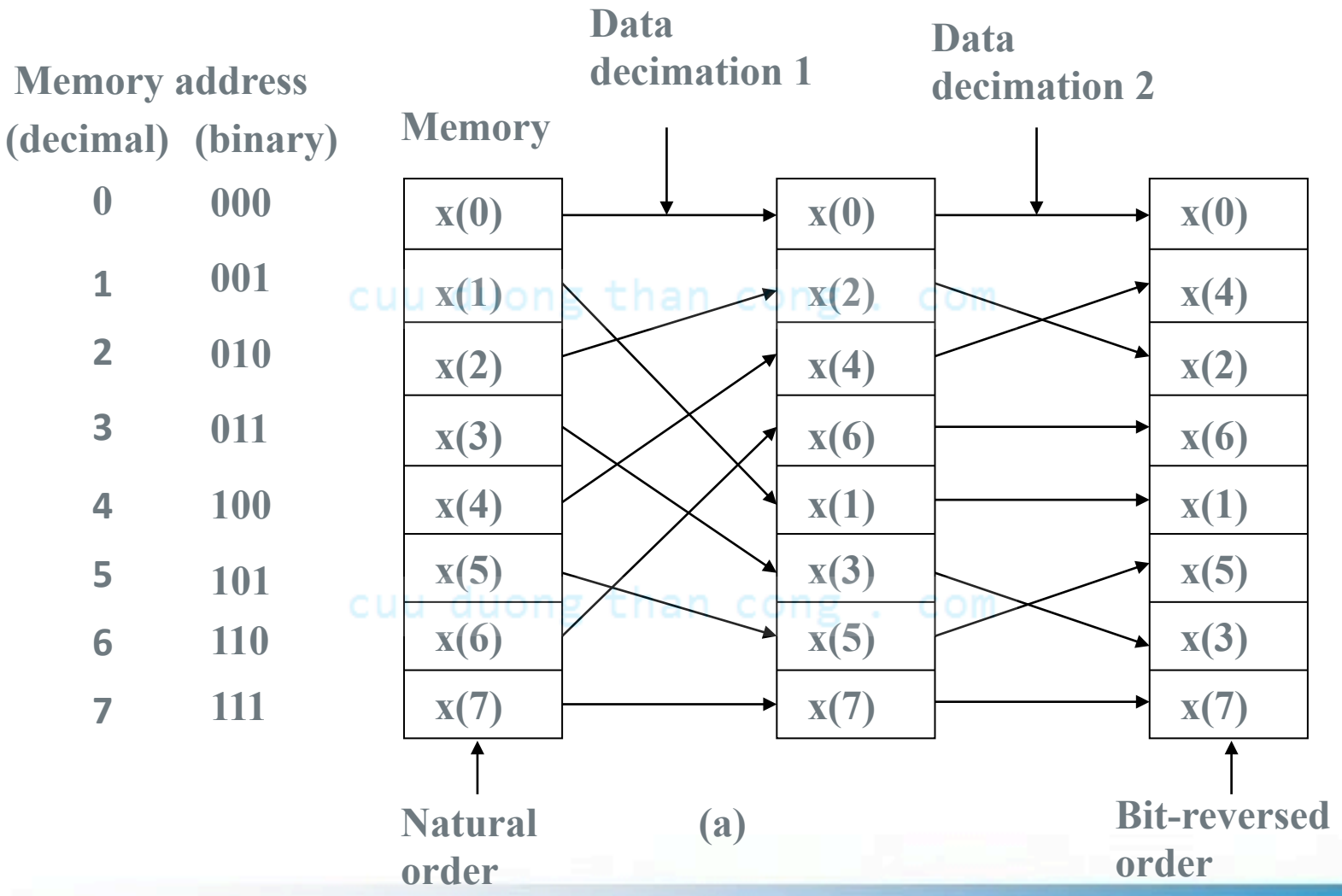
The order of the input data sequence after it is decimated (*v* – *1*) times.

This ***shuffling*** of the input data sequence has a well-defined order as can be ascertained from observing Fig 6.8.

Thus, we say that the data *x(n)* after decimation is stored in ***bit-revered order.***

6.1.3 Radix-2 FFT Algorithm.

Figure 6.8 Shuffling of the data and bit reversal



6.1.3 Radix-2 FFT Algorithm.

$(n_2n_1n_0)$	→	$(n_0n_2n_1)$	→	$(n_0n_1n_2)$
(000)	→	(000)	→	(000)
(001)	→	(100)	→	(100)
(010)	→	(001)	→	(010)
(011)	→	(101)	→	(110)
(100)	→	(010)	→	(001)
(101)	→	(110)	→	(101)
(110)	→	(011)	→	(011)
(111)	→	(111)	→	(111)

(b)

6.1.3 Radix-2 FFT Algorithm.

It is possible ***to arrange the FFT algorithm*** such that the input is ***left in natural order*** and the resulting output DFT will occur ***in bit-reversed order.***

We can impose the restriction that both the input data *x(n)* and the output DFT *X(k)* be ***in natural order***, and derive an FFT algorithm in which the computations are ***not done in place***.

6.1.3 Radix-2 FFT Algorithm.

The ***decimation-in-frequency algorithm*** is obtained by using the *divide-and-conquer approach* with the choice of $M = 2$ and $L = N / 2$

We obtain

$$X(k) = \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_{n=0}^{nk/2} x\left(n + \frac{N}{2}\right) W_N^{kn} \qquad (6.1.33)$$

Since $W_{n=0}^{kN/2} = (-1)^k$

then

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n + \frac{N}{2}\right)\right] W_N^{kn} \qquad (6.1.34)$$

6.1.3 Radix-2 FFT Algorithm.

let us split *X(k)*

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] W_{N/2}^{kn} \qquad (6.1.35)$$

and

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{\left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n\right\} W_{N/2}^{kn} \qquad (6.1.36)$$

$$k = 0, 1, \ldots, \frac{N}{2} - 1$$

6.1.3 Radix-2 FFT Algorithm.

If we define:

$$g_1(n) = x(n) + x\left(n + \frac{N}{2}\right)$$

$$g_2(n) = \left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n \qquad (6.1.37)$$

$$n = 0, 1, 2, \ldots, \frac{N}{2} - 1$$

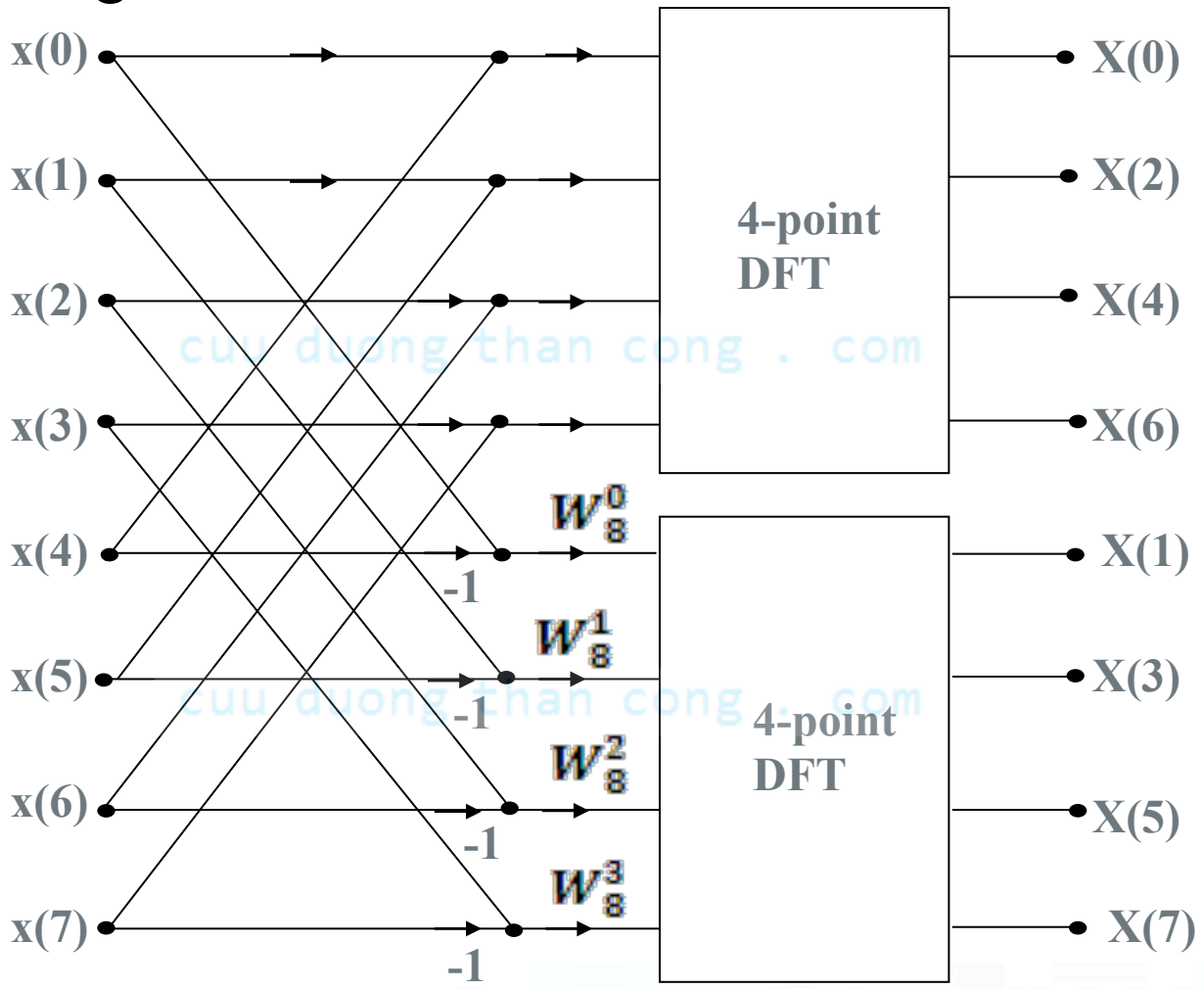
then

$$X(2k) = \sum_{n=0}^{(N/2)-1} g_1(n) W_N^{kn} \qquad (6.1.38)$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} g_2(n) W_N^{kn}$$

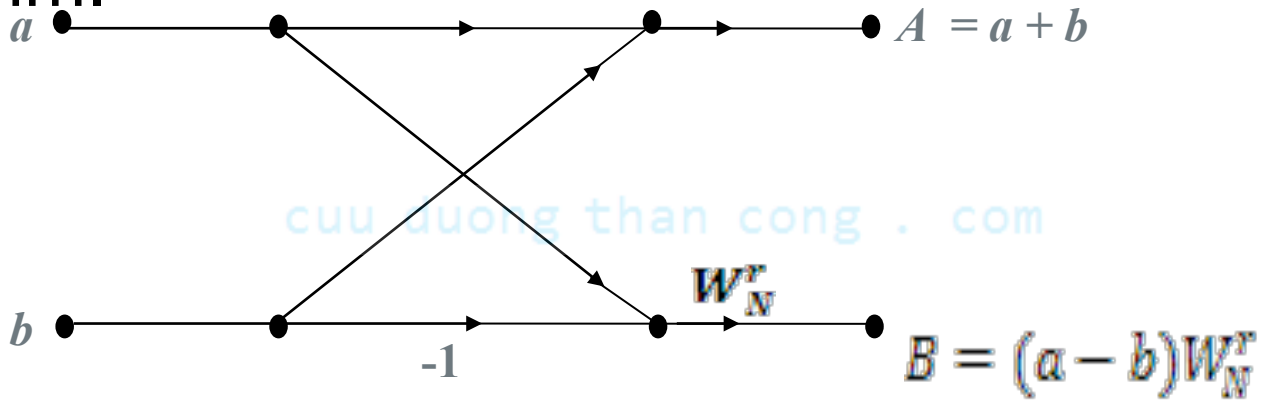
6.1.3 Radix-2 FFT Algorithm.

Figure 6.9 First stage of the decimation-in-frequency FFT algorithm.



6.1.3 Radix-2 FFT Algorithm.

Figure 6.10 Basic butterfly computation in the decimation-in-frequency FFT algorithm.



The entire process involves $v = \log_2 N$ ***stages*** of decimation.

Each stage involves ***N/2 butterflies*** of the type in Fig 6.10.

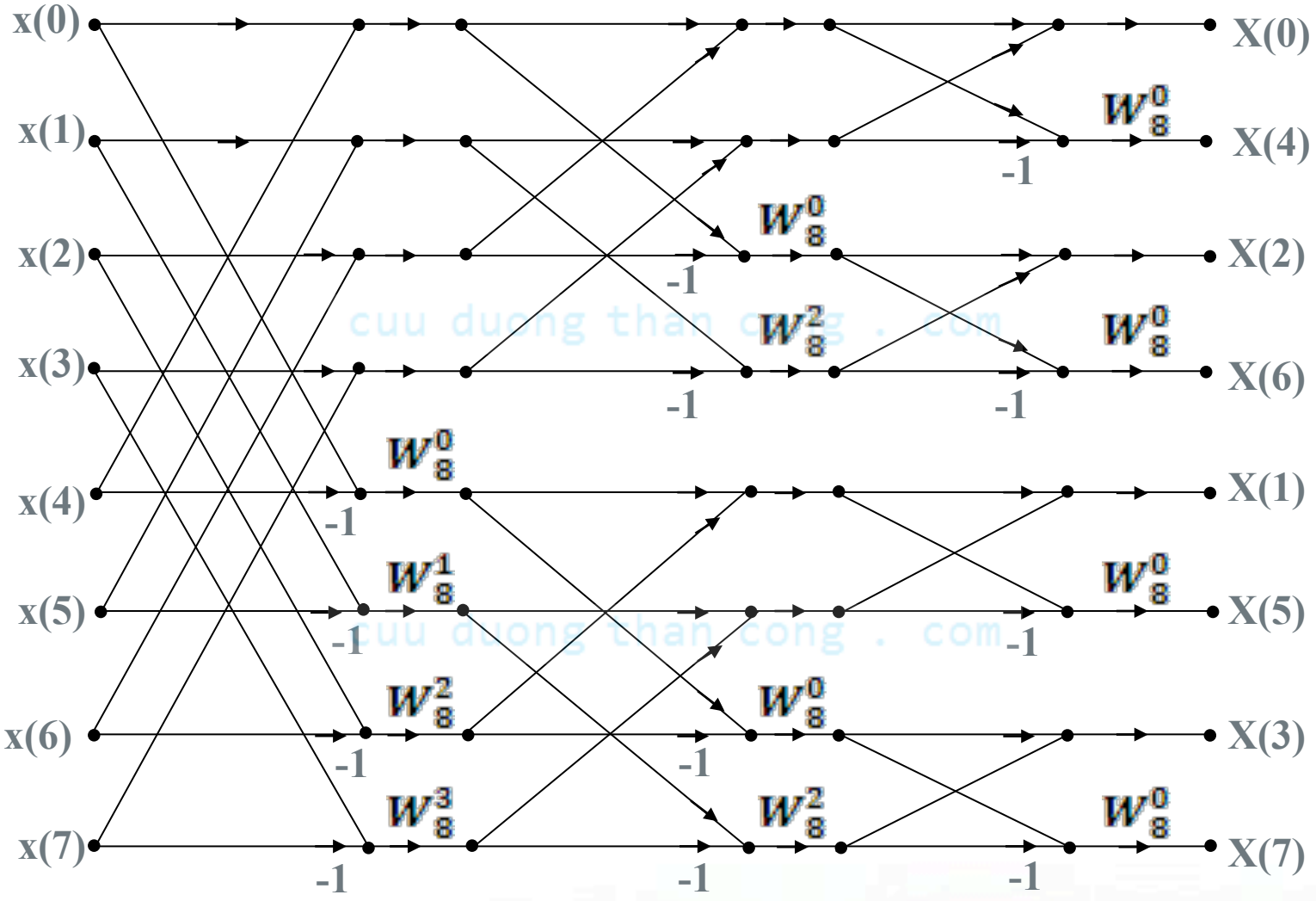
6.1.3 Radix-2 FFT Algorithm.

The computation of the *N*-point DFT via the decimation-in-frequency FFT algorithm, requires ***(N /2) $\log_2 N$*** *complex multiplications* and ***N $\log_2 N$*** *complex additions,* just as in the decimation-in-time algorithm.

From fig.6.11 that the input data *x(n)* occurs ***in natural order***, but the *output DFT* occurs ***in bit-reversed order***.

6.1.4. Radix-4 FFT algorithms

Figure 6.11 N = 8-point decimation-in- frequency FFT algorithm.



6.1.4. Radix-4 FFT algorithms

When the number data point N in the DFT is *power of 4*, we can employ a ***radix-4 FFT algorithm***.

Selecting $L = 4$ and $M = N/4$,

we have

$n = 4m + l;$ $\quad m, q = 0, 1, \ldots, N/4;$

$k = (N/4)\, p + q;$ $\quad l, p = 0, 1, 2, 3;$

Thus, we split the *N*-point input sequence into four subsequences :

$x(4n), x(4n+1), x(4n+2), x(4n+3)$

$n = 0, 1, \ldots, N/4 - 1.$

6.1.4. Radix-4 FFT algorithms

We obtain

$$X(p,q)=\sum_{l=0}^{3}\left[W_N^{lq}F(l,q)\right]W_4^{lp},\qquad p=0,1,2,3 \qquad (6.1.39)$$

$$F(l,q)=\sum_{m=0}^{(N/4)-1}x(l,m)\,W_{N/4}^{mq} \qquad (6.1.40)$$

$$l=0,1,2,3,$$

$$q=0,1,2,\ldots,\frac{N}{4}-1$$

and $x(l, m) = x(4m + l)$ (6.1.41)

$$X(p,q)=X\left(\frac{N}{4}p+q\right) \qquad (6.1.42)$$

6.1.4. Radix-4 FFT algorithms

from (6.1.39) we have a ***radix-4 decimation-in-time butterfly***.

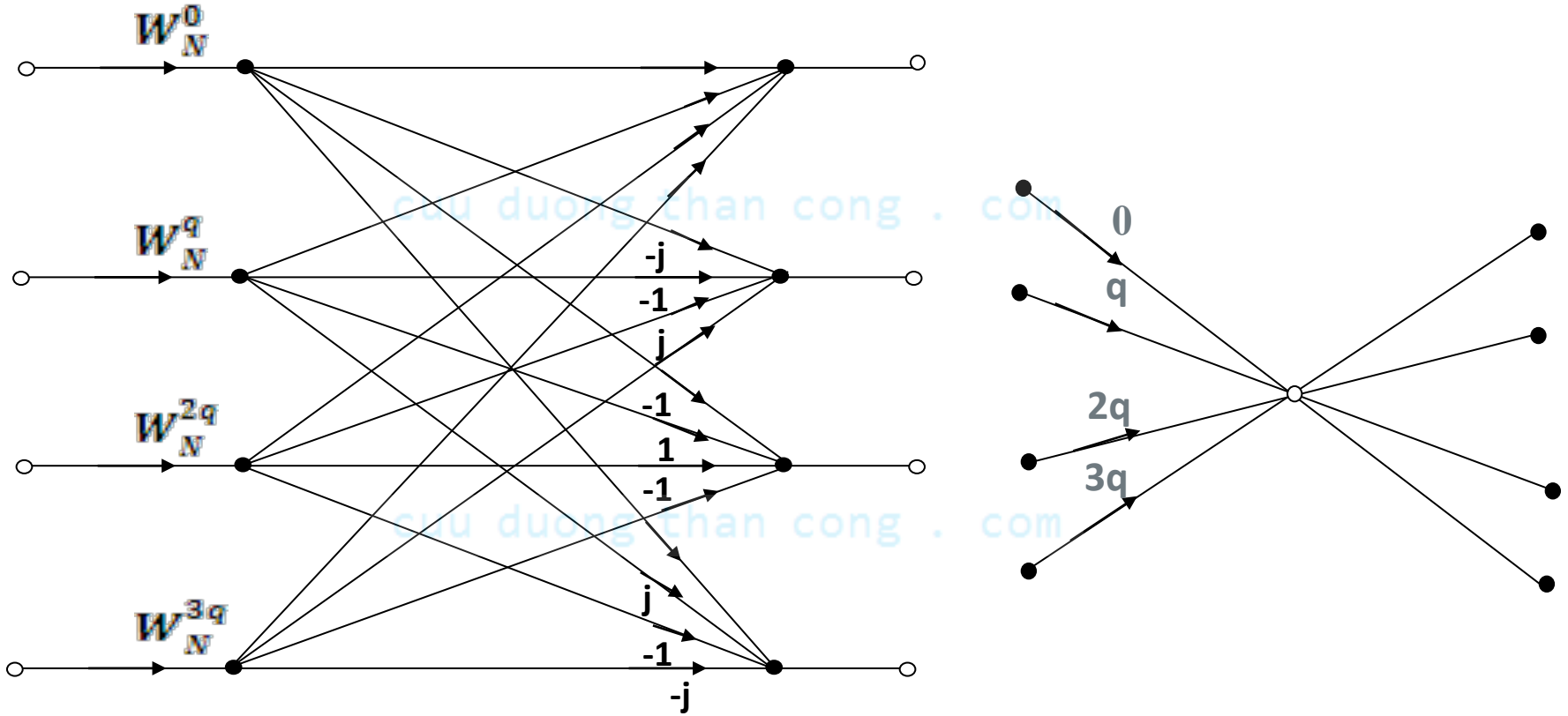
$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & 1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix} \qquad (6.1.43)$$

cuu duong than cong . com

Since $W_N^0 = 1$ each butterfly involves ***three complex multiplications***, and ***12 complex additions***.

6.1.4. Radix-4 FFT algorithms

Figure 6.12 Basic butterfly computation in a radix-4 FFT algorithm.



6.1.4. Radix-4 FFT algorithms

This *decimation-in-time procedure* can be repeated recursively ***v times***.

The resulting FFT algorithm consists of ***v stages***, where each stage contains ***N/4 butterflies***.

The computations burden for the algorithm is $3vN/4 = \mathbf{(3N/8)\ log_2 N}$ complex additions.

The number of multiplications is *reduced* by 25%, but the number of additions has *increased* by 50% from $N\ log_2 N$ to $\mathbf{(3N/2) log_2 N}$.

6.1.4. Radix-4 FFT algorithms

It is possible to ***reduce*** the number of a *additions per butterfly* from ***12*** to ***8*** by product of two matrices as follows:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix} \quad (6.1.44)$$

Now, each matrix multiplication involves four additions for a total of eight additions.

6.1.4. Radix-4 FFT algorithms

Thus the total number of complex additions is reduced to $N \log_2 N$.

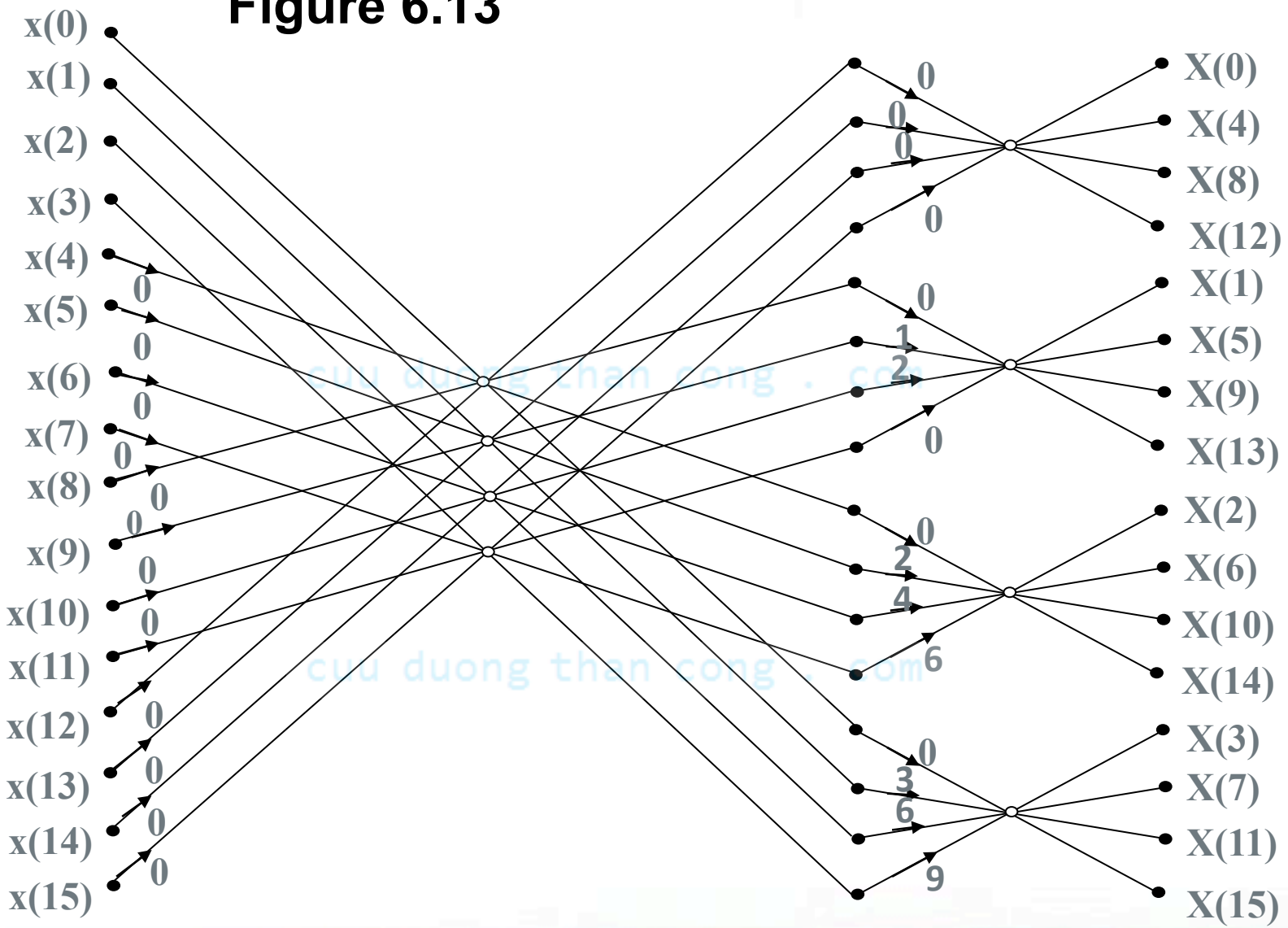
An illustration of a ***radix-4 decimation-in-time FFT algorithm*** is shown in Fig 6.13 for $N = 16$.

Figure 6.13 Sixteen-point radix-4 decimation-in-time algorithm with input in normal order and output in digit-reversed order.

The integer multipliers shown on the graph represent the exponents on W_{16}

6.1.4. Radix-4 FFT algorithms

Figure 6.13



6.1.4. Radix-4 FFT algorithms

A *radix-4 decimation-in-frequency* FFT algorithm can be obtained by selecting

$L = N / 4,\ M = 4,$

$l,p = 0, 1, \ldots, N / 4 - 1\ ;\ m, q = 0, 1, 2, 3;$

$n = (N / 4)\, m + l\ ;$ and $k = 4p + q$

From (6.1.15), we express as

$$X(p,q) = \sum_{i=0}^{(N/4)-1} G(l,q) W_{N/4}^{lp} \qquad (6.1.45)$$

6.1.4. Radix-4 FFT algorithms

where

$$G(l,q) = W_N^{lq} F(l,q) \qquad q = 0,1,2,3 \quad l = 0,1\ldots,\frac{N}{4}-1 \qquad (6.1.46)$$

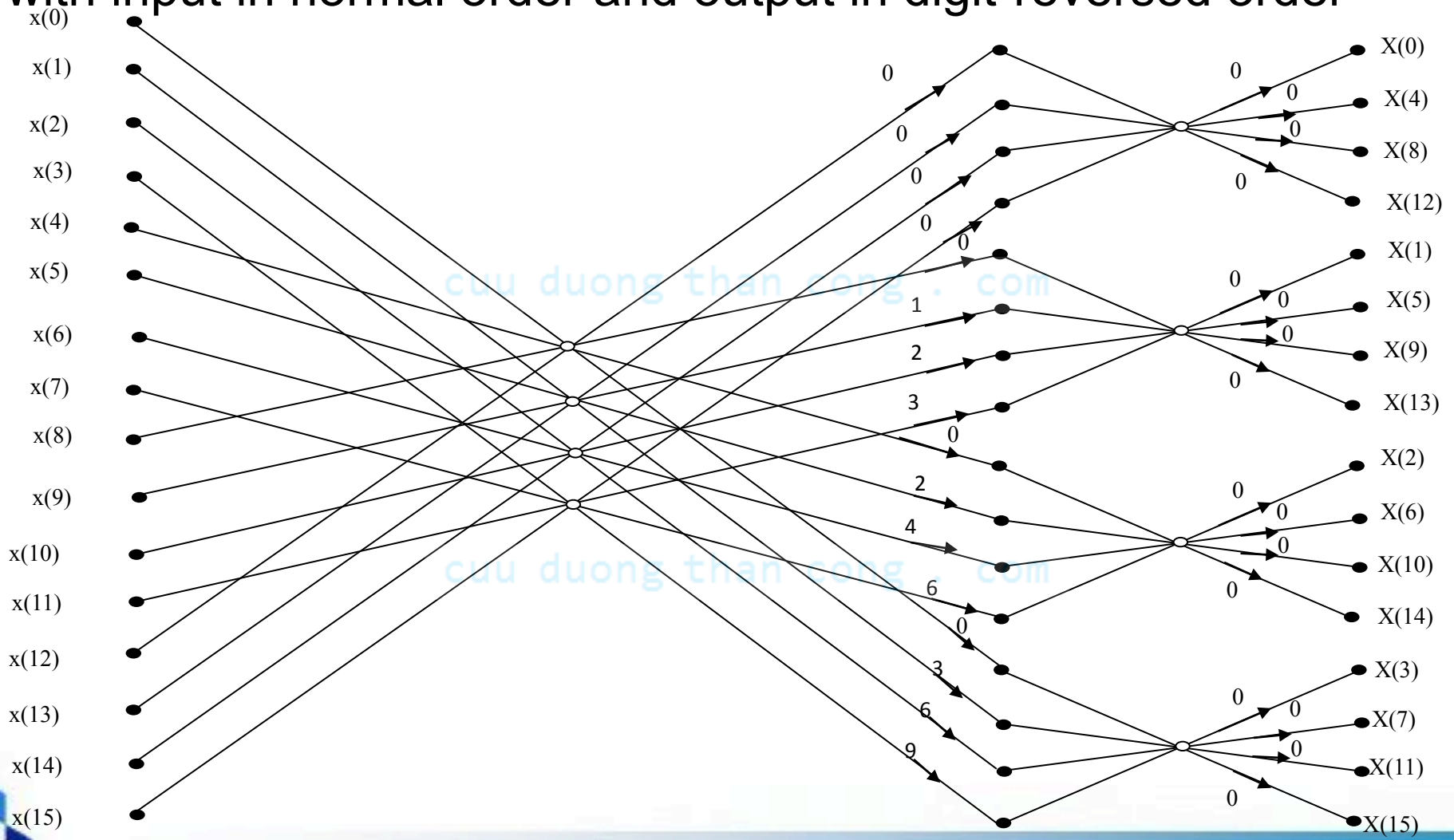
and

$$F(l,q) = \sum_{m=0}^{3} x(l,m) W_4^{mq} \qquad q = 0,1,2,3 \quad l = 0,1\ldots,\frac{N}{4}-1 \qquad (6.1.47)$$

The computation in (6.1.46) and (6.1.47) define the basic *radix-4 butterfly* for the decimation-in-frequency algorithm.

6.1.4. Radix-4 FFT algorithms

Figure 6.14 Sixteen-point radix-4 decimation-in-frequency algorithm with input in normal order and output in digit-reversed order



6.1.4. Radix-4 FFT algorithms

It has exactly the same computational complexity as the decimation-in-time radix-4 FFT algorithm.

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{kn}=\sum_{n=0}^{N/4-1}\left[x(n)+(-j)^k x\left(n+\frac{N}{4}\right)+(-1)^k x\left(n+\frac{N}{4}\right)+(j)^k x\left(n+\frac{3N}{4}\right)\right]W_N^{nk} \quad (6.1.50)$$

In this algorithm, the input sequence is in normal order while the output DFT is shuffled.

6.1.4. Radix-4 FFT algorithms

$$x(4k) = \sum_{n=0}^{N/4-1} \left[x(n) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right) \right] W_N^0 W_{N/4}^{nk} \qquad (6.1.51)$$

$$X(4k+1) = \sum_{n=0}^{N/4-1} \left[x(n) - jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) + jx\left(n+\frac{3N}{4}\right) \right] W_N^n W_{N/4}^{kn} \qquad (6.1.52)$$

$$X(4k+2) = \sum_{n=0}^{N/4-1} \left[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right) \right] W_N^{2n} W_{N/4}^{kn} \qquad (6.1.53)$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \left[x(n) + jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - jx\left(n+\frac{3N}{4}\right) \right] W_N^{3n} W_{N/4}^{kn} \qquad (6.1.54)$$

6.1.5 Split-Radix FFT Algorithm

The ***split-radix FFT*** (***SRFFT***) ***algorithms*** use both a *radix-2* and *radix-4 decomposition* in the same FFT algorithm.

The *even-numbered samples* of the *N*-point DFT

$$X(2k) = \sum_{n=0}^{N/2-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] W_{N/2}^{nk}, \quad K = 0, 1, \ldots, \frac{N}{2} - 1 \qquad (6.1.55)$$

6.1.5 Split-Radix FFT Algorithm.

The odd-numbered samples of the N-point DFT :

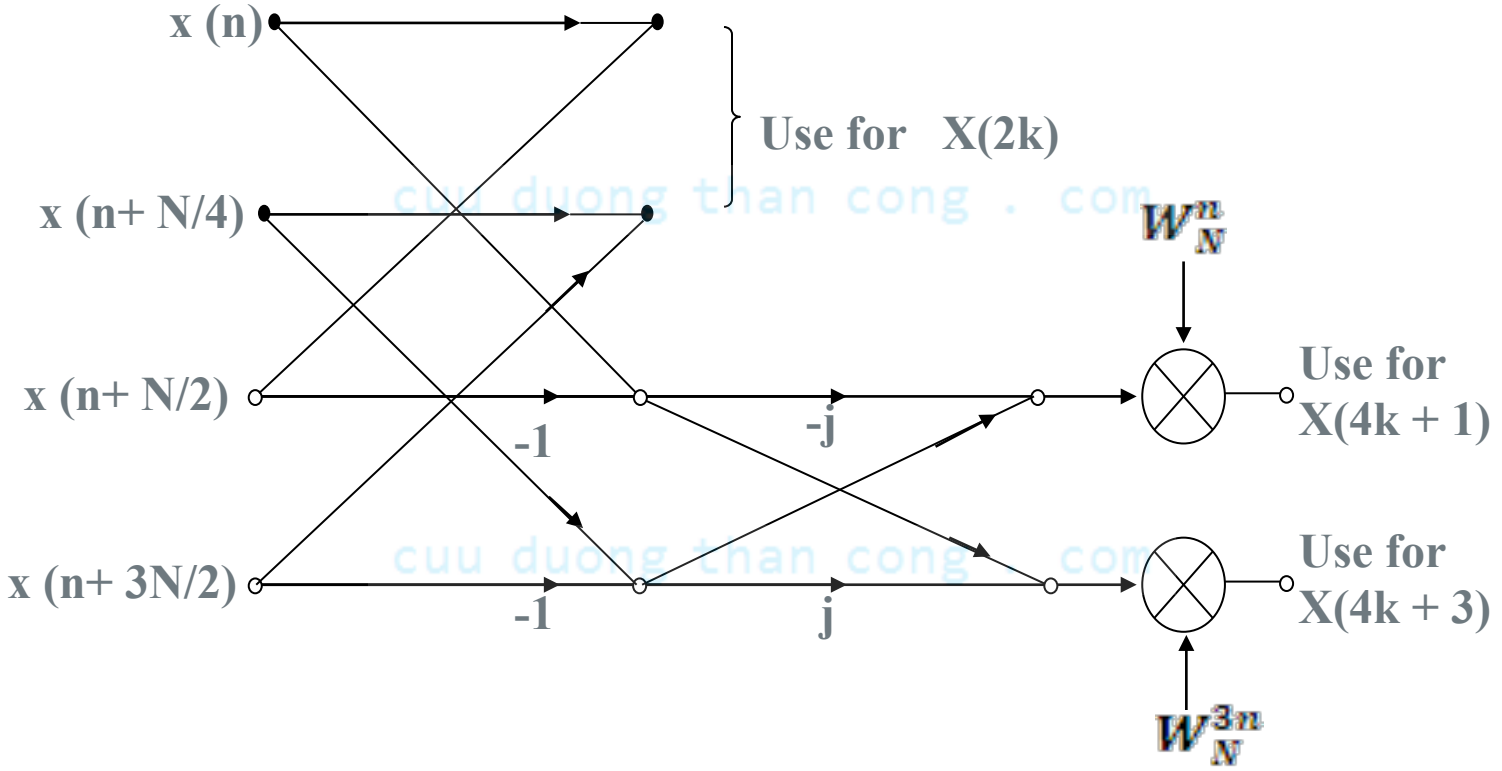
$$X(4k+1) = \sum_{n=0}^{N/4-1} \{[x(n) - x(n+N/2)] - j[x(n+N/4) - x(n+3N/4)]\} W_N^n W_{N/4}^{kn} \quad (6.1.56)$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \{[x(n) - x(n+N/2)] + j[x(n+N/4) - x(n+3N/4)]\} W_N^{3n} W_{N/4}^{kn} \quad (6.1.57)$$

The *N*-point DFT is decomposed into one *N/2*-point DFT without additional twiddle factors and two *N/4*-point DFTs with twiddle factors

6.1.5 Split-Radix FFT Algorithm

Figure 6.16 Butterfly for SRFFT algorithm



6.1.5 Split-Radix FFT Algorithm.

Table 6.2 Number of Nontrivial Real multiplication and addition to compute an N-point complex DFT.

	Real multiplications				Real additions			
N	Radix 2	Radix 4	Radix 8	Split Radix	Radix 2	Radix 4	Radix 8	Split Radix
16	24	20		20	152	148		148
32	88			68	408			388
64	264	280	204	196	1,032	976	972	964
128	712			516	2,504			2,308
256	1,800	1,392		1,284	5,896	5,488		5,380
512	4,360		3,204	3,076	13,566		12,420	12,292
1,024	10,248	7,856		7,172	30,728	28,336		27,652

Source: Extracted from Duhamel (1986)

6.1.6 Implementation of FFT Algorithms.

The butterfly computations is repeated many times (***(N log$_2$ N) /2 time***) for the computation of an *N*-point DFT.

The ***phase factors*** are computed once and stored in the table.

The number of memory locations required is ***2N*** since the numbers are *complex*.

We can instead double the memory to ***4N***, thus simplifying the *indexing* and *control operations* in the FFT algorithms.

6.1.6 Implementation of FFT Algorithms.

We can convert to an ***FFT algorithm*** for computing the ***IDFT*** by changing the ***sign*** on all the phase factors and dividing the final output of the algorithm by ***N***.

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad (6.1.63)$$

For DSP processors, radix-2 or radix-4 decimation-in-frequency FFT algorithms, are preferable in terms of speed and accuracy.

6.2 Applications of FFT Algorithms

The FFT algorithms find application in a variety of areas, including ***linear filtering***, ***correlation***, and ***spectrum analysis***.

We only consider the use of the FFT algorithms in *linear filtering* and in the computation of the *crosscorrelation* of two sequences.

6.2.1 Efficient computation of the DFT of two real sequences.

Suppose that: $x_1(n)$ and $x_2(n)$ are two real-valued sequences of length N, and let $x(n)$ be a complex-valued sequence defined as

$$x(n) = x_1(n) + jx_2(n) \qquad 0 \le n \le N-1 \qquad (6.2.1)$$

The DFT operations is linear and hence:

$$X(k) = X_1(k) + jX_2(k) \qquad (6.2.2)$$

$$x_1(n) = \frac{x(n) + x^*(n)}{2} \qquad (6.2.3)$$

$$x_2(n) = \frac{x(n) + x^*(n)}{2j} \qquad (6.2.4)$$

6.2.1 Efficient computation of the DFT of two real sequences.

Therefore,

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)] \qquad (6.2.7)$$

$$X_2(k) = \frac{1}{j2}[X(k) - X^*(N-k)] \qquad (6.2.8)$$

Thus, by performing a ***single DFT*** on the complex-valued sequence *x(n)*, we have obtained the DFT of the ***two real sequences*** with only a *small amount* of ***additional computation***.

6.2.2 Efficient computation of the DFT of a 2*N*-Point Real Sequences

Suppose that *g(n)* is a real-valued sequence of *2N* point, First we define

$$x_1(n) = g(2n) \qquad (6.2.9)$$

$$x_2(n) = g(2n + 1)$$

Let *x(n)* be the *N*-point complex-valued sequence

$$x(n) = x_1(n) + jx_2(n) \qquad (6.2.10)$$

We have

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)]$$

$$X_2(k) = \frac{1}{2j}[X(k) - X^*(N-k)] \qquad (6.2.11)$$

6.2.2 Efficient computation of the DFT of a 2*N*- point two real sequences

Finally

$$G(k) = \sum_{n=0}^{N-1} x_1(n) W_N^{nk} + W_{2N}^k \sum_{n=0}^{N-1} x_2(n) W_N^{nk}$$

Consequently

$$G(k) = X_1(k) + W_2^k N X_2(k) \qquad (6.2.12)$$

$$G(k+n) = X_1(k) - W_2^k N X_2(k) \quad k = 0, 1, \ldots, N-1$$

6.2.3 Use of the FFT Algorithm in Linear Filtering and Correlation

Let $h(n)$, $0 \le n \le M-1$, be the *unit sample response* of the FIR filter.

$x(n)$ denotes the input data sequence

The block size of the FFT algorithm is N

where $N = L + M - 1$

L – the number of new data samples being processed by the filter.

M – any giver value so that N is a *power of 2*.

The N-point DFT of $h(n)$, which is padded by $L-1$ zero, is denoted as $H(k)$, and use the *decimation-in-frequency FFT algorithm* to compute $H(k)$

6.2.3 Use of the FFT algorithm in linear filtering and correlation.

In the overlap-save method, the *first M – 1* data point of each data block are the *last M – 1* data points of the *previous* data block.

The *N*-point DFT of ***each data block*** is performed by the FFT algorithm.

Since this is exactly the order of *H(k),* we can multiply the DFT of the data, say $X_m(k)$, with *H(k)* and thus the result

$$Y_m(k) = H(k)\, X_m(k)$$

is also in bit-reversed order.

6.2.3 Use of the FFT algorithm in linear filtering and correlation

The ***inverse DFT*** can be computed by use an FFT algorithm that takes the *input in bit-reversed* and produces an *output in normal order*.

The computational burden is ***$N\log_2 N$ complex multiplications*** and ***$2N\log_2 N$ additions***.

We have ***$(N \log_2 2N)/L$ complex multiplications*** per ***output data point*** and approximately ***$(2N\log_2 2N) / L$ additions*** per ***output data point***.

6.2.3 Use of the FFT algorithm in linear filtering and correlation

It is interesting to ***compare*** the ***efficiency of the FFT algorithm*** with the ***direct form realization*** of the FIR filter.

suppose that $M = 128 = 2^7$ and $N = 2^v$

The number of complex multiplications per output for an FFT size of $N = 2^v$ is

$$c(v) = \frac{N\log_2 2N}{L} = \frac{2^v(v+1)}{N-M+1}$$

$$\approx \frac{2^v(v+1)}{2^v - 2^7}$$

6.2.3 Use of the FFT algorithm in linear filtering and correlation

Table 6.3 Computational complexity

Size of FFT $v = \log_2 N$	c (v) number of complex multiplication per output point
9	13.3
10	12.6
11	12.8
12	13.4
14	15.1

6.3 A Linear Filtering Approach To Computation of The DFT.

The ***FFT algorithm*** takes *N points of input data* and produces an *output sequence of N points* corresponding to the DFT of the input data.

The ***radix-2 FFT algorithm*** performs the computation of the DFT in $(N / 2)\ \log_2 N$ ***multiplications*** and $N \log_2 N$ ***additions*** for ***N-point sequence***.

6.3 A linear filtering approach to computation of the DFT.

When the desired number of values of the DFT is ***less than $\log_2 N$***, a ***direct computation*** of the desired values is ***more efficient***.

There are two algorithms:

The ***Goertzel Algorithm*** is particularly attractive where the DFT is to be computed at a relatively small number M of values, where $M \leq \log_2 N$.

The ***clip-z transform algorithm***.

Problems: 6.8, 6.11, 6.12, 6.13, 6.20, 6.27, 6.28