Chapter2　　快速傅立葉轉換數學演算法之分析與研究

2-0　簡介

快速傅立葉轉換（FFT）之 VLSI 技術發展相當進步，而硬體的實現（implement）都是根據最基本的數學而來，在本章將對快速傅立葉轉換的數學演算法做詳細的介紹，並對其不同演算法做特性上的分析與比較，以下各節會介紹的內容有：

1. ***以二點為基底之快速傅立葉轉換（Radix-2 FFT）***
2. ***以四點為基底之快速傅立葉轉換（Radix-4 FFT）***
3. ***以 2^2 為基底之快速傅立葉轉換（Radix-2^2 FFT）***
4. ***以 2^3 為基底之快速傅立葉轉換（Radix-2^3 FFT）***
5. ***混合基數快速傅立葉轉換演算法（Mixed-radix FFT）***

2-1　以二點為基底的快速傅立葉轉換（Radix-2 FFT）

快速傅立葉轉換的運算是根據離散傅立葉轉換（DFT）的數學而來，其利用離散傅立葉轉換的複數乘法在複數平面上的對稱性質，公具有對稱性質的多個乘法合併成一項，因此可以大大地減小數學運算量，在不變更原數學模型架構下，可以獲得較有效率的運算，此運算方法是在 1965 年由 Cooly and Tukey 所提出的，此即為現在著名的快速傅立葉轉換(FFT)，其數學模型大致上可分為二種不同的架構，一種為時間點分組架構（decimation in time），另一種為頻率點分組架構（decimation in frequency）。由系統可以知道規律性（regularity）和簡易性（simplicity）在 VLSI 的實作上是很重要的因數，其可以決定時脈速度（clock speed），系統設計時間（design time），及其它重要的特性，以下將推導 DIT 及 DIF 的數學式。

2-1.1　DIT 架構（decimation in time）

離散傅立葉（DFT）的數學模型為：

$$X[k]=\sum_{n=0}^{N-1}x[n]w_N^{kn} \qquad k=0.1\cdots\cdots.N-1 \qquad (2.1)$$

（其中 w_N=exp(-j2π/N)，X[k]，x[n]為複數）

將 x[n]分為奇數點跟偶數點二個不同的組合團體，即

$$X[k]=\sum_{n(even)} x[n]w_N^{kn}+\sum_{n(odd)} x[n]w_N^{Kn}\ \text{X[k]=} \qquad (2.2)$$

令偶數組合團體的 n=2r，奇數組合團體的 n=2r+1，因此

$$X[k]=\sum_{r=0}^{(N/2)-1} x[2r]w_N^{2rk}+\sum_{r=0}^{(N/2)-1} x[2r+1]w_N^{(2r+1)k} \qquad (2.3)$$

$$=\sum_{r=0}^{(N/2)-1} x[2r]w_N^{2rk}+w_N^k\sum_{r=0}^{(N/2)-1} x[2r+1]w_N^{2rk}$$

$$=\sum_{r=0}^{(N/2)-1} x[2r]w_{N/2}^{rk}+w_N^k\sum_{r=0}^{(N/2)-1} x[2r+1]w_{N/2}^{rk}$$

$$=G[k]+w_N^k H[k] \qquad (2.4)$$

其中 $G[k]=\sum_{r=0}^{(N/2)-1} x[2r]w_{N/2}^{rk}$ （2.5）

$$H[k]=\sum_{r=0}^{(N/2)-1} x[2r+1]w_{N/2}^{rk} \qquad (2.6)$$

以 N=8 為例子，其圖形為

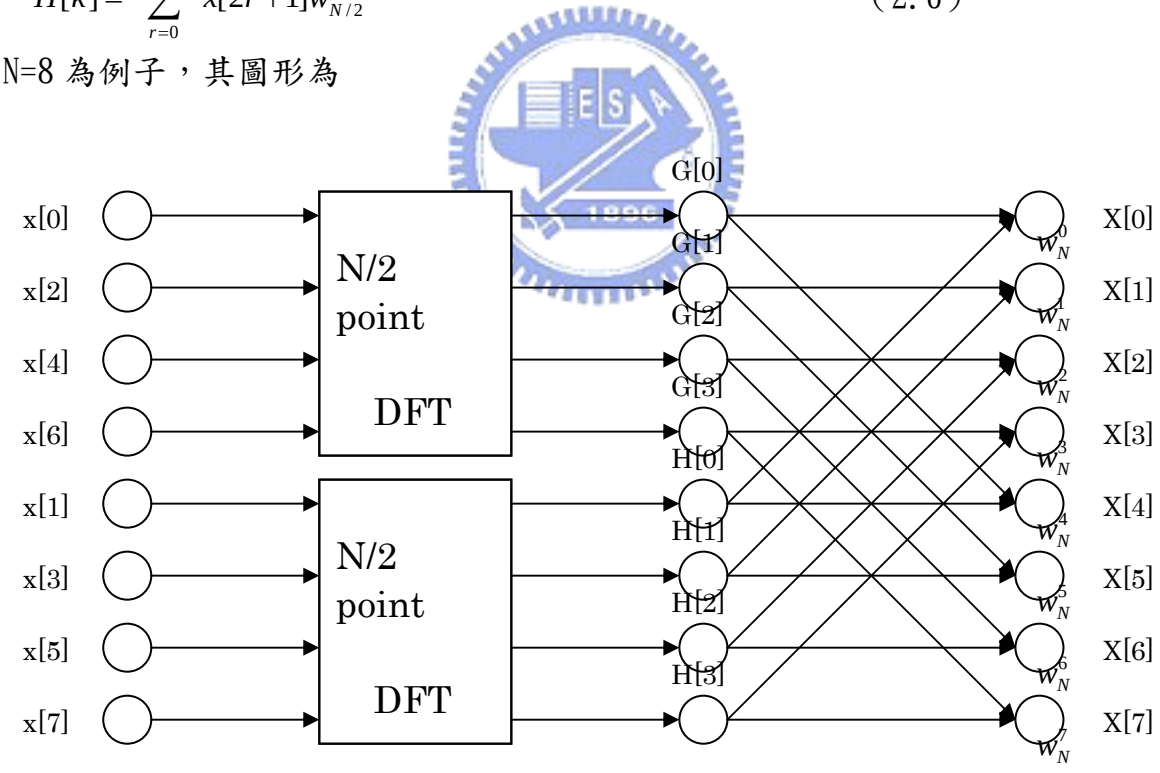

Fig 2-1.1　Simple DIT 8-point FFT

若將（N/2）點的 G[k]及（N/2）點的 H[k]以相同的方式來作偶數奇數分組動作，則可以得到類似的結果，其說明如後：

$$G[k]=\sum_{l=0}^{(N/4)-1} g[2l]w_{N/2}^{2lk}+\sum_{l=0}^{(N/4)-1} g[2l+1]w_{N/2}^{(2l+1)k}$$

$$=\sum_{l=0}^{(N/4)-1} g[2l]w_{N/4}^{lk}+w_N^{2k}\sum_{l=0}^{(N/4)-1} g[2l+1]w_{N/4}^{lk} \qquad (2.7)$$

$$H[k]=\sum_{l=0}^{(N/4)-1} h[2l]w_{N/4}^{lk}+w_N^{2k}\sum_{l=0}^{(N/4)-1} h[2l+1]w_{N/4}^{lk} \qquad (2.8)$$

因此 N/2 點的 G[k]，及 H[k]是由 N/4 點的 g，h 經過 DFT 而來的，其分解過後的訊號如 Fig2-1.2 所示：

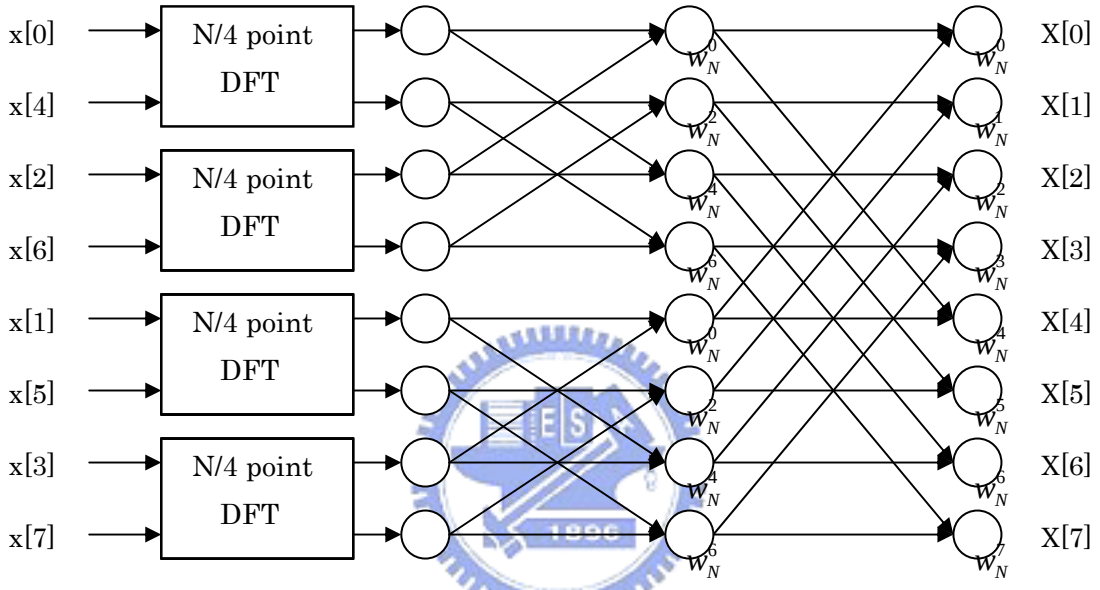

Fig2-1.2 Simple DIT 8-point FFT with radix-2

利用此原理繼續分解，則可得到如 Fig2-1.4 所示的訊號流圖形，由於其是由一種狀態類似蝴蝶的數學運算基本架構而來，因此這種架構即成為有名的時間點分組之蝴蝶圖（DIT butterfly）。其基本單位元件為：

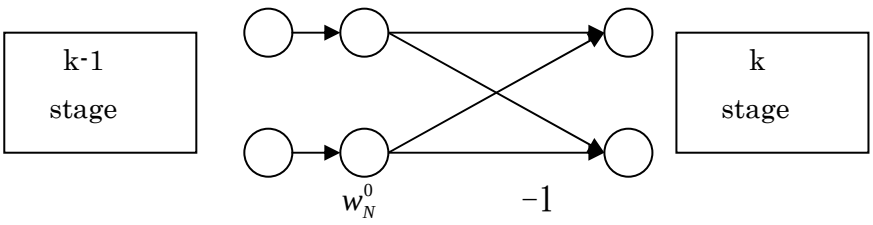

Fig2-1.3 Butterfly 基本單位元件圖

因為butterfly二個轉動因數乘法（twiddle factor multiplication）在相位上相差180^0，乘法只有正負號不同而已，因此可將其共同之乘法項提出，再加入正負號，即乘一次轉動因數乘法，再用加減法運算，可獲得和原本乘二次轉動因數乘法相同效果。Fig2-1.4 以及 Fig2-1.5 即簡化前與簡化後的訊號流圖形。

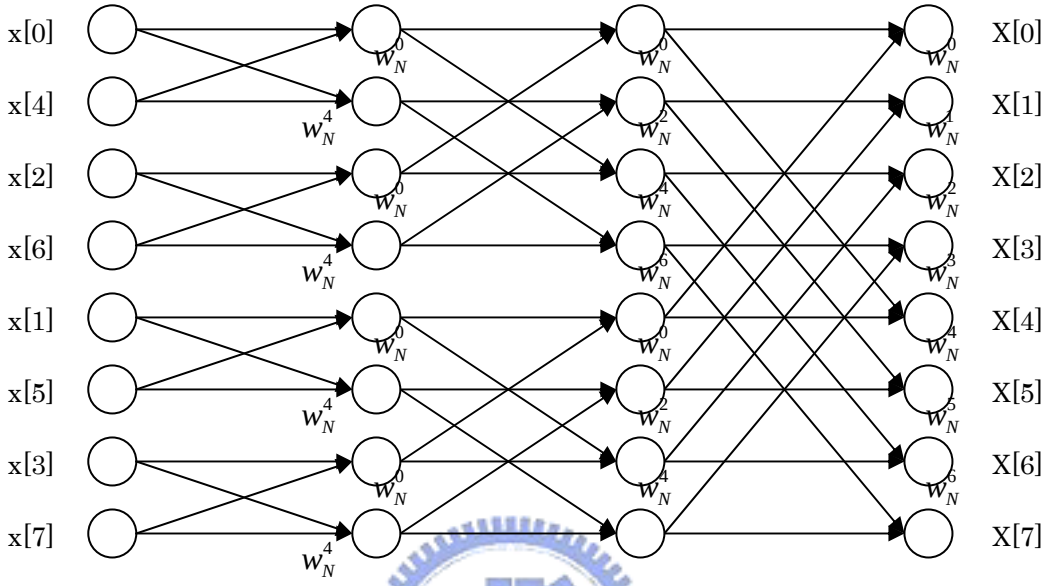

Fig2-1.4 8-point FFT with radix-2(未簡化)

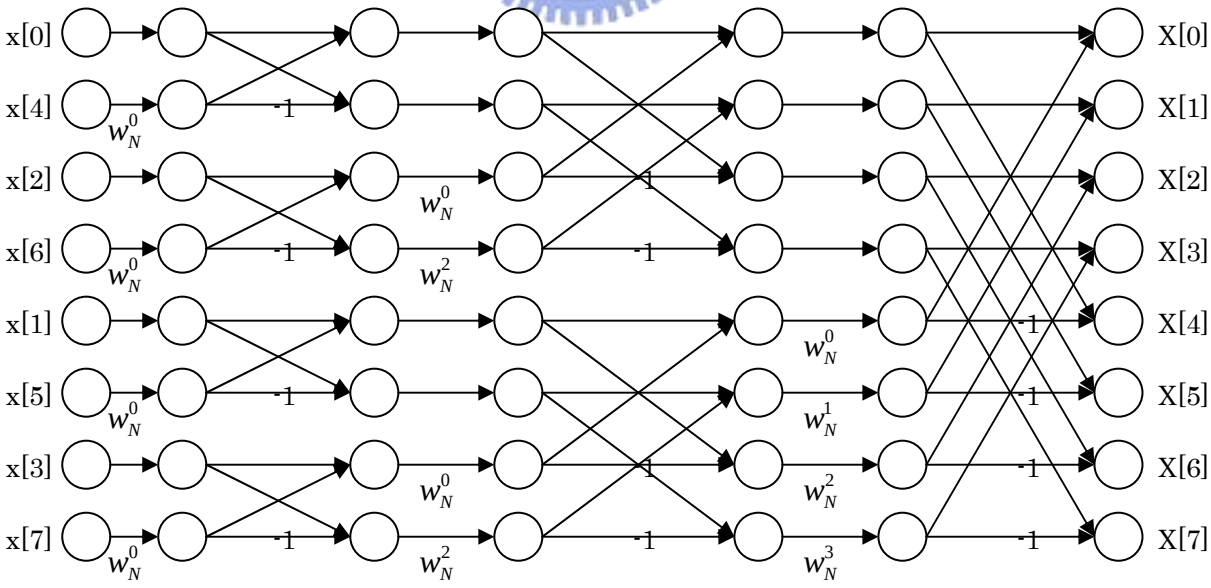

Fig2-1.5 8-point FFT with radix-2(簡化後)

2-1.2 DIF 架構（decimation in frequency）

在上一節介紹 DIT 架構，其是利用時間點來做分組動作，同樣的，可以利用附立葉轉換的輸出點，即頻率點做頻率上的分組動作，來構成 DIF 架構，其原理如下：

DFT 為 $X[k]=\sum_{n=0}^{N-1}x[n]w_N^{kn}$ ，　　$k=0.1\cdots\cdots.N-1$　　（2.9）

將 k 點分組為偶 數 k 點及奇數 k 點，首先先觀察偶數 k 點的組合
，令 k=2r，其中 r=0，1，2，…….（N/2）-1

$$
\begin{aligned}
X[2r]&=\sum_{n=0}^{N-1}x[n]w_N^{2rn}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{2rn}+\sum_{n=N/2}^{N-1}x[n]w_N^{2rn}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{2rn}+\sum_{n=0}^{(N/2)-1}x[n+(N/2)]w_N^{2r[n+(N/2)]}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{2rn}+\sum_{n=0}^{(N/2)-1}x[n+(N/2)]w_N^{2rn}\\
&=\sum_{n=0}^{(N/2)-1}(x[n]+x[n+(N/2)])w_{N/2}^{rn}
\end{aligned}
\qquad (2.10)
$$

此即為將輸入資料的上半平面和下半平面的訊號相加後做（N/2）點的離散傅立葉轉換（DFT）。
在奇數 k 點的組合方面，令 k=2r+1

$$
\begin{aligned}
X[2r+1]&=\sum_{n=0}^{N-1}x[n]w_N^{(2r+1)n}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{(2r+1)n}+\sum_{n=N/2}^{N-1}x[n]w_N^{(2r+1)n}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{(2r+1)n}+\sum_{n=0}^{(N/2)-1}x[n+(N/2)]w_N^{(2r+1)[n+(N/2)]}\\
&=\sum_{n=0}^{(N/2)-1}x[n]w_N^{(2r+1)n}+w_N^{(2r+1)(N/2)}\sum_{n=0}^{(N/2)-1}x[n+(N/2)]w_N^{(2r+1)n}\\
&=\sum_{n=0}^{(N/2)-1}(x[n]-x[n+(N/2)])w_N^{(2r+1)n}\\
&=\sum_{n=0}^{(N/2)-1}(x[n]-x[n+(N/2)])w_{N/2}^{rn}w_N^{n}
\end{aligned}
\qquad (2.11)
$$

其中 r=0，1，2，…….(N/2)-1

此即為將輸入資料的上半平面和下半平面的訊號相減後乘上轉動因數（twiddle factor）然後做(N/2)點的離散傅立葉轉換（DFT)。

其訊號流程圖如 Fig2-1.6 所示：

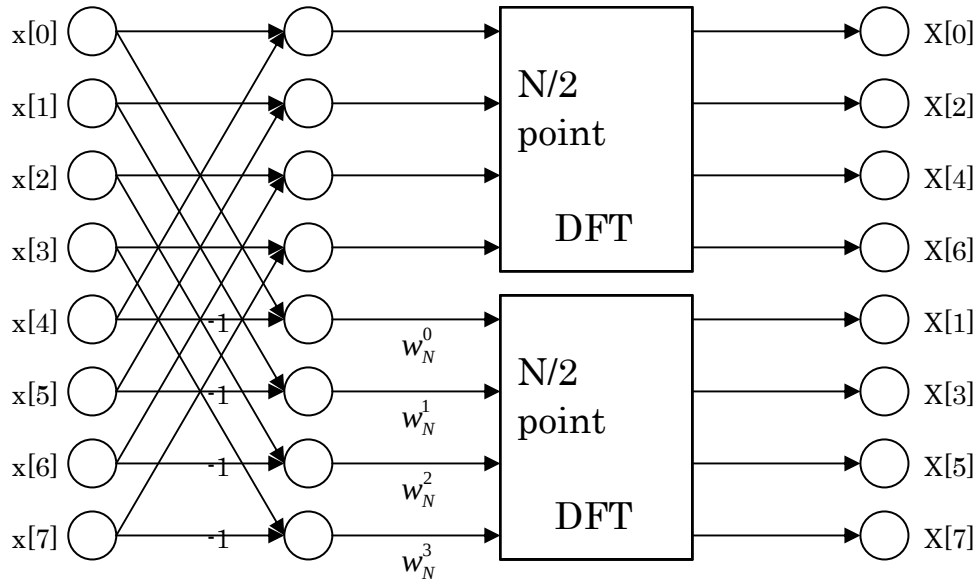

Fig2-1.6 Simple DIF 8-point FFT

將此分組後的訊號再用同樣的原理繼續重覆分組，即可獲得 Fig2-1.7 著名的頻率點分組之蝴蝶圖（ DIF butterfly)。

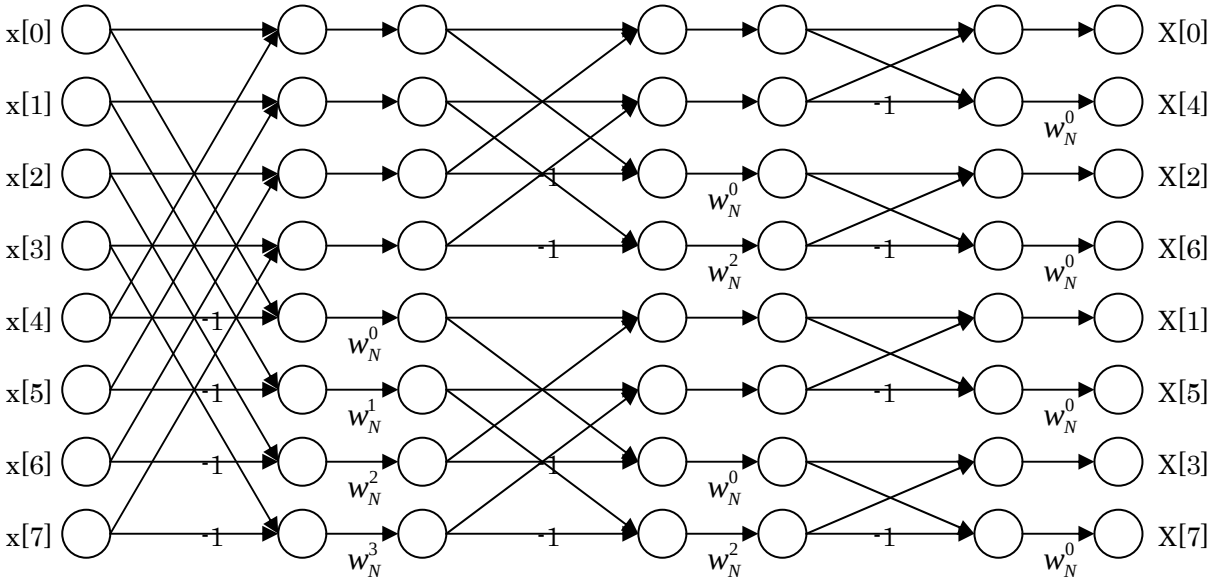

Fig2-1.7 DIF 8-point FFT with radix-2

DIT及DIF的架構很相似，其差別只有蝴蝶單位（BUTTERFLY UNIT）的跨越距離及轉動因數（twiddle factor）運算的先後不同而已，其共同特性就是可以把轉動因數的計算減低很多，再加上把原本的N級運算減小到$\log_2 N$級運算，因此總運算量從原本DFT需要的N^2次複數乘法降低到$N/2\log_2 N$次複數乘法，整個運算量減少，相對地運算速度可以大大地提昇，這也就是快速傅立葉轉換可以達到快速運算的原因。

2-2 以四點為基底之快速傅立葉轉換（Radix-4 FFT）

上一節介紹以二為基底的 DIT，DIF 快速傅立葉轉換，在這節要說明另一種比 radix-2 架構更有效率之 radix-4 架構。

由FFT的運算可以知道radix-2 架構是利用複數平面的 0^0與 180^0的對稱性，將乘法合併，因此運整算可以由原先每一級的N次複數運算降為每一級的（N/2）次複數運算，因此整體運算量可以降低，速度自然可以加快，現在將這樣的觀念推廣到 0^0，90^0，180^0，270^0。可以發現在複數平面上，其所對映的值分別是+1，+j，-1，-j，當一個複數輸入訊號乘上+j或-j，其結果是等效於實部與虛部交換後，再將其中一項變號，如式子（2.12）所示：

$$(a+jb)*(+j)=-b+ja$$
$$(a+jb)*(-j)=+b-ja \qquad (2.12)$$

所以了解其特性後，所有與+j 或-j 相乘之複數乘法運算將可變為簡單的實虛部交換，而不必使用到複雜的複數運算，也因此這樣的乘法就成為不主要的複數乘法運算（trivial multiplication），這也就是 radix-4 運算會比 radix-2 運算更有效率的地方，其 radix-4 DIT 架構的數學推導式子如下：

有限序列 x(n)之 N 點離散傅立葉轉換為

$$X(k)=\sum_{n=0}^{N-1} x(n)_N^{kn} \qquad (2.13)$$

假設 $N=4^i$，令 $n=4^{i-1}m_{i-1}+4^{i-2}m_{i-2}+\ldots+4^2m_2+4m_1+m_0$ （2.14）

$$k=4^{i-1}t_0+4^{i-2}t_1+\ldots+4^2t_{i-3}+4t_{i-2}+t_{i-1} \qquad (2.15)$$

其中 $0 \le m_{i-1}, m_{i-2}, \ldots m_1, m_0 \le 3$

$$0 \le t_{i-1}, t_{i-2}, \ldots t_1, t_0 \le 3 \qquad (2.16)$$

則

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

$$= \sum_{m_{i-1}=0}^{3} \ldots \sum_{m_1=0}^{3} \sum_{m_0=0}^{3} x(4^{i-1}m_{i-1} + 4^{i-2}m_{i-2} + \ldots + 4m_1 + m_0) W_N^{k(4^{i-1}m_{i-1} + 4^{i-2}m_{i-2} + \ldots + 4m_1 + m_0)} \qquad (2.17)$$

$$X(4^{i-1}t_0 + 4^{i-2}t_1 + \ldots 4^2 t_{i-3} + 4t_{i-2} + t_{i-1})$$

$$= \sum_{m_{i-1}=0}^{3} \ldots \sum_{m_1=0}^{3} \sum_{m_0=0}^{3} x(4^{i-1}m_{i-1} + 4^{i-2}m_{i-2} + \ldots + 4m_1 + m_0)$$

$$* W_N^{(4^{i-1}t_0 + 4^{i-2}t_1 + \ldots + 4t_{i-2} + t_{i-1})(4^{i-1}m_{i-1} + 4^{i-2}m_{i-2} + \ldots + 4m_1 + m_0)} \qquad (2.18)$$

此即為著名的以四點為基底之快速傅立葉轉換（Radix-4 FFT）。

以N=16 為例子， $n = 4m_1 + m_0$ ， $k = 4t_0 + t_1$

$$X(4t_0 + t_1) = \sum_{m_1=0}^{3} \sum_{m_0=0}^{3} x(4m_1 + m_0) W_{16}^{(4t_0 + t_1)(4m_1 + m_0)}$$

$$= \sum_{m_1=0}^{3} \sum_{m_0=0}^{3} x(4m_1 + m_0) W_{16}^{4t_0 m_0} W_{16}^{4t_1 m_1} W_{16}^{t_1 m_0}$$

$$= \sum_{m_1=0}^{3} \underbrace{W_{16}^{t_1 m_0} \sum_{m_0=0}^{3} x(4m_1 + m_0) W_{16}^{4t_0 m_0}}_{\text{first stage}} W_{16}^{4t_1 m_1} \qquad (2.19)$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\text{second stage}}$$

其中W_{16}^{t1m0}為轉動因數（twiddle factor）。Fig2-2.1 即為 16 點FFT之SFG。

由運算量分析可以知道，radix-4 FFT 的運算量會比 radix-2 FFT 的運算量減少 23% ，因此整體架構的運算會變得更有效率，也可以節省 23% 的功率，實在是不錯的方式，不過唯一缺點就是硬體設計複雜度會提高，Fig2-2.2 與 Fig2-2.3 為 radix-2 與 radix-4 的基本元件之比較：

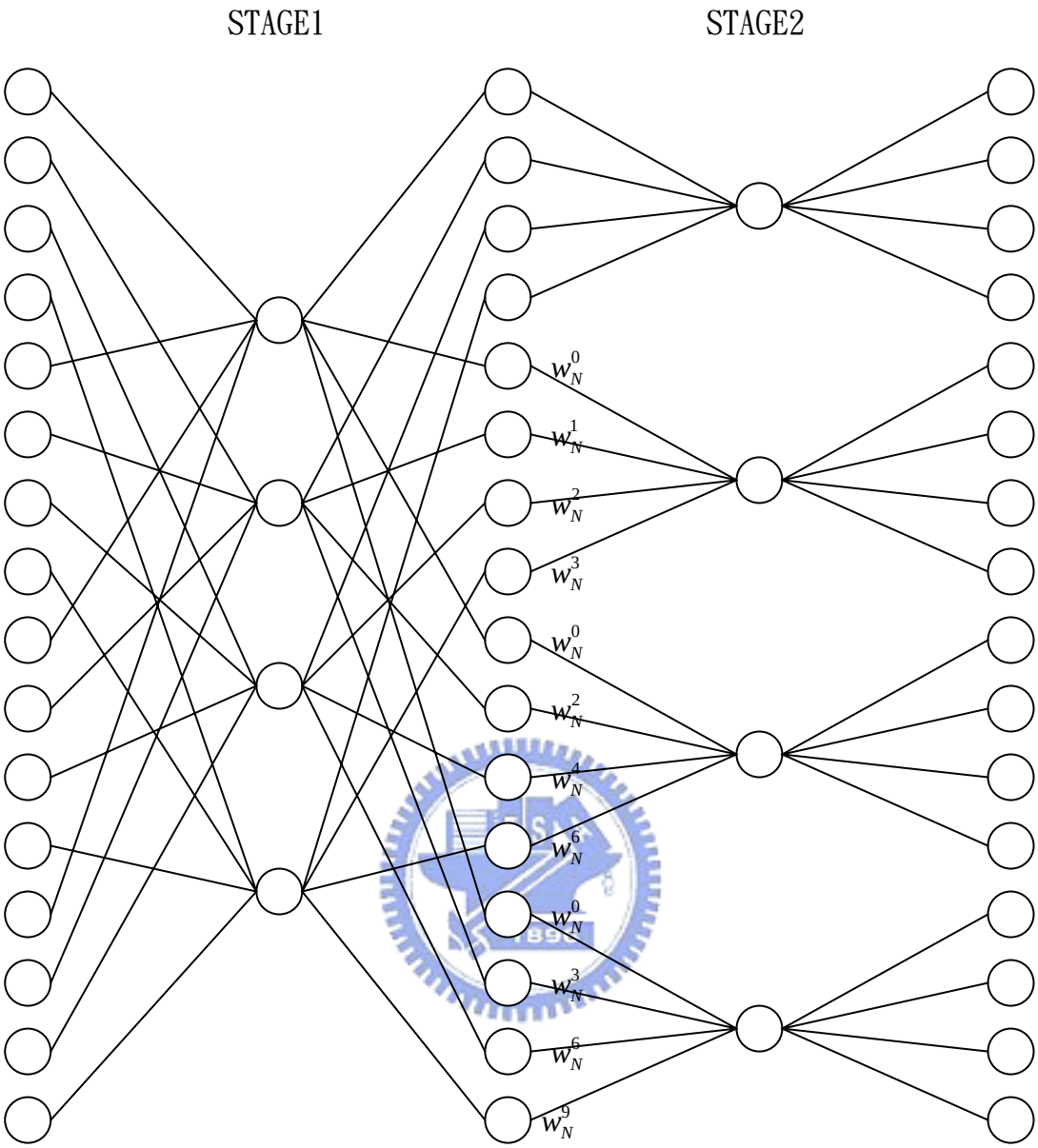

Fig2-2.1 FFT with Radix-4 簡化流程圖

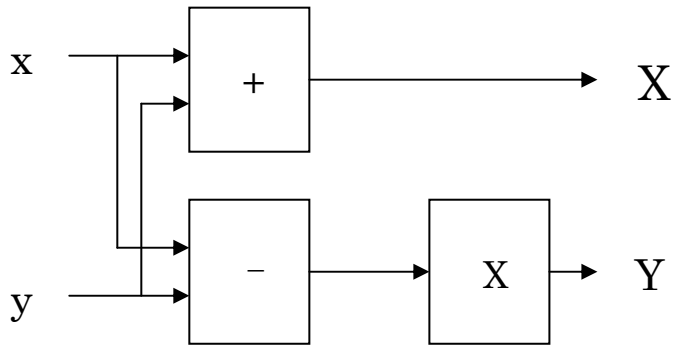

Fig2-2.2 Radix-2 基本元件

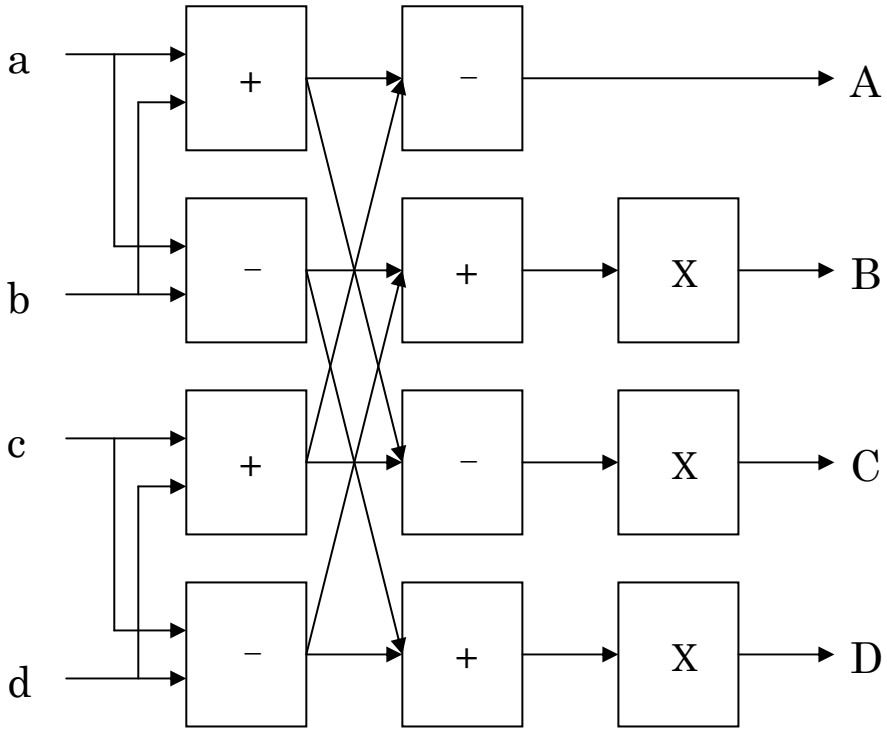

Fig2-2.3　　Radix-4 基本元件

可以看到一個基本的 radix-2 元件需要 2 個複數加法器及 1 個複數乘法器，而 radix-4 需要 8 個複數加法器及 3 個複數乘法器，其硬體複雜度約為 radix-2 的三倍，這也就是設計硬體力，在速度，功率與硬體複雜度需要做平衡考量（trade-off）的地方。

2-3　　以 2^2 為基底之快速傅立葉轉換（Radix-2^2 FFT）

Radix-2^2 FFT[1][20]具有和radix-4 減少計算複雜度的優點，而且仍可保持 radix-2 的butterfly之架構，相當有效率，因此常被使用在radix-4 系列的運算處理，其原理與推導如下：

DFT 為 $X(k)=\sum_{n=0}^{N-1}x(n)w_N^{nk}$ ，其中 $0\le k\le N-1$　　（2.20）

W_N是單位圓上N個根，假設

$$n=<\frac{N}{2}n_1+\frac{N}{4}n_2+n_3>_N$$

$$k=<k_1+2k_2+4k_3>_N \qquad (2.21)$$

根據共因數演算法(common factor algorithm)CFA

$$X(k_1+2k_2+4k_3)$$

$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)w_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}$$

$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\{\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)w_N^{(\frac{N}{2}n_1)(2k_2+4k_3)}w_N^{(\frac{N}{2}n_1)(k_1)}\}w_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}$$

$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\{\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)w_N^{(\frac{N}{2}n_1)(k_1)}\}w_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}$$

$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\{B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2+n_3)w_N^{(\frac{N}{4}n_2+n_3)(k_1)}\}w_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \qquad (2.22)$$

其中

$$B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2+n_3)=x(\frac{N}{4}n_2+n_3)+x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)w_N^{\frac{N}{2}k_1} \qquad (2.23)$$

$$=x(\frac{N}{4}n_2+n_3)+(-1)^{k_1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)$$

由上述的公式可知，與其radix-2 butterfly的架構相似，兩者之間的差別只在於radix-2 先將(2.23)和twiddle factor先行計算後再繼續下一部分解之動作，而radix-2^2演算方式是式子(2.23)先運算，括號內部的twiddle factor再和外部的twiddle factor進行合併運算，即可得到以下的式子:

$$w_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}=w_N^{(Nn_2k_3)}\ w_N^{(\frac{N}{4}n_2)(k_1+2k_2)}\ w_N^{(n_3)(k_1+2k_2)}\ w_N^{(4n_3k_3)}$$

$$=(-j)^{n_2(k_1+2k_2)}w_N^{(n_3)(k_1+2k_2)}\ w_N^{(4n_3k_3)} \qquad (2.24)$$

將所計算出來的的W帶入(2.22)即可得到下列式子:

$$X(k_1+2k_2+4k_3)=\sum_{n_3=0}^{\frac{N}{4}-1}[H(k_1,k_2,n_3)w_N^{n_3(k_1+2k_2)}]w_{\frac{N}{4}}^{k_3n_3} \qquad (2.25)$$

其中H如式子(2.26)所示:

$$H(k_1,k_2,n_3)=\{[x(n_3)+(-1)^{k_1}x(n_3+\frac{N}{2})]+(-j)^{k_1+2k_2}[x(n_3+\frac{N}{4})+(-1)^{k_1}x(n_3+\frac{3N}{4})]\}$$

(2.26)

從式子可以清楚看到，兩組不同的中括號為兩組不同的第一級butterfly，大括號則是算完第一級後的第二級butterfly，因此可以知道每一次的radix-2^2 FFT是由相當於二級的radix-2 所構成，這就是為什麼稱做 2^2。可以把第一次的radix-2^2公式由Fig2-3.1 之butterfly SFG來代表，其中在式子中的k_1，k_2即對應到butterfly不同級。

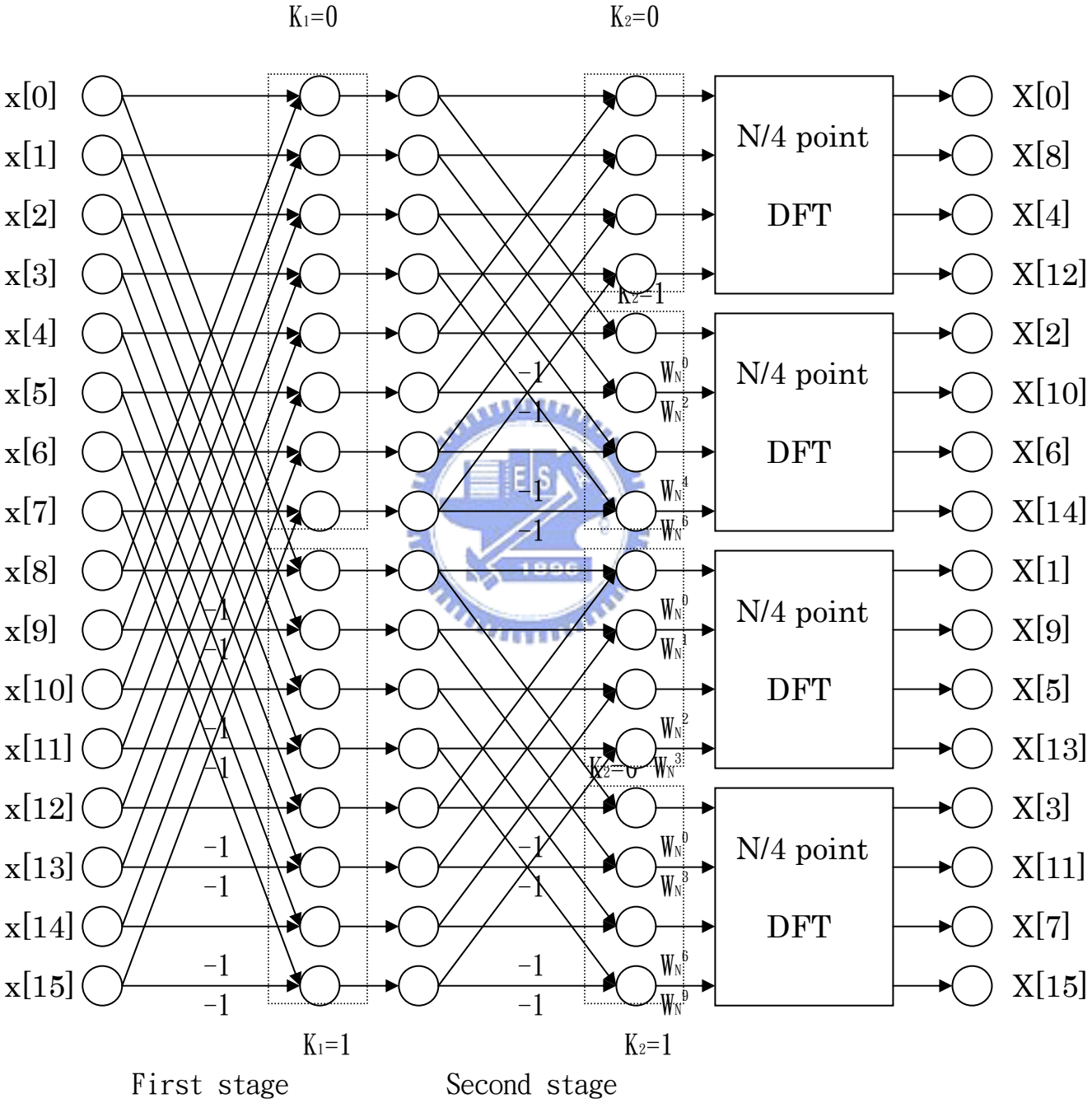

Fig 2-3.1　Simple 16-point FFT with Radix-2^2(DIF)

依照相同原理，將此演算法推廣到每一級的radix-2^2 butterfly，可以獲得

Fig2-3.2 的SFG，注意一看會發現此SFG和radix-2 以及radix-4 非常相似。和radux-2 的差別只在於twiddle factor的複數乘法配置位置不同。而和radix-4 的差別只在於整體訊號處裡架構不同。經過仔細分析，***可以知道radix-2^2 是結合了兩種演算法的優點，分別是具有radix-4 低運算量的優點和radix-2 簡單而具有彈性的butterfly架構之優點***。

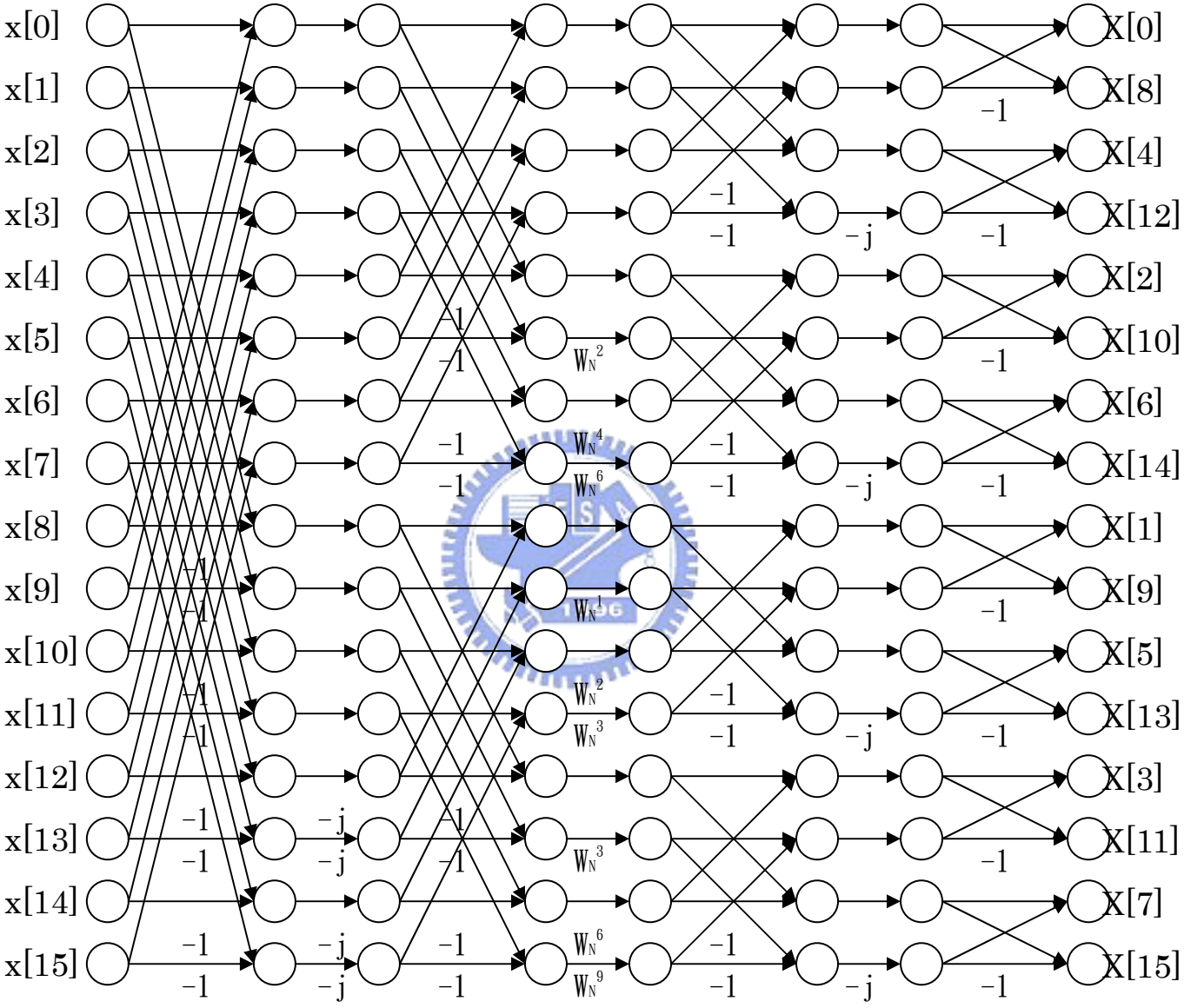

Fig2-3.2　Complete 16-point FFT with Radix-2^2

2-4 以 2^3 為基底之快速傅立葉轉換（Radix-2^3 FFT）

由數學分析可以得知，FFT 的 radix 等級愈高，運算量愈少，計算愈有效率，但是硬體的複雜度會提高，在這裡將介紹較有效率之 radix-8 演算法，其利用對稱性，將三個 radix-2 串接，具有 radix-2 簡單架構之優點極低複數運算量的特點，以下為 radix-8 之公式(2.27):

$$X[8k]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])]+[(x[n+\frac{N}{8}]+x[n+\frac{5N}{8}])+(x[n+\frac{3N}{8}]+x[n+\frac{7N}{8}])]\}w_N^{8nk}$$

$$X[8k+1]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]-x[n+\frac{N}{2}])-j(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])]+w_N^{\frac{8}{N}}[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])-j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{n}w_N^{8nk}$$

$$X[8k+2]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])]-j[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])+j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{2n}w_N^{8nk}$$

$$X[8k+3]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]-x[n+\frac{N}{2}])+j(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])]+w_N^{\frac{3N}{8}}[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])+j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{3n}w_N^{8nk}$$

$$X[8k+4]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])]-[(x[n+\frac{N}{8}]+x[n+\frac{5N}{8}])+(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{4n}w_N^{8nk}$$

$$X[8k+5]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]-x[n+\frac{N}{2}])-j(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])]-w_N^{\frac{8}{N}}[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])-j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{5n}w_N^{8nk}$$

$$X[8k+6]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])]-j[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])+j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{6n}w_N^{8nk}$$

$$X[8k+7]=\sum_{n=0}^{\frac{N}{8}-1}\{[(x[n]-x[n+\frac{N}{2}])+j(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])]-w_N^{\frac{3N}{8}}[(x[n+\frac{N}{8}]-x[n+\frac{5N}{8}])+j(x[n+\frac{3N}{8}]-x[n+\frac{7N}{8}])]\}w_N^{7n}w_N^{8nk}$$

(2.27)

此架構為三級 radix-2 之串接，其 SFG 如 2-4.1，可以發現其很多重要乘法(non-trivial multiplication)被不重要乘法(trivial multiplication)所取代，因此運算量減少很多。

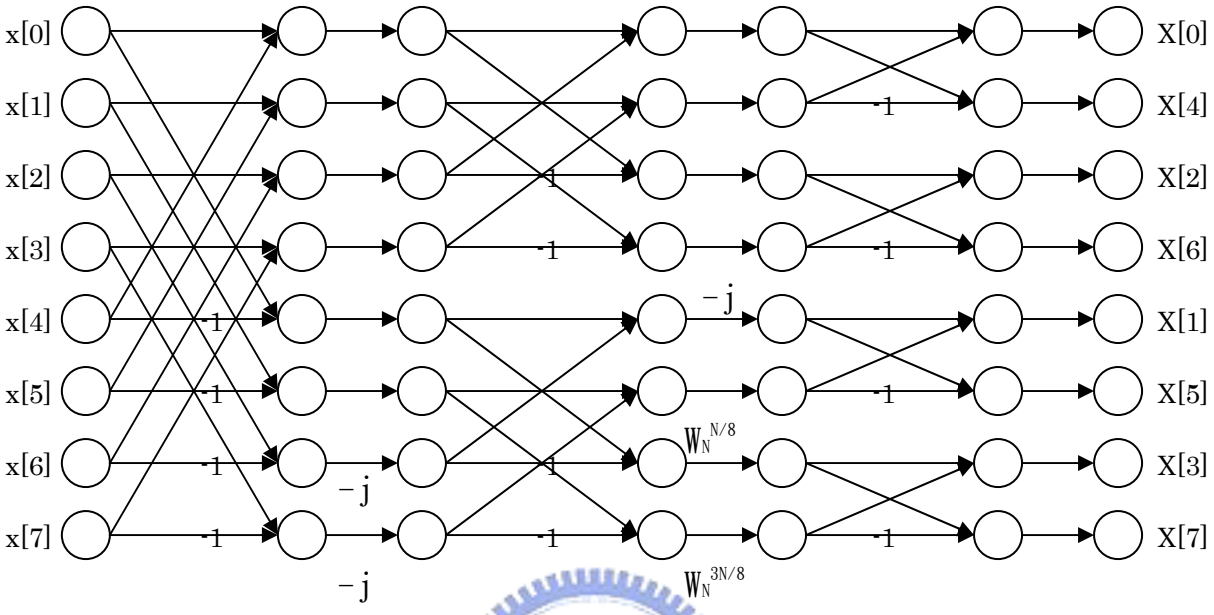

Fig 2-4.1 8-point FFT with Radix-2^3

由計算可以發現 radix 數愈大，計算愈有效率，重要乘法運算量會變少，但是當 radix 數大於 8 的時候，效率提升會漸漸緩和下來，比較不像低於 8 的時候那麼明顯，這是因為可以縮減的複數乘法對稱性變少，可縮減的運算量也跟著變少，而且當 radix 數大於 8 的時候硬體設計會比較複雜一點，***所以依照傳統演算法，大部分現在都使用 radix 不大於 8 的 FFT 演算法，但是 radix-16 演算法在當 FFT 點數很大的時候使用，再配合上特別的硬體設計改良，還是會比用不大於 radix 數 8 的時候來的好***，這一點後面章節會詳細討論。

2-5 不同演算法之比較

在看過以上幾種 FFT 演算法，我們在這裡將先前幾種演算法做綜合分析，並且可以將 FFT 歸納出幾個特性:

1. Radix 層次愈高，運算量愈少，效率愈高，但是硬體複雜度增加

2. Radix-2 的架構是所有 FFT 演算法之中最簡單也最具有彈性的架構

3. Radix-4 或是 radix-8 的運算量都比 radix-2 來的少，減少的主要原因是因為重要乘法被不要重乘法所取代，所以消耗功率小，效率也高

4. 若要獲得較有效率，低消耗功率，硬體設計簡單且有彈性，則適合用 radix-2 架構來串接的方式得到

所以根據以上幾點特點，現在極多數FFT設計架構都是用radix-2，radix-2^2是radix-2^3，而且再運用在pipeline FFT的技術上的話，速度和效率會更加明顯提升不少，而且 FFT 處理點數愈多所顯現出來的效果會愈明顯。

當然，對於不同點數的 FFT，我們也可以用 mix-radix 的架構來實現，好比說 32-point FFT可以用radix-2^2和radix-2^3來串接在一起，這會比單用radix-2 架構來的省功率和硬體面積。而往後的章節我們會討論不同點數 FFT 運用不同架構的設計所實現在硬體上出現不同的優缺點，並且會運用別的技術來替代一部分的重要乘法。