國立交通大學

電機與控制工程學系

碩士論文

以樹狀重組技術實現高延展性數位
傅立葉轉換器之研究

Design on Scalable FFT Processor Using Tree Transformation Technique

研 究 生：林毅慧
指導教授：董蘭榮 博士

中華民國九十三年七月

以樹狀重組技術實現高延展性數位傅立葉轉換器之研究

Design on Scalable FFT Processor Using Tree Transformation Technique

研究生：林毅慧　　**Student : Yi-Hui Lin**

指導教授：董蘭榮　　**Advisor : Lan-Rong Dung**

國立交通大學

電機與控制工程學系

碩士論文

A Thesis
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master in

Electrical and Control Engineering
July 2004
Hsinchu, Taiwan, Republic of China

中華民國九十三年七月

以樹狀重組技術實現高延展性數位傅立葉轉換器之研究

學生：林毅慧　　指導教授：董蘭榮

國立交通大學電機與控制工程學系

摘　要

近幾年來快速傅立葉轉換與反快速傅立葉轉換已經被廣泛的被提出並應用在 xDSL 上。文獻上可考之快速傅立葉轉換演算法很多，如:radix-2、radix-4 與 split-radix 等。這些演算法各有各的優點以符合應用的需求。已有很多論文針對其中一種演算法來實現硬體，但是很少有論文能提出彈性的方法或是硬體架構能允許設計者充分且適當探索分析演算法的平行度與元件配置可能。在本篇論文中，提出用樹狀的架構來詮釋各種演算法並充分呈現演算法平行化的各種可能性。我們所提的方法可以彈性的實現不同的快速傅立葉轉換運算，而且在硬體架構上也可依使用者所需增加平行度。為了證實我們提出的樹狀 FFT 架構的可行性，我們用 FPGA 實現 SR-2/4 64 點的 FFT 並完成軟體驗證。

Design on Scalable FFT Processor Using Tree Transformation Technique

student ：Yi-Hui Lin Advisor：Dr. Lan –Rong Dung

Department of Electrical and Control Engineering

National Chiao Tung University.

Abstract

FFT and IFFT have been broadly applied to xDSL technology. There have existed a large number of FFT algorithms, such as radix-2, radix-4 and split-radix. Designers may use any of them to implement FFT/IFFT computations according to system requirements. Papers have presented architectures for implementing specific FFT algorithms; however, few papers propose general architectures for exploring possibilities of parallelization. This thesis proposes a tree-based FFT algorithm that gives designers a good position to view the parallelism of FFT algorithms and hence provides a handy tool for FFT/IFFT architects. The tree-based FFT algorithm is highly scalable in terms of the number of processing elements and pipelining buffers. The scalability makes tradeoffs between memory and processing power tractable. To demonstrate the feasibility of proposed tree-Based FFT architecture and its correctness, we implemented a 64-point SR-2/4 FFT on FPGA platform and verified the functionality using FPGA emulation environment.

誌　謝

這篇論文得以完成，最要感謝的是董蘭榮教授，從題目的擬定、理論與架構的設計到結果的分析以及論文的撰寫，都全程在旁給予我最大的協助，尤其是老師認真的教學態度以及對學生的關懷更讓我印象深刻，也讓我獲益良多。口試當中，承蒙胡竹生老師、黃俊達老師與劉建男老師在百忙中撥冗參加，並給予精闢的建議，讓本論文更臻完備，在此獻上最誠摯的感謝。

研究所這兩年，讓我成長與學習了很多，感謝一路陪我走過來的實驗室伙伴們，顯文學長、學之學長、介皇學長、盟淳學長、健釗、玉書、昭維、樹德與芳彥等，給予我課業上以及生活上的幫助，因為有你們讓我有多采多姿的研究所生活。

最後，感謝我的家人一直支持著我，讓我在求學上無後顧之憂。在此，僅將本篇論文獻給我的家人以及所有我關心及關心我的人，人生的路上有你們相伴的我是何其幸運，謝謝你們！

林毅慧　謹識

中華民國九十三年七月

目　錄

表 目 錄

圖目錄

第一章 緒　論

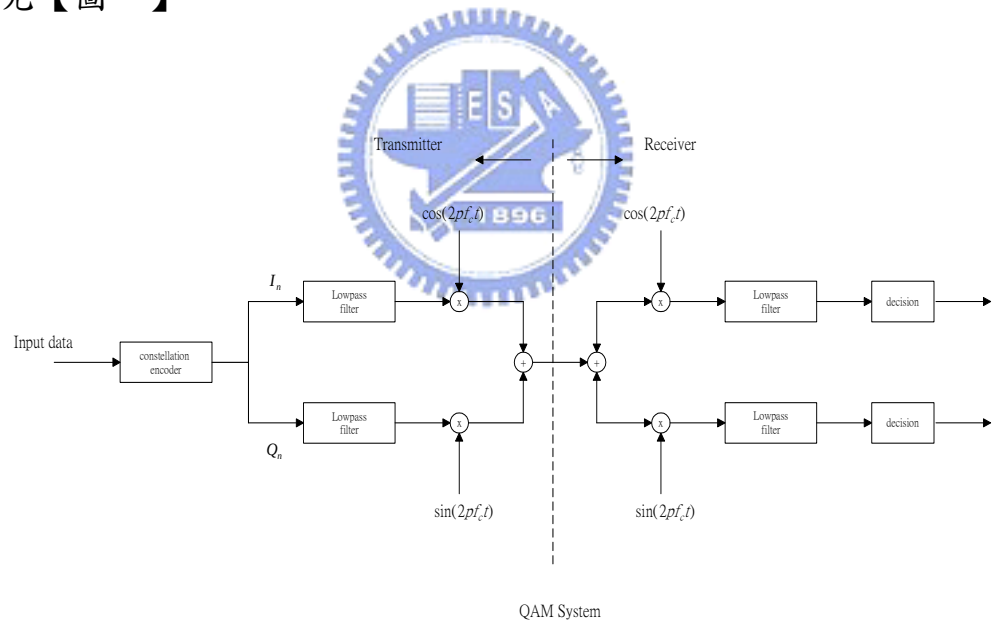
1.1 研究背景

快速傅立葉轉換(FFT)在通訊系統中多被使用，我們希望能夠提出一個有彈性的快速傅立葉轉換(FFT)硬體架構能夠適用於在各種數位用戶迴路(xDSL)[6]中，因此在數位用戶迴路(xDSL)的應用中，快速傅立葉轉換(FFT) 輸入資料不需要按照順序，在此應用層面下我們提出樹狀快速傅立葉轉換(FFT)的架構讓使用者能夠根據此硬體架構去實現硬體。以下為一些簡單的數位用戶迴路(xDSL)的介紹。數位資料的傳輸，最早一開始是用電話利用震動的方式只能傳輸類比的語音訊號，然而隨著科技的進步，目前已經發展至網路的型態，其中最廣為人知，非對偶數位用戶迴路(ADSL)[7][8] 即為其中一種。數位用戶迴路 (DSL,Digital Subscriber Line) 技術開始於 144kb/s 的 基本 ISDN(BRI)，之後才有 1.5 和 2.0Mb/s 的 HDSL 與 7Mb/s 的非對偶數位用戶迴路(ADSL)，一直到現在最為熱門的 52Mb/s 的高速數位用戶迴路(VDSL)，以上所有的傳輸技術皆簡稱為數位用戶迴路(xDSL)。在非對偶數位用戶迴路(ADSL)中使用 DMT(Discrete Multi-tone)的傳輸技術，而因為高速數位用戶迴路(VDSL)[4][5]速度的增快，因此傳輸技術有

QAM(Quadrature Amplitude Modulation)和 CAP(Carrierless AM/PM) 而 MCM 又有如 DMT(Discrete Multi-tone)等技術發展。[1][2][3] 以下先介紹其基本調變技術：

QAM 調變

先將 2 進位的資料輸入星狀圖編碼器(consterllation encoder)作對映的動作得到特定的編碼，之後再經由 sine 與 cosine 的弦波做調變再把訊號傳遞出去。在接收端也用同樣的兩組正交的波形再將其訊號解回來，見【圖一】

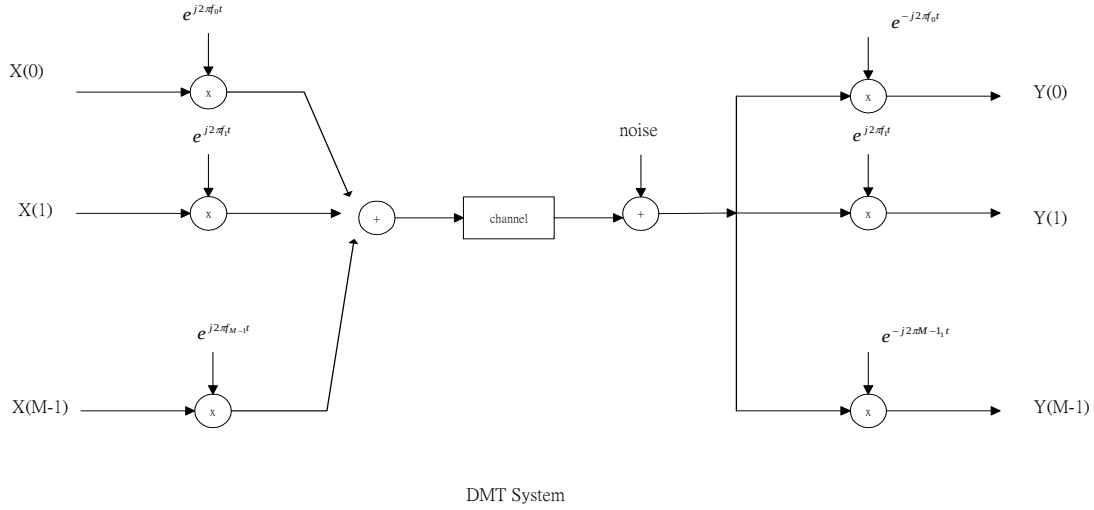


【圖一】 QAM 的架構圖

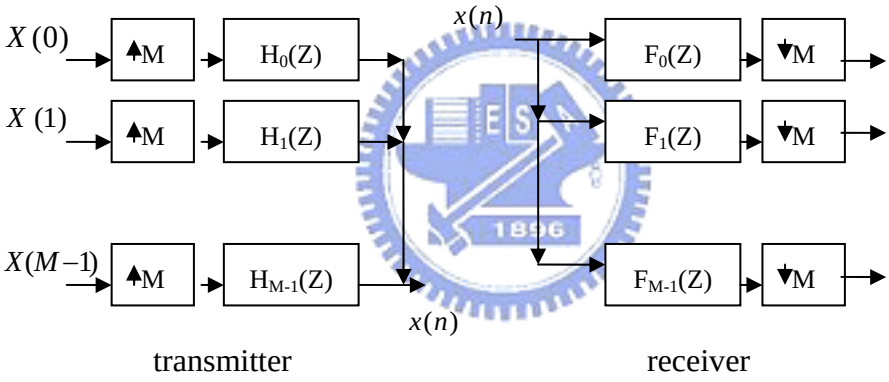
DMT 調變

DMT 調變技術主要是利用 MCM(multi-carrier modulation)的調變技術來傳輸，見【圖二】。把資料用並行的方式傳輸，然後再分別用不同的載波來調變，其中一個載波是一個子通道(subchannel)，每個子通道之間是互相獨立，可看成所有子通道合成一個寬頻來傳輸資料。因為子通道非常的龐大，因此後來使用正交多頻分工(OFDM)的方法，把每個載波重疊(overlap)然後再利用離散傅立葉轉換(Discrete Fourier Transform , DFT)使其有正交特性，見【圖二】。

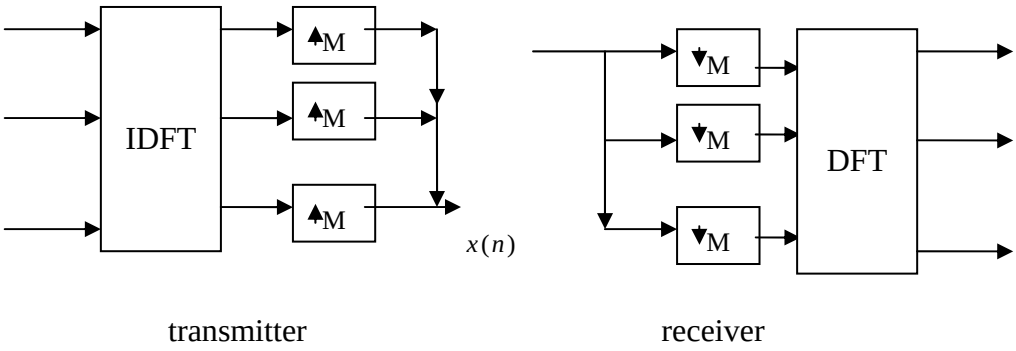
對於高速數位用戶迴路(VDSL)的原始的 DMT 系統中見【圖三】，$H_0(Z)$ 為原型濾波器(Prototype filter)，至於 $H_k(Z)$ $,k=1\sim M-1$ 皆為 $H_0(Z)$ 在頻率軸中移動 $\frac{2\pi}{M}$，傳送端實現的方式輸入端經過 IDFT 的方塊圖再經過升頻速率(upsampling rate) M，而接收端的實現方式即為，接收端先經過降頻(downsampling) M 再經過數位傅立葉轉換方塊(DFT block) ，見【圖四】。因此在 DMT 的系統中，最重要的地方還是在離散傅立葉轉換與反離散傅立葉轉換(DFT/IDFT) 如何實現。



【圖二】 DMT 系統



【圖三】 濾波器表示 DMT 系統



【圖四】 DFT /IDFT 表示 DMT 系統

因為濾波器頻帶(Filter bank)[3]的分解方式，之後會產生快速傅立葉轉換/反快速傅立葉轉換(FFT /IFFT)的方塊。 經由以上的調頻所需要的快速傅立葉轉換(FFT)硬體不需要按照順序輸入(in order)，所以才發展出我們樹狀的演算方式。

1.2 研究動機

以往所提出的快速傅立葉轉換(FFT)的硬體架構，大多著眼於單一的演算法，如基-2(radix-2), 基-4(radix-4)和分割-基數(split-radix)等，在彈性上面的選擇就因此限制住，當選擇使用分割-基數(split-radix)的演算法，就要因此而量身定做一套硬體，但是在分析當中，每一種演算法有其優點，應當讓使用者選擇何種演算法是最佳的。

所以，基於這樣的原因，試圖找出一套有效的理論，能夠符合任何一種快速傅立葉轉換的演算法，也讓使用者能夠有彈性的選擇所需。

1.3 提出的理論與方法

我們所提出的樹狀演算法，是先對各種不同的快速傅立葉轉換演算法先形成蝴蝶圖(butterfly)的形狀，再依照樹狀的方式讓每一級的蝴蝶圖(butterfly)皆能形成二元樹或四元樹，再依照排程方式，讓使用者可以選擇輸入的快速傅立葉轉換型態，以得到正確的快速傅立葉轉換結果。

1.4 章節安排

本論文的組織為：

第一章：說明研究背景與動機，以及簡單的介紹我們所提出的樹狀理論。

第二章：介紹各種不同的快速傅立葉轉換演算法與硬體，並比較其中的優缺點。

第三章：提出如何使用樹狀延展性技術，把蝴蝶(butterfly)轉變成平行的樹狀圖形並加以驗證。

第四章：提出樹狀延展性技術的硬體架構

第五章：FPGA 模擬圖形。

第六章：結論與未來展望。

第二章 各種快速傅立葉轉換演算法及架構

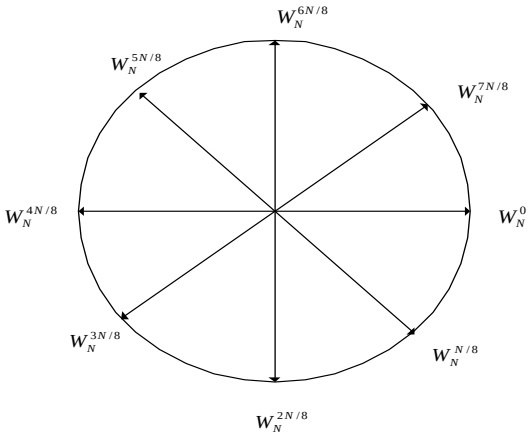
N 點輸入資料序列(input data sequence){x[n]}的離散傅立葉轉換(Discrete Fourier Transform，DFT)[9]定義為：

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn},\quad where\quad W_N=e^{-j\frac{2\pi}{N}}\quad k=0\sim N-1 \tag{2.1}$$

其中 N 稱為離散傅立葉轉換的長度(length)，而 n 代表的是時間域(time domain)k 代表的是頻率域(frequency domain)，N 點的資料序列(data sequence){x[k]}的反離散傅立葉轉換(Inverse Discrete Fourier Transform ，IDFT)定義為：

$$x[n]=\frac{1}{N}\sum_{k=0}^{N-1}X[k]W_N^{nk}\ ,\quad where\quad n=0\sim N-1 \tag{2.2}$$

由(2.1)與(2.2)式中其中W_N^{kn}稱為交互因子(twiddle factor)，可經由【圖五】發現交互因子(twiddle factor) 把 0 到 2π 分成所需的 N 等分。[11]



【圖五】 分割 Twiddle factor

如果直接計算 N 點的離散傅立葉轉換需要 N(N-1)的複數加法器與複數乘法器，因此對於 N 點離散傅立葉轉換的運算複雜度為 $O(N^2)$。因此發展出快速傅立葉轉換(Fast Fourier Transform，FFT)[10]演算法來降低運算複雜度 $O(N\log_2 N)$。

為了增加離散傅立葉轉換有效的運算，將用快速傅立葉轉換的運算方式去實現離散傅立葉轉換，而快速傅立葉轉換可以分為分時快速傅立葉轉換(Decimation-in-Time FFT ,DIT FFT) 和分頻快速傅立葉轉換(Decimation-in-Frequency FFT ,DIF FFT) 。[9][12]

2.1 各種快速傅立葉轉換(FFT)演算法

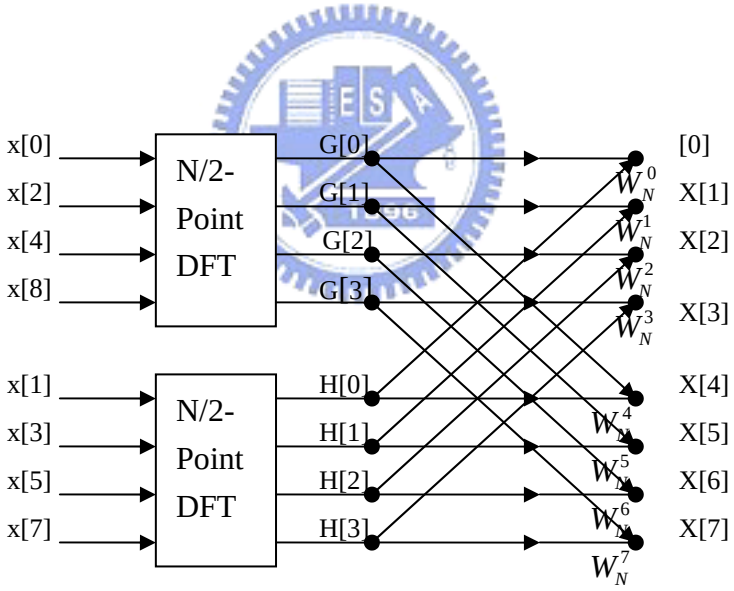
2.1.1 分時快速傅立葉轉換(DIT FFT)演算法

分時快速傅立葉轉換(DIT-FFT)演算法又稱為分時基-2 (DIT radix-2) 快速傅立葉演算法[13]，其基本的演算法定義如下：

取 $N = 2^{\nu}$

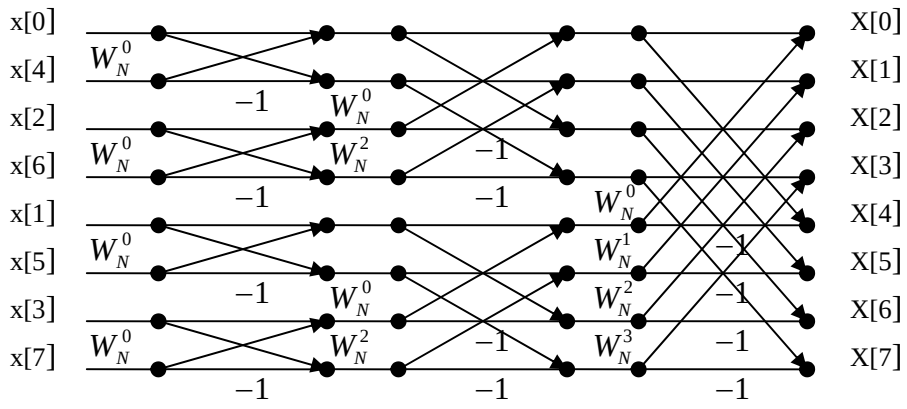
$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\
&= \sum_{n:even} x[n] W_N^{kn} + \sum_{n:odd} x[n] W_N^{kn} \\
&= \sum_{r=0}^{N/2-1} x[2r] W_N^{2rk} + \sum_{r=0}^{N/2-1} x[2r+1] W_N^{k(2r+1)} \\
&= \sum_{r=0}^{N/2-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x[2r+1] W_{N/2}^{rk} \\
&= G[k] + W_N^k H[k], \\
&\textit{where} \quad G[k] \quad \textit{and} \quad H[k] \quad \textit{are} \quad N/2 - \textit{point} \quad \textit{DFT}.
\end{aligned}
$$

以上式子由圖形表示(以八點為例子)[14]，見【圖六】:



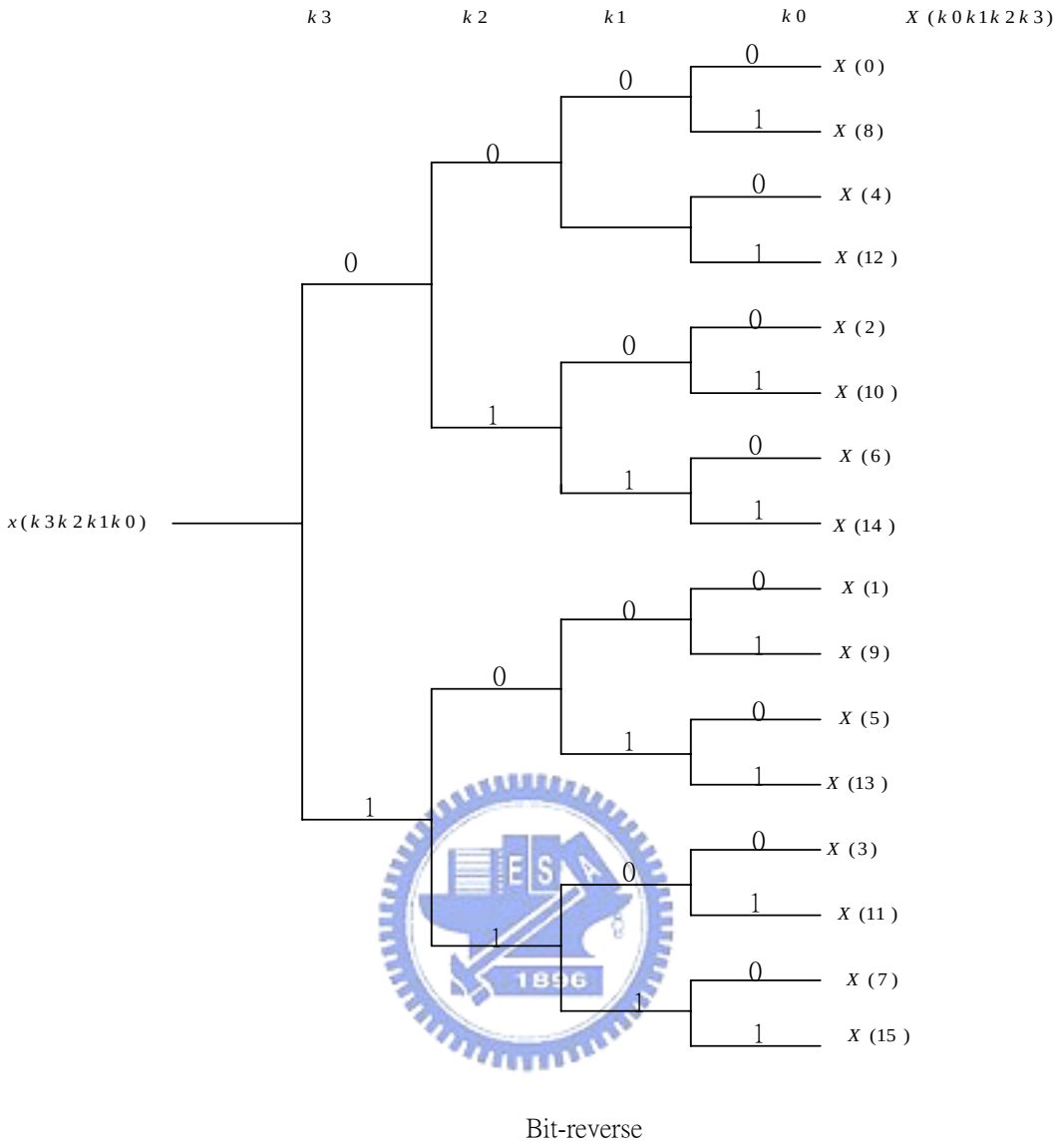
【圖六】 八點經過四點 DFT 轉換 butterfly 圖形

以同樣的方式，再把 N/2 點的快速傅立葉轉換再往下分成四塊 N/4 點的快速傅立葉轉換，最後圖形即變成以下之圖形(以八點為例子)，見【圖七】:



【圖 七】 八點的 DIT FFT butterfly 圖形

由【圖七】的圖形中可以發現其輸入端的資料必須以反向位元(bitreverse) 的方式輸入，輸出端才能夠依照順序輸出。其中所謂的反向位元(bitreverse)的名詞解釋見【圖八】，假設 N＝16，每一筆輸入的資料 $x(k3k2k1k0)$ 的註標(index)可以用位元的方式表示為 k0,k1,k2 與 k3，經由反向位元(bitreverse)所產生的輸出為 $x(k0k1k2k3)$，因此經由反向位元(bitreverse)即可快速的算出目前的輸出註標(index)。



【圖八】 Bit-Reverse 樹狀圖

為了讓輸入端的資料依照順序輸入接下來介紹分頻快速傅立葉轉換(DIF FFT)的演算法。

2.1.2 分頻快速傅立葉轉換(DIF FFT)演算法

以下為分頻快速傅立葉轉換演算法的過程，也可稱為基-2 分頻快速傅立葉轉換(radix-2 DIF FFT)演算法。

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \qquad , k = 0 \sim N-1$$

$$\textit{even} \quad \textit{term} : \quad X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)} \qquad , r = 0 \sim N/2 - 1$$

$$= \sum_{n=0}^{N/2-1} x[n] W_N^{2nr} + \sum_{n=N/2}^{N-1} x[n] W_N^{2nr}$$

$$= \sum_{n=0}^{N/2-1} x[n] W_N^{2nr} + \sum_{n=0}^{N/2-1} x[n + \frac{N}{2}] W_N^{2r(n+\frac{N}{2})}$$

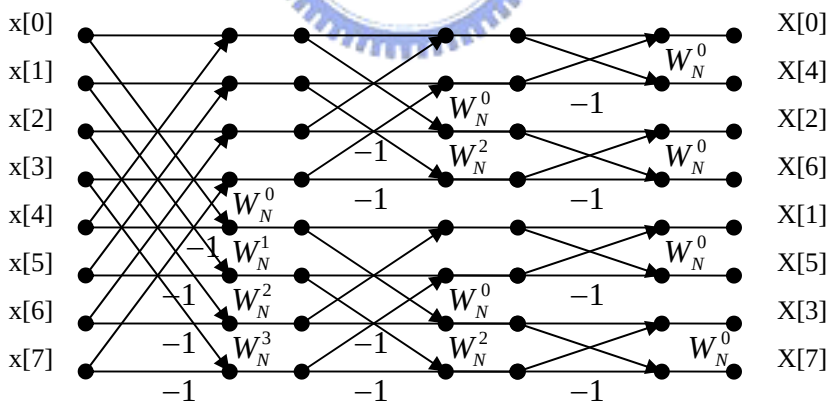
$$= \sum_{n=0}^{N/2-1} \{x[n] + x[n + \frac{N}{2}]\} W_{N/2}^{nr}$$

$$\textit{odd} \quad \textit{term} : \quad X[2r+1] = \sum_{n=0}^{N/2-1} x[n] W_N^{n(2r+1)} + \sum_{n=N/2}^{N-1} x[n] W_N^{n(2r+1)}$$

$$= \sum_{n=0}^{N/2-1} \{x[n] - x[n + \frac{N}{2}]\} W_N^{n(2r+1)}$$

$$= \sum_{n=0}^{N/2-1} \{x[n] - x[n + \frac{N}{2}]\} W_{N/2}^{nr} W_N^{n}$$

使用以上的演算法，可以得到以下的圖形(以八點為例)，見【圖九】：



【圖九】 八點的 DIF FFT butterfly 圖形

由此所有的輸入的資料皆可按照順序輸入，因此在輸入端資料流控制會比分時快速傅立葉轉換(DIT FFT)的控制容易。由以上的分析接下來討論其餘的快速傅立葉轉換演算法皆以分頻快速傅立葉轉換(DIF FFT)為基礎。

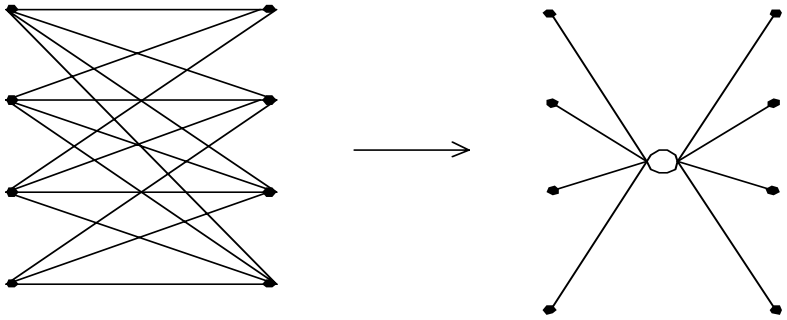
2.1.3 基-4 快速傅立葉轉換(Radix- 4 FFT)演算法

基-4 快速傅立葉轉換(radix-4 FFT)的演算法[15]其實主要是依基-2 快速傅立葉轉換(radix-2 FFT)所分出來的偶數輸出端部分與奇數輸出端部分再繼續以基-2 (radix-2)的方式再分，其演算法定義如下。

$$X[4k+l] = \sum_{n=0}^{N/4-1} [x[n] + x[n+\frac{N}{4}] \times W_4^l + x[n+\frac{N}{2}] \times W_4^{2l} + x[n+\frac{3N}{4}] \times W_4^{3l}] \cdot W_{N/8}^{nl} W_{N/4}^{nk}$$

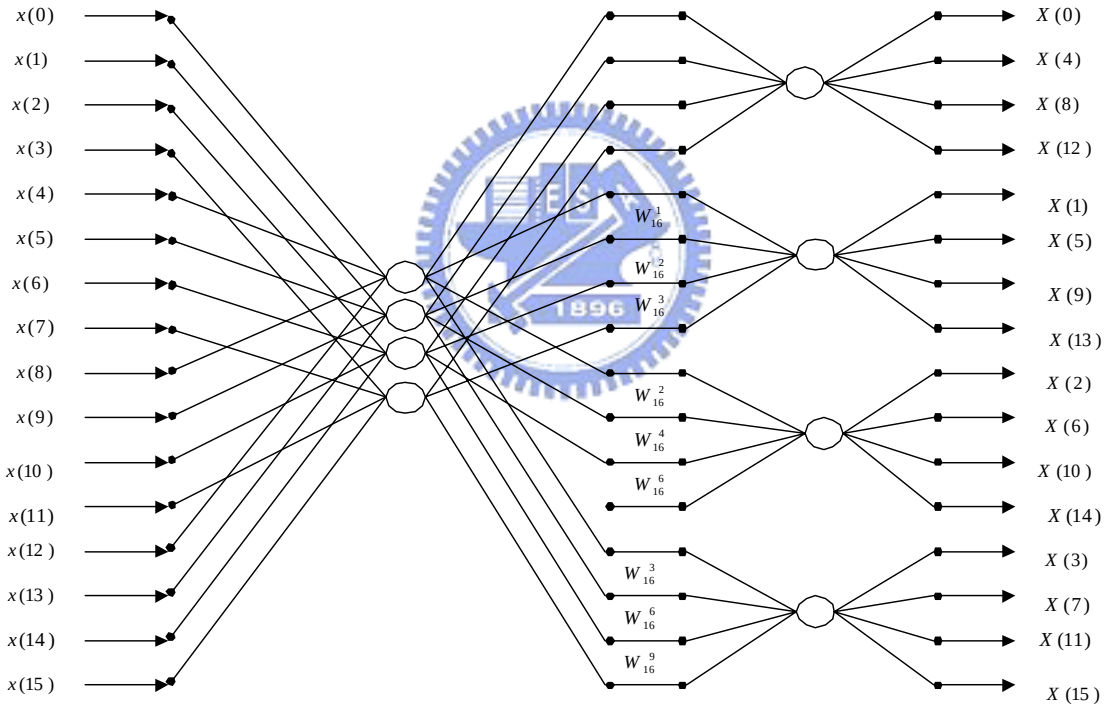
$$where \quad l = 0,1,2,3 \quad and \quad k = 0 \sim N/4-1$$

由此可發現能夠一次做好四點的運算，會比基-2 (radix-2)的運算快。但是基-4 (radix-4)的演算法只能適用於點數為四的指數。 而且其蝴蝶圖(butterfly)的形狀較基-2 (radix-2)的複雜，見【圖十】。



【圖 十】radix-4 butterfly 單元

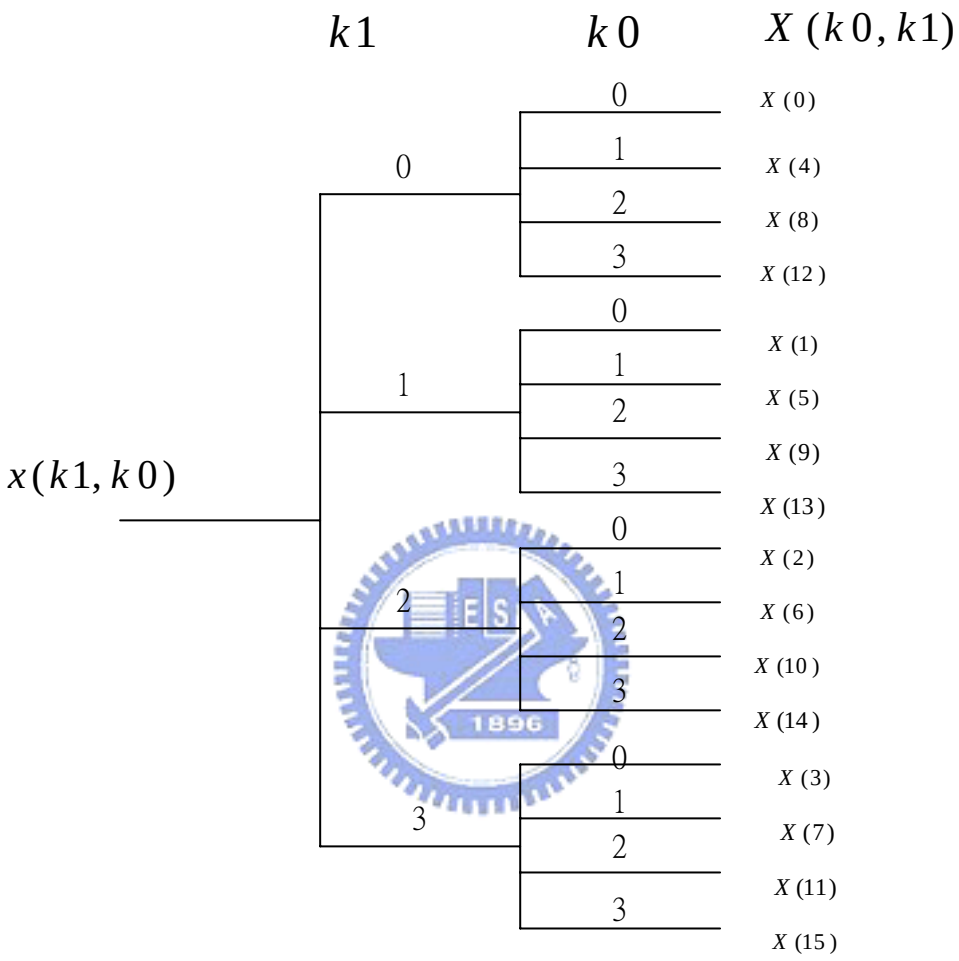
以 N=16 為例子，基-4(Radix-4) 的圖形如下：



【圖十一】 十六點的 radix-4 butterfly 圖形

由【圖十一】的輸出端發現，其輸出的順序並不像基-2(radix-2)一樣形成反向位元(bitreverse)，而形成反向數字(digitreverse)，以十

六點為例子同反向位元(bitreverse)產生的方式一樣，解釋記憶體的註標(index)如何變成 digitreverse，見【圖十二】。



【圖十二】 Digit-reverse 樹狀圖

因為輸出不為反向位元(bitreverse) 希望輸出能維持像基-2(radix-2)一樣的方式輸出，因此就產生了基-2^2(Radix-2^2)演算法。

2.1.4 基-2^2 快速傅立葉轉換(Radix-2^2 FFT)演算法

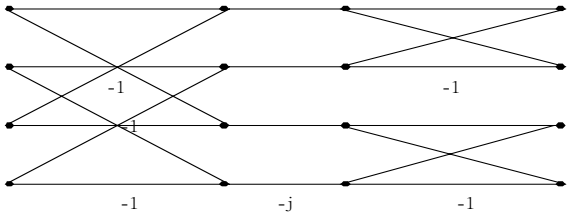
基-2^2 快速傅立葉轉換(radix-2^2 FFT)的演算法[16]與基-4(Radix-4)

快速傅立葉轉換的演算法類似只是其蝴蝶圖(butterfly)單元可以由兩個基-2(radix-2)的蝴蝶圖(butterfy)來實現。其演算法為：

$$
\begin{aligned}
&X(4k+2l_2+l_1)\\
&=\sum_{n=0}^{N/4-1}[x(n)+x(n+N/4)W_4^{2l_2+l_1}+x(n+N/2)W_4^{4l_2+2l_1}+x(n+3N/4)W_4^{6l_2+3l_1}]W_N^{n(2l_2+l_1)}W_{N/4}^{nk}\\
&=\sum_{n=0}^{N/4-1}\{[x(n)+(-1)^{l_1}x(n+N/2)]+(-1)^{l_2}(-j)^{l_1}[x(n+N/4)+(-1)^{l_1}x(n+3N/4)]\}W_N^{n(2l_2+l_1)}W_{N/4}^{nk}\\
&\quad where \quad l_1,l_2=0,1 \quad and \quad k=0\sim N/4-1
\end{aligned}
$$

在上式中最大的好處即能用兩個基-2(radix-2)的演算方式來實現基-2^2(Radix-2^2)的演算法，而且發現其中已經把含有 j 的係數項都合併在一起，相對於基-2(radix-2)快速傅立葉轉換的演算法，基-2^2(Radix-2^2)快速傅立葉轉換所要相乘的的交互因子(twiddle factor)會比較少，因此其乘法器的數目理論上會比較少。

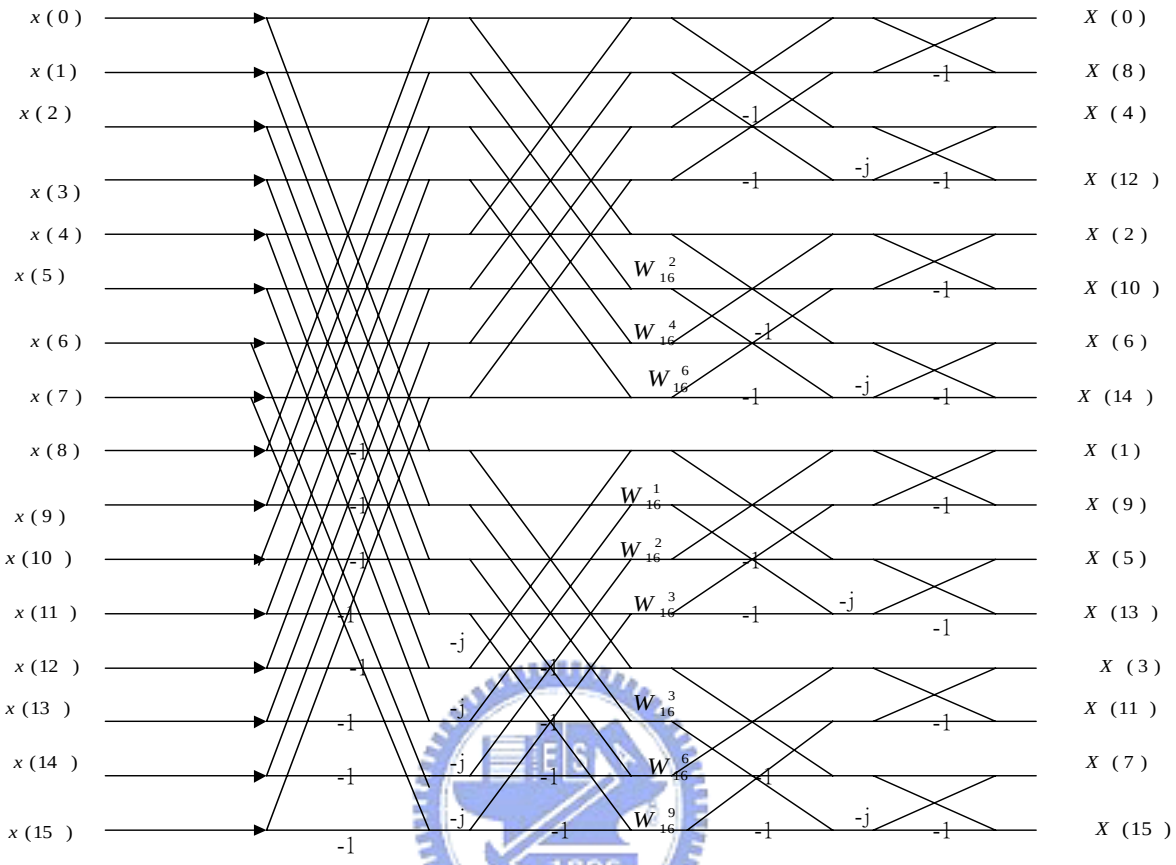
而其蝴蝶圖(butterfly)的元件會與之前介紹的基-4(radix-4)的不同，比基-4(radix-4)的圖形簡單，而且較易實現。見【圖十三】。



【圖十三】 R adix-2^2 FFT butterfly 單元

舉 N=16 點為例子，以下圖形即是基-2^2(Radix-2^2) 快速傅立葉轉

換的圖形 。



【圖十四】 十六點的 radix-2^2 butterfly 圖形

由【圖十一】與基-2(radix-2)的圖形來比較可以發現【圖十四】的交互因子(twiddle factor)會比較少。因此高基(radix)的演算法可以使真正所需要相乘的交互因子(twiddle factor) 減少，但是相對的高基(radix)的演算法其單一 PE(Processor Element)的控制與實現會比低基(radix)的實現複雜。因此接下來介紹基-8 快速傅立葉轉換(radix-8 FFT)的演算法。

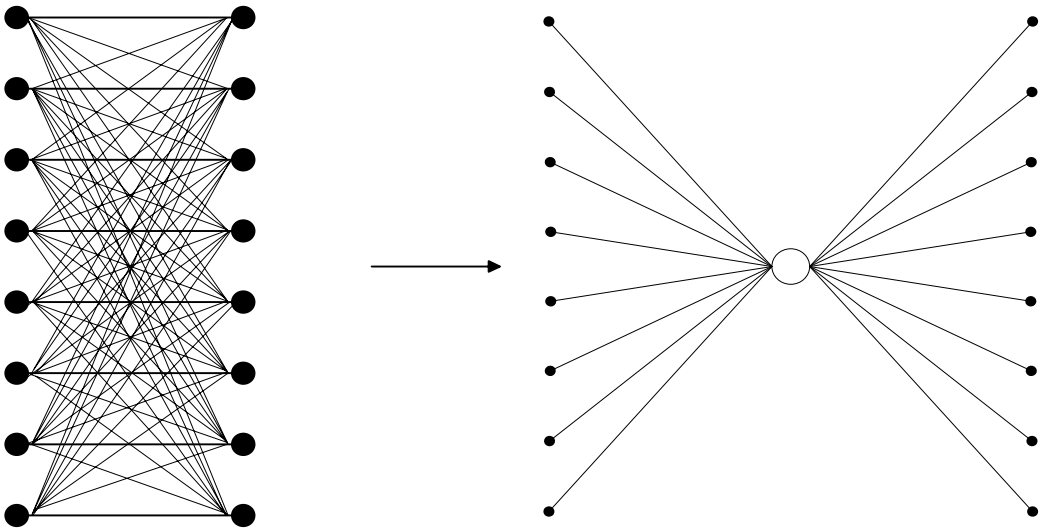
2.1.5 基-8 快速傅立葉轉換(radix-8 FFT)演算法

在介紹了基-2 快速傅立葉轉換(radix-2 FFT) 與基-4 快速傅立葉轉換(radix-4 FFT)之後，也能夠使用更高的基-8 快速傅立葉轉換(radix-8 FFT)的演算法去實現，以下為基-8 快速傅立葉轉換(radix-8 FFT)的演算法：

$$
\begin{aligned}
X(8k+l) &= \sum_{n=0}^{N-1} x(n) W_N^{(8k+l)n} \\
&= \sum_{m=0}^{7} \sum_{n=0}^{N/8-1} x\left(n+\frac{mN}{8}\right) W_N^{(8k+l)(mN/8+n)} \\
&= \sum_{m=0}^{7} \sum_{n=0}^{N/8-1} \left[x\left(n+\frac{mN}{8}\right) W_8^{lm} \right] W_N^{nl} W_{N/8}^{nk} \\
&= \sum_{n=0}^{N/8-1} \{ \left[x(n) + x(n+\frac{2N}{8}) W_4^{l} + x(n+\frac{4N}{8}) W_4^{2l} + x(n+\frac{6N}{8}) W_4^{-l} \right] \\
&\quad + \left[x(n+\frac{N}{8}) + x(n+\frac{3N}{8}) W_4^{l} + x(n+\frac{5N}{8}) W_4^{2l} + x(n+\frac{7N}{8}) W_4^{-l} \right] W_8^{l} \text{"} \} W_N^{nl} W_{N/8}^{nk}
\end{aligned}
$$

$$
l = 0 \sim 7 \qquad \textit{and} \qquad k = 0 \sim N/8 - 1
$$

由以上的演算法可得知基-8 快速傅立葉轉換(radix-8 FFT)的演算法只能適用於點數為 8 的指數。由以上的演算法將其圖形表示成基本的蝴蝶圖(butterfly)圖形表示之。



【圖十五】 Radix-8 FFT butterfly 單元

由基-8 快速傅立葉轉換(radix-8 FFT)的演算法所產生輸出端的結果排列順序會成 digit-reverse 的型態，同基-4 快速傅立葉轉換(radix-4 FFT)的演算法所說明之。

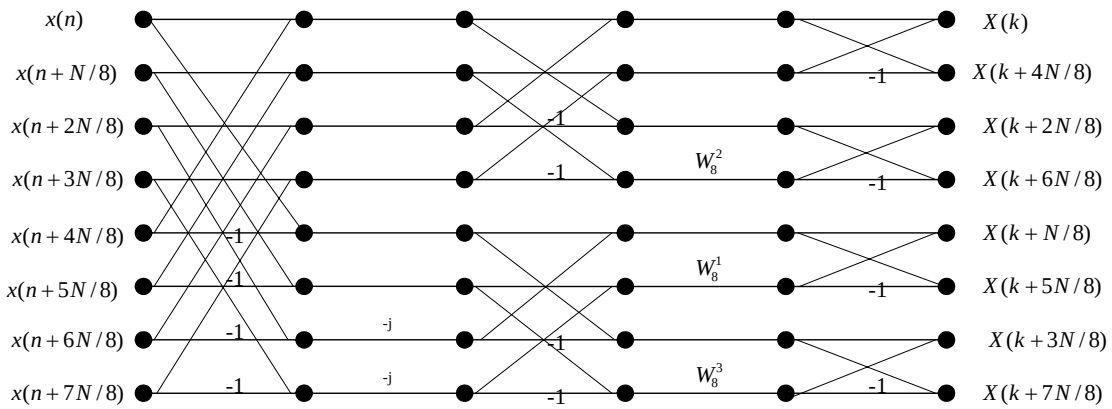
2.1.6 基-2^3 快速傅立葉轉換(Radix- 2^3 FFT)演算法

同樣使用三個基-2 (radix-2)去形成一個基-8 (radix-8)的方式，讓基-8 快速傅立葉轉換(radix-8 FFT)的演算法較容易用基-2(radix-2)的方式實現。其演算法為：

$$
\begin{aligned}
&X(8k+4l_3+2l_2+l_1)\\
&=\sum_{n=0}^{N/8-1}\{[x(n)+x(n+\frac{2N}{8})W_4^{l}+x(n+\frac{4N}{8})W_4^{2l}+x(n+\frac{6N}{8})W_4^{-l}]\\
&\quad+[x(n+\frac{N}{8})+x(n+\frac{3N}{8})W_4^{l}+x(n+\frac{5N}{8})W_4^{2l}+x(n+\frac{7N}{8})W_4^{-l}]W_8^{l}\}W_N^{nl}W_{N/8}^{nk}\\
&=\sum_{n=0}^{N/8-1}\{[(x(n)+W_2^{l_1}x(n+\frac{4N}{8}))+W_2^{l_2}W_4^{l_1}(x(n+\frac{2N}{8})+W_2^{l_1}x(n+\frac{6N}{8}))]\\
&\quad+[x(n+\frac{N}{8})+W_2^{l_1}x(n+\frac{5N}{8}))+W_2^{l_2}W_4^{l_1}(x(n+\frac{3N}{8})+W_2^{l_1}x(n+\frac{7N}{8}))]W_8^{2l_2+l_1}\}W_N^{n(4l_3+2l_2+l_1)}W_{N/8}^{nk}\\
&\quad \textit{where}\qquad l_1,l_2,l_3=0,1\qquad \textit{and}\qquad k=0\sim N/8-1
\end{aligned}
$$

由上面的演算法能夠把$W_8^1=\frac{\sqrt{2}}{2}(1-j)$和$W_8^3=\frac{\sqrt{2}}{2}(1+j)$合併在一起，而且這兩個交互因子(twiddle factor) 並不需要使用到複數乘法器，只需要使用實數乘法器的運算，因此單一處理單元(PE)的實現上的面積不會太大。但是如果想使用更高基(radix)的方式去實現單一處理單元(PE)，面積與控制方法都會比較不利，所以對於高基(radix)， 基-8(radix-8)是最高的考量。

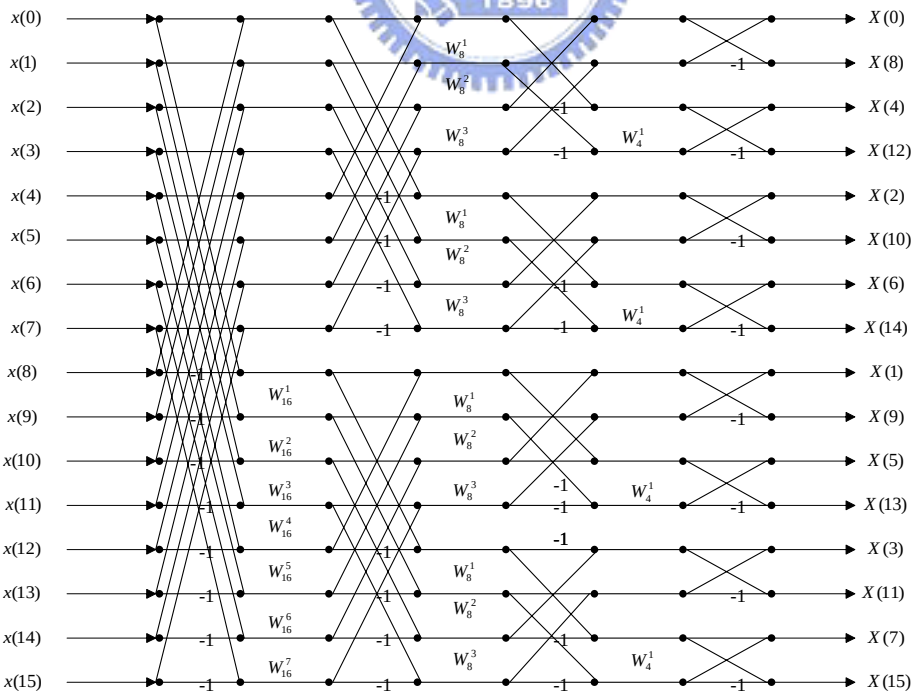
其基-2^3快速傅立葉轉換(Radix- 2^3 FFT)蝴蝶圖(butterfly)基本形狀為：



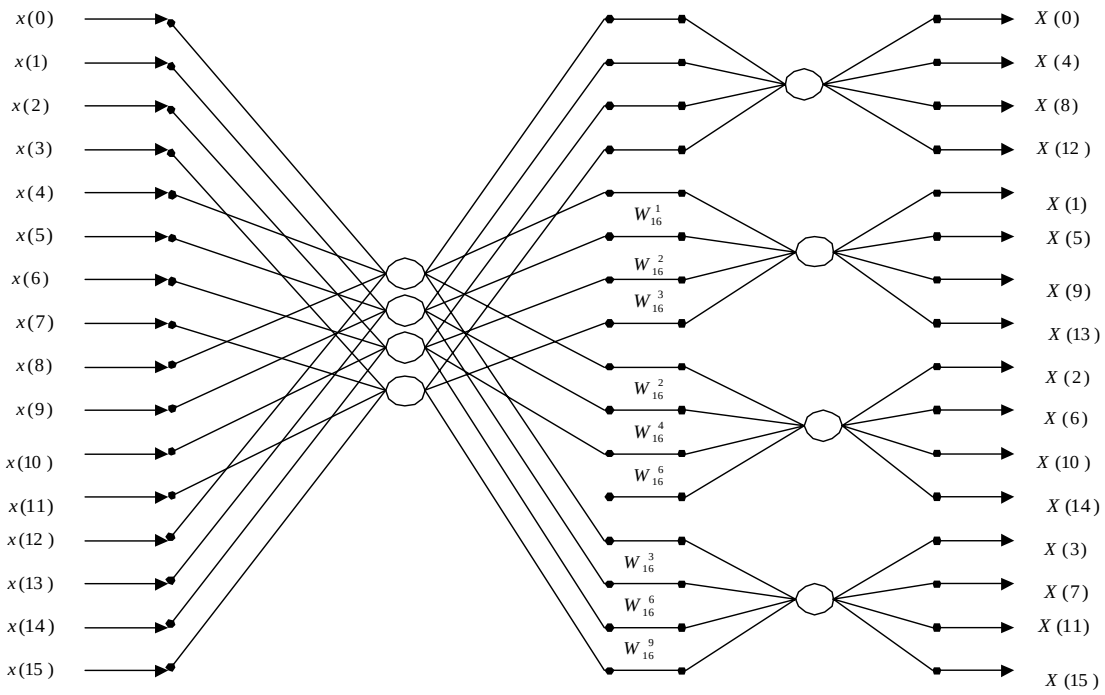
【圖十六】 Radix-2^3 FFT butterfly 單元

2.1.7 分割-基數 24 快速傅立葉轉換(Split-Radix24 FFT)演算法

分割-基數快速傅立葉轉換(Split-Radix FFT) [17]是對於交互因子(twiddle factor) 去做不同的分解，選擇最好的分解方式把運算複雜度降至最低。通常選擇分割-基數 24 快速傅立葉轉換(Split-Radix24 FFT)的演算法，因為基-2(radix-2)與基-4(radix-4)是比較常使用的演算法，因此選擇以此兩種混和的方式來形成分割-基數 24 快速傅立葉轉換(Split-Radix24 FFT)。以下先由十六點的基-2(radix-2)蝴蝶圖(butterfly)及基-4(radix-4)蝴蝶圖(butterfly)說明其如何演變成分割-基數(Split-Radix)。十六點的基-2(radix-2)蝴蝶圖(butterfly)見【圖十七】。十六點的基-2(radix-2)蝴蝶圖(butterfly)見【圖十八】。



【圖十七】 十六點的 radix-2 butterfly 圖形



【圖十八】十六點的 radix-4 butterfly 圖形

分割-基數 24 快速傅立葉轉換(Split-Radix24 FFT)是利用相同的方式使用基-2 (radix-2)與基-4 (radix-4)的演算法去擴充，由以上的基-2 快速傅立葉轉換(radix-2 FFT)圖形觀察之，可以發現偶數的輸出端其交互因子(twiddle factor)皆為 1，所以在偶數的輸出端不需乘法器 ，但在基-4(Radix-4)的圖形終究沒有像基-2 (radix-2)一樣的規則性，見【圖十八】。但是基-4 快速傅立葉轉換(radix-4 FFT) 卻比基-2 快速傅立葉轉換(radix-2 FFT)的乘法複雜度低。從以上的觀點來看，對奇數輸出端與偶數輸出端使用不同的基底可以有效的降低複數乘法與加法的的數目。因此對於偶數點的資料輸入使用基- 2(radix-2)的演算法：

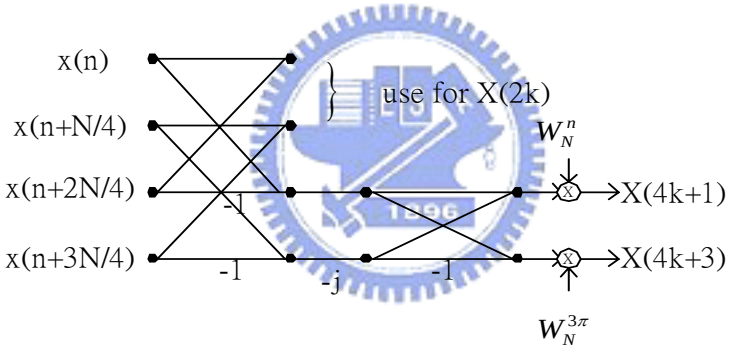
$$X(2k)=\sum_{n=0}^{N/2-1}(x[n]+x[n+\frac{N}{2}])W_{N/2}^{n} \qquad where \quad k=0\sim N/2-1$$

而對於奇數點的輸出端使用基-4(radix-4)的演算法：

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

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

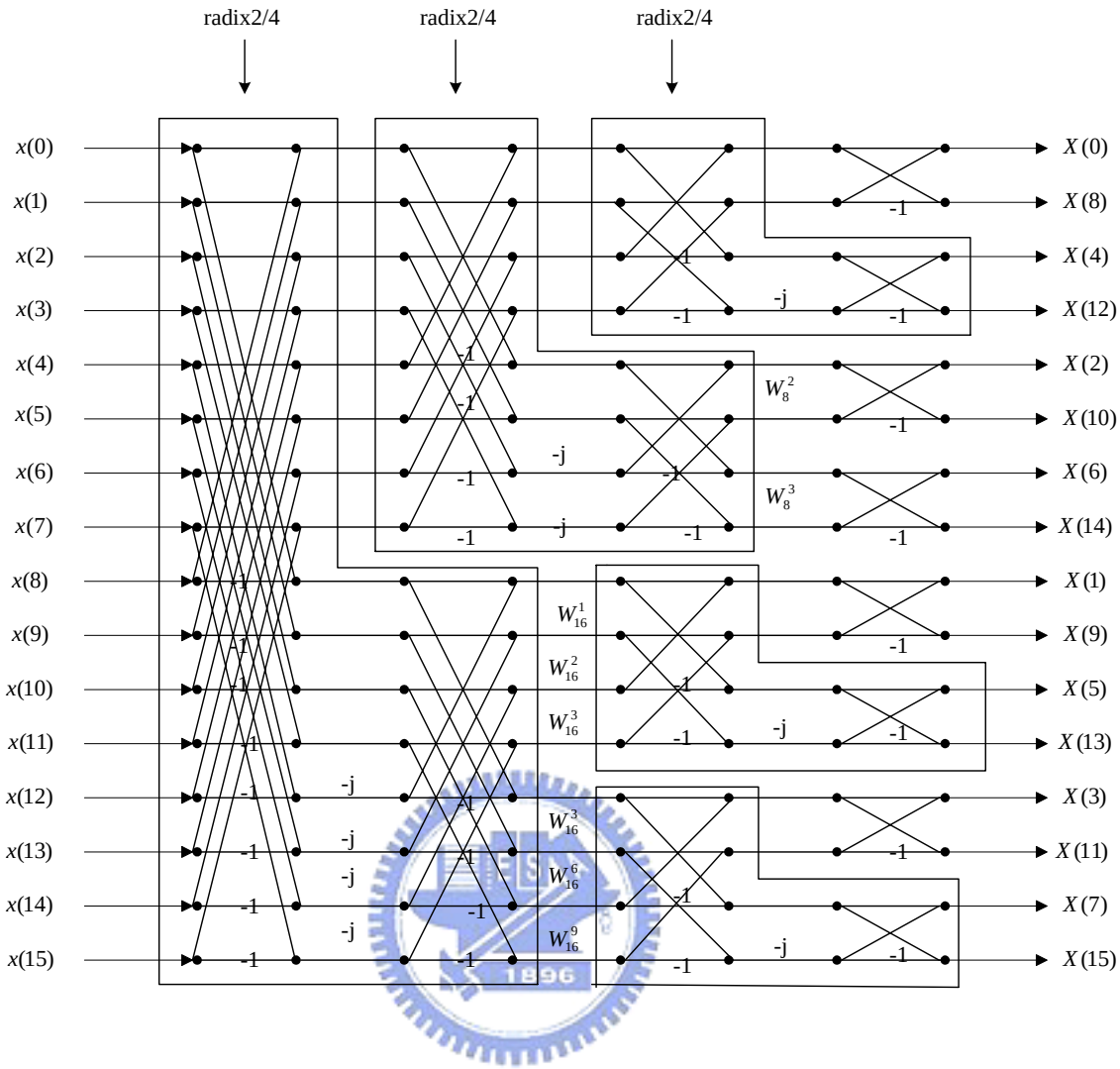
根據以上的演算法夠畫出分割-基數(SRFFT)蝴蝶圖 (butterfly)的基本單元，其圖形會形成 L 的形狀，見【圖十九】:



【圖十九】 split-radix2/4 FFT butterfly 單元

由基本單元來看就可發現其圖形並不規則。

以下用 N 為十六點為例子展開而成的分割-基數(SRFFT)圖形如下：



【圖二十】 十六點的 split-radix2/4 butterfly 圖形

雖然分割-基數(SRFFT)可以減少乘法的複雜度，使運算量可以減少，但因為其輸出端並不能很規則直接用基-2(radix-2)或基-4(radix-4)的處理器單元(PE)去實現。因此必須為此分割-基數(SRFFT)得演算法特別去想一套硬體來實現。因為基-r(radix-r)的選擇當中發現基-8(radix-8)為其最高基(radix)的選擇，因此分割-基數 24 快速傅立葉轉換(SR24FFT)也能擴充成分割-基數 28 快速傅立葉轉換(SR28FFT)。

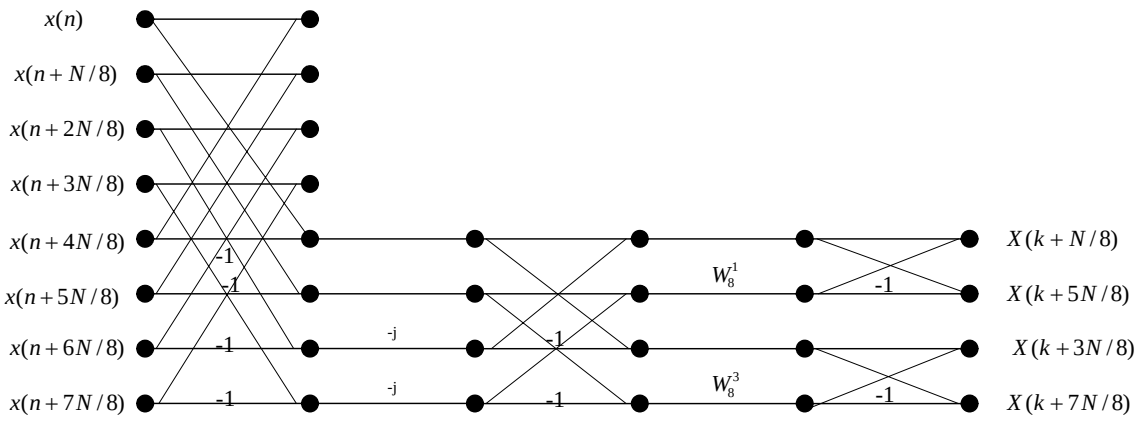
2.1.8 分割-基數 28 快速傅立葉轉換(Split-Radix2/8 FFT)演算法

依同樣的方式把偶數點使用基-2(radix-2)的演算法，而奇數點使用基-8(radix-8)的演算法去形成分割-基數 28 快速傅立葉轉換(Split-Radix2/8 FFT)的演算法：

$$\begin{cases} X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n+\frac{2N}{4})]W_N^{2nk} \\ X(8k_1 + l) = \sum_{n=0}^{N/8-1} \{[x(n) + x(n+\frac{2N}{8})W_4^l + x(n+\frac{4N}{8})W_4^{2l} + x(n+\frac{6N}{8})W_4^{-l}] \\ \qquad + [x(n+\frac{N}{8}) + x(n+\frac{3N}{8})W_4^l + x(n+\frac{5N}{8})W_4^{2l} + x(n+\frac{7N}{8})W_4^{-l}]W_8^l\}W_N^{nl}W_{N/8}^{nk_1} \end{cases}$$

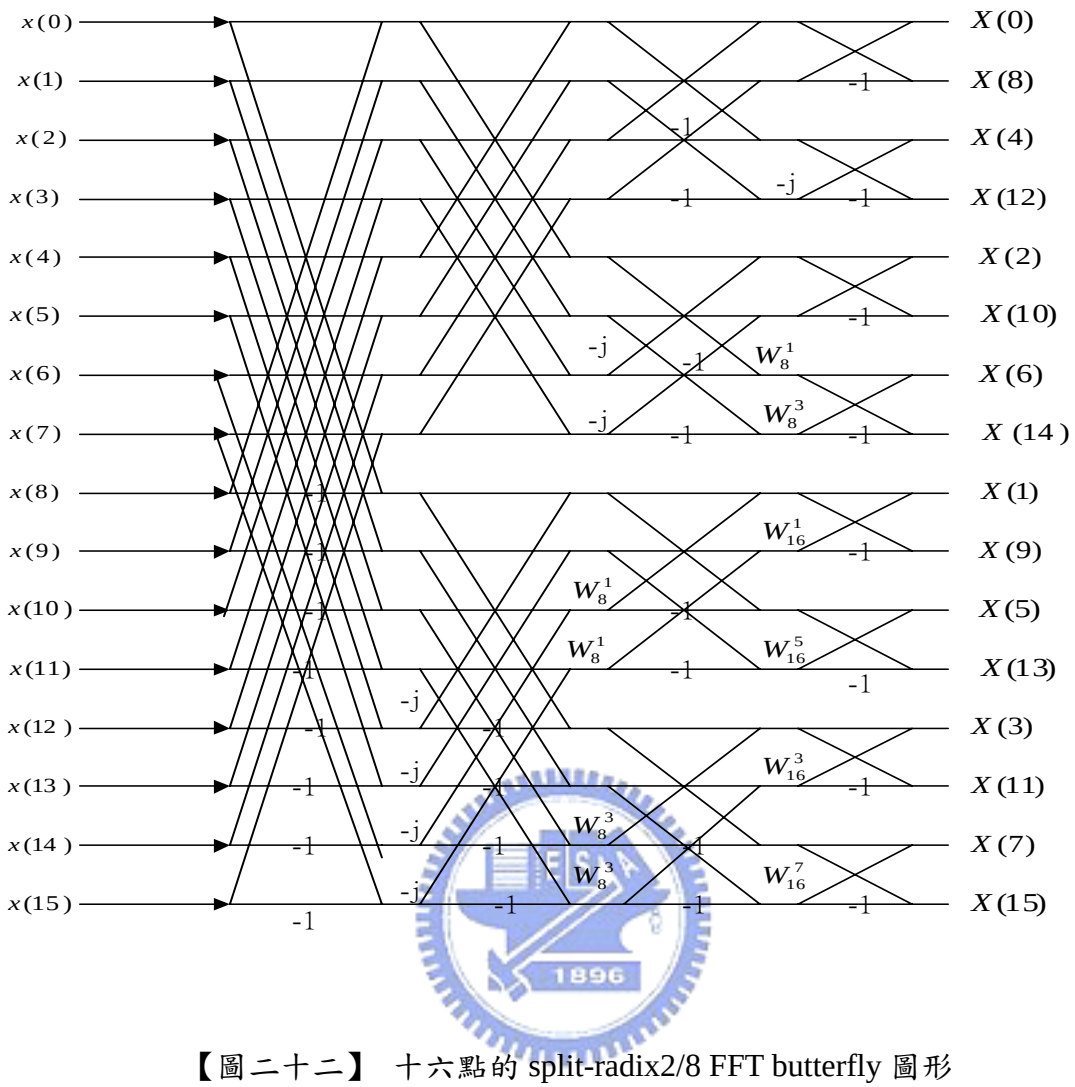
where $l = 1,3,5,7$, $k = 0 \sim N/2-1$ *and* $k_1 = 0 \sim N/8-1$

由以上的演算法也可發現其中也把奇數點的 $W_8^1 = \frac{\sqrt{2}}{2}(1-j)$ 和 $W_8^3 = \frac{\sqrt{2}}{2}(1+j)$ 合併在一起，因此其乘法器的運算量會降低。由以上的演算法可以形成下圖的蝴蝶圖(butterfly)的單元。



【圖二十一】 split-radix2/8 FFT butterfly 單元

取 N=16 為例子，其圖形如下。



【圖二十二】 十六點的 split-radix2/8 FFT butterfly 圖形

2.2 快速傅立葉轉換(FFT) 之實現架構

快速傅立葉轉換(FFT)的硬體架構中最快速的運算的硬體架構，主要是管線(Pipeline)的硬體架構，因此針對此做介紹。

對於管線(Pipeline)的架構當中，每一級的單元處理器(PE)皆相同，不同的是暫存器數目會成級數增加。輸入端的資料會依照順序進入，並且在硬體實現中，其產出率(througtput)高。

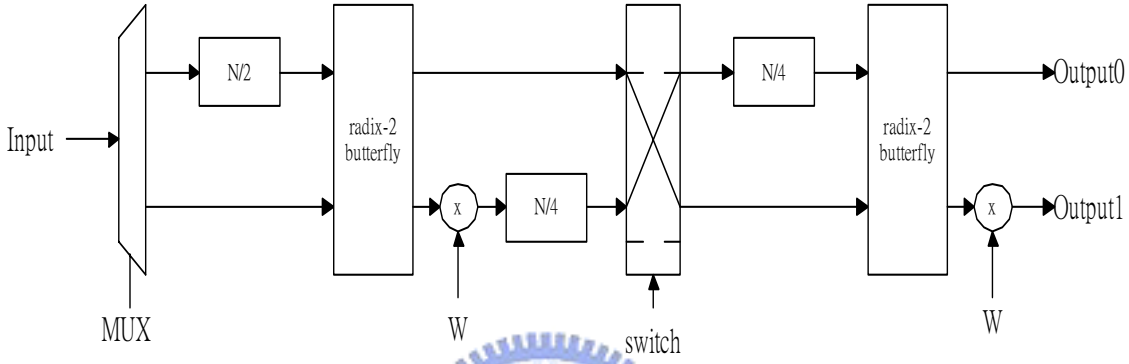
管線(Pipeline)快速傅立葉轉換(FFT)硬體架構可分為：

1. 基-2 多路徑延遲換向器(Radix-2 Multipath delay commutator ,R2 MDC)[21]
2. 基-2 單路徑延遲迴授器(Radix-2 single-path delay feedback ,R2SD F)[22]
3. 基-4 多路徑延遲換向器(Radix-4 Multipath delay commutator ,R4 MDC)[21]
4. 基-2^2 多路徑延遲換向器(Radix-2^2 Multipath delay commutator ,R 2^2MDC)[16]
5. 基-4 單路徑延遲迴授器(Radix-4 single-path delay feedback ,R4SD F)[23][24]
6. 基-2^2 單路徑延遲迴授器(Radix-2^2 single-path delay feedback ,R2^2 SDF)[16]
7. 基-8 多路徑延遲換向器(Radix-8 Multipath delay commutator ,R8 MDC)
8. 基-2^3 多路徑延遲換向器(Radix-2^3Multipath delay commutator ,R2^3 MDC)[16]
9. 基-8 單路徑延遲迴授器(Radix-8 single-path delay feedback ,R8SD F)
10. 基-2^3 單路徑延遲換向器(Radix-2^3 single-path delay commutator , R2^3SDF)[16]
11. 分割基數(Split-radix)架構 [18]

接下來分別仔細介紹以上之架構。

2.2.1 基-2 多路徑延遲換向器(Radix-2 Multipath delay commutator ,R2MDC)硬體架構

此硬體架構為最直接去實現基-2 快速傅立葉轉換(radix-2 FFT)管線(pipeline)的硬體架構，其硬體架構，見【圖二十三】。



【圖二十三】R2MDC 硬體架構

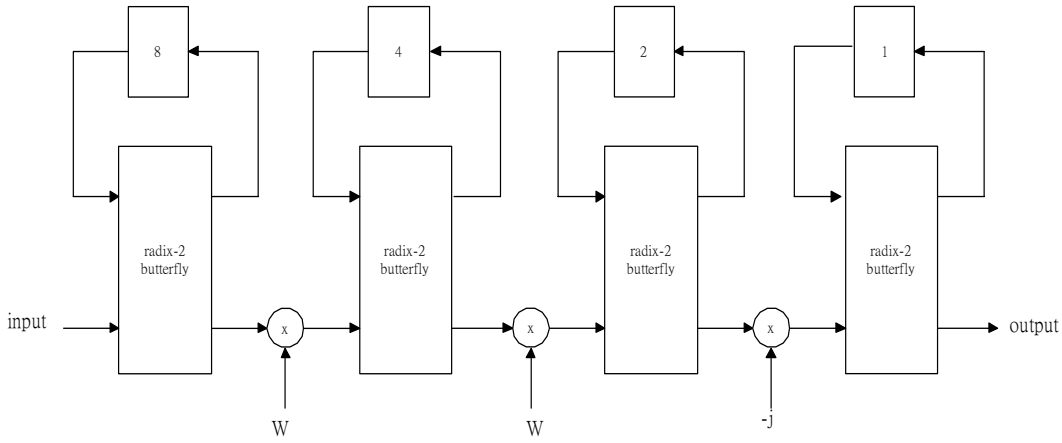
當輸入端的資料先經由上端的暫存器進入，直到 N/2 各資料都進入後，多工器就選擇下端讓第 N/2+1 筆資料進入，因此在第一級的蝴蝶圖(butterfly)的輸入端資料為第一筆與第 N/2+1 筆資料作運算，而其加法的結果選擇走上端，減法的結果走下端輸出與交互因子(twiddle factor)相乘之後又存入下端 N/4 的暫存器中同時又把第一級加法的輸出端先存入上端的 N/4 的暫存器中，接下來第一級加法再產生的結果開關切換至下端路線走，因此第一級的第一筆資料會與第一級的第 N/4+1 筆資料同時進入第二級的蝴蝶圖(butterfly)做運算，在此同時第一級的第一筆減法結果也會經由開關存入上端的

N/4 的暫存器中，以此同樣的運算方式，直至所有點數的資料都運算完畢。

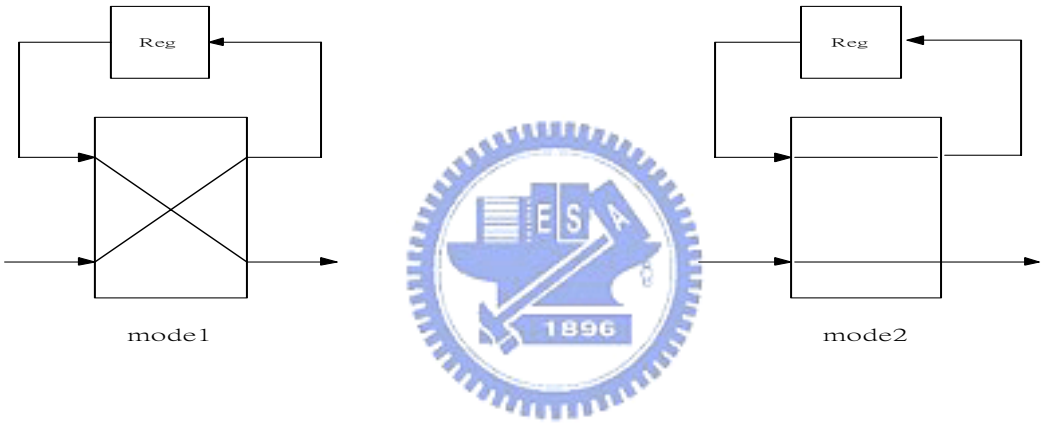
在此架構中可以發現，在蝴蝶圖(butterfly)和乘法器要做運算時會有一半的時間去等輸入端的資料進入，因此在乘法器與蝴蝶圖(butterfly)的使用率只有 50%。對於 8 點的 FFT 所需使用的暫存器 4+2+2+1+1=10 個。當做 N 點的 DFT 時需要 N/2+N/2+N/4+N/4+…＋2=3N/2 -2 個暫存器。在乘法器當中，因為每一級都需要一個乘法器，所以總乘法器為 $\log_2(N)-1$。

2.2.2 基-2 單路徑延遲迴授器(Radix-2 single-path delay feedback ,R-2SDF)硬體架構

為了要降低暫存器的數目，所以將每一級要輸出端的暫存器迴授回來成為每一級的輸入端使用，此種硬體架構我們稱為基-2 單路徑延遲迴授器(Radix-2 single-path delay feedback ,R2SDF)，取 N=16 點為例子，其圖形如下。



【圖二十四】 十六點 R2SDF 硬體架構



【圖二十五】 R2SDF 的 butterfly 單元的模式

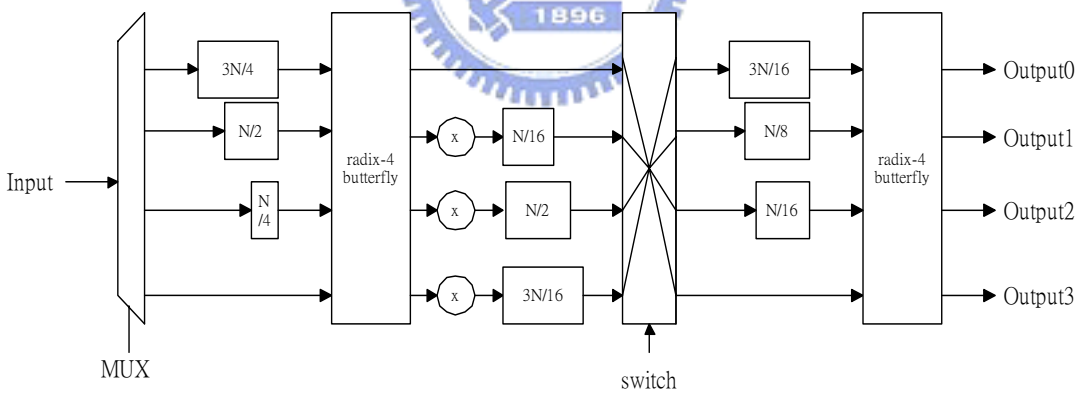
在一開始時先把 8 筆資料先存入第一級的暫存器中，等第 9 筆資料進入時再開始與第一筆資料同時進入第一級的蝴蝶圖(butterfly)中進行運算，再把算出加法結果傳至下一級的暫存器中，而減法的結果再存回同一級的暫存器中，因此此蝴蝶圖(butterfly)提供了一個完整的迴授迴圈。

雖然在暫存器上面把原本的 3N/2 -2 個暫存器降低成 N-1 個暫存器，但是乘法器還是不變為 $\log_2(N)-1$，因為每一級都需要一個乘

法器。且乘法器與蝴蝶(butterfly)的使用率也是為 50%。

2.2.3 基-4 多路徑延遲換向器(Radix-4 Multipath delay Commutator ,R4MDC) 硬體架構

基-4 多路徑延遲換向器(Radix-4 Multipath delay commutator ,R4MDC)硬體架構與基-2 多路徑延遲換向器(Radix-2 Multipath delay commutator ,R2MDC)的架構類似，只是輸入端的多工器變成 4-1 的多工器，而且在第一級有 3N/2 個暫存器，且在兩各級之間使用 4 條路徑的換向器(commutator)。因此在乘法器與蝴蝶(butterfly)的使用率上會變成 25%。因為所使用的為基-4(radix-4)所以只能適用於點數為 4 的 n 次方的點數。其硬體架構，見【圖二十六】。



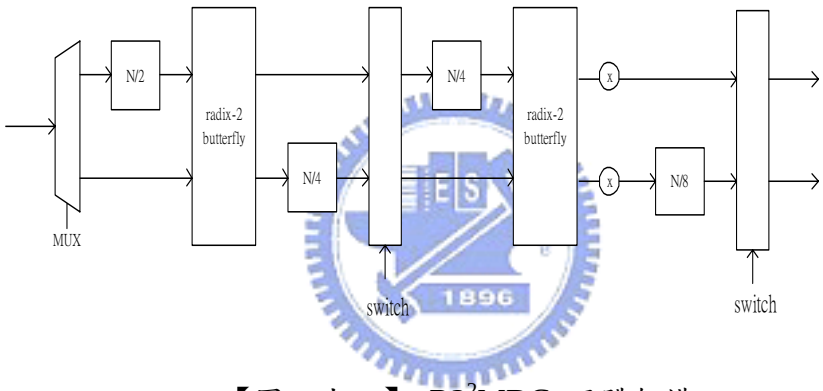
【圖二十六】 R4MDC 硬體架構

由以上的架構可以看出每一級皆需要 3 個乘法器，對於 N 點 DFT 需要 $3(\log_4(N)-1)$，由此可知其所需要的乘法器比 R2MDC 和 R2SDF 的多。而 R4MDC 需要 5N/2-4 個暫存器，也比之前所討論的

架構多。所以此架構並不如之前所討論的好。

2.2.4 基-2^2 多路徑延遲換向器(Radix-2^2 Multipath delay commutator,R2^2MDC)硬體架構

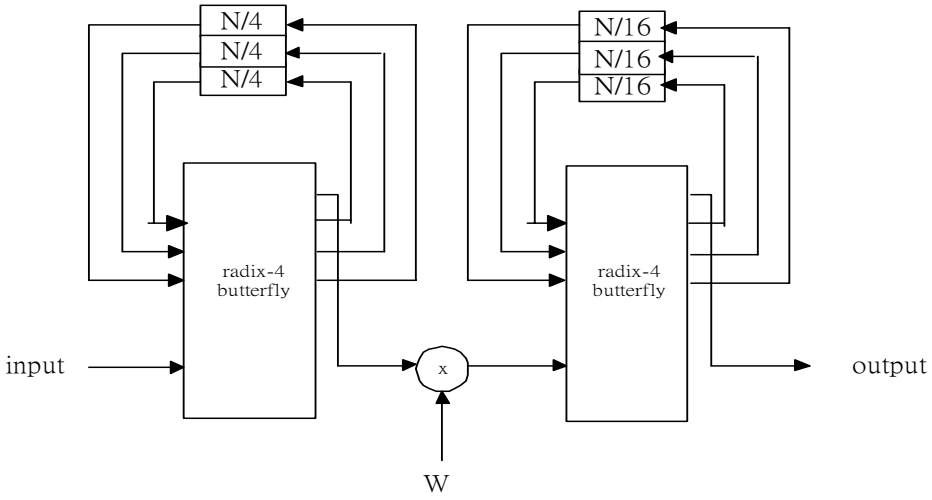
主要是為了把基-4(radix-4) 的硬體架構，使用兩個基-2(radix-2)的架構去實現，在每級 radix-2^2 需要兩個乘法法器總共需要 $\log_2$N-2 個，暫存器總共需要 3N/2-2 個，比 R4MDC 好。



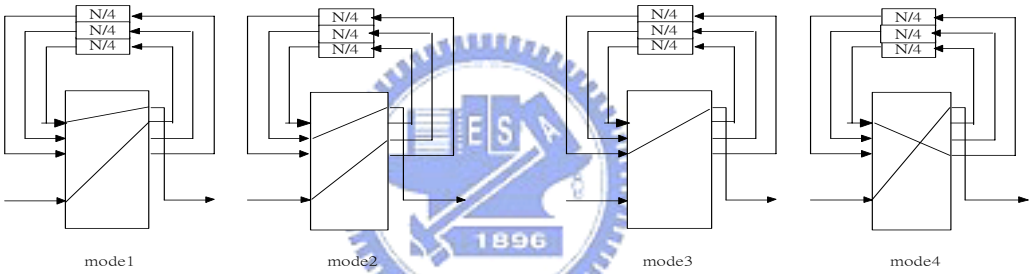
【圖二十七】 R2^2MDC 硬體架構

2.2.5 基-4 單路徑延遲迴授器(Radix-4 single-path delay feedback ,R-4SDF)硬體架構

R4SDF 的硬體架構與 R2SDF 的架構也類似，只是在於使用 Radix-4 的演算法時，可以發現 radix-4 的乘法使用會比 radix-2 的少，其數字可由 $\log_2$(N)-1 降低至 $\log_4$(N)-1 但同時蝴蝶圖(butterfly)與乘法的使用率會降低至 25%，而且 radix-4 的蝴蝶圖(butterfly)設計的複雜度也會比 radix-2 複雜一點。下圖即為 R4SDF 的硬體架構。



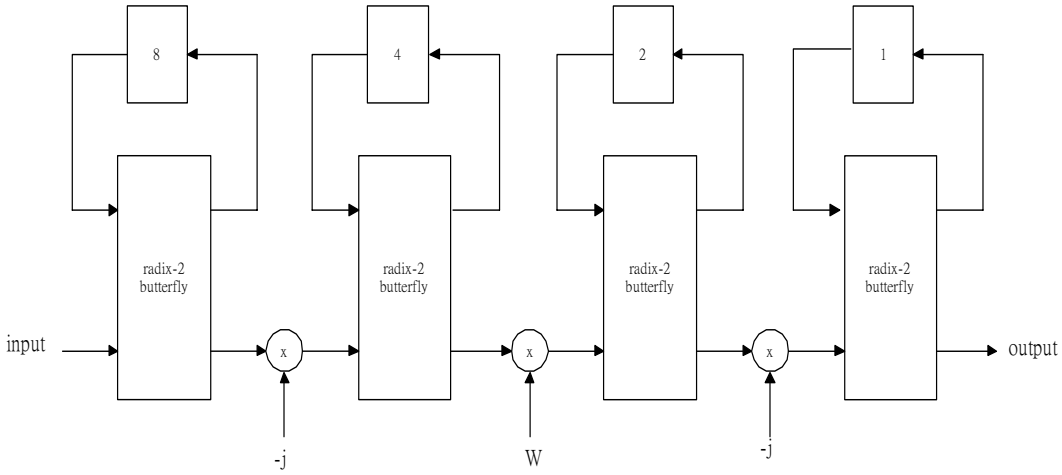
【圖二十八】 R4SDF 的硬體架構



【圖二十九】 R4SDF butterfly 單元的型態

2.2.6 基-2^2 單路徑延遲迴授器(Radix-2^2 single-path delay feedback,R2^2SDF) 硬體架構

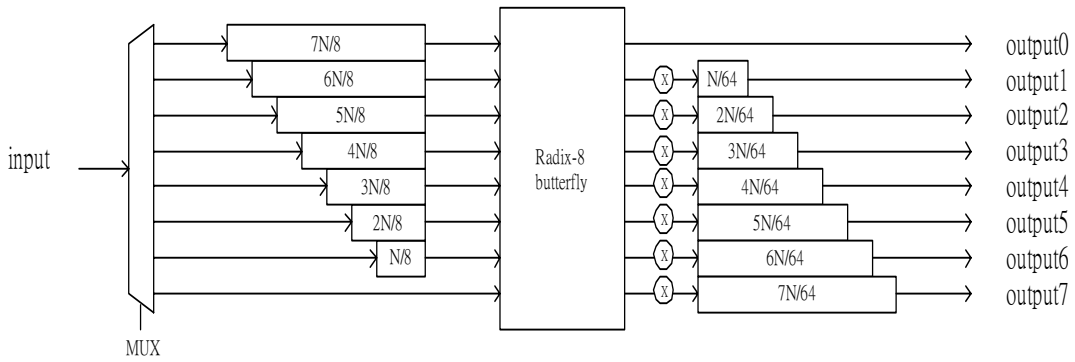
同樣的道理望能夠使用兩個基-2(radix-2)的蝴蝶圖(butterfly)去實現一個基-4(radix-4)的蝴蝶圖(butterfly)，其暫存器與乘法器與R4SDF 硬體架構相同，其硬體架構如下：



【圖三十】十六點 R2^{2}SDF 硬體架構

2.2.7 基-8 多路徑延遲換向器(Radix-8 Multipath delay commutator ,R8MDC)硬體架構

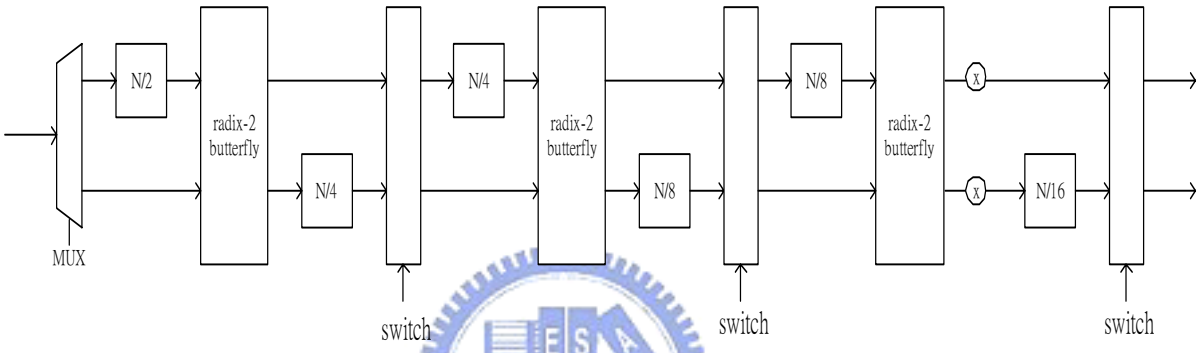
其架構與之前的 R2MDC 不同的是在於蝴蝶圖(butterfly)的不同，使用的為基-8(Radix-8)的蝴蝶圖(butterfly)，而且第一筆資料必須先經過 7N/8 的暫存器，才會進入蝴蝶圖(butterfly)去運算。需要乘法器 7($\log_2$(N)/3-1)，暫存器個數 9N/2-8 個。



【圖三十一】 R8MDC 硬體架構

2.2.8 基-2^3 多路徑延遲換向器(Radix-2^3Multipath delay commutator ,R2^3MDC)硬體架構

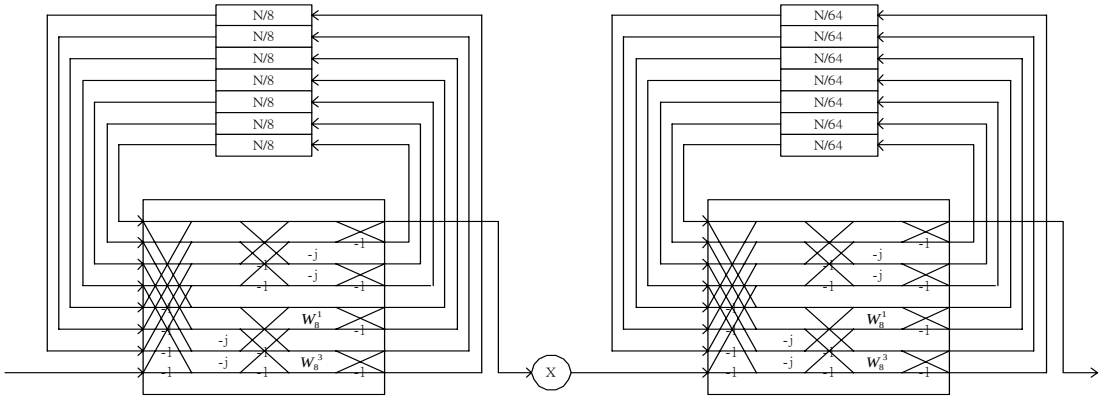
其硬體架構也是用三個基-2(radix-2)的蝴蝶圖(butterfly)來實現一個基-8(radix-8)的蝴蝶圖(butterfly)，讓乘法器能夠減少至$2(\log_2(N)/3-1)$，暫存器個數減少至(3N/2-2) 個。



【圖三十二】 R2^3 MDC 硬體架構

2.2.9 基-8 單路徑延遲迴授器(Radix-8 single-path delay feedback ,R8SDF)硬體架構

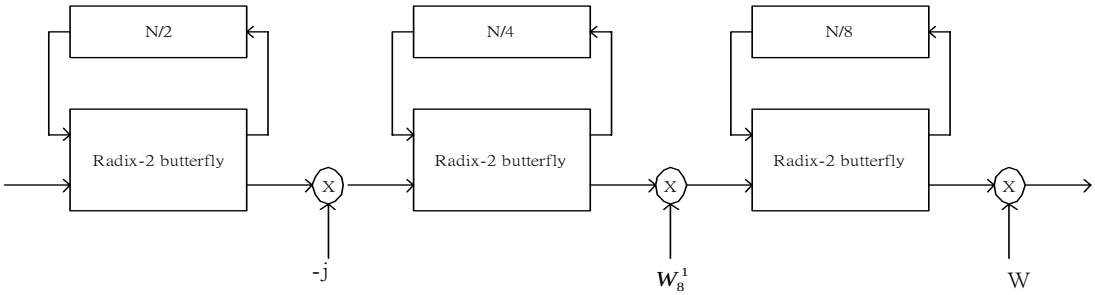
其硬體架構也是用一基-8(radix-8)的蝴蝶圖(butterfly)來實現一個 PE，讓乘法器能夠減少至 $\log_8(N)$-1 暫存器個數減少至(N-1) 個。



【圖三十三】 R8 SDF 硬體架構

2.2.10 基-2^3 單路徑延遲換向器(Radix-2^3 single-path delay commutator,R23SDF)硬體架構

其硬體架構也是用三個基-2(radix-2)的蝴蝶圖(butterfly)來實現一個基-8(radix-8)的蝴蝶圖(butterfly)，乘法器至 $\log_2(N)/3-1$ 暫存器個數減少至 (N-1) 個



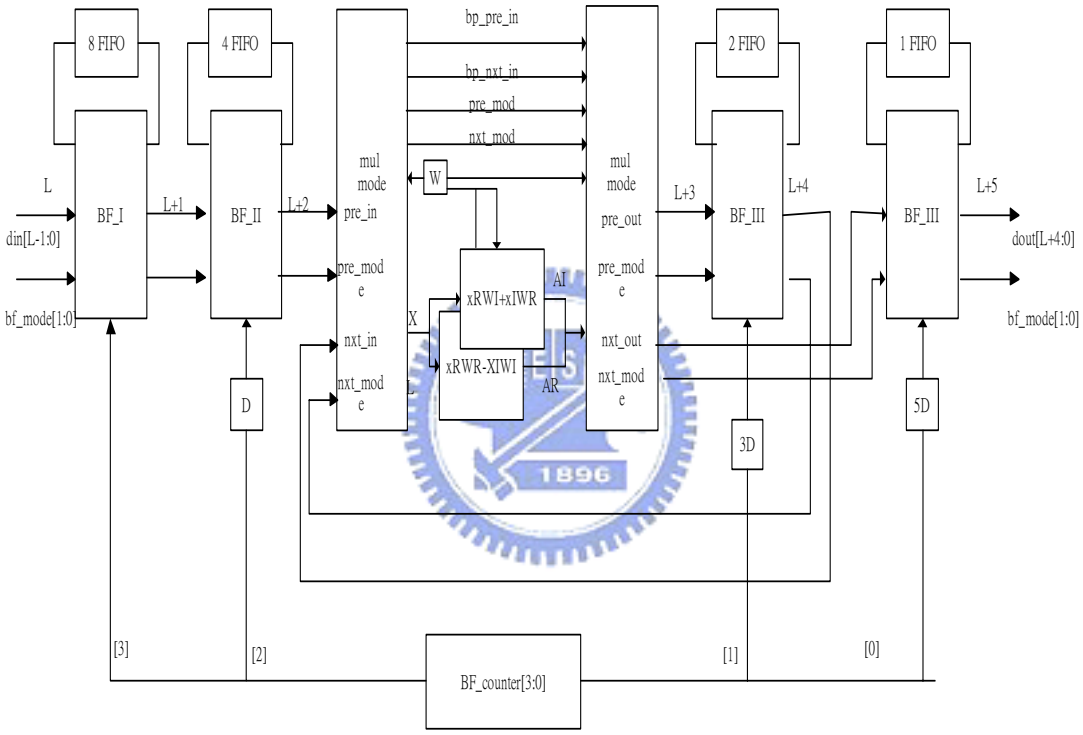
【圖三十四】 $R2^3$ SDF 硬體架構

以上四個硬體架構是跟基-4(radix-4)的介紹相同，不同的只是蝴蝶圖(butterfly)的實現不同，但是相對的階數(order)變高了，乘法的

使用率就會降低。

2.2.11 分割基數(Split-radix)硬體架構

參考[18]，其硬體架構如【圖三十五】:



【圖三十五】 split-radix 2/4 硬體架構

因為分割基數 24(Split-radix2/4)的蝴蝶圖(butterfly)圖形非常的不規則，在此硬體架構下，為了能夠有效的實現在此硬體架構中，使用了三種不同型態的蝴蝶圖(butterfly)當作運算單元，且在乘法器

的地方使用了共享的機制，但為了要能夠共享乘法器，必須在使每一級之間的排列順序重新排程不能用反向位元(bit-reverse)的方式輸出，但因為乘法的運算複雜度降低，在功率的消耗上有所改良。

2.3 結論與比較：

對於各種不同的基-r 快速傅立葉轉換(radix-r FFT)與分割-基數快速傅立葉轉換(split-radix FFT)可以分為幾個部分來比較與討論：

1. 複數乘法的複雜度。
2. 運算的速度。
3. 點數的限制。
4. 蝴蝶圖(butterfly)的規則性。

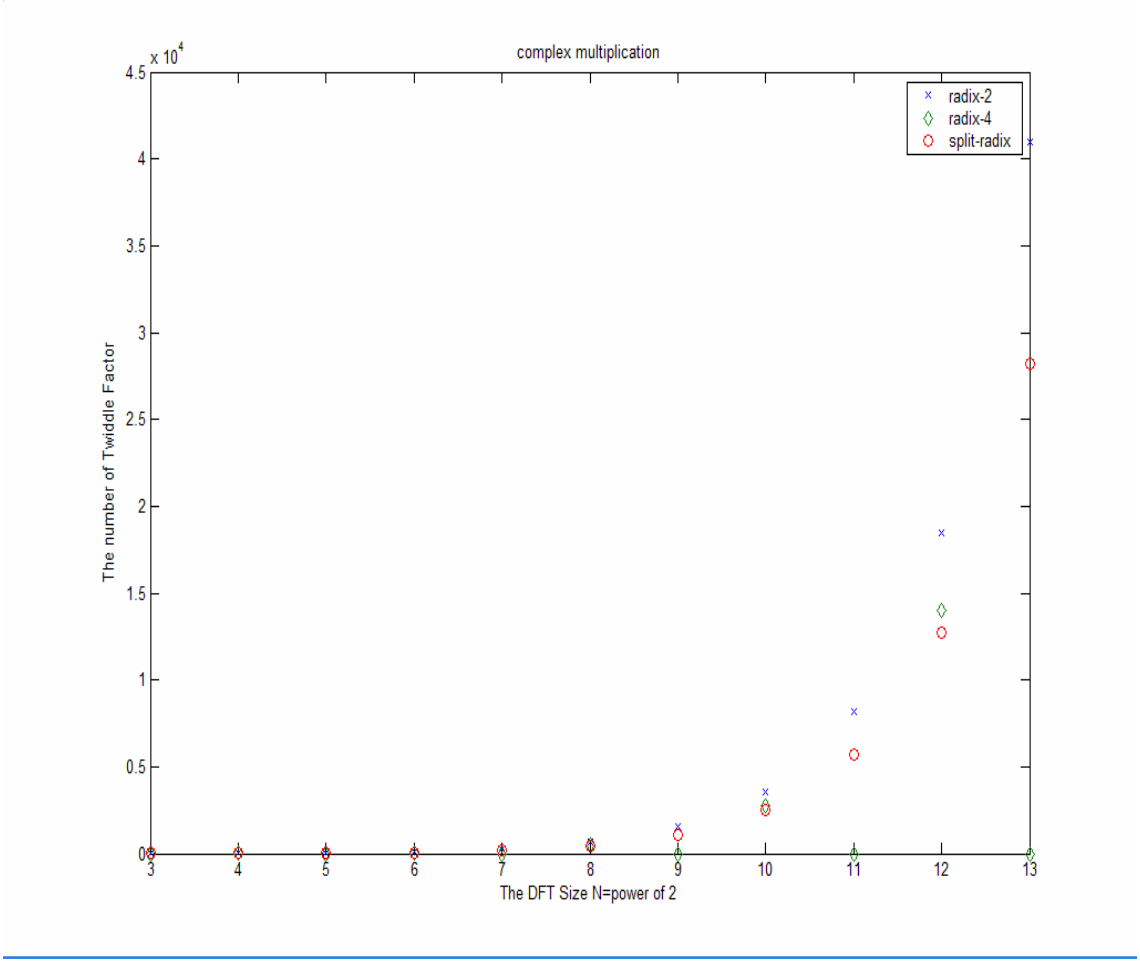
a. 複數乘法的複雜度

以複數的乘法器的複雜度來看我們能參考[18]，單位為交互因子的個數，見【表一】：

DFT size	Radix-2	Radix-4	Split-Radix
8	2		2
16	10	8	8
32	34		26
64	98	76	72
128	258		186
256	642	492	456
512	1538		1082
1024	3586	2732	2504
2048	8194		5690
4096	18434	13996	12744
8192	40962		28218

【表一】 乘法器的複雜度

由此表格將其用圖形表示出來，見【圖三十六】。橫座標為快速傅立葉轉換(FFT)的點數，而不同的演算法由不同的圖形表示之。由此圖形能夠發現在複數乘法的複雜度當中，以分割-基數 24 快速傅立葉轉換(Split-radix 2/4)為其運算度最低，如果要以此為考量，我們會選擇以分割-基數 24 快速傅立葉轉換(Split-radix 2/4) 為最佳選擇。



【圖三十六】 乘法器的複雜度的比較

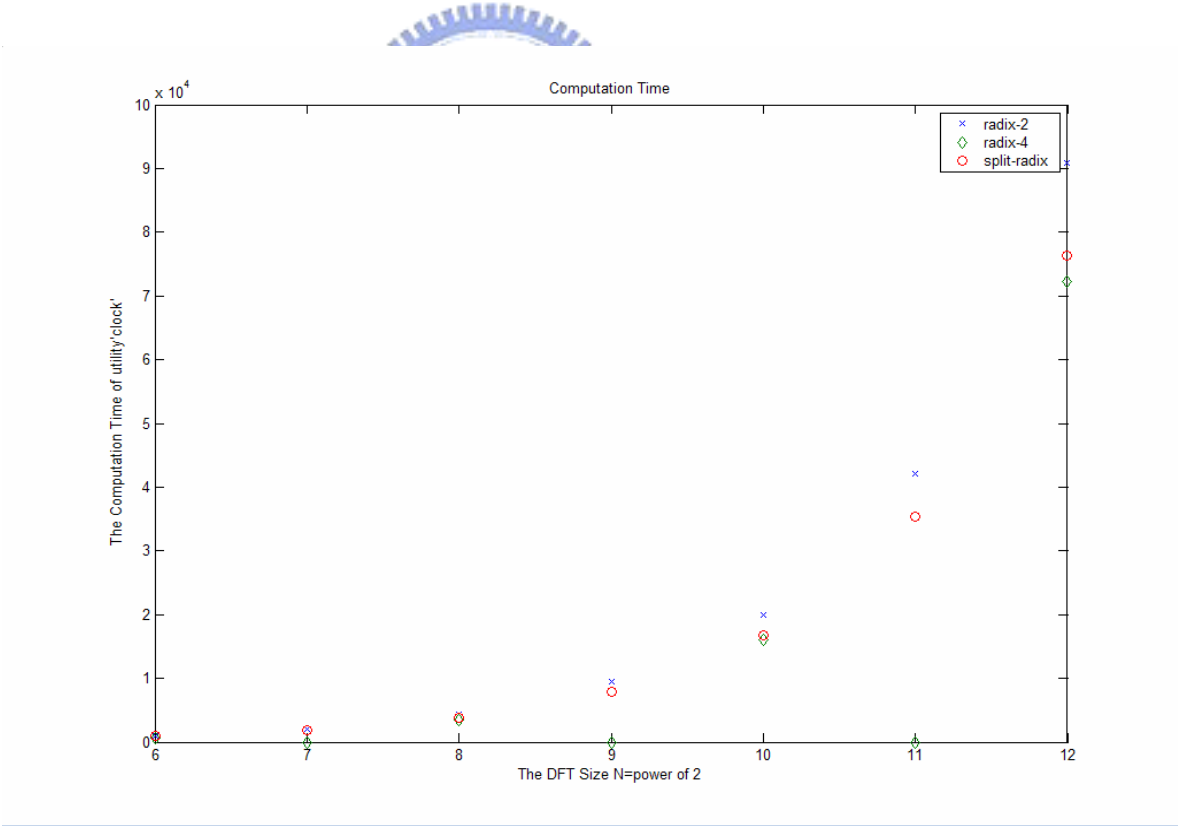
b.運算的速度:

以下為運算的時間，單位為 standard　unix clock，我們參考[19]，其數據為：

DFT size	Radix-2	Radix-4	Split-Radix
16	1000	700	900
128	2100	0	1800
256	4400	3500	3800
512	9400	0	7900
1024	19900	16000	16800
2048	42100	0	35400
4096	90900	72200	76400

【表二】運算的速度

把表格用圖形表示之，見【圖三十七】。可發現當點數為 4 的指數時，基-4 快速傅立葉轉換(radix-4 FFT)的運算時間最短，所以會最快，因此在此情況下我們選擇基-4 快速傅立葉轉換(radix-4 FFT)會是最好的選擇。



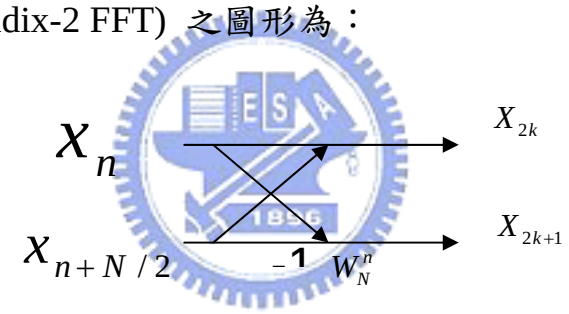
【圖三十七】 運算的速度比較

c. 點數的限制。

由前面幾個小節的演算法，可以發現基-r 快速傅立葉轉換(radix-r FFT)只能適用於當點數為 r 的指數時才能使用，而分割-基數 24 快速傅立葉轉換(Split-radix 2/4) 和基-2 快速傅立葉轉換(radix-2 FFT)的演算法是可以符合點數為 2 的指數，所以在選擇分割-基數 24 快速傅立葉轉換(Split-radix 2/4) 和基-2 快速傅立葉轉換(radix-2 FFT)。

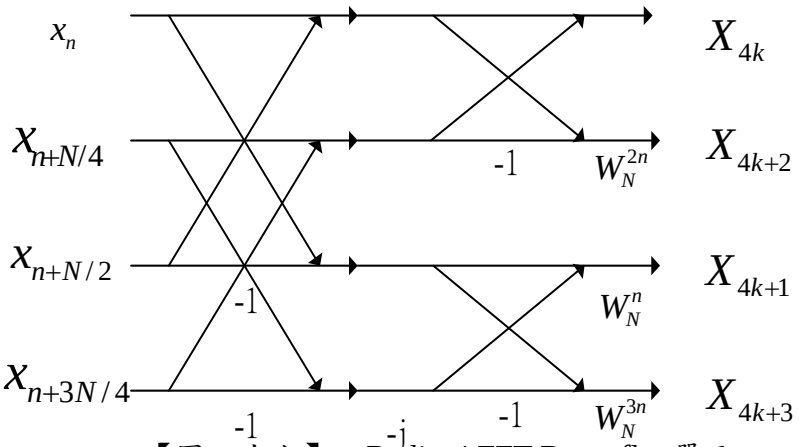
d. 蝴蝶圖(butterfly)的規則性:

我們可以由蝴蝶圖(butterfly)的基本單元來看其中的規則性，基-2 快速傅立葉轉換(radix-2 FFT) 之圖形為：



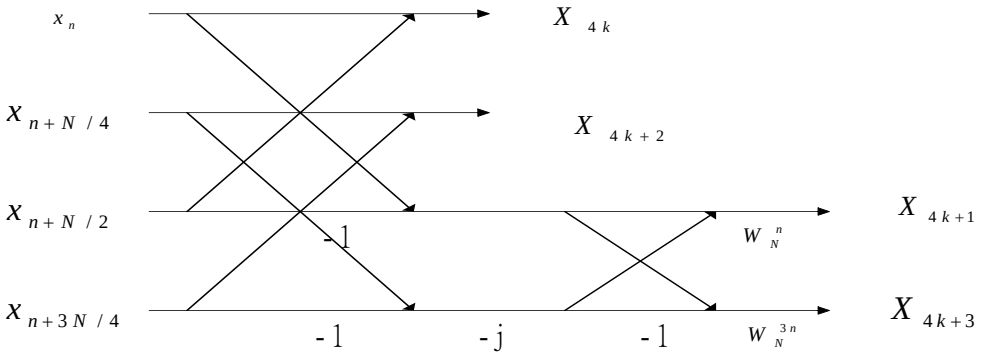
【圖三十八】 Radix-2 FFT Butterfly 單元

基-4 快速傅立葉轉換(radix-4 FFT) 之圖形為：



【圖三十九】 Radix-4 FFT Butterfly 單元

分割-基數 24 快速傅立葉轉換(Split-radix 2/4)之圖形為:



【圖四十】 Split -radix2/4 FFT Butterfly 單元

由以上的蝴蝶圖(butterfly)形狀可以發現分割-基數 24 快速傅立葉轉換(Split-radix 2/4)的形狀最不規則，在硬體實現上我們會較傾向選擇廣為使用的基-2 快速傅立葉轉換(radix-2 FFT)與基-4 快速傅立葉轉換(radix-4 FFT)的演算法。

因此所有比較的總結論以【表三】示之：

	複數乘法運算	運算速度	點數	Butterfly 規則性
Radix-2	高	慢	2^n	規則
Radix-4	低	最快	4^n	規則
Split-radix	最低	快	2^n	不規則
選擇	Split-radix	Radix-4	Radix-2 或 Split-radix	Radix-2 或 Radix-4

【表三】FFT 演算法比較

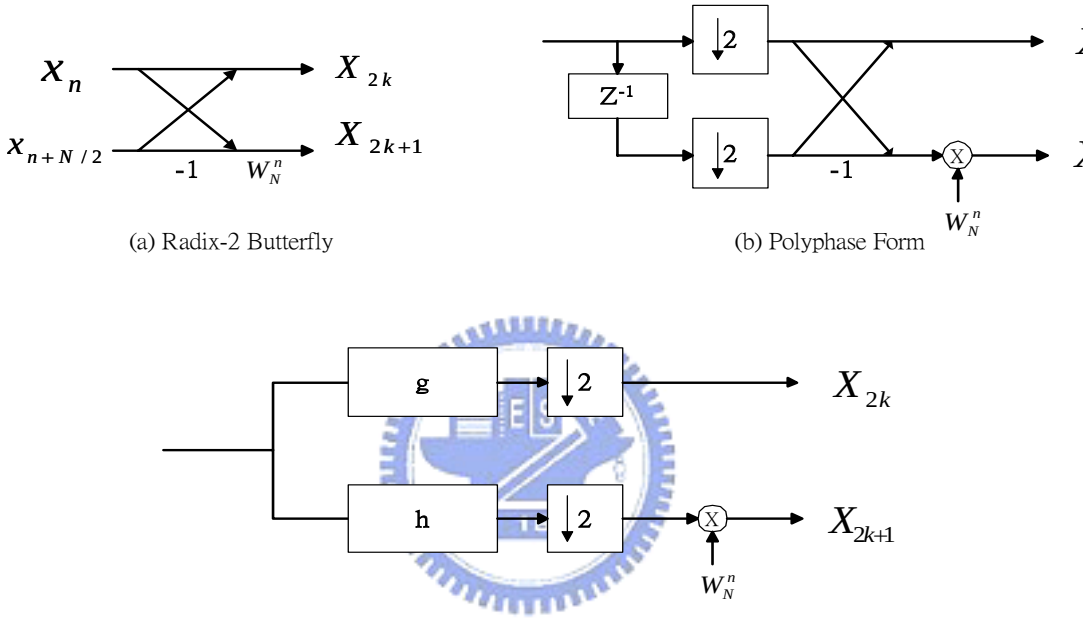
綜合所有的比較我們發現，大部分的硬體都是為不同的 FFT 演算法量身定做而成，但在應用層面來說，各種的 radix 都有其不同的好處，因此我們希望能夠發展出一種演算法讓硬體架構能夠用有效率的方式去實現不同的硬體，而讓使用者選擇想達到的硬體要求。

第三章 樹狀重組技術演算法

所有的 DFT/IDFT 皆可以用 FFT/IFFT 的演算法來實現，而其演算法後來皆可以形成蝴蝶(butterfly)的圖形，經由[20]我們瞭解到蝴蝶(butterfly)的圖形我們能夠同等於一個訊號經過 r 個濾波器，再經由不同的降頻(downsampling rate) r，其中 r 所指的即是看使用的演算法是基-r 快速傅立葉轉換(radix-r FFT)以下分別說明基-2 快速傅立葉轉換(radix-2 FFT)，基-4 快速傅立葉轉換(radix-4 FFT)及分割-基數 24 快速傅立葉轉換(Split-radix 2/4)如何轉成樹狀圖。

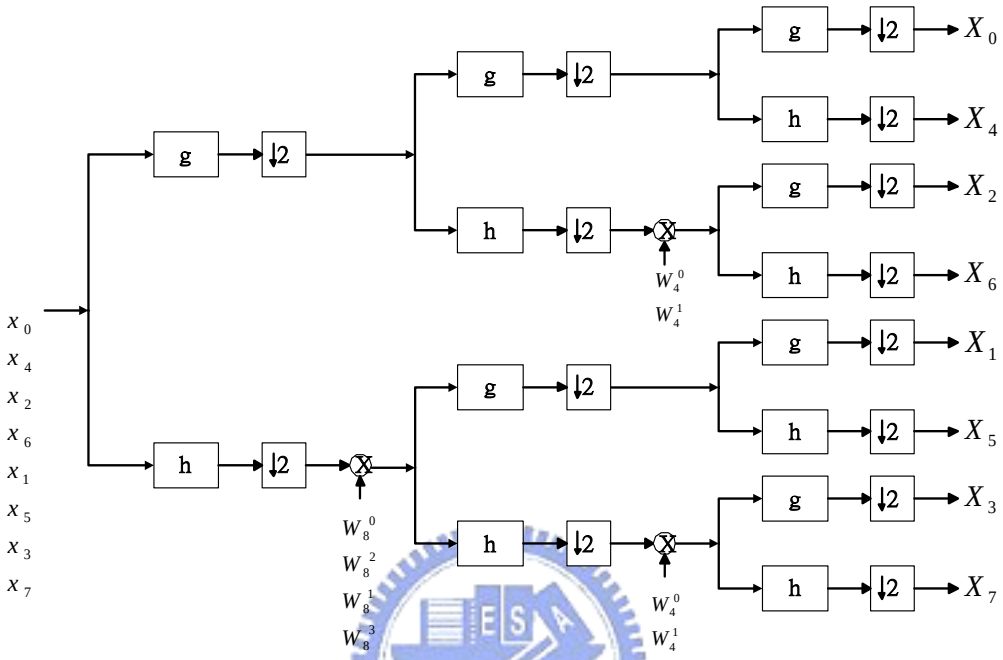
3.1 基-2 快速傅立葉轉換(radix-2 FFT) 演變二元樹

【圖四十一】即為基-2 快速傅立葉轉換(radix-2 FFT) (a)變成 Polyphase 型態(b)再演變成濾波器 (c)的基本圖形，其中 濾波器 中的係數分別為 g=[1 ,1] ,h=[-1 ,1]。



【圖四十一】 Radix-2 FFT 演變二元樹狀圖

由上圖的基本圖形的演變，我們舉之前的八點的例子將此展開成樹狀圖，見【圖四十二】:



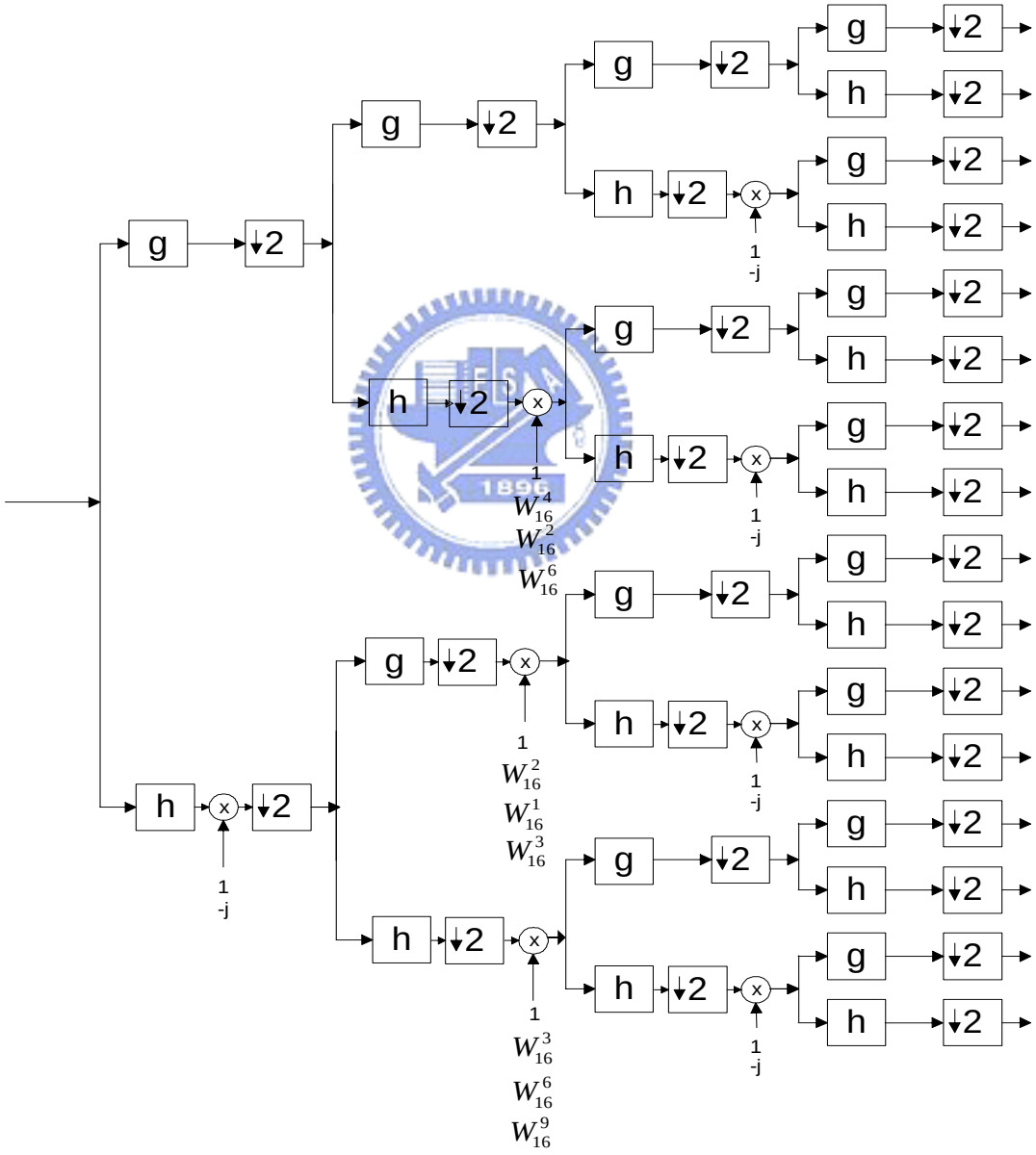
【圖四十二】 八點 radix-2 FFT 演變二元樹狀圖

由【圖四十二】我們所展開的樹狀圖中，輸入端的輸入順序一定為反向位元(bitreverse)而輸出端也為反向位元(bitreverse)，所以我們不用像 FFT 的演算法一樣要分分時(DIT)和分頻(DIF)，而且交互因子(twiddle factor) 輸入順序也是呈現反向位元(bitreverse) 的方式。

3.2 基-4 快速傅立葉轉換(radix-4 FFT)演變二元樹

同理我們能夠把基-4(radix-4)的蝴蝶圖(butterfly)展成二元樹狀圖，不同的只是交互因子(twiddle factor) 的排列方式，見【圖四十三】，但是濾波器中的係數皆沒有改變。

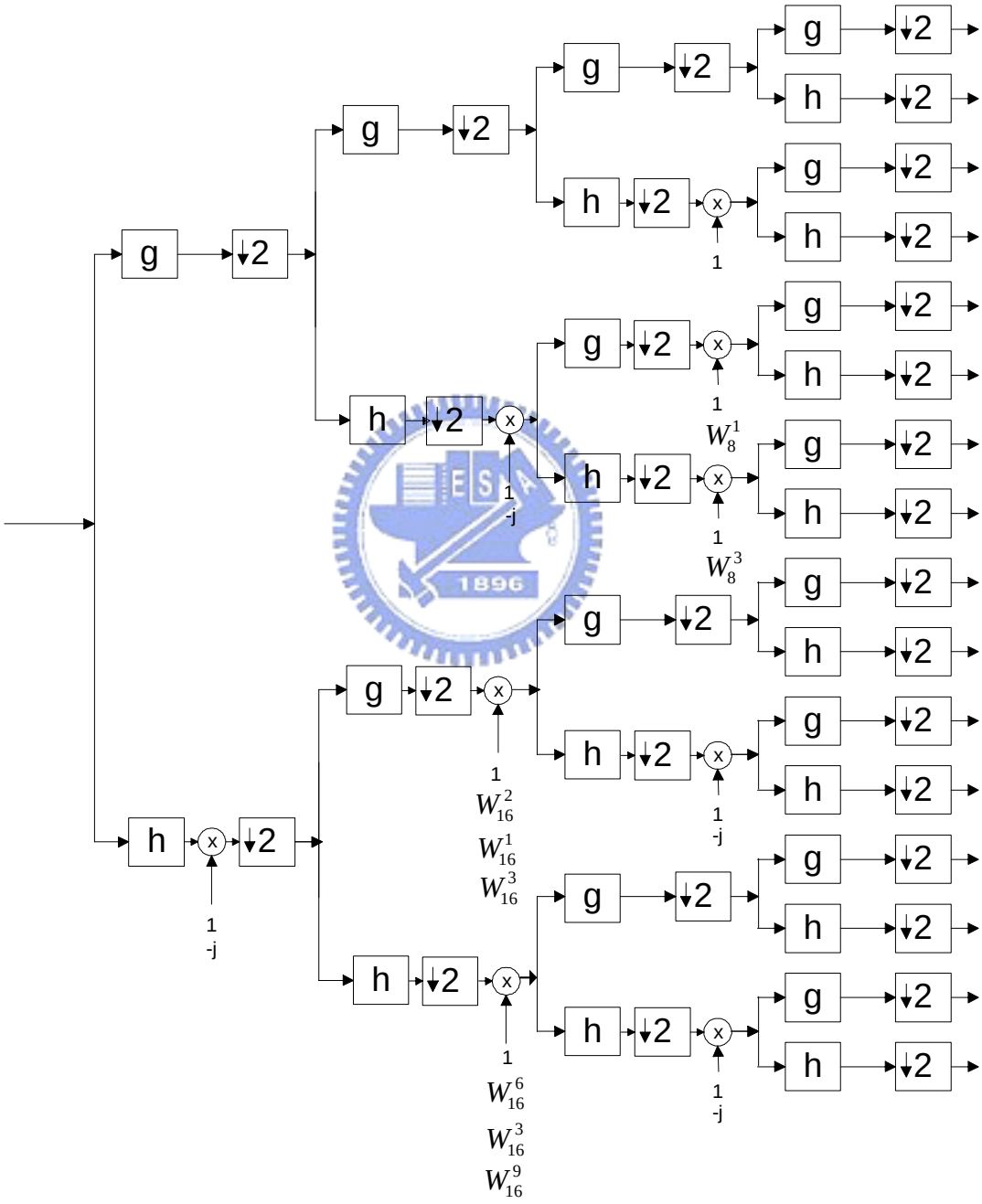
我們取 N＝16 的點數去轉成二元樹狀圖，見【圖四十三】。



【圖四十三】 十六點 radix-4 FFT 演變二元樹狀圖

3.3 分割-基數 24 快速傅立葉轉換(Split-radix 2/4) 演變二元樹

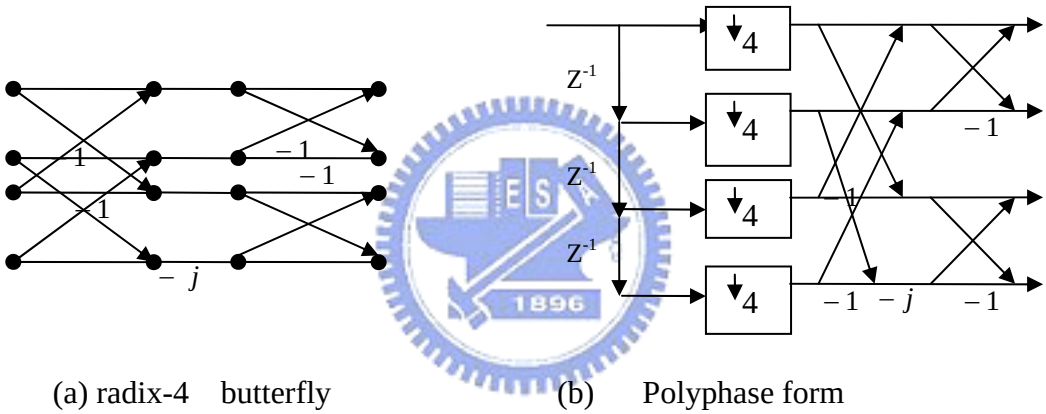
同理我們能夠把分割-基數 24 快速傅立葉轉換(Split-radix 2/4)的蝴蝶(butterfly)展成二元樹狀圖，以下以 N=16 為例子，見【圖四十四】。



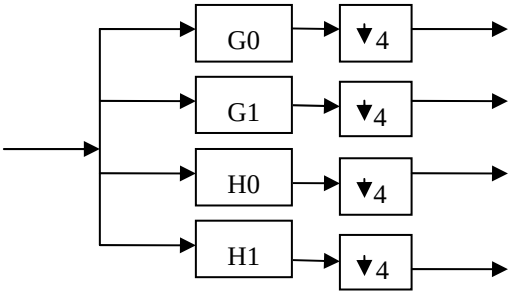
【圖四十四】 十六點 split-radix2/4 FFT 演變二元樹狀圖

3.4 基-4 快速傅立葉轉換(radix-4 FFT)演變四元樹

同理我們把基-4 快速傅立葉轉換(radix-4 FFT)的蝴蝶圖(butterfly)展成 4 元樹狀圖，去增加其平行度，而讓濾波器的係數由兩個變成四個，其基本單元轉變為以下的圖形，由(a)圖到(c)圖，其中濾波器的係數分別為：g0=[1 1 1 1]，g1=[-1 1 -1 1]，h0=[j -1 -j 1]和 h1=[-j -1 j 1]，見【圖四十五】。



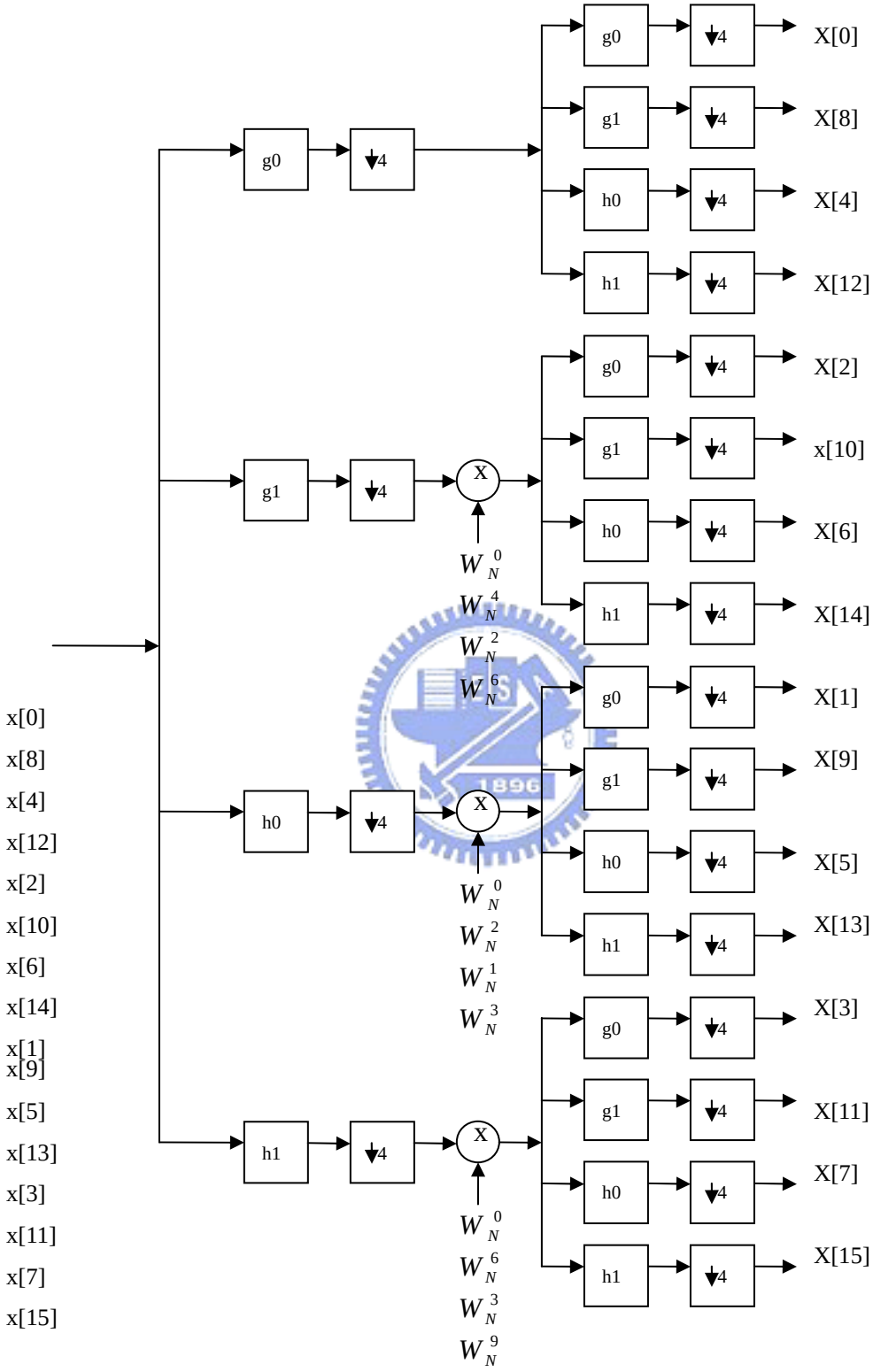
(a) radix-4 butterfly (b) Polyphase form



(c) butterfly of filter form

【圖四十五】 Radix-4 FFT 演變四元樹狀圖

我們以十六點為例子展成四元樹狀圖，見【圖四十六】。

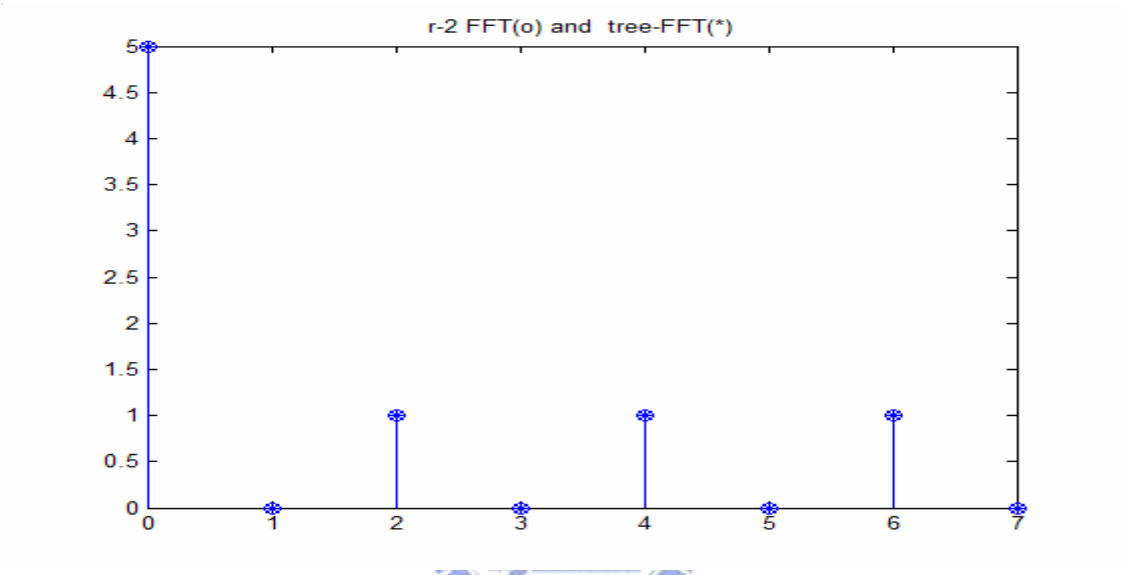


【圖四十六】 十六點 radix-4 FFT 演變四元樹狀圖

3.5 模擬驗證各種樹狀圖與 FFT 的結果

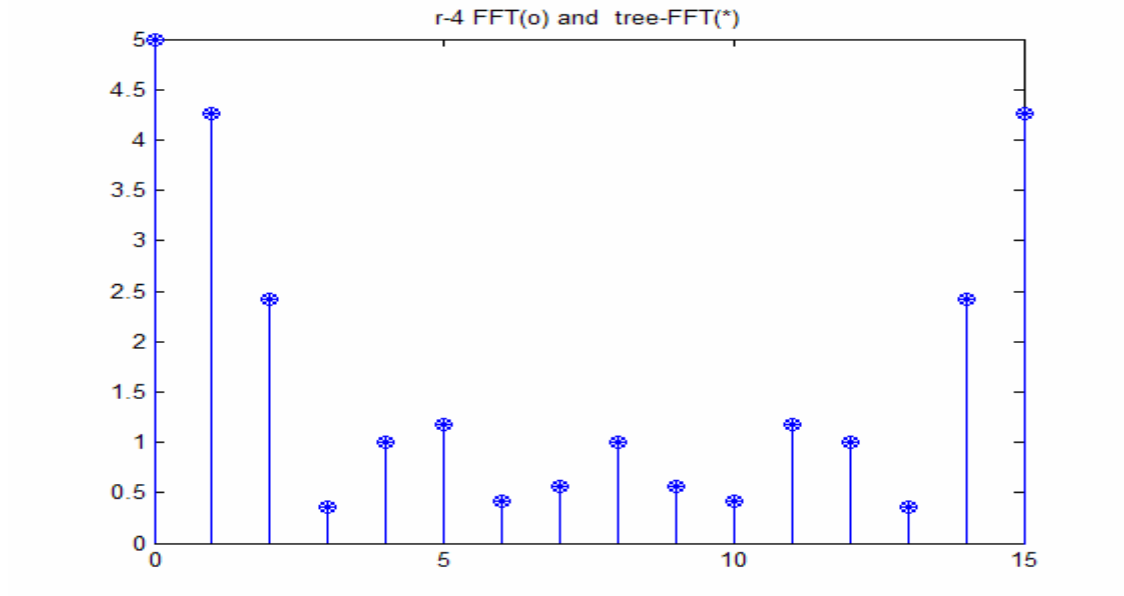
根據上一章的演算法，我們使用 matlab 去模擬驗證以上圖形的推導是正確的。

1. 驗證 N＝8 的 radix-2 轉成二元樹狀圖，見【圖四十七】。



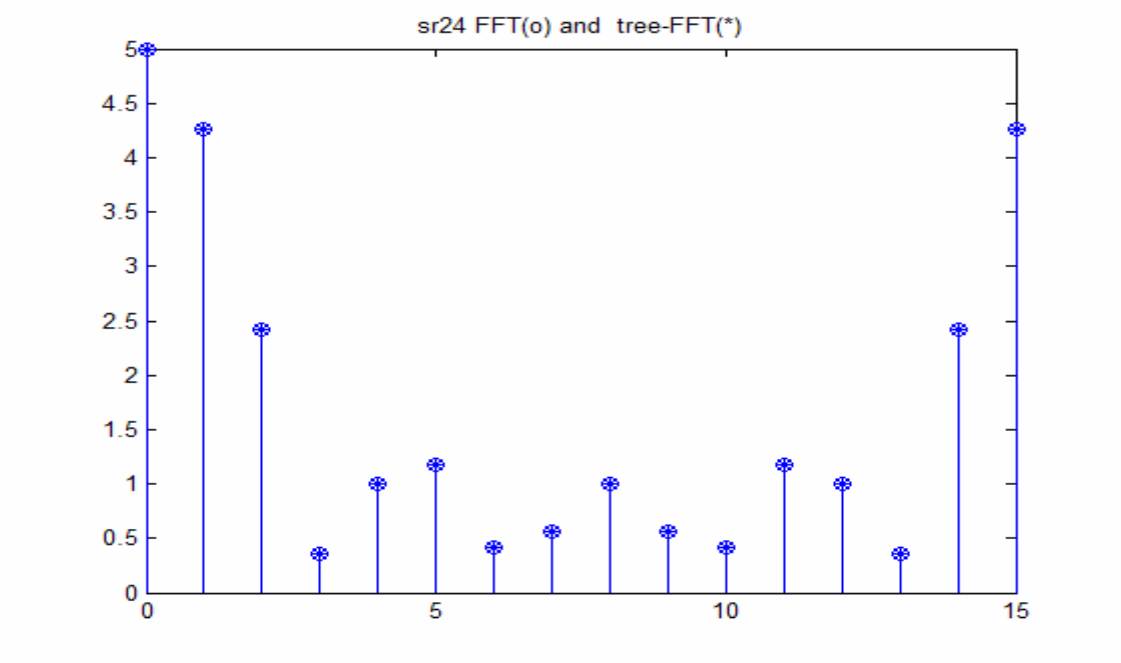
【圖四十七】八點 radix-2 FFT(o) 與 二元樹 FFT(*)

2. 驗證 N＝16 的 radix-4 轉成二元樹狀圖，見【圖 四十八】。



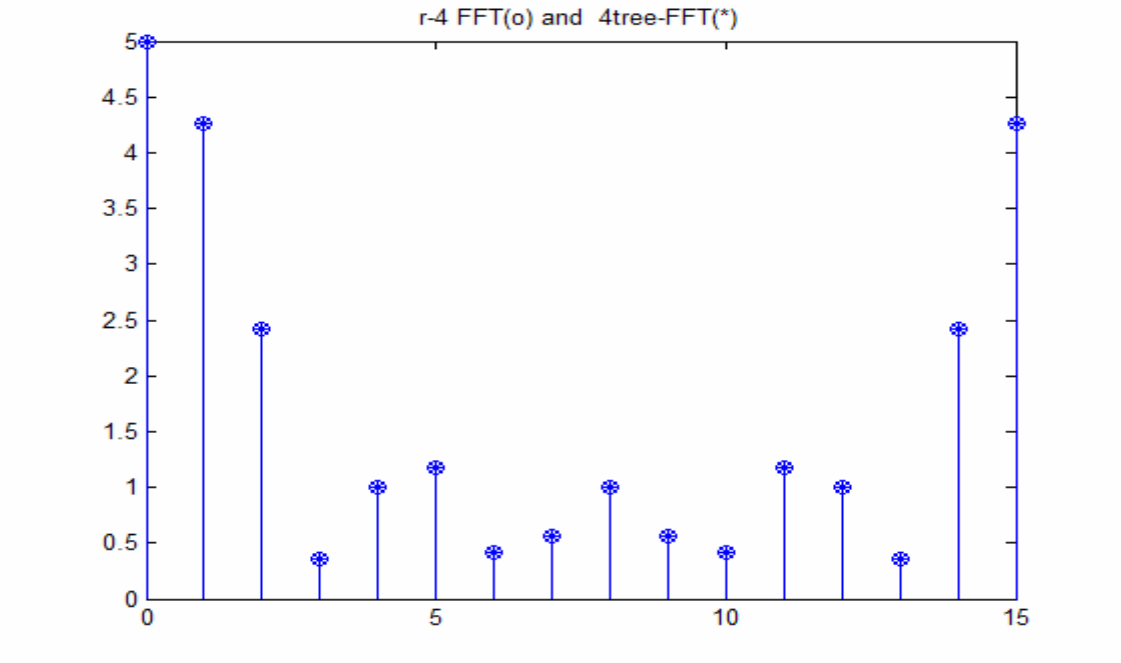
【圖四十八】十六點的 radix-4 FFT(o) 與二元樹 FFT(*)

3.驗證 N＝16 的 split-radix2/4 轉成二元樹狀圖，見【圖四十九】。



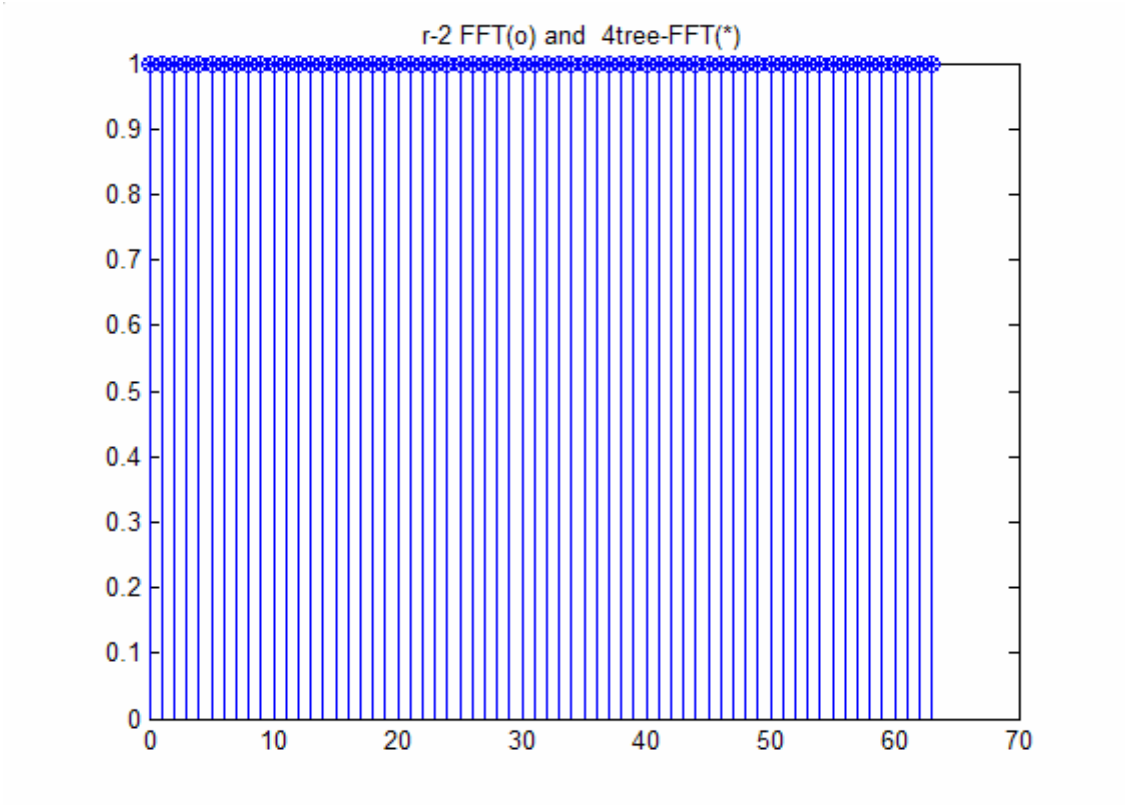
【圖四十九】十六點的 split-radix2/4 FFT(o) 與二元樹 FFT(*)

4.驗證 N＝16 的 radix-4 轉成四元樹狀圖，見【圖五十】。



【圖五十】十六點的 radix-4 FFT(o)與四元樹 FFT(*)

以上圖形的輸入為方波其結果應該為弦波(sinc function)，由觀察結果得知，其圖形完全吻合，由此可以得知各種 FFT 演算法可以由樹狀演算法分別轉成各種不同的樹狀圖，但因為以上的點數太少，因此我們在擴展成六十四點的圖形做驗證，此時的輸入波形使用脈衝(impulse)，其結果見【圖五十一】。



【圖五十一】六十四點的 radix-2 FFT(o) 與二元樹 FFT(*)

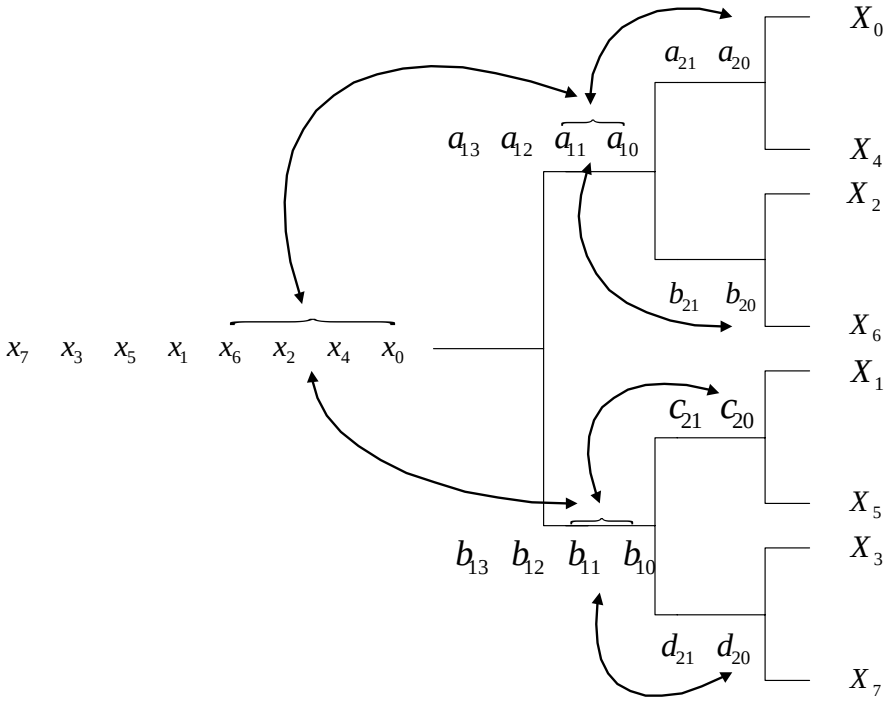
3.6 結論

根據樹狀演算法的方法拆解不同 FFT 演算法的 butterfly 圖形，經由以上的驗證知，使用者能夠有彈性選擇各種不同 FFT 的演算方法，皆能由樹狀的演算法去實現。

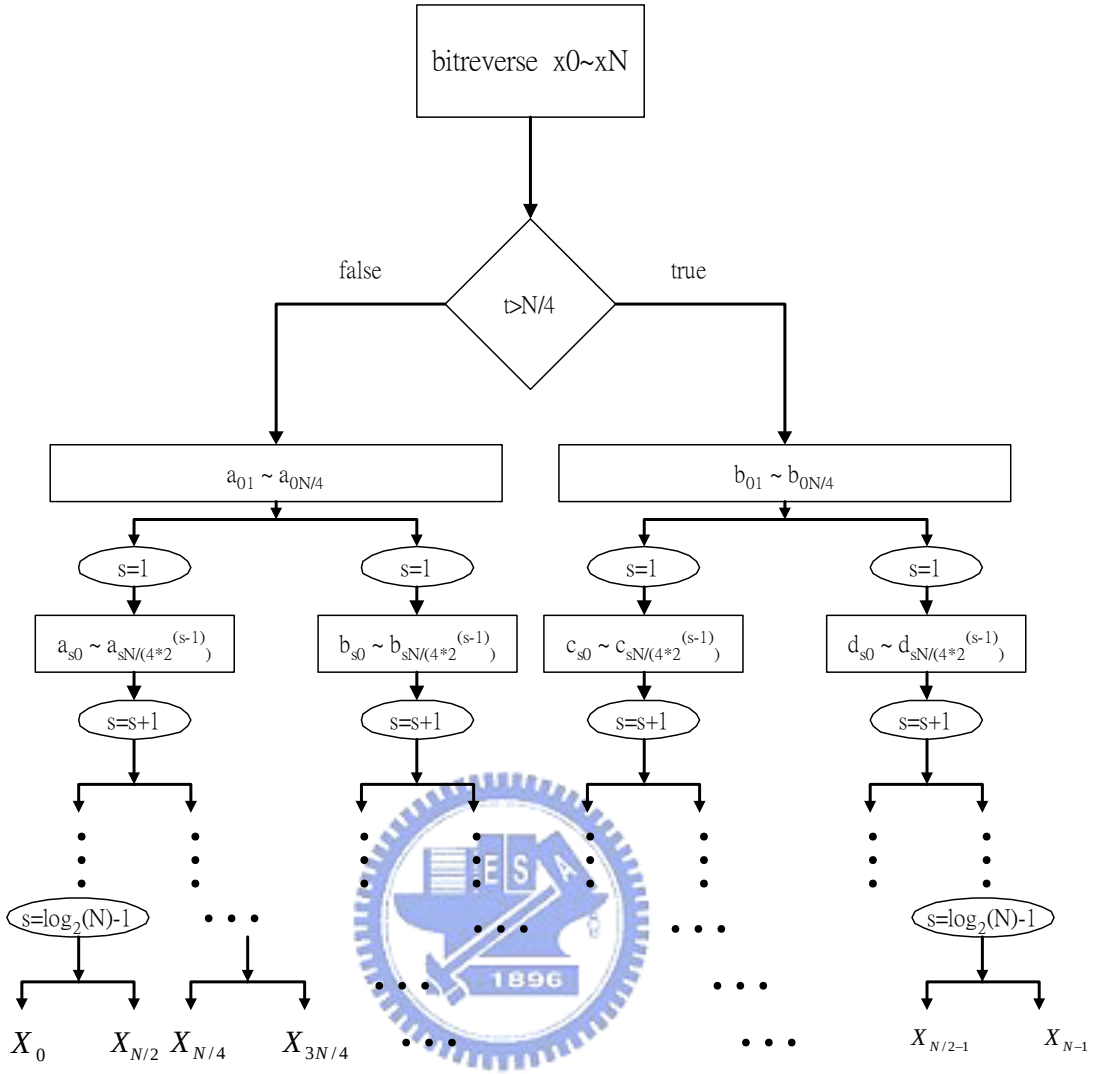
第四章 樹狀硬體之實現

4.1 排程

確定樹狀為可行的，接下來為排程方式，而其中排程的方式我們有兩種方式，一種為密集的排程，我們希望能夠在產出率(throughput)最高的情況下去排程，在這種情況下的排程我們會發現，在每一級的暫存器會成 2 的級數增加，如果依此種方式去做運算，當暫存器增加相對面積就會增加。因此我們只好想出另一種排程，及用 iteration 去尋找每一個樹的根，在此情形下暫存器的數目不會增加，每一級的暫存器只需要 4 個，但是產出率(throughput)在六十四點以前的 FFT 可以接近所要求的，可是當大於六十四點時，我們的產出率(throughput)也會成 2 的級數的增加，因此我們決定把兩個結合起來，在符合規格的產出率(throughput)下同時也讓暫存器的數目最少。其排程的流程圖我們以【圖五十二】與【圖五十三】示之。先處理上半個樹的資料，先產生偶數的 FFT 輸出結果，再處理下半個樹的資料，產生奇數的 FFT 輸出結果。


X_0
a_{21} a_{20}
a_{13} a_{12} a_{11} a_{10}
X_4
X_2
b_{21} b_{20}
X_6
x_7 x_3 x_5 x_1 x_6 x_2 x_4 x_0
X_1
c_{21} c_{20}
X_5
b_{13} b_{12} b_{11} b_{10}
X_3
d_{21} d_{20}
X_7

【圖五十二】排程的示意圖


bitreverse x0~xN
false
t>N/4
true
a01 ~ a0N/4
b01 ~ b0N/4
s=1
s=1
s=1
s=1
as0 ~ asN/(4*2(s-1))
bs0 ~ bsN/(4*2(s-1))
cs0 ~ csN/(4*2(s-1))
ds0 ~ dsN/(4*2(s-1))
s=s+1
s=s+1
s=s+1
s=s+1
s=log2(N)-1
s=log2(N)-1
X0
XN/2
XN/4
X3N/4
XN/2-1
XN-1

【圖五十三】排程的流程圖

因此以下為我所使用的 3 種排程方式演變(以 16 點為例子)：

第一種排程(高產出率(throughput))：

clk	input	1stMAC	R1	2nd MAC	R2	3rd MAC	R3	4rdMAC	output
1	x0x8x4x12	x0x8x4x12	a01a02	a01a02a01-a02	a11 b11				
2	x2x10x6x14	x2x10x6x14	a03a04	a03a04a03-a04	a12 b12	a11a12a11-a12	a21b21		
3	x1x9x5x13	x1x9x5x13	a05a06	a05a06a05-a06	a13 b13	b11b12b11-b12	c21d21		
4	x3x11x7x15	x3x11x7x15	a07a08	a07a08a07-a08	a14 b14	a13a14a13-a14	a22b22	a21a22a21-a22	a31b31
5	x0-x8x4-x12	x0-x8x4-x12	b01b02	b01b01b01-b02	c11d11	b13b14b13-b14	c22d22	b21b22b21-b22	c31d31
6	x2-x10x6-x14	x2-x10x6-x14	b03b04	b03b04b03-b04	c12d12	c11c12c11-c12	e21f21	c21c22c21-c22	e31f31
7	x1-x9x5-x13	x1-x9x5-x13	b05b06	b05b06b05-b06	c13d13	d11d12d11-d12	g21h21	d21d22d21-d22	g31h31
8	x3-x11x7-x15	x3-x11x7-x15	b07b08	b07b08b07-b08	c14d13	c13c14c13-c14	e22f22	e21e22e21-e22	i31j31
9						d13d14d13-d14	g22h22	f21f22f21-f22	k31l31
10								g21g22g21-g22	m31n31
11								h21h22h21-h22	o31p31

第二種排程(暫存器少)：

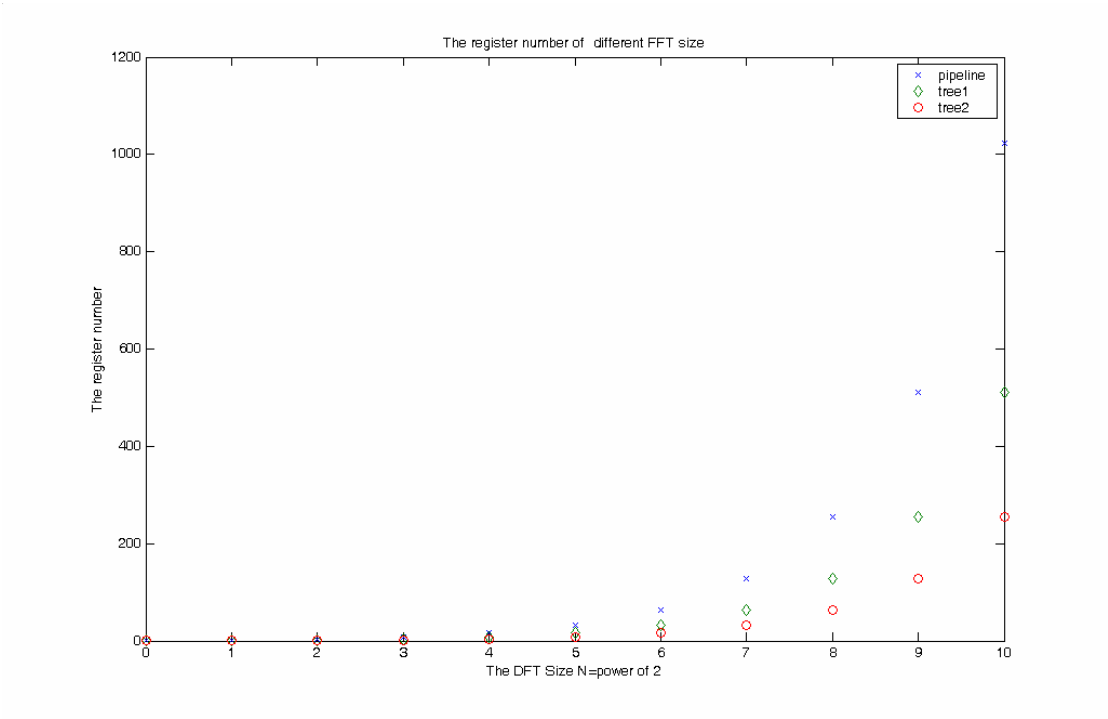
clk	input	1stMAC	R1	2nd MAC	R2	3rd MAC	R3	4rdMAC	output
1	x0x8x4x12	x0x8x4x12	a01a02						
2	x2x10x6x14	x2x10x6x14	a03a04	a01a02a03a04	a11a12				
3	x1x9x5x13	x1x9x5x13	a05a06						
4	x3x11x7x15	x3x11x7x15	a07a08	a05a06a07a08	a13a14	a11a12a13a14	a21a22	a21a22a21-a22	a31b31
5	x0x8x4x12	x0x8x4x12	a01a02						
6	x2x10x6x14	x2x10x6x14	a03a04	a01a02a03a04	a11a12				
7	x1x9x5x13	x1x9x5x13	a05a06						
8	x3x11x7x15	x3x11x7x15	a07a08	a05a06a07a08	a13a14	a11-a12a13-a14	b21b22	b21b22b21-b22	c31d31
9	x0x8x4x12	x0x8x4x12	a01a02						
10	x2x10x6x14	x2x10x6x14	a03a04	a01-a02a03-a04	b11b12				
11	x1x9x5x13	x1x9x5x13	a05a06						
12	x3x11x7x15	x3x11x7x15	a07a08	a05-a06a07-a08	b13b14	b11b12b13b14	c21c22	c21c22c21-c22	e31f31
13	x0x8x4x12	x0x8x4x12	a01a02						
14	x2x10x6x14	x2x10x6x14	a03a04	a01-a02a03-a04	b11b12				
15	x1x9x5x13	x1x9x5x13	a05a06						
16	x3x11x7x15	x3x11x7x15	a07a08	a05-a06a07-a08	b13b14	b11-b12b13-b14	d21d22	d21d22d21-d22	g31h31
17	x0x8x4x12	x0-x8x4-x12	b01b02						
18	x2x10x6x14	x2-x10x6-x14	b03b04	b01b02b03b04	c11c12				
19	x1x9x5x13	x1-x9x5-x13	b05b06						
20	x3x11x7x15	x3-x11x7-x15	b07b08	b05b06b07b08	c13c14	c11c12c13c14	e21e22	e21e22e21-e22	i31j31
21	x0x8x4x12	x0-x8x4-x12	b01b02						
22	x2x10x6x14	x2-x10x6-x14	b03b04	b01b02b03b04	c11c12				
23	x1x9x5x13	x1-x9x5-x13	b05b06						
24	x3x11x7x15	x3-x11x7-x15	b07b08	b05b06b07b08	c13c14	c11-c12c13-c14	f21f22	f21f22f21-f22	k31l31
25	x0x8x4x12	x0-x8x4-x12	b01b02						
26	x2x10x6x14	x2-x10x6-x14	b03b04	b01-b02b03-b04	d11d12				
27	x1x9x5x13	x1-x9x5-x13	b05b06						
28	x3x11x7x15	x3-x11x7-x15	b07b08	b05-b06b07-b08	d13d14	d11d12d13d14	g21g22	g21g22g21-g22	m31n31
29	x0x8x4x12	x0-x8x4-x12	b01b02						
30	x2x10x6x14	x2-x10x6-x14	b03b04	b01-b02b03-b04	d11d12				
31	x1x9x5x13	x1-x9x5-x13	b05b06						
32	x3x11x7x15	x3-x11x7-x15	b07b08	b05-b06b07-b08	d13d14	d11-d12d13-d14	h21h22	h21h22h21-h22	o31p31

第三種排程(綜合)：

clk	input	1stMAC	R1	2nd MAC	R2	3rd MAC	R3	4rdMAC	output
1	x0x8x4x12	x0x8x4x12	a01a02	a01a02	a11				
2	x2x10x6x14	x2x10x6x14	a03a04	a03a04	a12	a11a12a11-a12	a21b21		
3	x1x9x5x13	x1x9x5x13	a05a06	a05a06	a13				
4	x3x11x7x15	x3x11x7x15	a07a08	a07a08	a14	a13a14a13-a14	a22b22	a21a22a21-a22	a31b31
5	x0x8x4x12	x0x8x4x12	a01a02	a01-a02	b11			b21b22b21-b22	c31d31
6	x2x10x6x14	x2x10x6x14	a03a04	a03-a04	b12	b11b12b11-b12	c21d21		
7	x1x9x5x13	x1x9x5x13	a05a06	a05-a06	b13				
8	x3x11x7x15	x3x11x7x15	a07a08	a07-a08	b14	b13b14b13-b14	c22d22	c21c22c21-c22	e31f31
9	x0-x8x4-x12	x0-x8x4-x12	b01b02	b01b01	c11			d21d22d21-d22	g31h31
10	x2-x10x6-x14	x2-x10x6-x14	b03b04	b03b04	c12	c11c12c11-c12	e21f21		
11	x1-x9x5-x13	x1-x9x5-x13	b05b06	b05b06	c13				
12	x3-x11x7-x15	x3-x11x7-x15	b07b08	b07b08	c14	c13c14c13-c14	e22f22	e21e22e21-e22	i31j31
13	x0-x8x4-x12	x0-x8x4-x12	b01b02	b01-b01	d11			f21f22f21-f22	k31l31
14	x2-x10x6-x14	x2-x10x6-x14	b03b04	b03-b04	d12	d11d12d11-d12	g21h21		
15	x1-x9x5-x13	x1-x9x5-x13	b05b06	b05-b06	d13				
16	x3-x11x7-x15	x3-x11x7-x15	b07b08	b07-b08	d14	d13d14d13-d14	g22h22	g21g22g21-g22	m31n31
17								h21h22h21-h22	o31p31

根據以上的排程，我們能進一步的分析點數與暫存器的關係。又因為當蝴蝶圖(butterfly)拆解成樹狀時，我們也能夠去探討當資源不同時，MAC 與暫存器的關係。

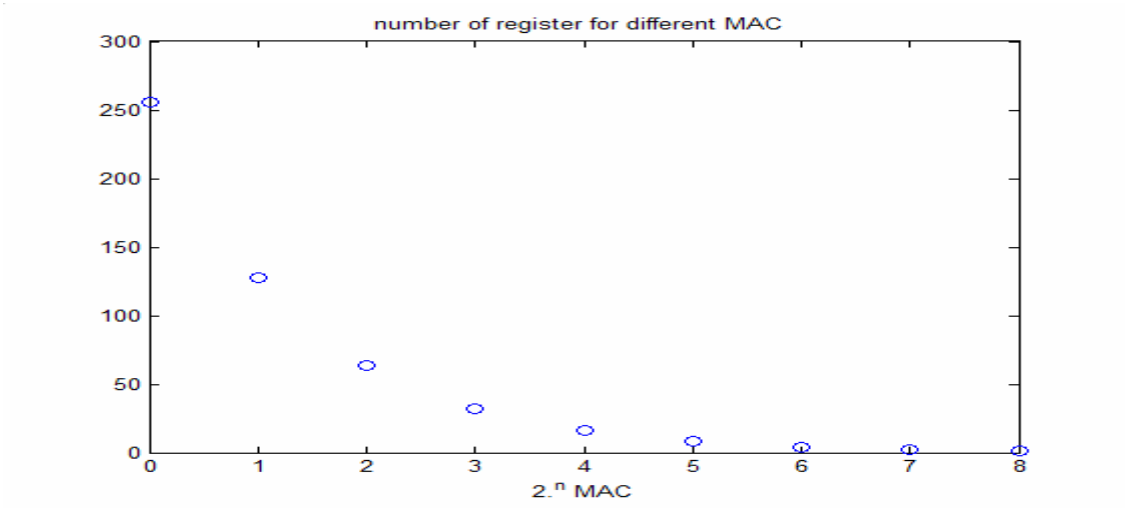
以下的模擬圖形為我們對暫存器的分析:



【圖五十四】暫存器個數比較

由以上暫存器的數目來看我們所使用的 pipeline 排程所需暫存器為 N-1 個、第一種排程需要 N/2，而第三種排程需要 N/4 個暫存器明顯已經降低許多接下來要想辦法設計交互因子(twiddle factor)的部分使得運算能夠更快速。

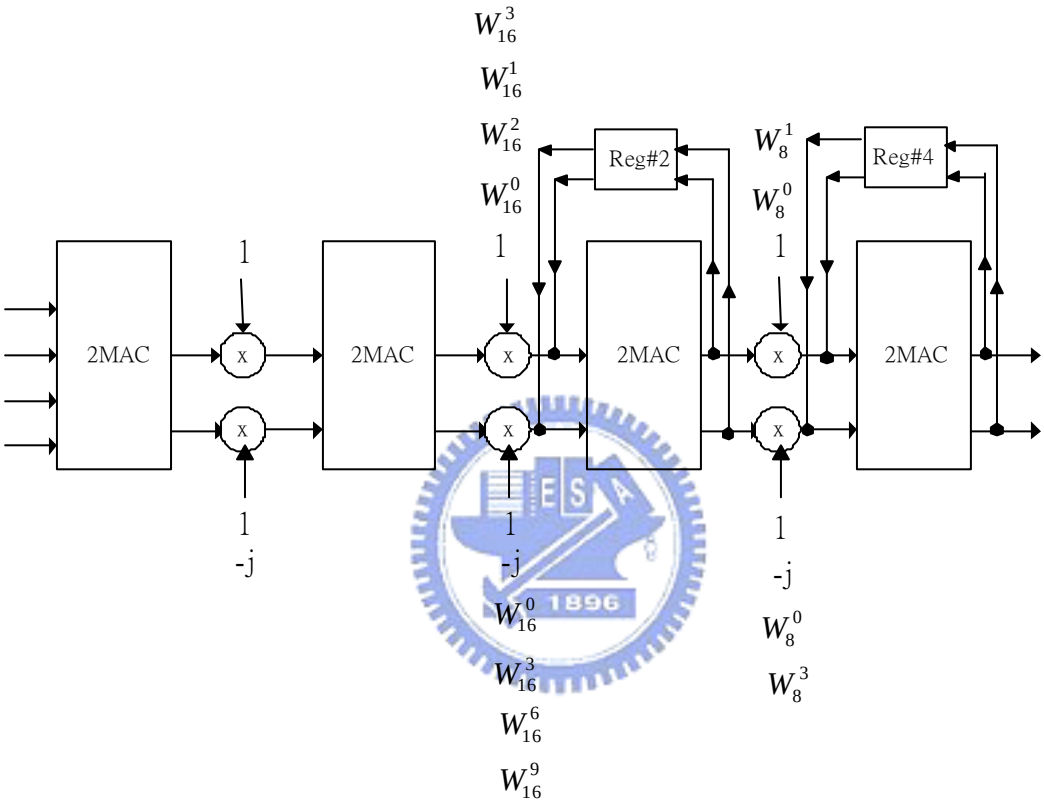
因為使用樹狀的演算法，在分析當中，可以使用我們的排程，去增加平行度，當每一級的 MAC 增加為兩倍時，暫存器也會隨著減少一級的暫存器數目，其圖形如【圖五十五】。



【圖五十五】 MAC 數目與暫存器的關係

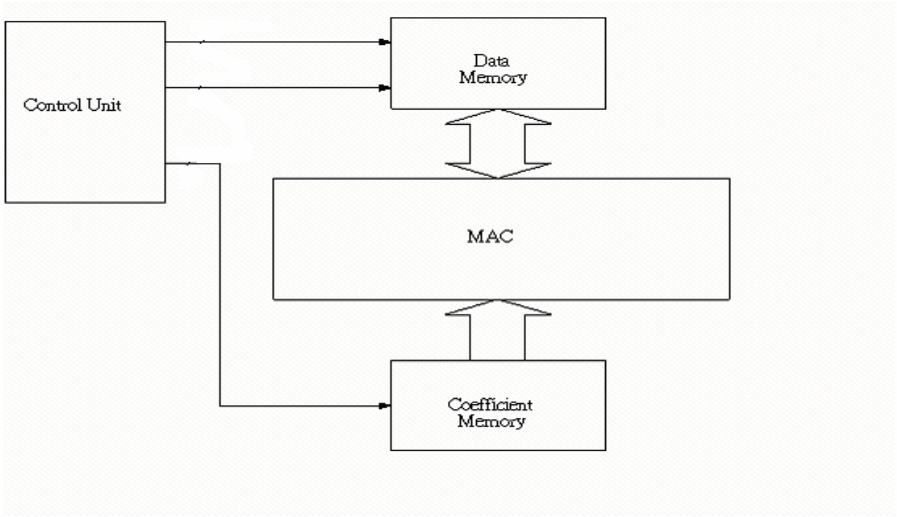
4.2 樹狀硬體架構設計

我們使用以上排程的分析，以下為樹狀硬體系統架構的基本圖形，見【圖五十六】。



【圖五十六】 樹狀的基本架構

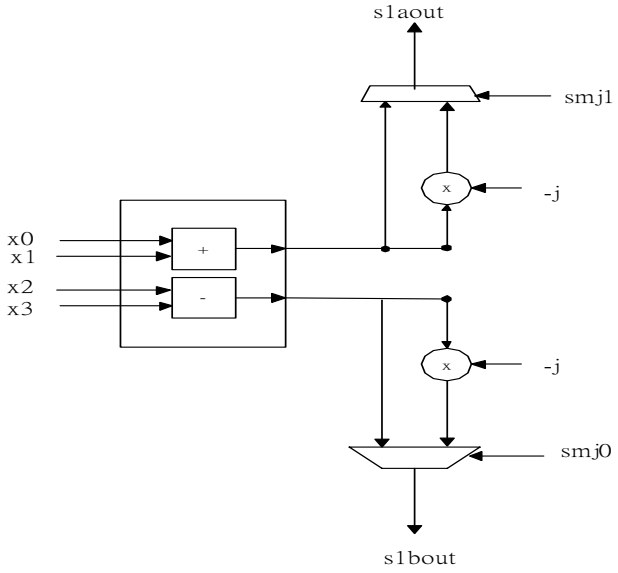
但實際硬體可分成四個部分，見【圖 五十七】。控制單元，乘加器單元及兩個記憶體：一個稱為係數記憶體(coefficient memory，CM)，存放係數的記憶體，可以讓使用者存入所需要的資訊，也就是交互因子(twiddle factor)的存放，另一個稱為資料記憶體(Data memory，DM)，即為硬體本身的暫存器。



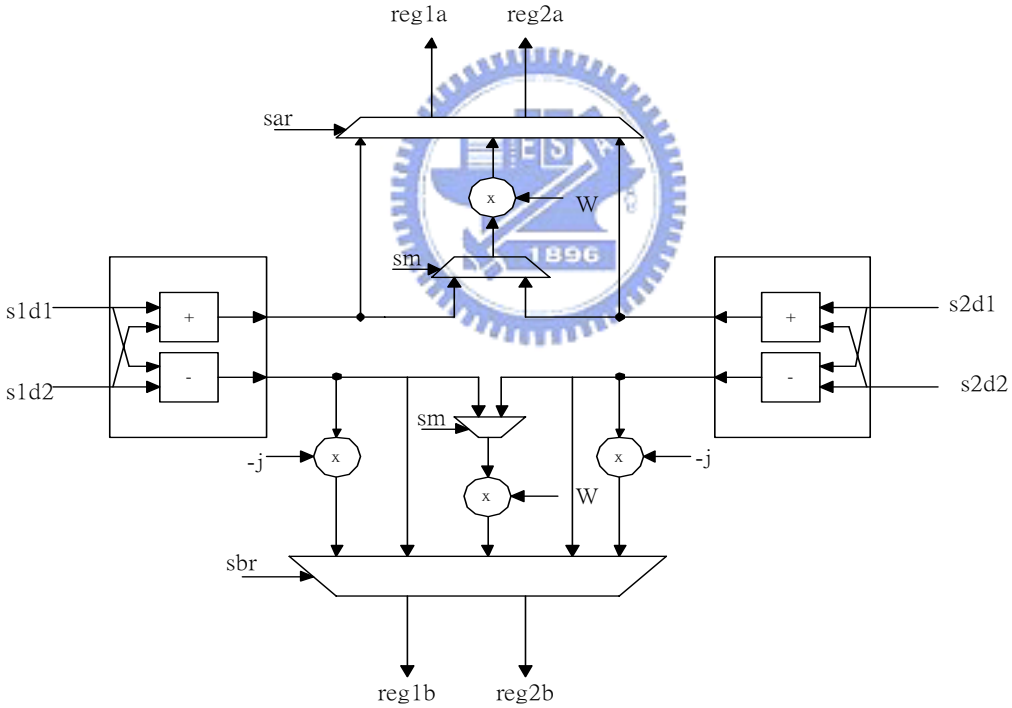
【圖五十七】 樹狀 FFT 的實際硬體架構系統圖

4.2.1 乘加器的邏輯電路：

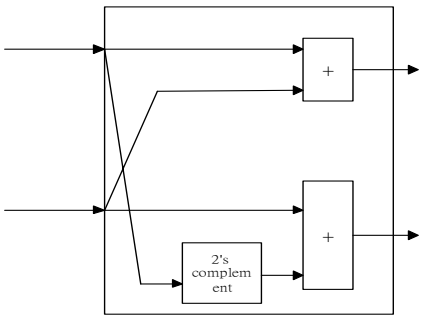
我們分成三種不同的運算單元，第一種乘加器是需要算出乘 1 與乘-j 的運算，電路設計見【圖 五十八】。第二種乘加器則是需要算出乘 1、乘-j 的運算與乘一般的交互因子(twiddle factor)電路設計。而第三種則是一般的加法器，電路設計見【圖 六十】。在整體硬體當中，以 16 點的 FFT 為例，我們在第一級使用第一種乘加器，在最後一級使用第三種加法器，中間的兩級皆使用第二種的乘加器。在第二種的乘加器，因為需要有複數的乘法器，而且又因為乘法器的面積很大，第一步先把兩個相鄰的兩級之間的乘法器共享，就可以把原來的乘法器個數先減半。見【圖五十九】。乘法器我們不使用一般的複數乘法器，因為一般的複數乘法器需要 4 個實數的乘法器，因此我們選擇使用座標旋轉演算法(Coordinate Rotation DIgitial Coputer, CORDIC)的乘法器。其基本的原理我們接下來介紹。



【圖五十八】 第一種乘加法單元



【圖五十九】 第二種乘加法單元



【圖六十】 第三種乘加法單元

4.2.2 座標旋轉演算法(Coordinate Rotation DIgitial Coputer,CORDIC)乘法器

CORDIC 演算法：

因為 DFT 的運算當中，每一個乘交互因子(twiddle factor)的方式，在複數的平面上可用極座標的方式來計算，先把原來的值 A(x1,y1) 依旋轉的角度至 B (x2,y2)，而 B 點座標即可表示成：

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \qquad (4.2.1)$$

若 $\theta_k = \tan^{-1} 2^{-k}$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{1+\tan^2\theta_k}} & \frac{\tan\theta_k}{\sqrt{1+\tan^2\theta_k}} \\ -\frac{\tan\theta_k}{\sqrt{1+\tan^2\theta_k}} & \frac{1}{\sqrt{1+\tan^2\theta_k}} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \frac{1}{\sqrt{1+2^{-2k}}} \begin{bmatrix} 1 & 2^{-k} \\ -2^{-k} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \qquad (4.2.2)$$

而任何的角度皆可以依下列的式子計算：

$$\theta = \pm\frac{\pi}{2} + \sum_{k=0}^{\infty}(\pm\tan^{-1}2^{-k})$$

$$\theta = \sum_{k=-1}^{n-1}\mu(k)\theta_k + \varepsilon,\ \ where\ \ \mu(k) = 1\ \ or\ \ -1$$

$$and\quad \theta_k = \begin{cases}\pi/2 & k=-1\\ \tan^{-1}2^{-k}, otherwise\end{cases} \qquad (4.2.3)$$

CODIC 的演算法：

1.依照以上的角度運算先求出 $\mu(k)$ 的值。

2.再計算出比例係數(scaling factor)

$$K = \left(\prod_{k=0}^{n-1}\sqrt{1+2^{-2k}}\right)^{-1} \qquad (4.2.4)$$

3.先把原來的座標 A(x1,y1)旋轉 $\pm\pi/2$，也就是

$$x(0) = \mu(-1)y\ ,\ y(0) = -\mu(-1)x \qquad (4.2.5)$$

4.之後都是做一樣的計算方式，i 從 0 到(n-1)

$$\begin{bmatrix}x(i+1)\\ y(i+1)\end{bmatrix} = \begin{bmatrix}1 & \mu(i)2^{-k}\\ -\mu(i)2^{-k} & 1\end{bmatrix}\begin{bmatrix}x(i)\\ y(i)\end{bmatrix} \qquad (4.2.6)$$

5.最後只要把最後算出來的值乘上比例係數 K，即可算出正確要求出的 B (x2,y2)。

$$x2 = Kx(n)$$
$$y2 = Ky(n) \qquad (4.2.7)$$

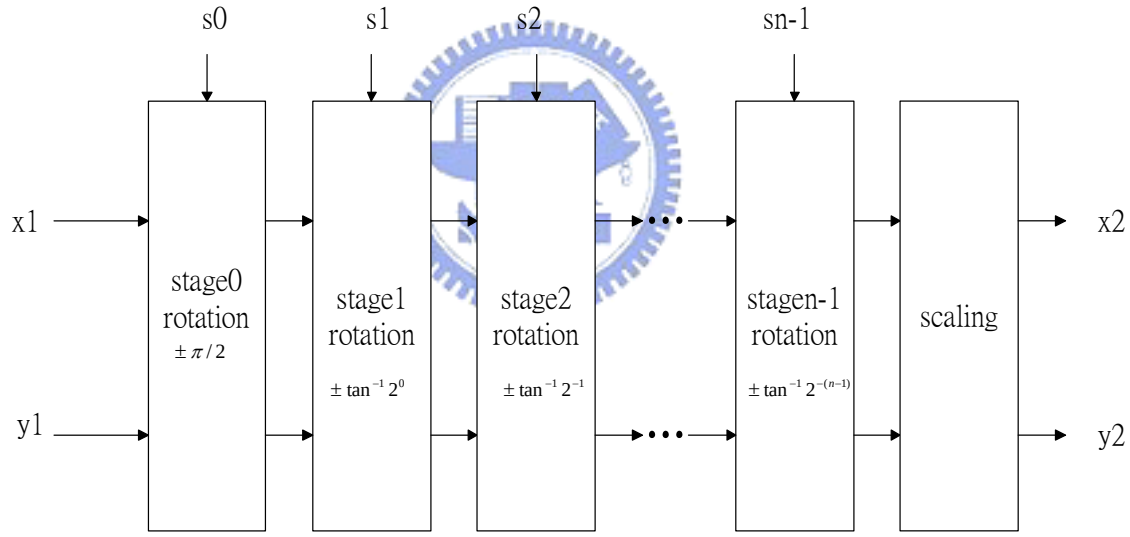
由此演算法發現我們只需要移位和加法的運算。

CORDIC 硬體實現：

給定所需要乘的交互因子(twiddle factor)角度，我們可算出 $\mu(i)$ 的值，接下來依照下列的式子產生位元碼 $s_{n-1}s_{n-2}\cdots s_0 s_{-1}$ 。

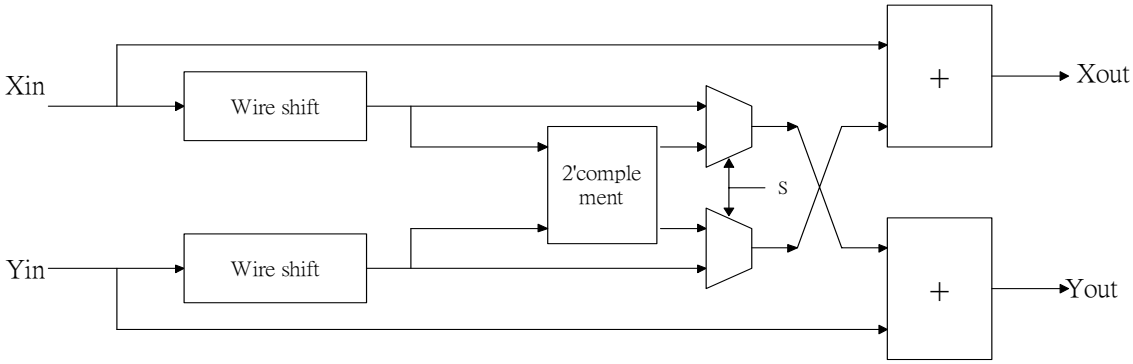
$$s_i = \begin{cases} 0 & \mu(i) = 1 \\ 1 & \mu(i) = -1 \end{cases} \qquad (4.2.8)$$

其中硬體架構圖見【圖六十一】。



【圖六十一】 CORDIC 的架構圖

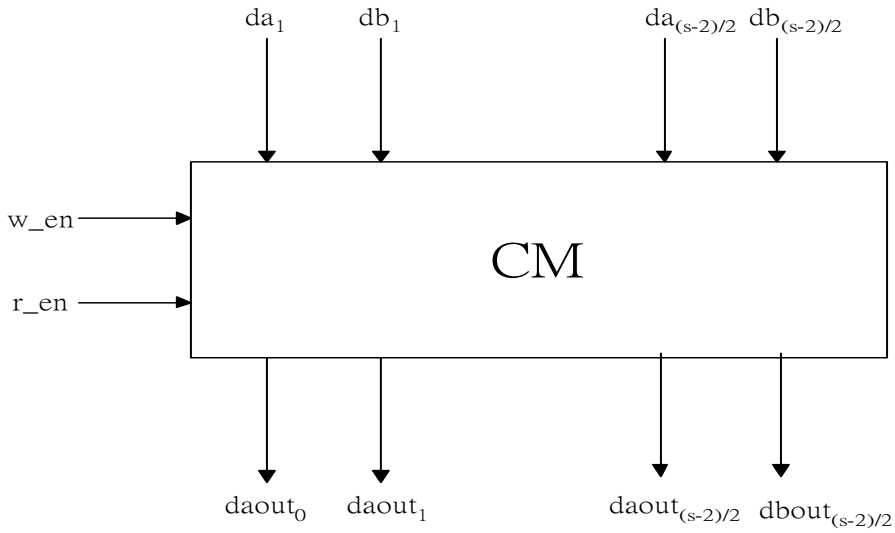
其中每一級的電路設計，見【圖六十二】。



【圖六十二】 CORDIC 每一級的架構

4.2.3 資料記憶體(DM)的設計

資料記憶體，也就是每一級當中，所需存放資料的記憶體。以 16 點為例，我們所需要記憶體大小為 N/2-2=6 個暫存器。見【圖六十三】。因為我們同時使用兩套硬體，所以在同一時間必須而有因為每一級有兩筆資料進入暫存器，也同時會有兩筆資料從暫存器輸出，所以對於 16 點的 FFT 為例，我們在同時會有 2(s-2)=4 個輸入端與輸出端，s 即為 $\log_2(N)$。

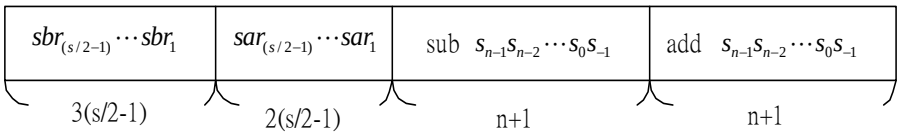


【圖六十三】 資料記憶體架構

4.2.4 係數記憶體(CM)的設計

此部分是給使用者使用，其中除了要存放 twiddle factor 之外，增加 1+5(s/2-1)個位元去控制每一級何時該乘 1、乘-j 與乘複數。而其中交互因子(twiddle factor)的係數即為存放 CORDIC 的碼，此長度會因為 CORDIC 的級數不同而長度會不同。

所以輸入的規格見【圖六十四】

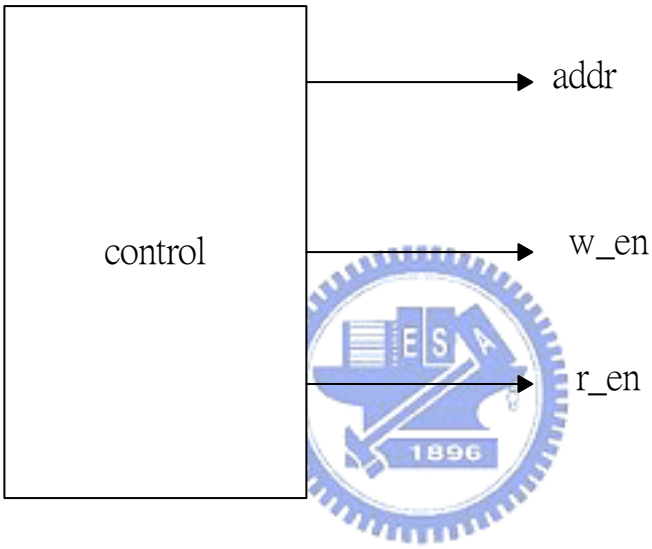


【圖六十四】 係數記憶體的輸入規格

其中後面兩個皆為 CORDIC 的輸入係數，而 sar 與 sbr 為控制何時要做 CORDIC 的運算。

4.2.5 控制單元的設計

控制器，要同時控制資料記憶體與係數記憶體的動作，所以我們的控制訊號線有 3 條，一條為讀出係數記憶體的位址為 $\log_2(N)$個位元，另兩條為產生要資料記憶體的寫入與讀出的控制線其中各有 N/2-2 個位元，見【圖六十五】。



【圖六十五】 控制單元架構

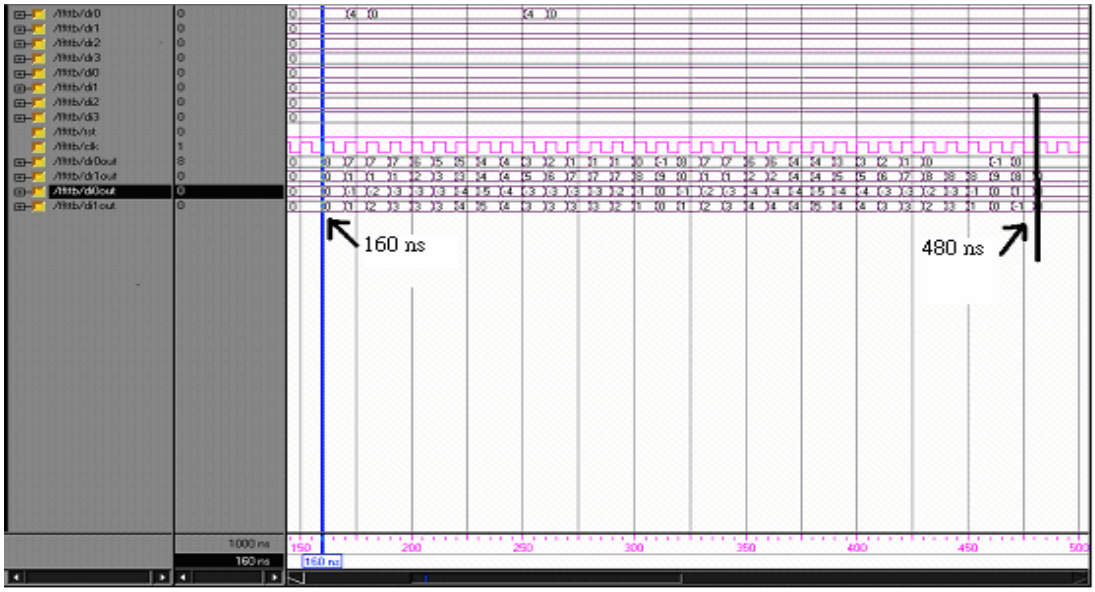
第五章 FPGA 之實現

為了要驗證我們的架構是可以套用到其他演算法，我們特別挑選了分割基數 24(Split-Radix)的演算法來做驗證，我們使用的輸入與輸出為 8 位元，8 位元加法器，乘法器則選擇用 9 階的 CORDIC，其使用的加法器為 10 位元。以下為其 behavior 和 RTL 的驗證。

5.1 Behavior 的驗證

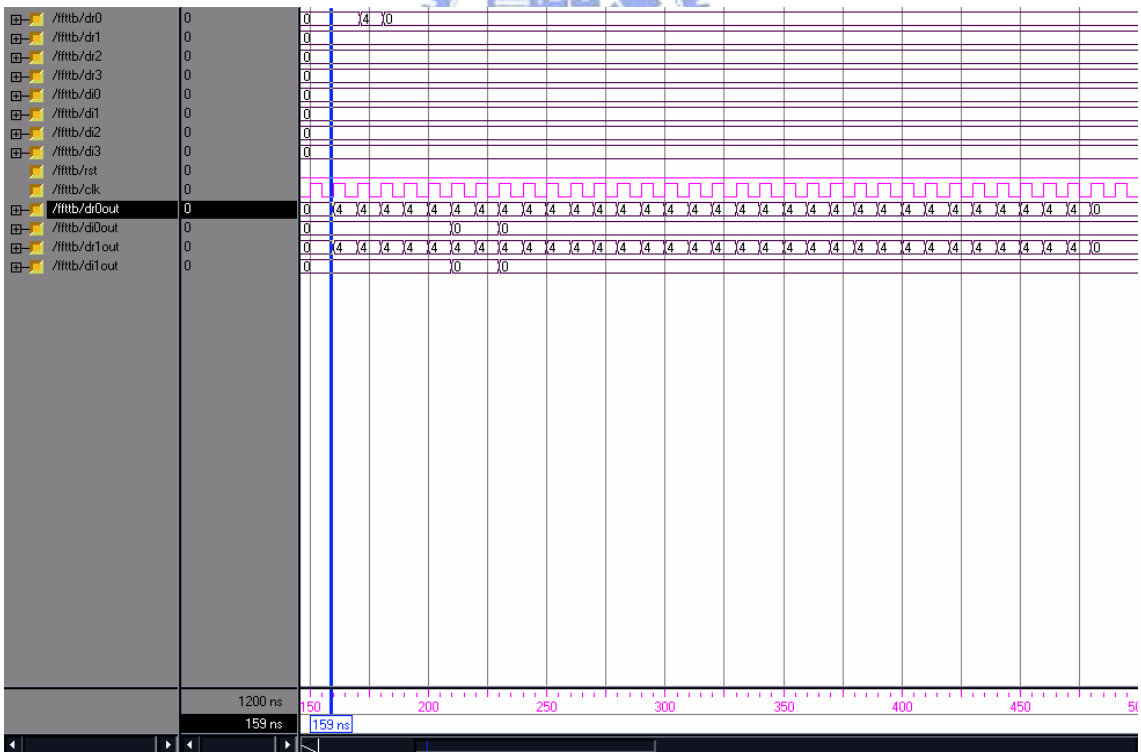
依照我們前一章的排程來控制每一級的流程。以下為 N=64 點的 VHDL behavior 模擬圖形。

其中輸入的資料為 x(1)=0.0625　與 x(2)=0.0625。輸出為一弦波。延遲時間(Latency)為 N/4 -1　clock cycle=160ns(clock =10ns)。總共計算完成為 3N/4　clock cycle = 480 ns。Behavior 的程式碼確定無誤。

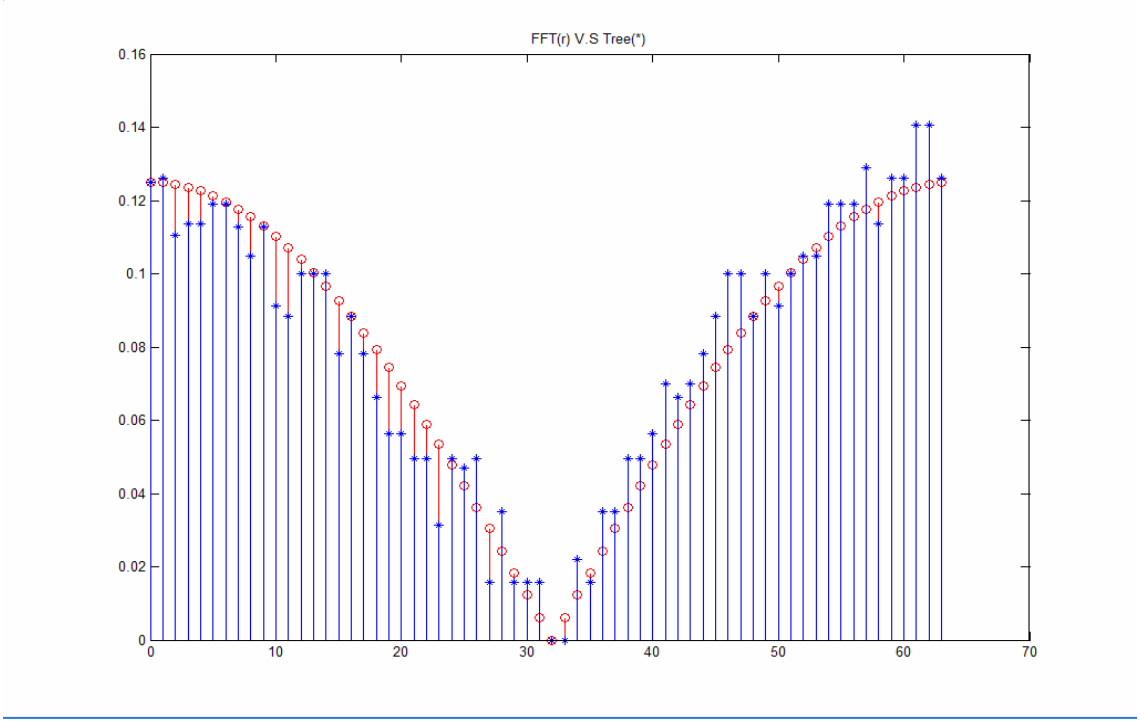


【圖六十六】 六十四點樹狀 FFT 硬體的模擬圖形

輸入的資料為 x(1)=0.0625。輸出為一方波。



【圖六十七】 六十四點樹狀 FFT 硬體的模擬圖形



【圖六十八】 六十四點樹狀 FFT 與 FFT 的比較

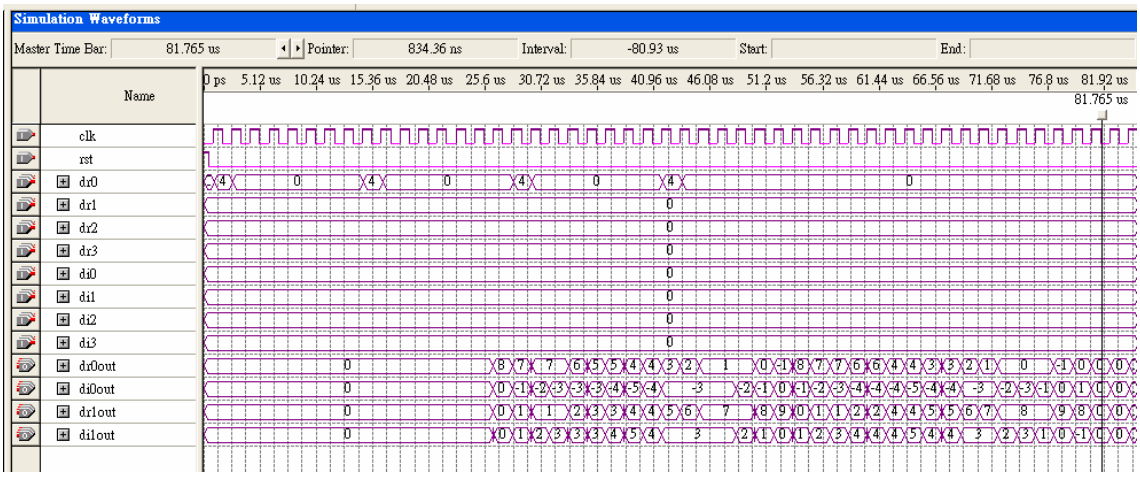
最大的誤差為：0.0222

5.2 RTL 的驗證

5.2.1 波形圖結果

我們使用 FPGA 的軟體做 RTL 的驗證，加入了 gate 的延遲。

以下為驗證圖形。



【圖六十九】 六十四點樹狀 FFT　RTL 的模擬圖形

5.2.2 面積結果

總面積：

總共需要 5012 個 LE，見【圖七十】。

Processing status	Fitting Successful
Timing requirements/analysis status	No requirements
Chip name	dpfft64
Device for compilation	EP20K1500EBC652-1
Total logic elements	5012 / 51840 (9 %)
Total pins	98 / 493 (19 %)
Total ESB bits	0 / 442368 (0 %)
Device for timing analysis	EP20K1500EBC652-1

【圖七十】 六十四點樹狀 FFT 總面積報告

係數記憶體的面積：

總共所需要 121 個 LE，佔 2.4%，見【圖六十九】。

Processing status	Fitting Successful
Timing requirements/analysis status	No requirements
Chip name	ctrs
Device for compilation	EP20K1500EBC652-1
Total logic elements	121 / 51840 (< 1 %)
Total pins	52 / 493 (10 %)
Total ESB bits	0 / 442368 (0 %)
Device for timing analysis	EP20K1500EBC652-1

【圖七十一】 六十四點樹狀 FFT 係數記憶體面積報告

資料記憶體面積：

【圖七十二】為一個實數的記憶體面積，所以複數的記憶體要乘 2，其面積為 1744 個 LE。佔 34.8%。

Processing status	Fitting Successful
Timing requirements/analysis status	No requirements
Chip name	fifo1
Device for compilation	EP20K1500EBC652-1
Total logic elements	872 / 51840 (1 %)
Total pins	190 / 493 (38 %)
Total ESB bits	0 / 442368 (0 %)
Device for timing analysis	EP20K1500EBC652-1

【圖七十二】 六十四點樹狀 FFT 資料記憶體面積報告

控制單元面積：

其面積為 62 個 LE。佔 2%，見【圖七十三】。

Processing status	Fitting Successful
Timing requirements/analysis status	No requirements
Chip name	codegen
Device for compilation	EP20K1500EBC652-1
Total logic elements	102 / 51840 (< 1 %)
Total pins	62 / 493 (12 %)
Total ESB bits	0 / 442368 (0 %)
Device for timing analysis	EP20K1500EBC652-1

【圖七十三】 六十四點樹狀 FFT 控制單元面積報告

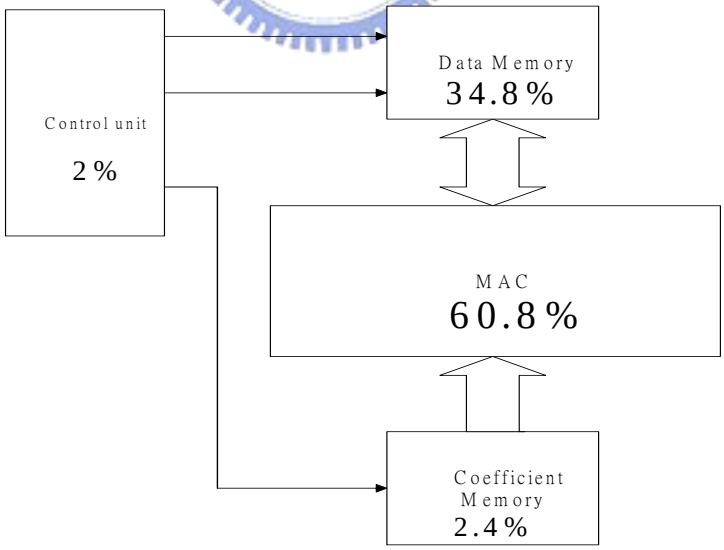
CORDIC 的乘法器面積：

單一個乘法器面積為 568 個 LE，見【圖七十四】。64 點需要 4 個乘法器，568x4=2272 個 LE。佔總面積的 45.3%。

Processing status	Fitting Successful
Timing requirements/analysis status	No requirements
Chip name	cord
Device for compilation	EP20K1500EBC652-1
Total logic elements	568 / 51840 (1 %)
Total pins	42 / 493 (8 %)
Total ESB bits	0 / 442368 (0 %)
Device for timing analysis	EP20K1500EBC652-1

【圖七十四】 六十四點樹狀 FFT 單一乘法器面積報告

根據以上的報告，我們算出六十四點的各個部分的面積分佈比例，見【圖七十五】。

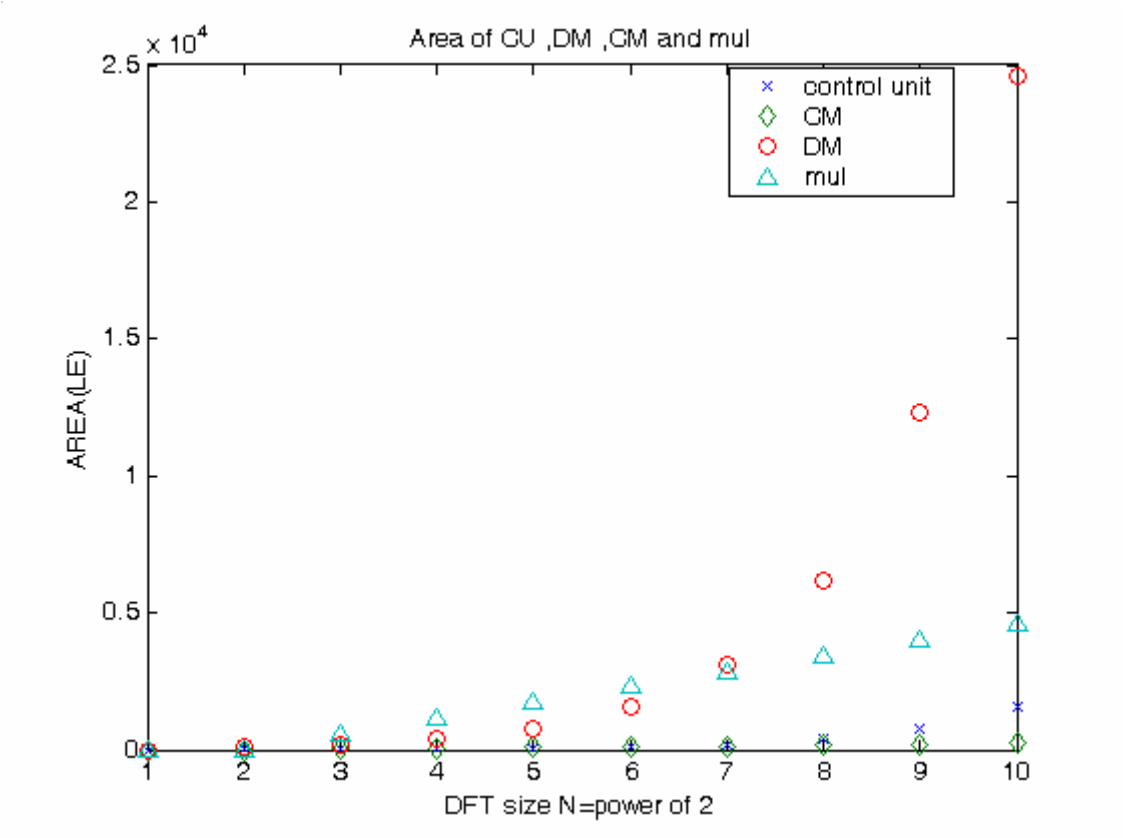


【圖七十五】 六十四點樹狀 FFT 面積分佈

5.3 硬體的分析與比較

5.3.1 面積分析

由以上的實現，我們可以發現當 N 擴大為 2^m 次方時，其面積的成長分佈圖，我們使用面積的單位為 LE。



【圖七十六】 FFT 面積分佈圖

由此可以發現，當點數越大，資料記憶體佔的面積會遠遠超過其他的面積。

5.3.2 硬體比較

以下為之前的一些硬體與我們提出的演算法的比較，樹狀的硬體在速度上有增快，為了要讓多種型態的 FFT 硬體需求可以使用，所以其乘法器的數目會比一般多一倍，在面積上就會增加。但乘法器我們已經使用較小的 CORDIC 方式改良。且由面積分佈圖發現，當點數到達高點數，資料記憶體的面積會佔比較多的比率。

硬體架構	複數乘法	複數加法	複數記憶體大小	乘法運算度
R2MDC	$2(\log_4 N-1)$	$4\log_4 N$	1.5N-2	Radix-2
R2SDF	$2(\log_4 N-1)$	$4\log_4 N$	N-1	Radix-2
R4MDC	$3(\log_4 N-1)$	$8\log_4 N$	2.5N-4	Radix-4
R4SDC	$(\log_4 N-1)$	$43og_4 N$	2N-2	Radix-4
R4SDF	$(\log_4 N-1)$	$48og_4 N$	N-1	Radix-4
$R2^2SDF$	$(\log_4 N-1)$	$4\log_4 N$	N-1	Radix-4
SRSDF	$(\log_4 N-1)$	$4\log_4 N$	N-1	Split-Radix
tree	$2(\log_2 N-1)$	$2\log_2 N$	N/2-2	Every type

【表四】 硬體的比較

第六章 結論與未來展望

6.1 結論

因為在應用面上不需要按照順序(in order)，所以在此情形下經過之前從排程至演算法的逐一驗證，確定我們發展出來樹狀的演算法是適用於各種在通訊系統上 FFT 的演算法，且因為把原本交叉的蝴蝶圖(butterfly)的圖形，轉成平行的樹狀圖，因此我們能夠去探討 MAC 數與暫存器之間的關係，讓使用者去選擇需要何種硬體需求，因為當 MAC 數增加了，相對的速度也可以增加，暫存器也可以減少，平行度也由以上的驗證得知。

6.2 未來展望

1.FFT 轉成四元樹

此一部份的硬體還在實現當中，目前演算法已經驗證完畢，確實無誤，接下來要著手此硬體的設計與驗證，讓平行度能夠比現在的架構增加一倍。

2.折疊方式(folding)

目前因為乘法器會佔去大部分的面積，所以應該接下來要著手如何設計使用硬體折疊(folding)的方式，如何去共享乘法器，藉此減少乘

法器的數目，對於面積應該會有所改善。

3.功率

因為乘法器如果減少，功率相對也會降低，所以如果能夠將乘法器的個數減少，之後希望用軟體能夠仔細去驗證這一方面，並設計出低功率的樹狀 FFT 硬體架構。

Reference

[1] T. Starr., J. M. Cioffi and P.J. Silverman, Digital Subscribe Line Technology.Englewood Cliffs, NJ: Prentice-Hall 1999.

[2] J. M. Cioffi and S. Olcer “Very high speed digital subscriber line” IEEE Communication Magazine, pp. 62-64, May 2000.

[3] M. D. Nava and C. Del-Toso, ”A short overview of the VDSL system requirements,” IEEE Communication Magazine, pp. 82-90, Dec. 2002.

[4] ”VDSL metallic interface. Part 1. ”Committee T1, Working Group T1E1.4, Contribution T1E1.4 /2000-009R3, Feb. 2003.

[5] ”VDSL metallic interface. Part 3.” Committee T1 , Working Group T1E1.4 , Contribution T1E1.4 /2000-0013R4, Feb. 2003.

[6] Lawrence Harte and Roman Kikta , Delivering xDSL. McGraw-Hill, New York, 2001.

[7] Dr. Dennis J. Rauschmayer, ADSL/VDSL Principles. Macmillan Technical Publishing, 1999.

[8] John A. C. Bingham, ADSL, VDSL, and Multicarrier Modulation. John Wiley & Sons, Inc, Canada, 2000.

[9] Oppenheim, Alan V., Ronald W. Shafer. Discrete-Time Signal Processing. Prentice Hall. Englewood Cliffs, New Jersey, pp. 587-610, 1989.

[10] Bracewell, Ronald N., The Fourier Transform and Its Applications, Second Edition. McGraw-Hill Book Company. New York, 1978. pp. 356-381.

[11] Roberts, Richard A., Clifford T. Mullis. Digital Signal Processing , Addision Wesley, Reading, Massachusetts, 1987. pp. 148-162.

[12] Saidi, Ali., “Decimation-In-Time-Frequency FFT Algorithm.” ICASSP94 Digital Signal Processing. Vol III. Institute for Electrical and Electronics Engineers. pp. 453-456, 1994.

[13] J. W. Cooley and J. W. Turkey, ”An algorithm for the machine calculation of complex Fourier series.” Math. Comput., vol.5, no.5, pp. 87-109, 1965.

[14] Tatyana D. Roziner, et. al., “Fast Fourier transform over finite groups by multiprocessor system,” IEEE Trans. on ASSP, vol. 38, no.2, pp. 226-239, Feb. 1990.

[15] P. Duhamel, “Algorithm meeting the lower bounds on the multiplicative complexity of length-2n DFT’s and their connection with practical algorithms”, IEEE Trans. Acoust., Speech, Signal Processing, vol. 38, September 1990.

[16] S. He and M. Torkelson, “Designing pipeline FFT processor for OFDM (de)Modulation,” in Proc. ISSSE, pp. 257-262, 1998.

[17] P.Duhamel and H. Hollomann, “Split-radix FFT algorithm,” Electron. Lett., vol. 20, pp. 14-16, Jan. 1984.

[18].W. C. Yen and C. W. Jen ,“High-speed and low- power split-radix FFT,” IEEE Trans. Acoust., Speech , Signal Processing, vol. 51, pp. 864-874, March. 2003.

[19] Michael Balducci, Aji. Choudary, J. Hamaker, ”comparative analysis of FFT algorithms in sequential and parallel form,” parallel DSP Group, pp. 5-16 , 1996 .

[20] H. Guo and C.S. Burrus. “Fast Approximate Fourier Transform via Wavelet Transform.” Proceedings of SPIE Conference on Mathematical Imaging. Denver, CO. August, 1996.

[21] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, 1975

[22] E. H. World and A. M. Despain, “Pipeline and parallel popeline FFT processors for VLSI implementation ,” IEEE Trans. Comput., vol. C-33, pp. 414-426, May 1984.

[23] A. M. Despain, “Fourier transform computer using CORDIC iterations,” IEEE Trans. Comput., vol. C-23, pp.993-1001, Oct. 1974.

[24] G. BI. And E. V. Jones, “A pipelined FFT processor for word-sequential data,” IEEE Trans. Acoust., Speech , Signal Processing, vol. 37, pp. 1982-1985, Dec. 1989.