

4-0 簡介

上一章是專門介紹快速傅立葉轉換之架構分類，在這一章會主要將 pipeline 架構(SDF)直接實現到硬體上測得其面積(gate count)，速度以及所消耗的平均動態 power，並且將前一章所介紹的架構做一些改良，以用來省下面積(gate count)或是動態 power。本章將實現的 FFT 點數(SDF 架構)有以下幾類：

1. 16-point FFT
2. 64-point FFT
3. 256-point FFT
4. 1K-point FFT

並且我們會針對乘法器部分以及 cordic 部分做些改良。

而基本上我們在做 pipeline 架構(SDF)FFT 的時候，記憶體會運用到兩種記憶體記憶方式



1. Register base
2. Ram base

Register base 的記憶體主要是運用 register 來儲存資料，而每經過一次 clock cycle 則資料便會自動做一次 shift，很適合用做在 pipeline(SDF)架構上使用，但是缺點就是和 Ram base 比較起來面機會較大一點，而且消耗 power 會多一點點(因為每一次 clock cycle 所以資料會做一次 shift)

而 Ram base 的架構就沒有 register base 架構上的缺點，但是卻必須在每一組 butterfly 之內再加上一組”指位器”，因為 ram 必須要用指位功能來存取資料，所以相較於 register base 架構，其 butterfly 的 control 會比較複雜一些，整體 FFT system 速度會被拉低一些

而以下我們討論的架構完全以 register base 為主，當然在此之前我們也要考慮到所有複數乘法器的架構比較以及之間的優缺點

目前我們所知道用在 FFT system 上的乘法器，除了最基本的複數乘法器以外，最常見的就是 cordic system，當然在前一章節我們也有說到所謂的 W8 block 架構，接著我們就是要先比較這些乘法器的硬體架構以及之間運行的優缺點

4-1 Cordic 的硬體架構簡介

對於一組 FFT 用的乘法器，我們先前已經敘述過，可以用 cordic 的方法來實現，而 cordic 是利用逼近的方法將原數值做極座標上的角度旋轉。但是因為 cordic 是利用逼近的方法，所以自然會花上比較多的時間來做逼近的步驟，而每一步驟其實只是將角度愈轉愈小直到角度的誤差值趨近零。

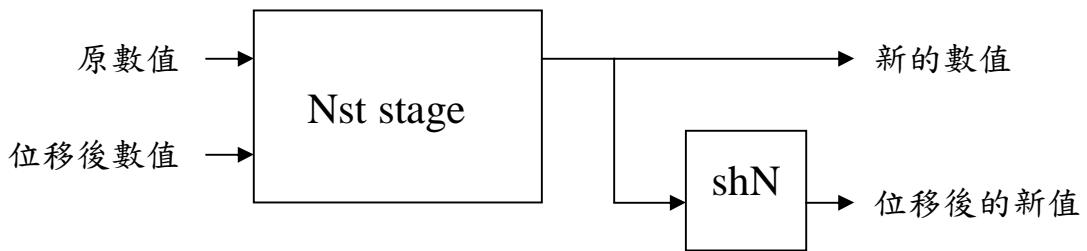
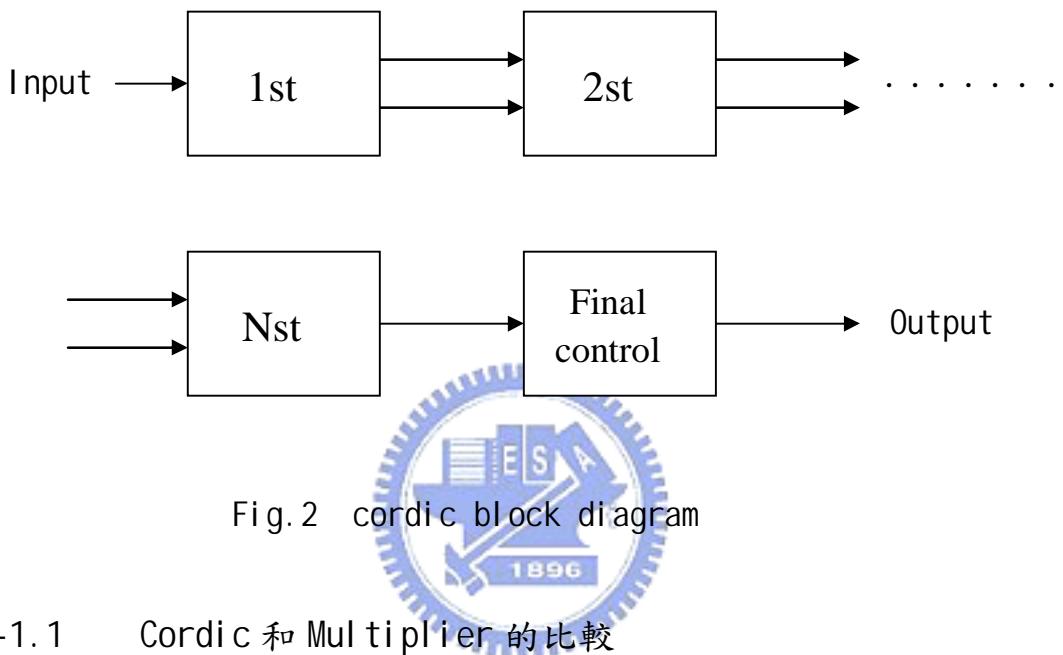


Fig 4-1.1 cordic 第 N 組逼近 block

如上圖 Fig 4-1.1 所顯示，每一組逼近都為一組 block，而每一組逼近用的 block 都由 *control*，三組加法器和兩組 *shifter* 組成。假設我們為了數值精準而逼近十五次，如此一來每做一次數值運算都要動用到四十五組加法器和十五組 *shifter*，並且耗時十七個 *clock*，並且最後的數值會放大約 1.64676 倍。(數值放大倍數會因為逼近次數多寡而改變，逼近次數愈多會愈接近 1.64676) 而整體的 cordic block diagram 如下圖 Fig. 2 所示：



4-1.1 Cordic 和 Multiplier 的比較

為什麼一般 FFT 系統的複數乘法器都用 Cordic 來替代呢？這裡有兩個主要的理由：

一. 加法器所用的個數：

我們知道，一個系統所用的加法器多寡決定了面積大小以及速度的快慢，假設我們輸入的數值長度為 16bits 長，那麼用一般的複數乘法器就等於是用了 62 組長度為 31bits 的加法器，比起 cordic 的大約 40~45 組 16bits 長度的加法器自然大上了許多面積以及 power 消耗。

二. 複數乘法器和 cordic 的 rom 難易度

對於一般複數乘法器的乘數必須要用 rom 來讀取，但是因為

一組複數有分實部跟虛部，假設現在是做 N-point FFT 轉換，那麼我們就必須要儲存 $2 \times N$ 個數值在 rom 裡面。而 cordic 我們要輸入的是轉的角度值，也就是說我們只需要儲存 N 個角度值到 rom 裡即可，甚至還可以利用 control 和 counter 來直接產生角度值，這是複數乘法器在此無法使用的一個優勢。所以比起複數乘法器的 rom 自然省上了許多 memory，並且方便設計。

所以就上面兩點來看，cordic 比一般乘法器來要優秀和方便這是確定無誤的，只差在 delay 時間過多。

4-1.2 Cordic 的缺點和避免之處

就如先前對 cordic 所敘述的，雖然消耗的 power 和面積比複數乘法器來的小，但是缺點就是時間延遲比較多，並且會放大逼近 1.64676 倍(就逼近 15 次狀況看)。這兩個缺點會對整個 FFT 的架構有很大的影響。

一. 時間延遲過久：

假設現在我們要做 8K-point FFT，那麼保守估計將要用到四組 cordic，也就是說，整個系統至少會多延遲了將近 68 的 clock cycles，對於一個講究快速的 pipeline-FFT 系統而言，這不是一個好現象。

二. 數值會放大 1.64676 倍(逼近 15 次狀況下)

數值放大 1.64676 倍，若是我們向第一章所說的在 cordic 最後一端接上一個實數乘法器(1/1.64676)的話，基本上還要再用到至少十組加法器和一些 shift，所以我們通常最後出來的結果就直接多用一個 bit 來表示(因為等於將近是將結果放大兩倍)簡單說，假設現在數值進入 cordic 之前是 12bits，從 cordic 出來之後要變成用 13bits 表示。這樣下來只要每經過一次 cordic，不管是使用實數乘法器或是加一個 bit 來表示結果，面積都會漸漸增大，消耗的 power 也會會漸漸變大，甚至到最後四捨五入過後精準度也會大打折扣。

所以為了避免上面的兩個問題，簡單的說，就是看看一個 FFT 系統中，能否盡量讓 *cordic* 使用次數減少(也就是盡量避免使用複數乘法器)，也就是說能否產生一種方法替代掉一些 *cordic* 的存在，這就是下面即將要討論的問題。

4-2 16-point FFT(SDF)的硬體實現討論

先前我們提過整個替代 W_8 乘法的產生我們稱作 W_8 block，是由 W_8 diagram 和其 control 所組成，而我們在此 block 之前再加上一組斷路控制是為了控制不必要多餘的加法或是處理發生，這樣可以省下一些 power，整個 W_8 block 的架構如下圖 Fig 4-2.1 所示：

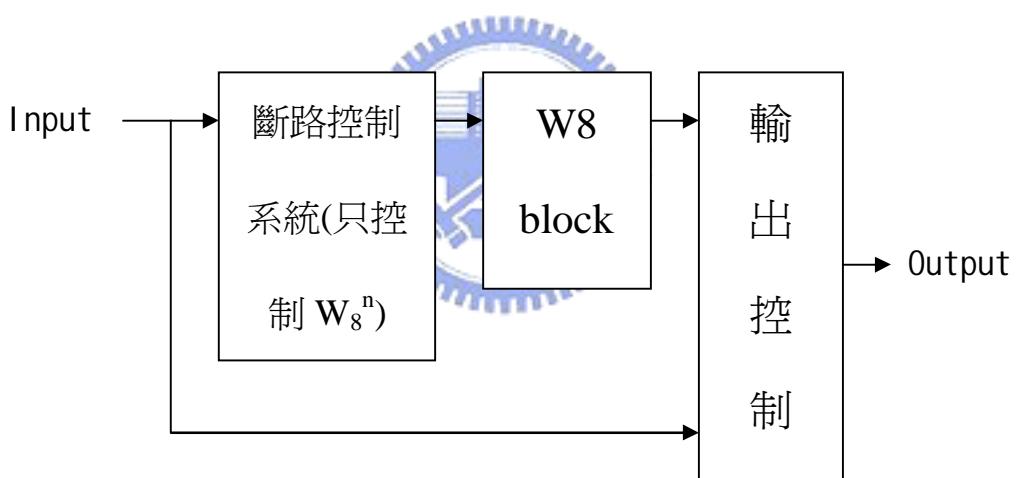
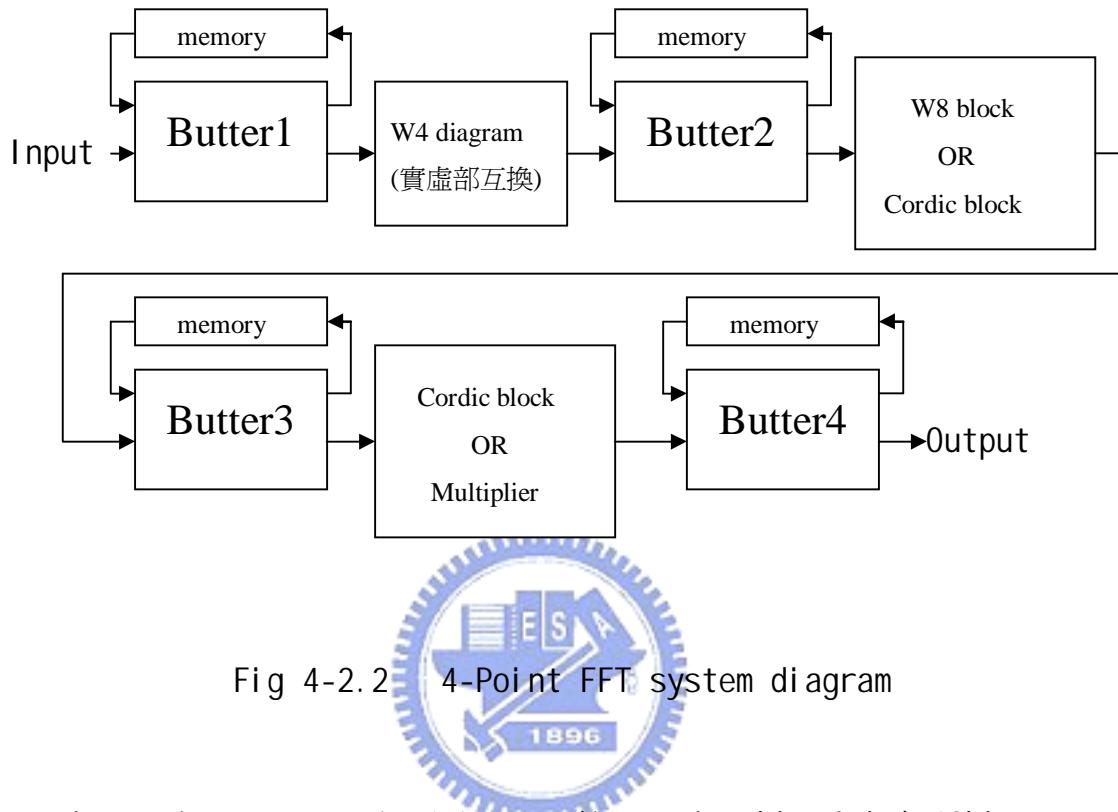


Fig 4-2.1 W_8 block(輸出控制要看 system 而定)

對於斷路控制是有必要的，因為所佔的面積其實非常有限，但是卻能夠省下一定量的 power，當然 W_8 block 的設計未必只有這種設計方法，還有別種設計方法可以省下更多的 power，但是缺點就是會用到非常大的面積去設計，這一點在後面的章節會加以討論。

利用 W8 block 和 cordic 的原理，我們可以很輕鬆的實現一組 16-point 的 FFT system，在此我們利用 radix- 2^3 的理論架構來實現此 FFT system，架構圖如下圖 Fig 4-2.2 所示：



在這裡我們可以做一個測試，假設輸入信號的 bit 長度為 12bits，如上圖，在用不同的乘法器條件之下，我們可以組合出六種組合，我們討論其中三種組合：

- 一. 第二組乘法器使用 Cordic block, 第三組乘法器使用 Multiplier:
- 二. 第二組乘法器使用 Cordic block, 第三組乘法器使用 Cordic block
- 三. 第二組乘法器使用 W8 diagram, 第三組乘法器使用 Cordic block

我們將以上三種組合實現在硬體上測試出來的面積，所消耗的平均動態 power，最後輸出的 bit 長度以及精準度(SNR)如下表(Table 4-2.1)所示：

	第一組	第二組	第三組
Gate count	58972	46646	28553
Dynamic Power	8.24mW	6.72mW	5.93mW
輸出 bit 長度	17bits	18bits	17bits
精準度(SNR)	45.08dB	44.89dB	45.20dB

Table 4-2.1: 三種不同架構 16-point FFT 數據比較

由上方數據我們可以很明顯的比較出來優劣以及得到一個重要的結論，那是一個複數乘法器的面積比 cordic 大約多了 10000 個 gate counts，而 cordic 又比 W8 block 大約多了 18000 個 gate counts，也證實了 W8 block 比 cordic 和複數乘法器來的省面積。

再者我們看 Power 和輸出 bit 長度的部分，複數乘法器所消耗的 power 比 cordic 多，而 cordic 又比 W8 block 消耗多 power，也證實了 W8 block 比 cordic 和複數乘法器來的省 power。而輸出 bit 的長度，我們可以觀察到若是每多一個 cordic 的經過，輸出的 bit 長度就必須多一個 bit，這對所消耗的 power 和所占面積也有不好的影響。

最後我們再看精準度，其實對於精準度而言三種方法其實都差不多，真的要勉強比出好壞的話，那們 cordic 是其中比較不好的，畢竟 cordic 是利用逼近的方法來旋轉角度，而且最後所放大的 1.64676 倍也有一定的誤差存在，而複數乘法器和 W8 block 則是差不多，因為都是利用確切的位數相加而成，所以誤差值都非常的小。

經由上面的綜合討論下來，W8 block 和 cordic 的原理和實現的確是比用最原始都只使用複數乘法器來的好得多。

4-2.1 對於 16-point FFT 的改良

對於 16-point FFT 我們再先前已經做過三組硬體上的實現和比較，但是值得思考的一件事情就是，作為 16-point FFT 的最後一組複數乘法器(W_{16})一定要用 *cordic* 來取代 *multiplier* 嗎？能否有更簡單的方法，更省面積的方法，更省 power 的方法？這是我們值得去長是看看的。

先前我們有提過 W8 block 的原理和設計，這的確讓我們省去不少面積和 power，同樣的道理，我們能否產生 W16 block 呢？若是可以的話，不只是原本的 16-point FFT 可以省下不少面積，更可以省下不少 power。

4-2.1.1 以 16 為基底的乘法替代法(W16 的乘法)

首先我們知道 W_{16} 都是以 22.5 度的角度來做旋轉，所以我們必須要先知道怎樣產生 $\sin(22.5^\circ)$ 以及 $\cos(22.5^\circ)$ ：

$$\begin{aligned} \sin(22.5^\circ) &= \cos(67.5^\circ) = 0.3926834 & \text{轉成二進位} &=> 0.011000100 \\ \sin(67.5^\circ) &= \cos(22.5^\circ) = 0.9238795 & \text{轉成二進位} &=> 0.111011001 \end{aligned}$$

由此我們可以知道，和 W8 diagram 是一樣的原理，只需要將信號進入幾組 shifter 之後即可得到旋轉過 22.5 度的值，架構如圖 Fig. 6 所示。而我們可以發現一件事情，就是 W16 diagram 和 W8 diagram 不一樣的是多了一組輸出信號，這是因為 W_{16} 是為 16 等分角，所以組合自然比 W_8 多了一倍之多，自然輸出也必須多一組才行。所以不難理解假設我們想做 W32 diagram 的話，輸出將會多達四組，所以相對的其 control 會變比較複雜。

而形成 W16 block 的 control 就比 W8 block 的 control 複雜了一點，下列舉幾個簡單的例子說明，假設原本實數部份為 A，虛數部分為 B，轉過 22.5 度實數部份 $a1=A*\sin(22.5^\circ)$ ，轉過 22.5 度虛數部份 $b1=B*\sin(22.5^\circ)$ ，轉過 67.5 度實數部份 $a2=A*\cos(22.5^\circ)$ ，轉過 67.5 度虛數部份 $b2=B*\cos(22.5^\circ)$ ：

若此複數轉了 W_{16}^1 ，則最後結果為：

實數部份為=>a2+b1

虛數部份為=>b2-a1

若此複數轉了 W_{16}^5 ，則最後結果為：

實數部份為=>b2-a1

虛數部份為=>-a2-b1

剩下的 W_{16}^n 以此類推，所以整個替代 $W16$ 乘法的產生我們稱作 $W16$ block (圖 Fig 4-2.4)，是由 $W16$ diagram 和其 control 所組成，相同的我們會加上斷路控制是為了控制不必要多餘的加法或是處理發生，而在此加上斷路控制的重要性就遠比 $W8$ block 來的重要的多，後面章節會詳細討論。

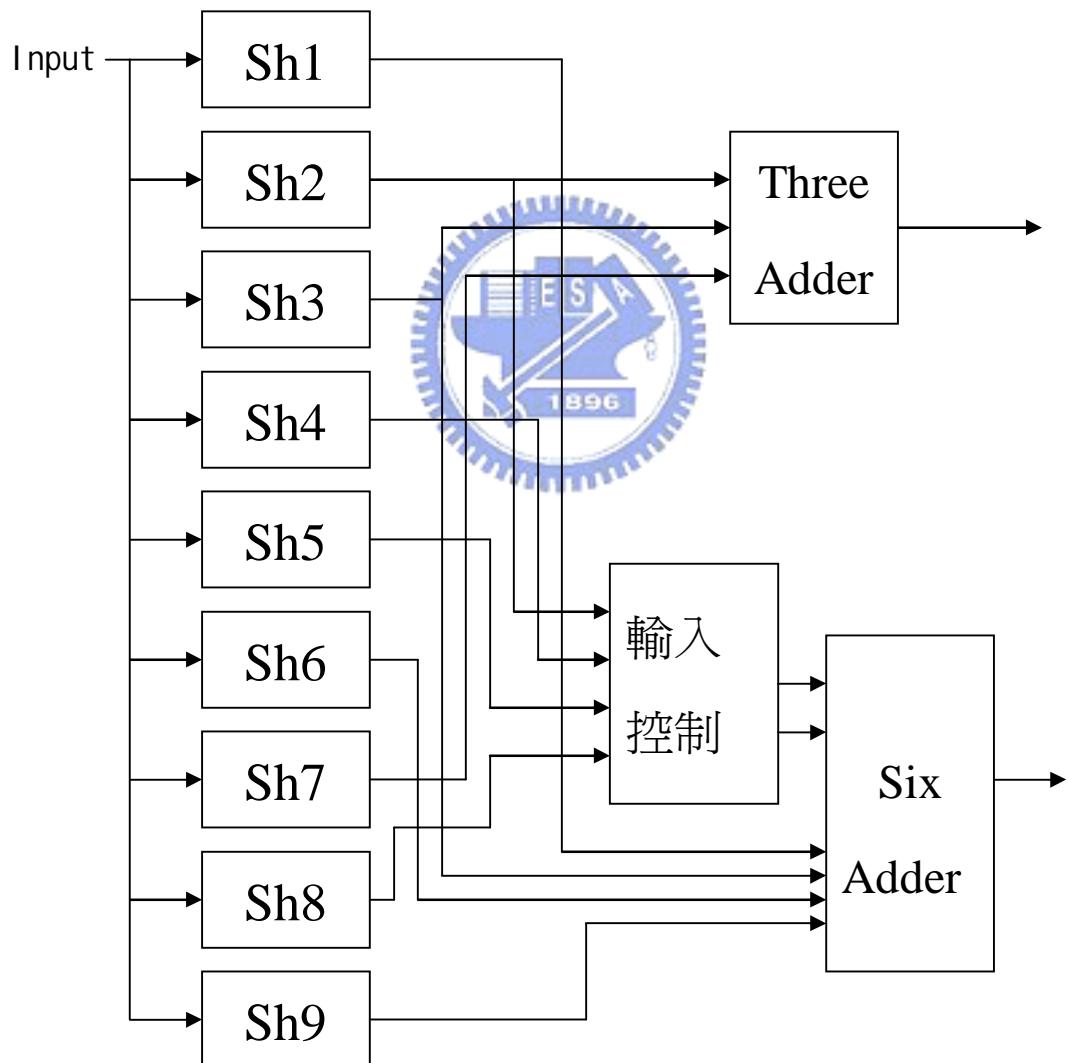


Fig 4-2.3 $W16$ diagram

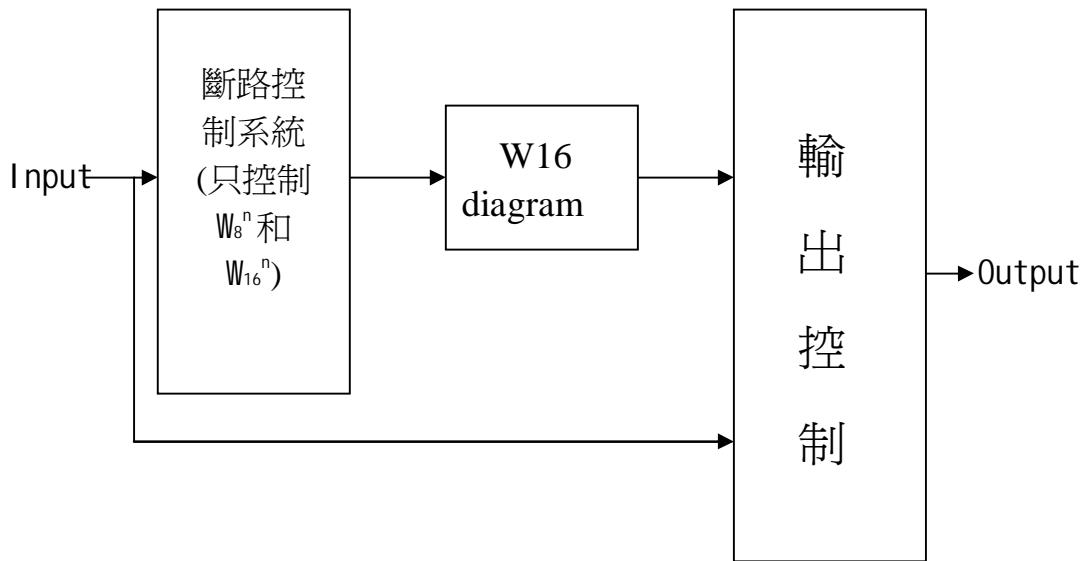


Fig 4-2.4 W16 block(輸出控制要看 system 而定)

Fig 4-2.3 運用了一組輸入控制，這是因為 W16 diagram 除了要產生 $W_{16}^{2^{n+1}}$ 系列的數還要產生 $W_{16}^{2^m}$ 系列的數，所以將 sixadder 拿來共用，只需要在其加法器之前加上一組控制輸入即可以達到其效果，面積也會節省不少(省了一組 sixadder 以及數組位移器)

在此我們將 W8 diagram 和 W16 diagram 拿來做比較，如下表 Table 4-2.2:

	W8 diagram(12bit)	W16 diagram(12bit)
面積(gate count)	1716	2942
平均消耗 power	0.89mW	1.12mW

Table 4-2.2 W8 diagram VS W16 diagram

可以看出，W16 diagram 因為比 W8 diagram 大約多了五組加法器，所以面積也大了一些，當然消耗 power 自然多一點，若是再拿 W16 block 和 W8 block 來做比較，因為 W16 block 包含了 W8 diagram 和 W16 diagram，並且加上較複雜的 control，面積會將近是 W8 block 的兩倍，但是卻還是遠小於 cordic 的面積。

4-2.2 經由 W16 block 改良過後的 16-point FFT

經由先前討論的 W16 block，我們現在將此運用到 16-point FFT 上，也就是將先前所設計的 16-point FFT 其中的 cordic 部分用 W16 block 來取代。首先我們比較一下 W16 block 和 cordic 的面積和消耗 power，如 Table 4-2.3 所示：

	W16 block(12bit)	Cordic system(12bit)
面積(gate count)	5978	14524
平均消耗的 power	1.91mW	2.38mW

Table 4-2.3 W16 block VS Cordic system

很明顯的可以看出來，不論是面積或是消耗 power 都是 W16 block 佔盡優勢，而且 W16 block 算出的結果並不會像 cordic 會放大 1.64676 倍，並且只需要用一個 clock cycle time 即可以算出結果，不像 cordic 必須要用到 14~17 個 clock cycle time 來算出結果。所以簡單說，W16 block 比起 cordic 的優勢有四點：

- 
- 一. 將近省了一半的面積(bit 長度愈大會差異愈明顯)
 - 二. 消耗的 power 比較少
 - 三. 算出結果不用多出一個 bit 來顯示(省面積和 power)
 - 四. 速度快上許多(1 cycle VS 14~17 cycles)

就以上四點的優勢，利用 W16 block 來設計 16-point FFT 可以價低整個 FFT 系統的面積和 power，並且增加速度。而我們將討論三種 16-point FFT 的架構來比較：

- 一. radix-2⁴ with cordic (W₁₆部分)
- 二. radix-2⁴ with W16 block and W8 block
- 三. radix-2² with W16 block

我們將以上三種組合實現在硬體上測試出來的面積，所消耗的平均動態 power，最後輸出的 bit 長度以及精準度(SNR)如下表 Table 4-2.4 所示(輸入信號長度為 12 bit)：

	第一組	第二組	第三組
Gate count	28553	20946	18013
Dynamic power	5. 93mW	5. 22mW	4. 85mW
輸出 bit 長度	17bits	16bits	16bits
精準度(SNR)	45. 20dB	46. 31dB	47. 34dB

Table 4-2.4: 三種不同架構 16-point FFT 數據比較之二

由此可以很明顯的比較出來先前所預測的結果，運用 W16 block 來實現 16-point FFT 的確比用 cordic，不論在哪一方面，都還要來的優秀。

在此，要特別先說明，並不是在任何點數架構上的 FFT system 運用 W16 block 來替代 cordic 就會比較省面積和 power，這要看 FFT system 的運用架構(radix-N)以及 FFT 的點數而言。換言之，若是運用架構和 FFT 的點數都適宜，會省上更多的面積和消耗 power(Table 1-4 的第二組和第三組即是如此)，但是相反的，若是運用架構和 FFT 點數都不適宜甚至是衝突，不但面積增大，連 power 都會反增加，但是不論如何，速度變快是不變的事實，這些細節部分會在以後討論。

現在先討論為什麼第二組設計和第三組設計都是用 W16 block 代替 cordic，但是第三組卻比第二組還要好。這是因為差異在架構上，radix- 2^2 以及 radix- 2^4 。若是運用 radix- 2^4 的架構我們會發現三組乘法器分別是 W4 block, W8 block, W16 block；而運用 radix- 2^2 的架構我們會發現三組乘法器分別是 W4 block, W16 block, W4 block。所以這兩組架構其實就是差在 radix- 2^4 用到了 W8 block 而 radix- 2^2 用到的只是 W4 block，其中的面積，power，SNR 上的差異就是由這個地方所產生的不同。

所以這裡已經先映證了，架構上的差異性會影響整個 FFT system 的面積大小以及消耗 power，甚至是 SNR。

4-3 Radix-2² 和 Radix-2³ 以及 Radix-2⁴ 之數學演算式簡單化

第二章我們有介紹過 Radix-2² 和 Radix-2³ 的數學演算式，乍看之下感覺很複雜，其實這些是有一定的簡單規律，可使得我們輕易的找出任意點數 FFT system 運用 Radix-2² 或是 Radix-2³ 演算法的情況下，把每一階的 twiddle factor 給找出來，以下我們先用 16-point FFT with Radix-2² 做例子：

首先我們必須知道 N-point FFT 運用 Radix-2² 的時候，其中為第奇數個乘法器的 twiddle factor 必定為 w_4^0 ， w_4^0 ， w_4^0 ， w_4^1 的順序排列，好比說 16-point FFT，一共有三個乘法器，其中第一個和第三個乘法器就是依照這個順序來產生 twiddle factor，但是重點是我們一定要先決定最後一個奇數組乘法器(第三組)的 twiddle factor，其實也就是最基本的 w_4^0 ， w_4^0 ， w_4^0 ， w_4^1 的順序一值重複下去，然後再推到前一組第奇數組(第一組)乘法器的 twiddle factor，其推算方法就是將後一奇數組(第三組)的 twiddle factor 每一次都做四次重複動作，也就是 w_4^0 四次， w_4^0 四次， w_4^0 四次， w_4^1 四次，然後再全部一值重複下去，依此類推，第 $2n-1$ 組的 twiddle factor 一定是第 $2n-3$ 組 twiddle factor 個個先四倍重複之後才開始做全部循環。

至於偶數部分的 twiddle factor 比較麻煩一點，基本上是要先求第一組偶數組的 twiddle factor。要求第一組偶數組的 twiddle factor 之前，我們必須先了解一種簡單的數列，假設這種數列有 8 點，則此數列為 [0 4 2 6 1 5 3 7]，假設有 16 點則此數列為 [0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15]，大家應該可以發現，這數列其實就是 N-point FFT 的最後輸出順序。其實在每一階的 twiddle factor 中我們都會用到此數列來找到該 stage 的 twiddle factor 排序，以下我們就先利用這數列來求 16-point FFT with Radix-2² 第一組偶數組(第二組)的 twiddle factor。

首先，16-point FFT 第二個乘法器之前有 2 個 stage，所以我們要求 $4(16/2^2)$ 點的上述數列，也就是 $[0\ 2\ 1\ 3]$ ，然後我們以 $4(2^2)$ 點為一組，將 twiddle factor 排出如 Fig4-3.1：

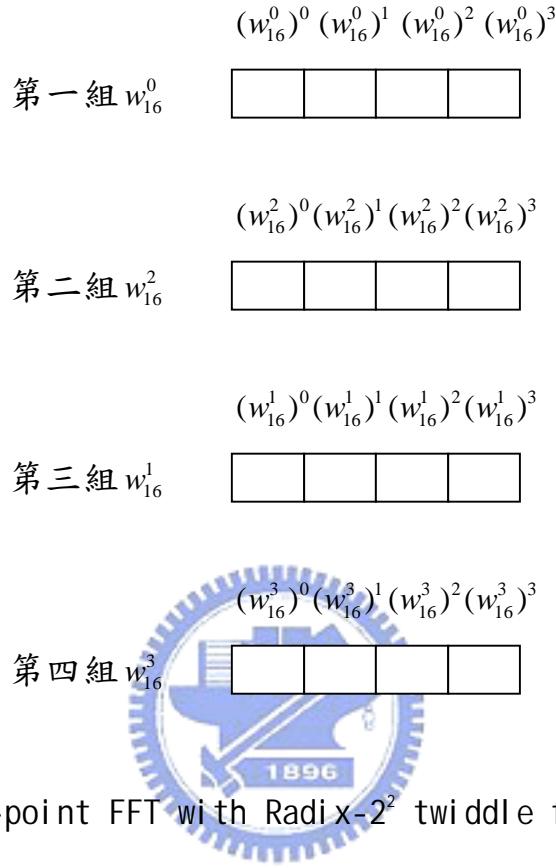


Fig 4-3.1 16-point FFT with Radix-2² twiddle factor 排序法

我們可以觀察前一章 Fig3-2.4，我們上敘所說導出來的 twiddle factor 和用演算是求出來的是一樣的。

我們在舉一個例子，64-point FFT with Radix-2²，其第奇數乘法器部分的 twiddle factor 求法跟上敘一樣，只要先把最後一個奇數(第五個)乘法器的 twiddle factor 求出來就可以求出前面奇數(第一個和第三個)乘法器的 twiddle factor。而偶數部分的求法也一樣，我們可以知道第二個乘法器之前還有 2 個 stage，所以我們要求 $16(64/2^2)$ 點的上述數列，也就是 $[0\ 8\ 4\ 12\ 2\ 10\ 6\ 14\ 1\ 9\ 5\ 13\ 3\ 11\ 7\ 15]$ ，然後我們以 $4(2^2)$ 點為一組，即可以將第二組乘法器的 twiddle factor 排出；同理，跟在球奇數點乘法器的 twiddle factor 幾乎一樣，第四個乘法器的 twiddle factor 就是將第二組乘法器的前四組 twiddle factor(0 8 4 12)每一個重複四次動作，即是第四組乘法器的 twiddle factor

第二組和第四組乘法器的 twiddle factor 排列如 Fig4-3.2 和 Fig4-3.3 所示：

第 1 組 $(w_{64}^0)^0 \ (w_{64}^0)^1 \ (w_{64}^0)^2 \ (w_{64}^0)^3$

--	--	--	--

第 3 組 $(w_{64}^4)^0 \ (w_{64}^4)^1 \ (w_{64}^4)^2 \ (w_{64}^4)^3$

--	--	--	--

第 5 組 $(w_{64}^2)^0 \ (w_{64}^2)^1 \ (w_{64}^2)^2 \ (w_{64}^2)^3$

--	--	--	--

第 7 組 $(w_{64}^6)^0 \ (w_{64}^6)^1 \ (w_{64}^6)^2 \ (w_{64}^6)^3$

--	--	--	--

第 9 組 $(w_{64}^1)^0 \ (w_{64}^1)^1 \ (w_{64}^1)^2 \ (w_{64}^1)^3$

--	--	--	--

第 11 組 $(w_{64}^5)^0 \ (w_{64}^5)^1 \ (w_{64}^5)^2 \ (w_{64}^5)^3$

--	--	--	--

第 13 組 $(w_{64}^3)^0 \ (w_{64}^3)^1 \ (w_{64}^3)^2 \ (w_{64}^3)^3$

--	--	--	--

第 15 組 $(w_{64}^7)^0 \ (w_{64}^7)^1 \ (w_{64}^7)^2 \ (w_{64}^7)^3$

--	--	--	--

第 2 組 $(w_{64}^8)^0 \ (w_{64}^8)^1 \ (w_{64}^8)^2 \ (w_{64}^8)^3$

--	--	--	--

第 4 組 $(w_{64}^{12})^0 \ (w_{64}^{12})^1 \ (w_{64}^{12})^2 \ (w_{64}^{12})^3$

--	--	--	--

第 6 組 $(w_{64}^{10})^0 \ (w_{64}^{10})^1 \ (w_{64}^{10})^2 \ (w_{64}^{10})^3$

--	--	--	--

第 8 組 $(w_{64}^{14})^0 \ (w_{64}^{14})^1 \ (w_{64}^{14})^2 \ (w_{64}^{14})^3$

--	--	--	--

第 10 組 $(w_{64}^9)^0 \ (w_{64}^9)^1 \ (w_{64}^9)^2 \ (w_{64}^9)^3$

--	--	--	--

第 12 組 $(w_{64}^{13})^0 \ (w_{64}^{13})^1 \ (w_{64}^{13})^2 \ (w_{64}^{13})^3$

--	--	--	--

第 14 組 $(w_{64}^{11})^0 \ (w_{64}^{11})^1 \ (w_{64}^{11})^2 \ (w_{64}^{11})^3$

--	--	--	--

第 16 組 $(w_{64}^{15})^0 \ (w_{64}^{15})^1 \ (w_{64}^{15})^2 \ (w_{64}^{15})^3$

--	--	--	--

Fig 4-3.2 64-point FFT with Radix-2² 第二組乘法器 twiddle factor 排序法

$$(w_{64}^0)^0 (w_{64}^0)^0 (w_{64}^0)^0 (w_{64}^0)^0 (w_{64}^0)^1 (w_{64}^0)^1 (w_{64}^0)^1 (w_{64}^0)^1 (w_{64}^0)^2 (w_{64}^0)^2 (w_{64}^0)^2 (w_{64}^0)^2 (w_{64}^0)^3 (w_{64}^0)^3 (w_{64}^0)^3 (w_{64}^0)^3$$

--	--	--	--	--	--	--	--	--	--	--	--	--	--

第一組

$$(w_{64}^8)^0 (w_{64}^8)^0 (w_{64}^8)^0 (w_{64}^8)^0 (w_{64}^8)^1 (w_{64}^8)^1 (w_{64}^8)^1 (w_{64}^8)^1 (w_{64}^8)^2 (w_{64}^8)^2 (w_{64}^8)^2 (w_{64}^8)^2 (w_{64}^8)^3 (w_{64}^8)^3 (w_{64}^8)^3 (w_{64}^8)^3$$

--	--	--	--	--	--	--	--	--	--	--	--	--	--

第二組

$$(w_{64}^4)^0 (w_{64}^4)^0 (w_{64}^4)^0 (w_{64}^4)^0 (w_{64}^4)^1 (w_{64}^4)^1 (w_{64}^4)^1 (w_{64}^4)^1 (w_{64}^4)^2 (w_{64}^4)^2 (w_{64}^4)^2 (w_{64}^4)^2 (w_{64}^4)^3 (w_{64}^4)^3 (w_{64}^4)^3 (w_{64}^4)^3$$

--	--	--	--	--	--	--	--	--	--	--	--	--	--

第三組

$$(w_{64}^{12})^0 (w_{64}^{12})^0 (w_{64}^{12})^0 (w_{64}^{12})^0 (w_{64}^{12})^1 (w_{64}^{12})^1 (w_{64}^{12})^1 (w_{64}^{12})^1 (w_{64}^{12})^2 (w_{64}^{12})^2 (w_{64}^{12})^2 (w_{64}^{12})^2 (w_{64}^{12})^3 (w_{64}^{12})^3 (w_{64}^{12})^3 (w_{64}^{12})^3$$

--	--	--	--	--	--	--	--	--	--	--	--	--	--

第四組



Fig 4-3.3 64-point FFT with Radix-2² 第二組乘法器
twiddle factor 排序法

接著我們必須知道 N-point FFT 運用 Radix-2³ 的時候，其中為第 3n+1 個乘法器的 twiddle factor 必定為 $w_4^0, w_4^0, w_4^0, w_4^1$ 的順序排列，第 3n+2 個乘法器的 twiddle factor 必定為 $w_4^0, w_4^0, w_4^0, w_4^1, w_4^0, w_4^0, w_8^0, w_4^0, w_8^0$ 的順序排列，只是當 Radix-2² 是每向前求一次 twiddle factor 等於要重複四次此時的 twiddle factor，Radix-2³ 則是每向前求一次 twiddle factor 等於要重複八次此時的 twiddle factor。至於第 3n 個乘法器的求法跟在 Radix-2² 的偶數像乘法器求法是一樣的，只是也差在是要取前 N/8 個 twiddle factor 然後每一個重複八次，即可以組成後一組第 3n 組乘法器的 twiddle factor

所以依照前面的規律，我們可以很輕易的推算出 N-point FFT 運用 Radix-2⁴ 的 twiddle factor 的規律法則。其中為第 4n+1 個乘法器的 twiddle factor 必定為 w_4^0 ， w_4^0 ， w_4^0 ， w_4^1 的順序排列，第 4n+2 個乘法器的 twiddle factor 必定為 w_4^0 ， w_4^0 ， w_4^0 ， w_4^1 ， w_4^0 ， w_8^1 ， w_4^0 ， w_8^3 的順序排列，第 4n+3 個乘法器的 twiddle factor 必定為 w_4^0 ， w_4^0 ， w_4^0 ， w_4^1 ， w_4^0 ， w_8^1 ， w_4^0 ， w_8^3 ， w_4^0 ， w_4^1 ， w_{16}^0 ， w_4^0 ， w_{16}^5 ， w_4^0 ， w_{16}^3 ， w_4^0 ， w_{16}^7 的順序排列，而 Radix-2⁴ 則是每向前求一次 twiddle factor 等於要重複十六次此時的 twiddle factor。至於第 4n 個乘法器的求法跟在 Radix-2² 的偶數像乘法器求法是一樣的，只是也差在是要取前 N/16 個 twiddle factor 然後每一個重複十六次，即可以組成後一組第 4n 組乘法器的 twiddle factor

所以很明顯的可以知道，只要運用此規律法則，不必去推導出複雜的數學演算式，也可以很方便的找出 N-point FFT 運用不同架構時乘法器所需要的 twiddle factor，這對於架構上的修改方面會比較方便。而接下來我們也可以發現，Radix-2² 的第 2n 組乘法器 twiddle factor，Radix-2³ 的第 3n 組乘法器 twiddle factor，Radix-2⁴ 的第 4n 組乘法器 twiddle factor，幾乎都是要用 cordic(或是一般複數乘法器)才行，而剩下部份都可以用 W4 block，W8 block 以及 W16 block 來實現。

4-4 64-point FFT(SDF)的硬體實現討論

對於 64-point FFT 的 SDF 硬體架構大概可以用下方 Fig 4-4.1 來表示：

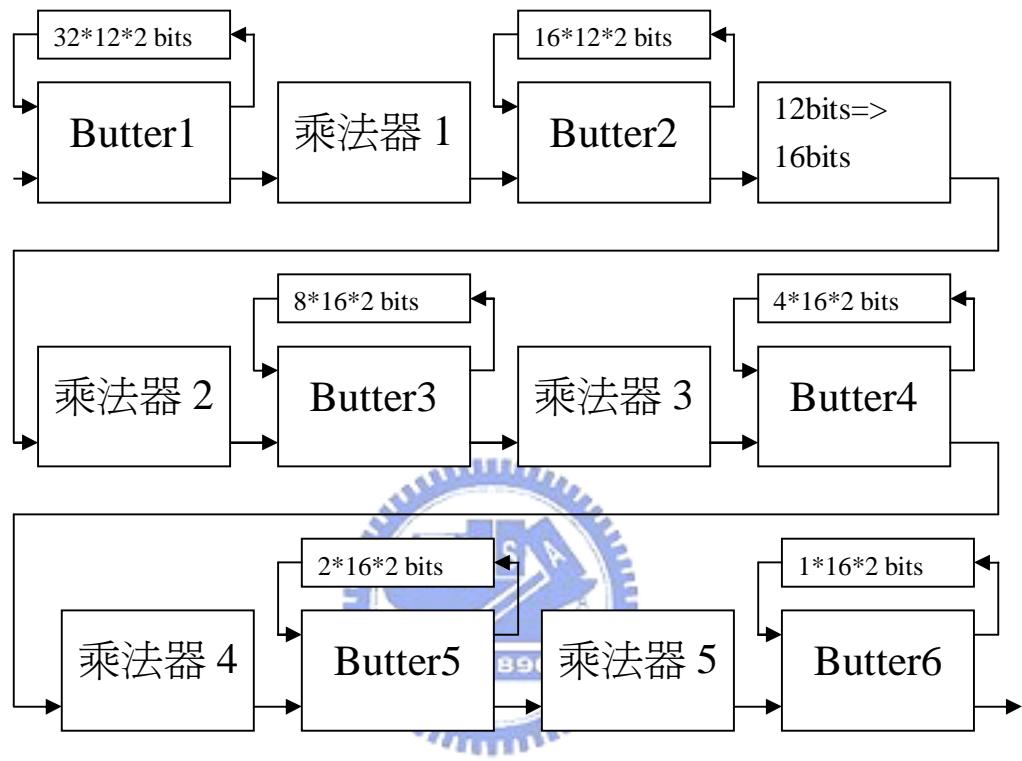


Fig 4-4.1 64-point FFT(SDF)通用硬體架構

對於輸入信號都以 12bits 輸入，其絕對值不超過 1，並且前八個 bit 用來表示小數部分，如 Fig 4-4.2 所示。

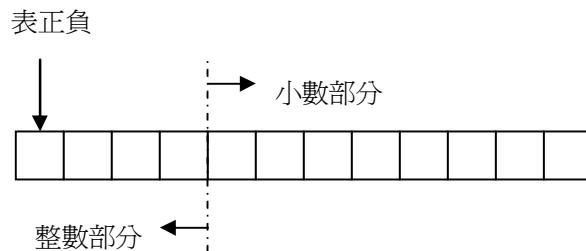


Fig 4-4.2 輸入信號的表示

而此架構中 *butterfly* 是不會有任何更改的問題，至於為什麼要加上”12 bit to 16 bit”這個 block，是因為每經過一次 *butterfly* 之後其數值最大可能會放大成兩倍，而經過 *butterfly* 之後，其數值已最少放大四倍左右，所以之後的數值 bit 數必須擴充或是做 quantize 的動作，而我們為了簡單起見，每一次遇到此問題都以四個 bits 來做擴充，因為在此做 quantize 動作會使得精準值誤差實在太嚴重，故在中間處理階段不做任何 quantize 動作，直到最後輸出結果再做適當的 quantize。

而在這一節主要要討論的就是乘法器 1 到乘法器 5 的設計，哪一個該用 W4 block，哪一個該用 W8 block，哪一個該用 W16 block 或是使用 cordic system。基本上我們可以將上述分成以下幾種組合類型來討論：

- 一. *Radix-2² with Cordic system*
- 二. *Radix-2² with W16 block and Cordic system*
- 三. *Radix-2³ with W8 block and Cordic system*

經由先前所述，我們已經可以知道 *Radix-2² 64-point FFT(SDF)* with Cordic system，其中第二組和第四組乘法器要用 cordic 來做運算，而至於第一組，第三組和第五組乘法器則用 W4 block 來替代一般複數乘法器來做運算。

Radix-2² 64-point FFT(SDF) with W16 block and Cordic system，其中第二組乘法器要用 W16 block 來做運算，第四組乘法器要用 cordic 來做運算，而至於第一組，第三組和第五組乘法器則用 W4 block 來替代一般複數乘法器來做運算。

Radix-2³ 64-point FFT(SDF) with W8 block and Cordic system，其中第一組和第四組乘法器要用 W4 block 來做運算，第二組和第五組乘法器要用 W8 block 來做運算，而至於第三組乘法器則用 cordic 來替代一般複數乘法器來做運算。

而每一組架構的詳細硬體圖以及每一架構每一級所需要的 bit 數請參考附錄一和附錄二。

我們求出每一級至少所需要的 bit 數之後，便可以找出哪一個地方需要接上”12 to 16 bits block”，甚至是需要加上”16 to 20 bits block”。

而此三個架構所分析出來的數據如下表 Table 4-4.1 所示：

	第一組	第二組	第三組
Average Dynamic Power	8.43mW	7.33mW	7.92mW
Gate count (面積)	57619	49705	52029
Total delay	97 cycles	81 cycles	82 cycles
Memory size(bits)	1656 bits	1632 bits	1632 bits
Performance(SNR dB)	43.435 dB	42.483 dB	42.127 dB

Table 4-4.1 三種 64-Point FFT(SDF) 數據整理比較

由此表我們可以很明顯的發現各種架構的差異性，首先我們看第一組 (Radix-2² 64-point FFT(SDF) with Cordic system) 以及第二組 (Radix-2² 64-point FFT(SDF) with W16 block and Cordic system) 之間，第一組架構比第二組架構多用了一個 cordic system，而第二組架構是運用 W16 block 替代了 cordic system，所以面積省了大約 8000 gate counts，而 power 也省了大約有 1mW 左右，也因為 cordic system 少了一組的關係，所以速度也比第一組快了 16 clock cycles。

至於 performance 部分，因為第二組是在最後一階級的時候做了 1bit 的 quantize 動作，所以 SNR 比第一組低了一些，但是也不過差了 1dB 左右，所以就第一組和第二組來比較，我們可以發現一個重點，就是我們一直強調的”減少 cordic system 的運用”，顯現出來的優點：

- 一. 面積明顯減少許多
- 二. power 減少許多
- 三. 速度比較快

而實際上要是我們在最後一即不做 quantize 動作的話，SNR 會比第一組來的好(大約為 44.433dB)，我們也可以發現第一組所用的 memory size

比第二組和第三組來的大一點點，這是因為第一組多用了一組 cordic system 造成進位(多一個 bit)所影響，面積也會隨之放大。

接著我們觀察第二組(Radix-2² 64-point FFT(SDF) with W16 block and Cordic system)和第三組(Radix-2³ 64-point FFT(SDF) with W8 block and Cordic system)架構的差異。第三種架構是現在 64-Point FFT(SDF)最普遍最常使用的架構，這是因為利用 Radix-2³ 數學演算架構的關係，而使得這架構剛剛好是呈現對稱的型態，所以就設計層面上比起前兩種架構簡單許多，也比較直觀。

我們把這架構拿來跟第二種架構比較的話，其實會比第二種架構來的稍微差一點點而已，其實真正用在實作上，這一點點的小差異不會去太在意，畢竟這只是 64-point FFT 而已，但是若是點數變大，假設用 256-point FFT 或是 1K-point FFT 上，那這差異就會被慢慢拉大，所以我們還是要找出其中差異原因所在。

我們可以由附錄一所顯示的架構看出，第二種架構 Radix-2² 64-point FFT(SDF) with W16 Block and Cordic System，運用了三個 W4 Block，一個 W16 Block，一組 cordic system，而第三種架構 Radix-2³ 64-point FFT(SDF) with W8 Block and Cordic System，則是運用了兩個 W4 block，兩個 W8 Block 以及一組 cordic system，剩下的架構則完全一樣。所以簡化一下，兩種架構關鍵差異性就在：

第二架構中 12bits 信號長度的 W16 block 加上 16bits 信號長度的 W4 block V.S 第三架構中 12bits 信號長度的 W8 block 加上 16bits 信號長度的 W8 block

因為 W4 block 的硬體架構根本佔不到什麼硬體空間(因為只是做實部虛部交換，用不到任何加法器或是位移器)，而我們前面也提過說，雖然 W16 block 面積將近是 W8 block 的兩倍，但是這裡有一組 W8 block 的信號長度卻是 16bits，再加上兩組 W8 block 有不同的 system control，所以才會造成面積上還是以 W16 block(12 bits)小於兩組 W8 block(12bits and 16bits)約有 3000 個 gate count。

至於 power 的方面，我們先分析出來了 W4 block, W8 block, W16 block(12bits)單獨做運算時所消耗的 Average dynamic power，如下表 Table 4-4.2 所示：

System	Average power
W4 block(12 bits)	0.48mW
W8 block(12 bits)	1.36mW
W16 block(12 bits)	1.91mW

Table 4-4.2 W4 W8 W16 block average dynamic power

依照上表，我們已經可以明顯發現為什麼兩組 W8 block 所消耗的 power 會比一組 W16 block 加上一組 W4 block 來的大，原因有二：

- 一. W8 block 所消耗的 power 並不是 W16 的一半，況且兩組 W8 block 中有一組信號長度是為 16bits
- 二. W4 block 所消耗的 power 非常的小，幾乎可以省略掉

綜合上面所有的比較，其實將 64-point FFT(SDF)以 $Radix-2^2$ 64-point FFT(SDF) with W16 Block and Cordic System 來實現是最好的結果，但是或許 $Radix-2^3$ 64-point FFT(SDF) with W8 Block and Cordic System 比較好設計，再加上其中差異性非常有限，所以目前來是比較常用 $Radix-2^3$ 架構。但是當 FFT 點數加大的時候，利用 W16 block 的優點就會更明顯的顯現出來，所以接下來我們要做 256-point FFT(SDF)的實作和分析。

4-5 256-point FFT(SDF)的硬體實現討論

對於 256-point FFT 的 SDF 硬體架構大概可以用下方 Fig 4-5.1 來表示：

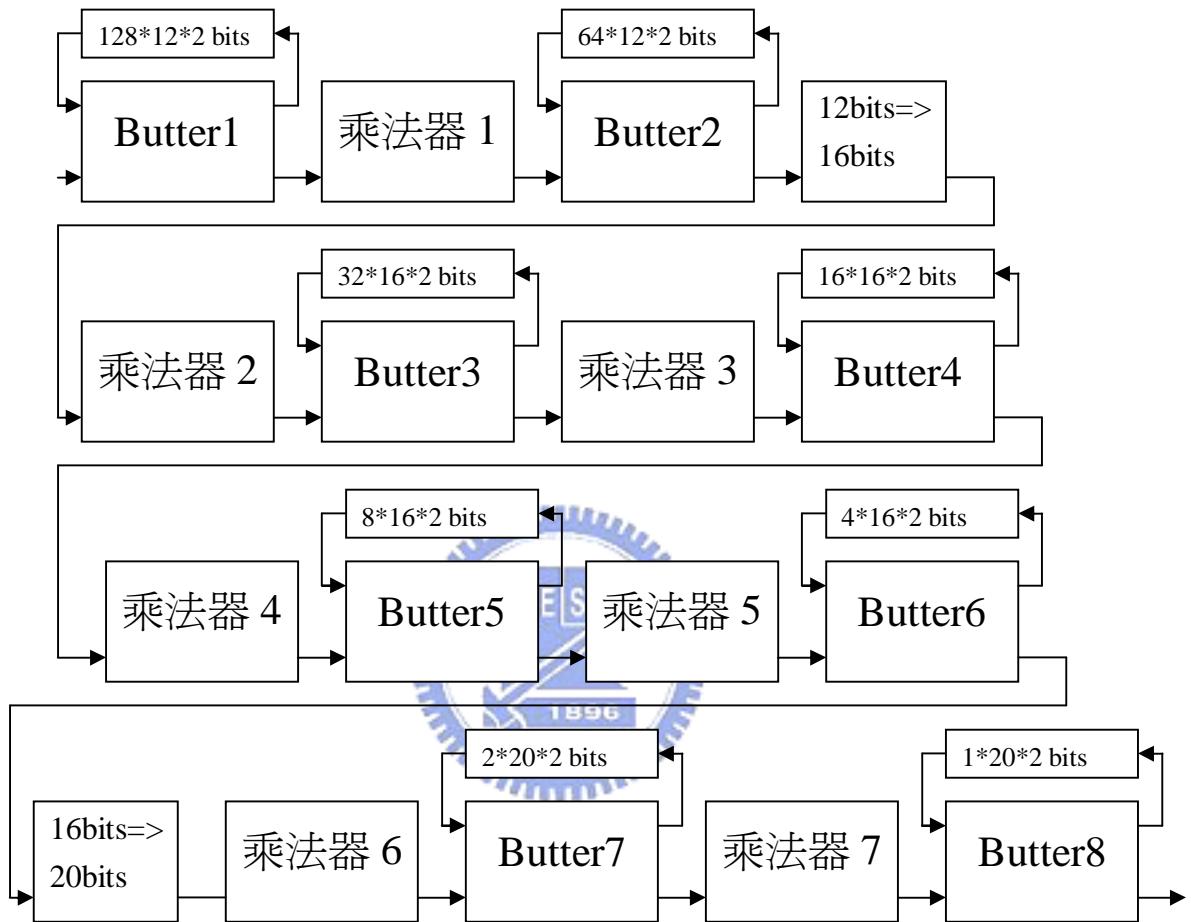


Fig 4-5.1 256-point FFT(SDF)通用硬體架構

當然此架構的”12bits to 16bits block” 和”16bits to 20 bits block”的位置還是看其真正架構為何才能決定真正擺設位置，這就如上一節所說的，這兩個 block 的位置會影響其總體架構面積以及 power。

而在這一節主要討論的是乘法器 1 到乘法器 7 的設計，哪一個該用 W4 block，哪一個該用 W8 block，哪一個該用 W16 block 或是使用 cordic system。基本上我們可以將上述分成以下幾種組合類型來討論：

- 一. Radix-2² with W16 block and Cordic system
- 二. Radix-2³ mix Radix-2² with Cordic system
- 三. Radix-2⁴ with W16 block and Cordic system
- 四. 改良後的 Radix-2² with W16 block and Cordic system

經由先前簡化演算法所述，我們已經可以知道 Radix-2² 256-point FFT(SDF) with W16 block and Cordic system，其中第一組和第三組和第五組以及第七組乘法器要用 W4 block 來做運算，第二組乘法器要用 W16 block 來做運算，而至於第四組和第六乘法器則用 cordic 來替代一般複數乘法器來做運算。

Radix-2³ mix Radix-2² 256-point FFT(SDF) with Cordic system，其中第一組和第四組以及第七組乘法器要用 W4 block 來做運算，第二組和第五組乘法器要用 W8 block 來做運算，而至於第三組和第六乘法器則用 cordic 來替代一般複數乘法器來做運算。

Radix-2⁴ 256-point FFT(SDF) with W16 block and Cordic system，其中第一組和第五乘法器要用 W4 block 來做運算，第二組和第六組乘法器要用 W8 block 來做運算，第三組和第七乘法器用 W16 block 來做運算，而第七組乘法器要用 cordic 來替代一般複數乘法器來做運算。

至於第四種架構是運用經驗法則所產生的架構，結合了以上三種架構的優點所產生，我們稍後再討論這第四種架構。

而每一組架構的詳細硬體圖以及每一架構每一級所需要的 bit 數請參考附錄三和附錄四。

我們求出每一級至少所需要的 bit 數之後，便可以找出哪一個地方需要接上” 12 to 16 bits block ” 以及” 16 to 20 bits block ”。而此三架構信號長度並不會超過 20 bits 。

而此三個 256-point FFT 架構所分析出來的數據如下表 Table 4-5.1 所示：

	第一組	第二組	第三組
Average Dynamic Power	11.80mW	11.94mW	10.98mW
Gate count (面積)	121788	122764	108866
Total delay	290 cycles	292 cycles	276 cycles
Memory size(bits)	6680 bits	6680 bits	6680 bits
Performance(SNR dB)	37.943 dB	37.169 dB	39.276 dB

Table 4-5.1 三種 256-Point FFT(SDF) 數據整理比較

由以上的分析數據可以先比較第一組(Radix-2² 256-point FFT(SDF) with W16 block and Cordic system)和第二組(Radix-2³ mix Radix-2² 256-point FFT(SDF) with Cordic system)架構，這兩架構都運用到了兩組 cordic system，雖然運用架構不一樣，但是其結果卻很接近，所以我們可以得到一個訊息，就是一個 FFT system 的面積大小，power 消耗等等，幾乎可以說是由所運用的 cordic system 數量決定。

那麼我們在來看，為什麼第一組架構會比第二組架構來的好一些。在第一組架構中，除了兩組 cordic system 之外，我們運用了幾乎都是 W4 block(有一組 W16 block)，而第二組架構中是運用了兩組 W8 block 以及兩組 W16 block。而我們先前比較過知道，一組 W16 block 的綜合效能比兩組 W8 block 來的好一些，而 W4 block 又是最好的效能，所以經過綜合比較可以知道，第一組架構因為 W4 block 用的比較多，所以整體效能比第二組來的好一些(沒有差很多是因為 W16 block 把效能降下來了一點)。所以在此我們可以知道一個重點，就是一個 FFT system 中，W4 block 愈多會使得整體面積和消耗 power 都會降低。

接著我們觀察第三組(Radix-2⁴ 256-point FFT(SDF) with W16 block and Cordic system)架構和前兩組架構的差異，由於是運用了 W16 block 以及 Radix-2⁴ 架構的原因，所以整體架構我們只用到了一組 cordic system 而已。

首先我們看面積的部分，因為少了一組 cordic system 而且可以說是用 W16 block 來替代它，所以面積大大減少了將近有 14000~15000 個 gate

counts，也因為這原因 *average dynamic power* 也幾乎降低了 1mW 左右，而速度也因為減少了一組 *cordic system* 以致於快了 14~16 個 *clock cycles*。甚至我們觀察一下 *performance*，也因為是利用 *W16 block* 替代了 *cordic system* 的原因，所以 *SNR* 升高了約有 2dB 左右，因此也可以證明出來 *W16 block* 不只是面積，*power*，速度比 *cordic system* 好，甚至連精準度都比 *cordic system* 好一些。

所以由這一組架構已經很清楚的說明，減少 *cordic system* 的存在是可以將整個架構的性能提升最好方法，但是前提就是要先設計出能夠適當替代 *cordic system* 的有效架構。接下來我們利用一些經驗法則，將 *Radix-2⁴* 以及 *Radix-2²* 的優點結合在一起，設計出第四種 256-point FFT(SDF) system。

首先我們要知道 *Radix-2²* 以及 *Radix-2⁴* 的優點在哪裡。

Radix-2² 的優點：利用到多組的 *W4 block*，省面積和 *power*

Radix-2⁴ 的優點：利用到 *W16 block*，可減少 *cordic system* 使用

換言之，我們希望能夠設計出一種能多利用 *W4 block* 並且又可以適當的使用 *W16 block* 來替代 *cordic system* 的 256-point FFT(SDF) 架構。而在此前提下，我們必須先觀察一下 *Radix-2⁴* 256-point FFT(SDF) with *W16 block* *Cordic system* 架構。

依照 *Radix-2⁴* 的數學演算法則，我們可以發現第五組乘法器的 *twiddle factor* 是呈現 $w_4^0, w_4^0, w_4^0, w_4^1$ 的循環，第六組乘法器的 *twiddle factor* 是呈現 $w_4^0, w_4^0, w_4^0, w_4^1, w_4^0, w_8^1, w_4^0, w_8^3$ 的循環，而第七組乘法器的 *twiddle factor* 是呈現 $w_4^0, w_4^0, w_4^0, w_4^1, w_4^0, w_8^1, w_4^0, w_8^3, w_4^0, w_16^1, w_4^0, w_16^5, w_4^0, w_{16}^3, w_4^0, w_{16}^7$ 的循環。當然，依照先前提過的簡化法則，第一組第二組第三組乘法器的 *twiddle factor* 只是分別將第五組第六組第七組的 *twiddle* 個別先重複 16 次之後再開始循環運作。

接著我們再仔細觀察第五第六第七組乘法器的 twiddle factor，和先前設計 Radix- 2^4 16-point FFT(SDF)的各階段乘法器的 twiddle factor，我們可以觀察到這兩組 twiddle 是一樣的，這顯示了一個非常重要的訊息，就是 256-point FFT(SDF)其就是先用十六組 16-point FFT(SDF)並聯之後再串接到一組 16-point FFT(SDF)上，如 Fig 4-5.2 所示：

接著我們可以回想一下 16-point FFT(SDF)最省 power，面積的是哪一種架構？就是 Radix- 2^2 16-point FFT(SDF) with W16 block 架構，所以我們只需要將其要並連的 16-point FFT(SDF)和要串接的 16-point FFT(SDF)都以 Radix- 2^2 16-point FFT(SDF) with W16 block 架構替換掉原先的 Radix- 2^4 16-point FFT(SDF) with W16 block 即可，這樣子我們可以想像的出來，先前所希望的兩個優點：

- 一. 希望 W4 block 能夠多一些(此架構有 4 組 W4 block)
- 二. 希望能夠適當的用 W16 block 替代 cordic system(此架構用到兩組 W16 block 以及一組 cordic system)

都已經在此架構中實現出來，當然我們也要先看一下每一個 butterfly 以及每一個乘法器至少所需要的 bit 數為多少。其每個階段所需要的 bit 數如下表 Table 4-5.2 所示：

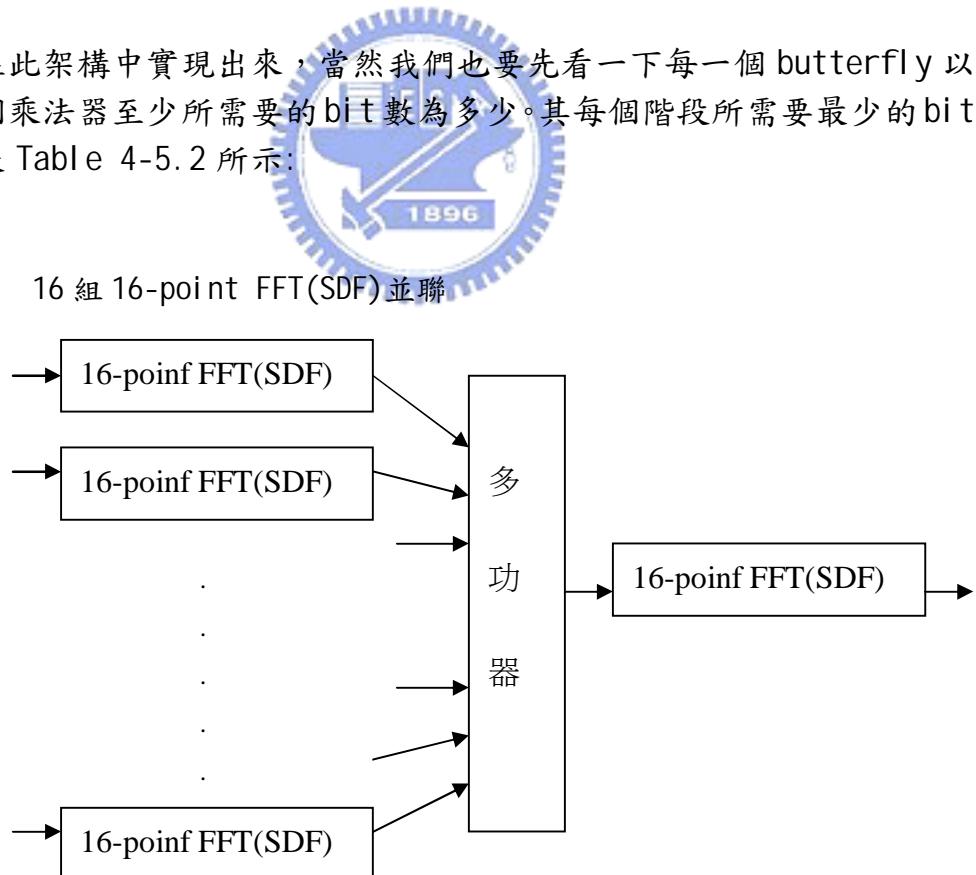


Fig 4-5.2 256-point FFT(SDF)以 16-point FFT(SDF)組成圖

Stage	Bit 數	Stage	Bit 數	Stage	Bit 數
Butter1	11bits	W4block2	13bits	Butter6	17bits
W4 block1	11bits	Butter4	14bits	W16block2	17bits
Butter2	12bits	cordic	15bits	Butter7	18bits
W16block1	12bits	Butter5	16bits	W4 block4	18bits
Butter3	13bits	W4block3	16bits	Butter8	19bits

Table 4-5.2 改良後 Radix-2² 256-point FFT with W16 block and Cordic system 各階層至少所需要的 bit 數

而此架構所設計出來的硬體圖如下圖 Fig 4-5.3 所示：

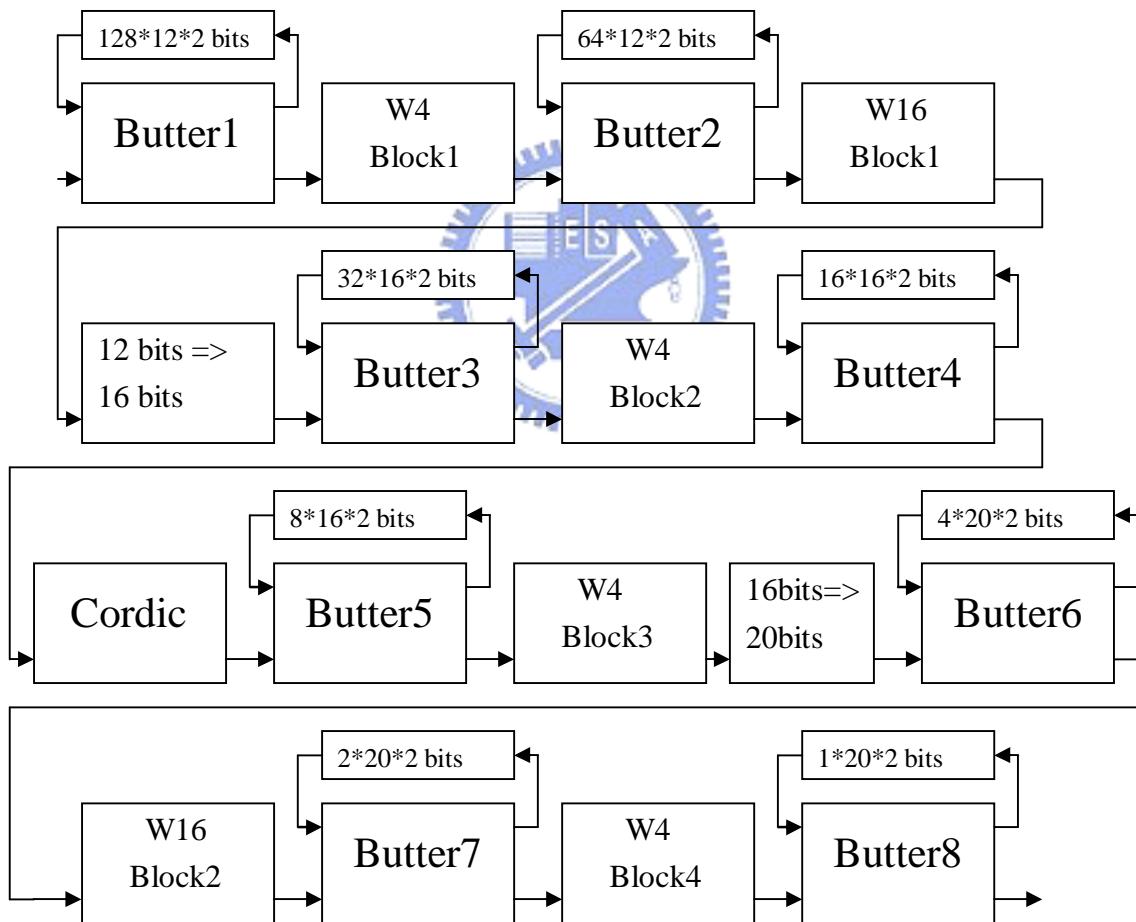


Fig 4-5.3 改良後 Radix-2² 256-point FFT with W16 block and Cordic 的硬體架構

在分析其數據之前我們可以先拿 Radix-2^4 256-point FFT with W16 block and Cordic 架構來做猜測性的比較，此架構跟 Radix-2^4 架構其實主要就是差在此架構利用了兩組 W4 block(12 bits 和 16 bits)替代掉了 Radix-2^4 架構的兩組 W8 block(12 bits and 16 bits)，所以我們可以預期新架構的 SNR，面積和消耗 power 都會進一步的改善，而此架構運作分析數據如表 Table 4-5.3 所示：

Average Dynamic Power	10.26mW
Gate count (面積)	94088 gate count
Total delay(clock cycles)	272 clock cycles
Memory size(bits)	6680 bits
Performance(SNR dB)	40.014 dB

Table 4-5.3 改良後 Radix-2^2 256-point FFT with W16 block and Cordic system 分析數據

根據以上數據已經很明顯可以看出來，此種新架構的確是四種 256-point FFT(SDF)架構中最優秀的一個，不論是面積，power，速度以及 performance 都是最好的，但是其數學演算式的用法就比較特殊一點，這靠最基本 Radix-2^2 以及 Radix-2^3 的數學演算式是無法推算出來的。

我們做完了以上四種 256-point FFT(SDF)的硬體架構實現，得到了以下幾點重要設計概念：

- 一. *Cordic system*(複數乘法器)能盡量用別種方法替代則盡量替代
- 二. W4 block 能多加運用則加運用，因為是最簡單的架構
- 三. 多觀察多用經驗法則，FFT system 存在著許多規律性可以去突破傳統的演算式子，進而使得硬體架構更簡化

我們針對第一種架構和第二種架構來看，因為都使用了兩組 cordic system，所以其整體效應都是屬於比較差的，而第三和第四種架構因為只運用了一組 cordic system，所以整體效應比起第一第二架構來明顯好。

我們針對第三架構和第四架構來看，面積，power 以及 performance 都是第四種架構比較好，就是因為第四種架構運用了比較多 W4 block (架構簡單並且精準度 100%)。

最後我們比較起第三架構和第四架構，第三架構是用 Radix-2⁴ 來推導出來的，而第四架構是無法用傳統的數學演算推導出來，則是用觀察規律性質而衍生出來的，其實這要用數學演算推論不是不行，只是其數學過程會非常之複雜和繁瑣，故不建議用慢慢推導的方式。

4-6 1K-point FFT(SDF)的硬體實現討論

對於 1K-point FFT 的 SDF 硬體架構大概可以附錄五來表示：

我們在此節只討論兩種架構



- 一. 傳統 mix Radix 以及 Radix-2³ 架構
- 二. 運用 mix Radix 以及 Radix-2⁴ 綜合改良架構

當然此架構的”12bits to 16bits block”和”16bits to 20 bits block”以及”20bits to 24 bits block”的位置還是看其真正架構為何才能決定真正擺設位置。所以我們還是先決定每一級的 bit 數，如附錄六所示

第一種架構將會用到三組 cordic system，三組 W8 block 以及三組 W4 block。而第二組架構用到了兩組 cordic system，兩組 W16 block 以及五組 W4 block。

而第二組架構其實就是運用前一章節第四種架構 256-point FFT 再接上一個 4-point FFT，即形成了 1K-point FFT，而且 cordic system 減少以及 W4 block 運用次數比較多。

而此兩種架構的分析數據如下表 Table 4-6.1 所示：

	第一組架構	第二組架構
Average Dynamic Power	17.85mW	16.32mW
Gate count (面積)	249871	230158
Total delay	1075 cycles	1057 cycles
Memory size(bits)	26792 bits	26792 bits
Performance(SNR dB)	32.457dB	36.947dB

Table 4-6.1 第一種和第二種 1K-point FFT 架構比較結果

由此兩架構比較可以得知，只要運用到以下兩重點：

- 一. *W4 block* 愈多愈好
- 二. 盡量減少 *cordic system* 的數量

即可以將 FFT system 面積，power 以及 SNR 都會變好，不管是用在 64-point，256-point 或是 1K-point FFT system 都適用，但是要適當的配上用經驗法則來改變架構。

以上兩種架構的硬體圖如附錄七所示：

4-7 對於其他架構改善的優缺點分析

我們以上所討論的都主要以 FFT 的乘法器組成下手做架構改良，那麼我們或許也可以從乘法器本身的架構下手做改良，比如說 W8 block, W16 block 或是 cordic system 做改良，但是一定會有其負面的影響存在，接下來我們將討論以下問題：

- 一. W8 block 以及 W16 block 的改良
- 二. Cordic system 的改良

4-7.1 W8 Block 以及 W16 Block 的改良

首先我們要說明先前所用的 W8 block 以及 W16 block 的細部構造，主要是以接收輸入信號的斷路控制系統為主。

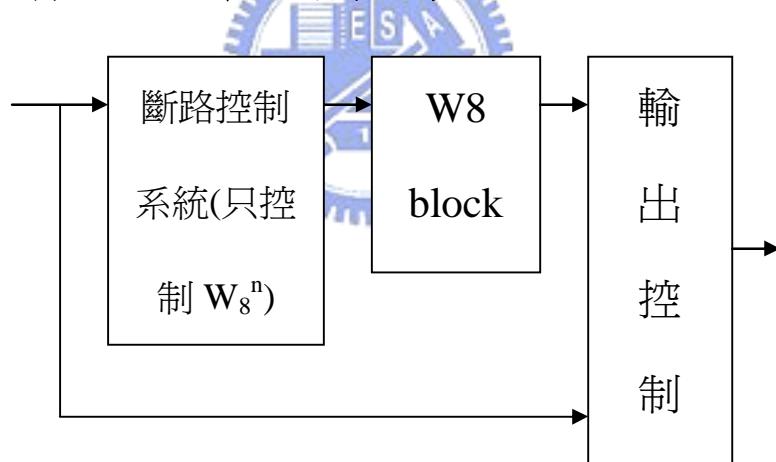


Fig 4-7.1 W8 block(輸出控制要看 system 而定)

Fig 4-7.1 為 W8 block 的架構圖，由斷路控制系統，W8 diagram 以及輸出控制組成，而其中 W8 diagram 以及輸出控制是固定的，而斷路控制系統可以做一些修正。

我們前面所用的 W8 block 的斷路控制系統其實是分成兩個層面做控制：

- 一. 假設此時刻 *twiddle factor* 為 w_8^{2n+1} ，則將輸入信號送到 W8 diagram 中做位移相加，則另一條線路則送零信號。
- 二. 假設此時刻 *twiddle factor* 為 w_8^{2n} ，則送零信號到 W8 diagram 中，另一條線路則送原輸入信號

之所以這樣做是為了避免當 *twiddle factor* 為 w_8^{2n} 的時候其 W8 diagram 還要做位移的動作，如此一來即可以省一些 power，當然多了這一個斷路控制系統的代價就是面積會增大一點，不過其值不會很大。最後兩組信號同時送入輸出控制系統做輸出的運算。

而以上的架構算是比較平衡的設計，因為以下的 W8 block 設計雖然 power 省下很多，但是面積卻暴增好幾倍，不符合設計理念。



我們觀察一下輸出控制系統，其中的運算可能包括四到五個加法器以及一些繁瑣的 control，所以我們想說能否避免掉不必要的運算和 control，也就是說當 *twiddle factor* 為什麼的時候就做什麼運算，其他的硬體就完全不動作。

所以光基於這一點想法，首先就要把輸入信號分成好幾組線路，而每一組線路都專處理不同的 *twiddle factor*，並且在一個 *clock cycle* 之內，只會有一組線路會工作(此由斷路控制系統來控制)，最後結果在用 OR block 來做整合，所以面積勢必會大上很多。此種設計方法如下圖 Fig 4-7.2 所示：

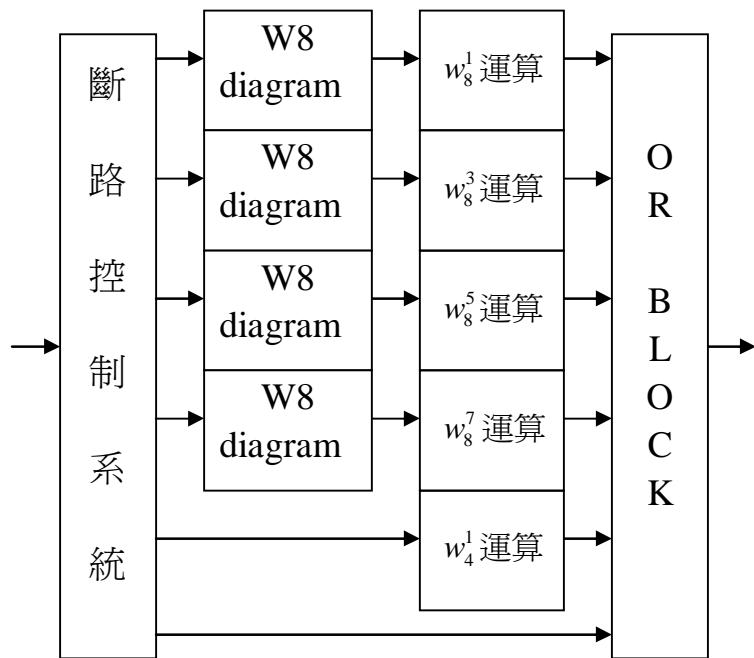


Fig 4-7.2 W8 block 第二種設計架構(省 power)

若將此種架構運用到先前做過的 Radix-2³ 64-point FFT(SDF) with W8 Block and Cordic 架構下，則其結果和原先結果如下表 Table 4-7.1 所示：

	W8 block 原先架構	W8 block 省電架構
平均消耗 power	7.92mW	6.64mW
Gate counts	52029	76429
Total delay	82 cycles	82 cycles
Memory size	1632 bits	1632 bits
Performance	42.127dB	42.127dB

Table 4-7.1 運用原本 W8 block 和省電 W8 block 的比較

依照上方的圖我們可以發現，消耗 power 的確是省下了 1.3mW 左右，但是面積卻足足變成了大約為原本的 1.5 倍，面積增加率比 power 減低率快上許多

當然此種想法也可以用在 W16 block 上，但是 W16 block 本身的輸出控制系統所做的運算比起 W8 block 多將近一倍。所以 W16 block 省電版本的分之一定也會比 W8 block 省電版本的分之多上快一倍，換言之，就是 W16 diagram 個數以及 W8 diagram 的個數在 W16 block 省電版本中會很多，這會使得面積暴增許多。

我們將 W16 block 省電版本用在改良後 Radi x-2² 256-point FFT(SDF) with W16 block and cordic 架構中，並且和用原本 W16 block 時候的結果來做比較，比較數據如 Table 4-7.2 所示：

	W16 block 原先架構	W16 block 省電架構
平均消耗 power	10.26mW	9.04mW
Gate counts	94088	182451
Total delay	272cycles	272cycles
Memory size	6680 bits	6680 bits
Performance	40.014 dB	40.014 dB

Table 4-7.2 運用原本 W16 block 和省電 16 block 的比較

其實我們發現消耗 power 也大約降低了 1.2mW 左右，但是此時所銷耗的面積卻是原本的兩倍之多，這實在不符合設計理念，因為增加面積變成兩倍多而 power 却只降低 1.2mW 左右。

所以這種架構就看看設計者是要求低 power 為主還是以面積和 power 都顧慮的設計，決定採用何種方式 W8 block 以及 W16 block 設計，不過一般的設計者是不會犧牲這麼大的面積去省那麼一點點的 power 的。

4-7.2 Cordic system 的改良

我們先前有說明過，cordic system 是採用逼近法來做運算，而逼近階數看所要求的精準度而定。我們設計了這麼多組的 FFT system 不難發

現，其實在使用 cordic system 的時候，其 twiddle factor 很多都是轉零度角(25%~40%)，也就是將原數值乘以 1.64676 倍而已，而此步驟其實使用一些位移器就可以完成，所以我們的想法就是，當輸入 cordic 的角度為零度角的時候，此時 cordic system 不工作，則用另一塊由一些位移器組成的 block 來做運算。其架構圖如 Fig 4-7.3 所示：

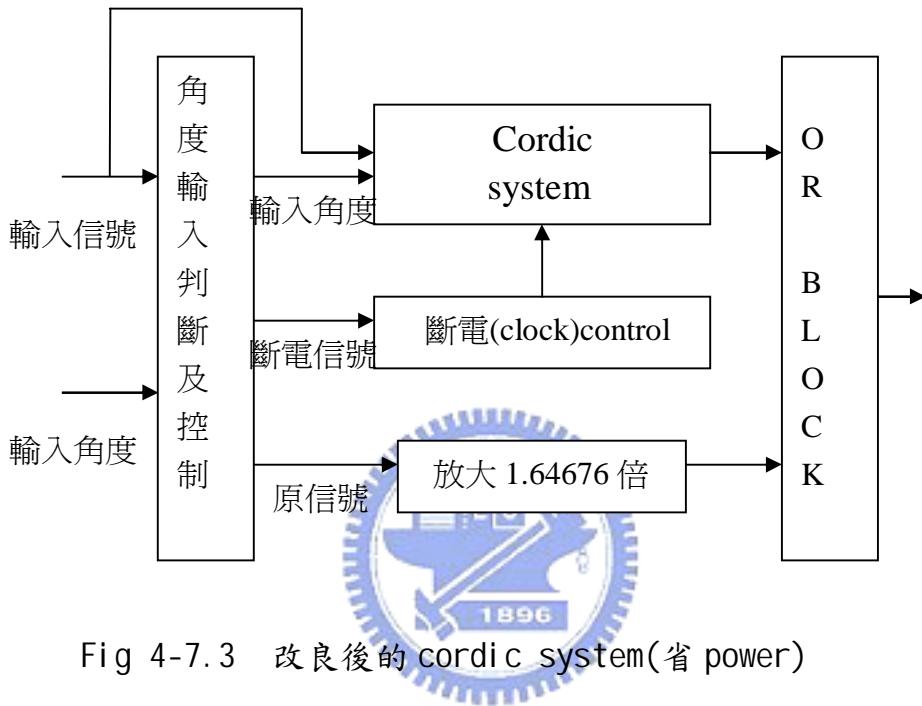


Fig 4-7.3 改良後的 cordic system(省 power)

上圖其中的 *斷路 control block* 是當接收到斷電信號的時候，分別將 cordic system 每一階層依序斷電一個 clock cycle，而不是同時斷電一個 clock cycle，這是因為當你輸入一個信號進入 cordic system 的時候，其後方階層已經有先前輸入處理到一半的信號了，要是同時斷電的話則這些資料將會全部消失，所以必須要依序斷電一個 clock cycle 才行。

當然，因為這個系統比原先的 cordic system 了一些 control 來控制，所以面積當然會加大，但是 power 會下降(因為 cordic 本身佔的 power 消耗比例很大)而且 performance 會稍微提升一點點，而下表 Table 4-6.3 是原 cordic system 和改良後的 cordic system 做比較(以 Radi x-2³ 64-point FFT with W8 block and cordic system 測量)

	原本的 cordic	改良後的 cordic
消耗 Power	7. 92mW	7. 18mW
Gate counts	52029	58647
Performance	42. 127dB	42. 954dB

Table 4-7.3 原始的 cordic 和改良後 cordic 的分析比較

由上表可以看出，改良後的 cordic 面積比原本大約多了 7000 個 gate counts，而 power 比原本的 cordic 降低了大約 0.7mW，這和前一章節所說的 W8 block 和 W16 block 改良比較起來，此種改良的結果(面積和 power 的互抵)比較符合設計理念。

而為什麼改良後的 Performance 會比原本的 cordic 好一些些呢？這是因為我們將轉零度角的時候做比較精確的放大(利用位移器)，而不是再利用 cordic system 的逼近法，所以當轉零度角的時候，其輸出值會比原本的精確一點點，不過基本上幾乎是沒有差異。



4-8 對於 FFT 硬體架構設計的總結和感想

經由這一整章的設計，我們可以知道其實重點主要是繞在 *cordic system* 的減少使用以及改良，而我們在此章特別設計了 W16 block，在某些 FFT system(SDF)架構下，適當的使用，的確可以將 Radix-2^4 效果發揮的很好。

或許會有人說，既然有所謂的 W8 block 和 W16 block 來替代 *cordic system*，而且也有好的結果出現，那麼可不可以有 W32 block(即用來處理 $n \times 11.25$ 度的處理器)呢？答案是，當然可以，但是非常不切實際，因為我們在設計 W16 block 的時候已經得知，W16 block 包括了 W8 diagram 和 W16 diagram，所以 W16 block 的面積是 W8 block 的兩倍多。

而 W32 block 就包括了 W8 diagram，W16 diagram 以及 W32 diagram，再加上其 control 會變成更加複雜，所以面積將近會是 W16 block 的 2.5 倍左右。這樣下去 W32 block 的面積會跟 *cordic system* 差不多，下表 Table 4-8.1 是 W8 block，W16 block，W32 block 以及 *cordic system* 的面積大小：

system	W8 block	W16 block	W32 block	Cordic
Gate count	3462	5978	19281	21943

Table 4-8.1 W8, W16, W32 block(12bits) 以及 cordic(16bits) 比較

我們可以發現，W32 block(信號長度 12 bits)的面積已經快跟 *Cordic system*(信號長度為 16 bits)一樣大了，所以要是 W32 block 的信號長度為 16 bits 的話，其面積必定會大於 *cordic system*。

有人可能會認為如果做 1024-point FFT(SDF)的話，使用 W32 block 說不定會比較好(因為使用 Radix-2^5 剛剛好對稱)，實際上他只有在速度上會比其他設計法好，剩下反而會更差。我們可以經由 Table 4-8.2 看出運用 Radix-2^5 以及運用 Radix-2^4 (mix Radix-2^2)產生架構所需的乘法器內容：

	Radix-2 ⁴ (mix Radix-2 ²)	Radix-2 ⁵
W4 block	三組	兩組
W8 block	兩組	兩組
W16 block	兩組	兩組
W32 block	零組	兩組
Cordic system	兩組	一組

Table 4-8.2 Radix-2⁵ 以及 Radix-2⁴(mix Radix-2²) 產生架構所需的乘法器內容

我們可以看出來，雖然 Radix-2⁵ 架構用到的 cordic system 只有一組，但是其 W32 block 的大小已經等於是 cordic system 的大小，雖然速度會變快大約 16 clock cycles，但是其面積會變大大約 20000 個 gate counts，而且 power 和 SNR 並不會有什麼明顯改善，因為 W32 block 所消耗的 power 以及 SNR 跟 cordic system 都差不多。

依照以上的分析，我們可以發現 W32 block 已經不適用實現在硬體架構上，更不用說是 W64 block 之類的推導了。而且我們也要注意一點，W16 block 的使用時機也要注意，並不是說在任何架構之下使用 W16 block 都會使整體架構改良的比較好。好比拿 32-point FFT 做例子，我們拿 Radix-2³ 以及 Radix-2⁴ 兩架構來做比較：

	Radix-2 ³	Radix-2 ⁴
W4 block	兩組	一組
W8 block	一組	一組
W16 block	零組	一組
Cordic system	一組	一組

Table 4-8.3 Radix-2³ 以及 Radix-2⁴ 產生架構所需的乘法器內容

依照上表可以發現，32-point FFT(SDF) 運用架構 Radix-2⁴ (W16 block) 反而會使得面積和消耗 power 變大 (因為 W16 block V.S W4 block)，所以這例子證實了，W16 block 用在不適當的架構下，反而會使得整體效能變更差。

最後，基於上面幾節設計的原理以及重點，不管是設計 1024-point FFT(SDF)還是到 8K-point FFT(SDF) ， 最好都要按照以下幾點原則：

- 一. 先找出最好的架構，使得 *cordic system* 愈少愈好
- 二. *Radix* 最高用到 2^4 ，要是超過的話反而會出現反效果
- 三. 若架構本身很大(如 4K 或是 8K-point FFT) ， 則可以再改良 *W8 block* 或是 *cordic system*(因為相對比較之下面基增加量比率很小)
- 四. 運用經驗法則思考架構能否使用愈多 *W4 block* 則愈好

相信按照這些原則，然後再適當的使用 *W16 block*，便可以設計出較小面積，較低消耗 power 以及較快速的 FFT system。

當然，相信一定還有別種方法來改良 FFT system 的效率，而我們要改善的地方主要還是著重在替代複數乘法器的部分，所以未來還是要發展出更快速或是更省 power 的複數乘法器替代品，這是我們的目標(*cordic system* 已經是其一)。

