第二章　封包偵測與頻率偏移補償

第一章中，我們已經介紹過了 IEEE 802.16a 實體層的架構，因此接下來我們就應該針對其這樣的一個規格，設計出一個內接收機，以求能達到偵測、同步、解調之用，而這正是本論文的目的。

在本章節中，將會介紹如何偵測封包以及如何估計、補償頻率偏移；我們會從問題的起因開始說起，接著是演算法的推導，而最後是硬體的設計與實現，這些都會在本章節中有著詳細的解說。

2.1 封包的偵測

對一個接收機而言，一開始所面臨到的第一個問題，便是要如何知道是否有該接收的資料在傳送；因為在傳送的媒介當中，有時是不只一種標準的訊號在傳播的，而就算是真的只有一種標準，那也有可能是沒有資料傳送而媒介中只有單單雜訊的這種情況；所以，對一個接收機來說，第一個要克服的問題就是要如何判斷是否有資料需要接收；若無，則應該停止硬體的運作，以達到節省能量消耗的目的；而這個判斷是否有資料在傳送的動作，我們稱之為封包的偵測(Packet Detection)。

2.1.1 封包偵測的演算法

問題已經如前言所述，而解決方法的關鍵，我們放在短調整符元(Short Training Symbol)上面，因為短調整符元具有兩個特性：一是具有週期性；二是不論在時域上的取樣點或是頻域上所載的資料都是已知的。

根據這兩個特性，我們可以提出兩個方法來做封包的偵測：一是使用自相關函數(Autocorrelation Function)的方法，因為短調整符元是具有週期性的，所以當我們取相隔一個週期的兩個取樣點作內積，並且累加一定的長度後，若選擇

的取樣點大部分都落於短調整符元內的話，可預期地，其內積累加的結果應該會是一個極大的數值；反觀，若選擇的取樣點是一群雜亂無規則的雜訊取樣點的話，那相成累加的結果應該會是一個極小的數值；因此我們可以使用自相關函數的方法來作為封包偵測的依據。除了使用週期的這個特性外，我們也可以使用資料已知的這個特性；既然資料已知，所以我們可以使用匹配濾波器(Match Filter)，對一個區塊的取樣點作掃描的動作，直到這個區塊剛好就是一個短調整符元時，可預期地，會有一個峰值(Peak Value)產生。

可是因為通道(Channel)的特性是不可預知的，連帶著造成的結果是當初傳送已知的資料，在經過一個未知的通道後，會使得已知的特性弱化，因此使用匹配濾波器就不適當；這也是本論文會採用自相關函數方法的原因之一；而另一個原因是使用自相關函數可以減少硬體的負擔，在作頻率偏移的估計時，我們亦會使用到自相關函數的方法，這將會在本章節 2.2 中說明。

在使用自相關函數的方法前，我們還有兩個問題要解決：第一個是累加長度(Accumulation Length)的大小；若選擇的累加長度過長，則因為累加結果的動態範圍(Dynamic Range)增大，所以相對應會增加硬體的負擔；而且過大的累加長度未必便會有好的效能(Performance)，因為過長的累加長度反而可能因為包括太多的非短調整符元取樣點，造成效能的降低。因此適當地選擇累加長度是必須的，在我們的硬體架構中，累加長度設計與短調整符元的週期一樣為六十四個取樣點。另一個問題是該如何界定一個累加的結果是極大還是極小的值；一群沒有週期性的取樣點，可能因為取樣點本身能量很大，所以自相關函數的結果會是個極大的值；相反地，一群具有週期性的取樣點，卻因為取樣點本身能量太小，所以自相關函數的結果卻是一個極小的值；這兩種情況都是我們不願意見到的。解決的方法便是對自相關函數的結果再作一個取樣點能量正規劃(Normalize)的動作；圖(2.1.1)便是整個自相關函數的示意圖。

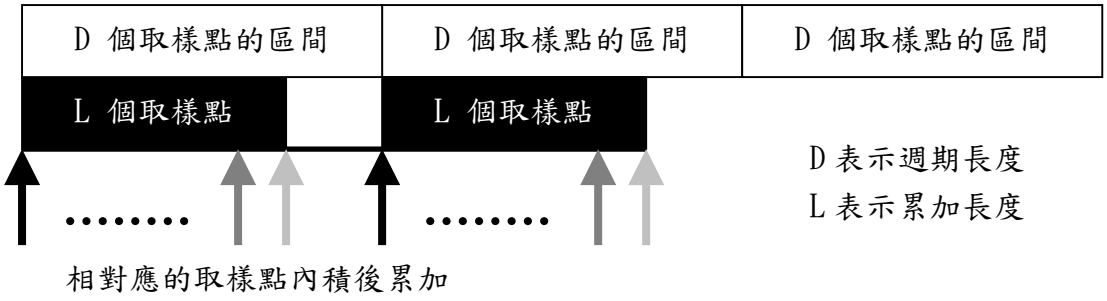


圖 2.1.1 自相關函數示意圖

為了解釋的方便以及將來硬體的講解，所以我們在式子(2.1.1)中定義兩個參數 Cn 和 Pn 如下：

$$Cn = \sum_{n=k}^{k+L-1} x^*(n) \cdot x(n+D)$$

$$Pn = \sum_{n=k}^{k+L-1} |x(n+D)|^2 \qquad mp = |Cn|^2 \Big/ |Pn|^2 \qquad \text{式(2.1.1)}$$

其中 x(n)表示的是接收端的取樣點。由這個數學的表示式我們可以知道，其實 Cn 就是自相關函數的累加值，而 Pn 就是該對應取樣點的能量，所以 mp，也就是兩者絕對值平方的比值，當大於一個界限(Threshold)時，我們便可判定現在有封包在傳送；這個臨界值的設定，跟通道的特性有關，針對不同特性的通道可以設定不同的大小；在本論文硬體的設計中，我們設定為 0.3798828125，如此利於將一個常數乘法器化簡為四個加法器的組合。

2.1.2 滑動框架

在理論上，每接收到一個取樣點我們就必須去計算一次相對應的 Cn 和 Pn；以週期為六十四、累加長度六十四、x(k)表示現在所接收到的取樣點為例：

$$Cn = \sum_{n=k-64+1}^{k} x^*(n-64) \cdot x(n)$$

$$Pn = \sum_{n=k-64+1}^{k} |x(n)|^2 \qquad mp = |Cn|^2 \Big/ |Pn|^2 \qquad \text{式(2.1.2.1)}$$

依照式子(2.1.2.1)，每接收到一個取樣點，我們就需要作六十四個內積運算、六十三個加法運算以求得 Cn；這對硬體來說是個很大的負擔，因為要在短時間內完成這些運算，除了在空間上展開外別無他法，且更不用說還需求得另一個常數 Pn。因此在硬體的實現上，我們還需要使用滑動框架(Sliding Window)的概念來節省硬體的需求。

所謂滑動框架的概念就是：當要求得一個新的取樣點所對應的 Cn、Pn 時，我們將利用前一個取樣點的 Cn、Pn 來取得；若以 Cn(k)表示第 k 個取樣點的 Cn 值的話，其數學表示如下式(2.1.2.2)：

$$
\begin{aligned}
Cn(k-1) &= \sum_{n=k-1-64+1}^{k-1} x^*(n-64)\cdot x(n) \\
Cn(k) &= \sum_{n=k-64+1}^{k} x^*(n-64)\cdot x(n) \\
&= \sum_{n=k-1-64+1}^{k-1} x^*(n-64)\cdot x(n) + x^*(k-64)\cdot x(k) - x^*(k-128)\cdot x(k-64) \\
&= Cn(k-1) + x^*(k-64)\cdot x(k) - x^*(k-128)\cdot x(k-64)
\end{aligned}
$$

式(2.1.2.1)

所以當我們求 Cn(k)時，我們可以用 Cn(k-1)加上新求得的內積值再扣去裡面最舊的內積值，如此形同一個滑動的框架，而框架的大小固定為累加的長度，也就是六十四個取樣點；利用這個滑動框架的方法，我們可以將原本的硬體降低到只需要一個內積運算、一個加法運算以及一個減法運算，這和原本需要六十四個內積運算和六十三個加法運算的硬體相比，複雜度大幅地降低。

2.1.3 封包偵測的硬體架構

整個硬體的架構中，我們總共使用到了六塊記憶體，其寬度都是十二位元，以深度來說，六塊皆為六十四的大小，其詳盡的時序圖，我們會在第五章中介紹。在封包偵測的硬體中，我們會使用到前三塊深度大小六十四的記憶體作

FIFO(First In First Out)使用，以達成我們自相關函數以及滑動框架的運算，下圖(2.1.3)便是整個封包偵測的硬體架構圖：

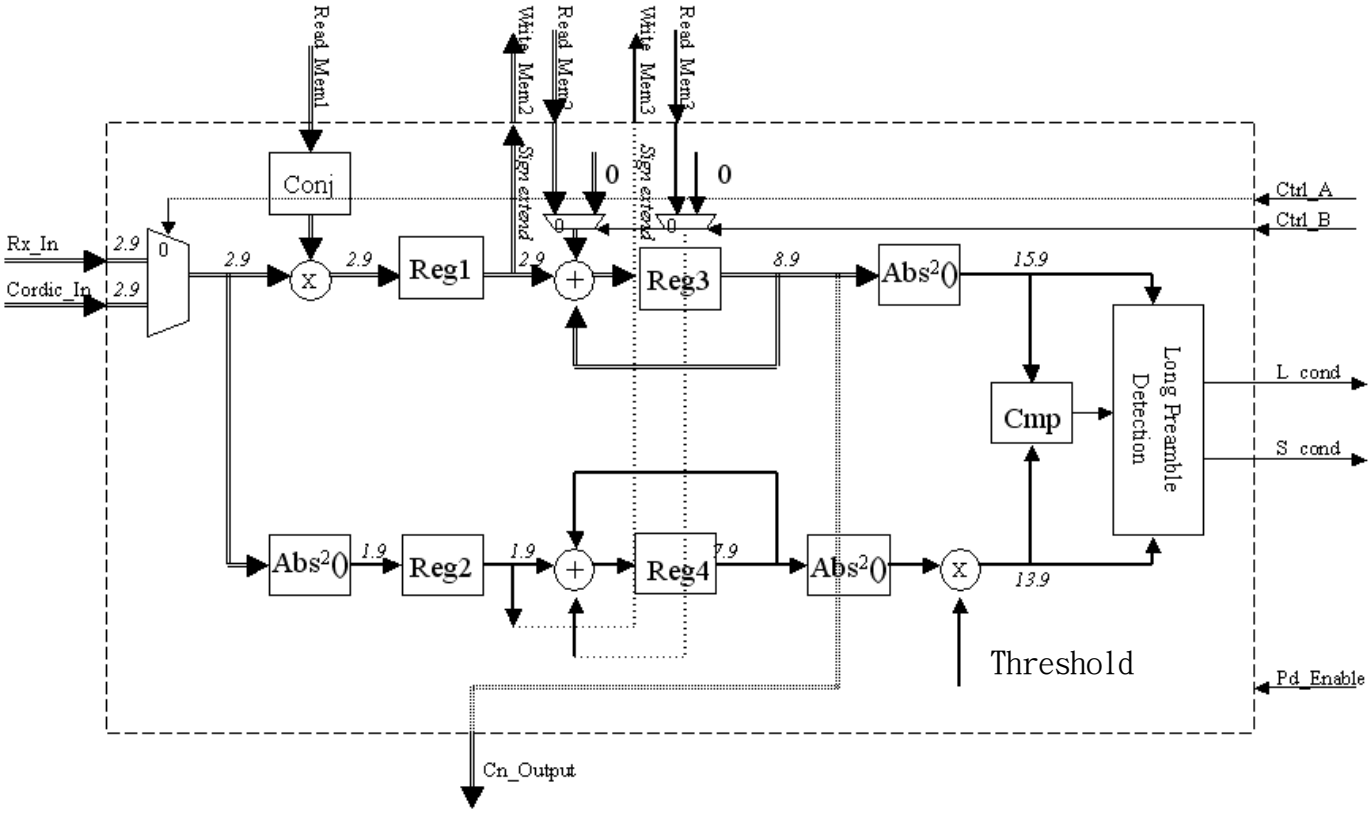


圖 2.1.3 封包偵測的硬體架構圖

2.1.3.1 輸入輸出以及控制訊號

資料輸入訊號：

- Rx_In：沒經過頻率偏移補償的資料取樣點，有實部、虛部訊號之分
- Cordic_In：經過頻率偏移補償的資料取樣點，有實部、虛部訊號之分
- Read_Mem1：從第一塊記憶體得到的輸入資料，有實部、虛部訊號之分
- Read_Mem2：從第二塊記憶體得到的輸入資料，有實部、虛部訊號之分
- Read_Mem3：從第三塊記憶體得到的輸入資料，只有實部

資料輸出訊號：

- Write_Mem2：寫入第二塊記憶體的輸出資料，有實部、虛部訊號之分
- Write_Mem3：寫入第三塊記憶體的輸出資料，只有實部訊號
- Cn_Output：將 Cn 的值輸出到 CORDIC 中使用，有實部、虛部訊號之分

控制訊號：

- Ctrl_A：輸入控制訊號，控制封包偵測的輸入資料是否經過頻率偏移補償
- Ctrl_B：輸入控制訊號，決定硬體是作累加還是滑動框架的運算
- Pd_Enable：輸入控制訊號，控制封包偵測的硬體是否運作
- L_cond：輸出控制訊號，表示是否有偵測到長調整符元
- S_cond：輸出控制訊號，表示是否有偵測到短調整符元

2.1.3.2 硬體運作的機制

我們在之前 2.1.3 節時已經說過在封包偵測中，我們使用三塊記憶體作 FIFO 使用，而這三塊記憶體的深度都為六十四，所以在圖 2.1.3 中 Reg1 前乘法器的兩個輸入會在時間上差六十四個取樣點，也就是說當現在輸入的資料 Rx_In 是 x(k)的話，那從記憶體一取得的資料 Read_Mem1 就會是 x(k-64)，因此，Reg1 所儲存的資料在演算法上，其實就是現在取樣點和其相對應的取樣點的內積值，同時這個內積值會藉由 Write_Mem2 的資料排線寫入記憶體二的 FIFO 中，以供後面作滑動框架之用。

接著我們再來看 Reg3，Reg3 和其之前的加法器形成的就是一個滑動框架的架構，如果我們先不看從記憶體二來的資料排線，那 Reg3 其實就是一個累加器，一個一直把從之前到現在，所有取樣點和其相對應的取樣點，內積後累加的積分器，也就是不設定累加長度的 Cn 值；但是滑動框架的運作是只需要六十四的累加長度，因此，用來作 FIFO 的記憶體二便派上了用場；如前一段所述，記憶體二中所存放的資料就是取樣點和其相對應取樣點內積的結果，同時在配合上 FIFO 的原理以及記憶體二深度六十四的影響，使得 Reg3 前加法器從記憶體二的 Read_Mem2 資料排線所取得的內積值，相對於從 Reg1 取得的內積值，將會有延遲六十四個取樣點的關係，因此 Reg3 形成了式(2.1.3.2.1)：

$$Cn(k) = Cn(k-1) + x^*(k-64)\cdot x(k) - x^*(k-128)\cdot x(k-64) \quad \text{式(2.1.3.2.1)}$$

而這式其實就是滑動框架的數學式，所以 Reg3 所儲存的值在演算法上就是

Cn。同理，Reg2 所存的是現在取樣點的能量大小，而 Reg4 便是從現在開始，往前回算六十四個取樣點能量大小的累加值，在演算法中，這也就是 Pn。

有了 Cn 和 Pn 的值，接著便應該著手求得 mp 以和界線 0.38 相比，依照數學式我們將會需要一個除法器來求得 mp，但是除法器對硬體的複雜度上來說，非常的龐大，所以我們改用乘法器來取代，取代的方法很簡單就是利用交叉相乘的運算，將原本所需的數學式(2.1.3.2.2)：

$$mp = \left|Cn\right|^2 \Big/ \left|Pn\right|^2 \geq Threshold \qquad \text{式(2.1.3.2.2)}$$

替換為式(2.1.3.2.3)：

$$\left|Cn\right|^2 \geq Threshold \cdot \left|Pn\right|^2 \qquad \text{式(2.1.3.2.3)}$$

如此一來，我們可以用一個常數的乘法器來代替原本需要求得 mp 的除法器，藉此以降低硬體的負擔；同時更進一步地，因為使用的是常數乘法器，所以我們還可以用加法器來簡化硬體，這也就是圖(2.1.3)中硬體實現的方法；在圖中，我們將 Reg3 經過絕對值平方後，和 Reg4 經過絕對值平方且乘以常數 Threshold 後作比較，當 Reg3 取絕對值平方後的值比較大時，便表示有個短調整符元接收到，同時把 S_cond 信號拉起為高電位(high)，供控制單元作後續的處理。

2.1.3.3 字元長度的選擇

所謂的字元長度(World Length)就是指在硬體中，資料間互相傳輸所使用的排線字元數，因為硬體的設計不比演算法可以使用任意個精密度來表示一個數，相反的其必須有所限制，不然可能會為了增加那一點的訊號雜訊比，但卻付出了十幾二十倍的硬體負擔，這是得不償失的，因此我們必須要在訊號雜訊比和硬體負擔中，有所取捨，而適當的選擇字元長度便是一件很重要的事情。

在本論文所設計的硬體中，資料的取樣是採用十位元、二的補數表示法的類比數位轉換器(Analog to Digital Converter)，而且這個取樣過後的資料本身是個複數，也就是說對任意一個取樣點，我們用十位元、二的補數表示法表示該取樣點的實部，同時也用了十位元、二的補數表示法表示該取樣點的虛部。而

這個取樣點在送至封包偵測的硬體前，會先經過一個複數平面的旋轉後，才是圖(2.1.3)中的 Rx_In 或是 Cordic_In 輸入訊號；而需要先經過旋轉的原因，是為了作頻率補償的效用，這將會在本章的 2.2 節中解釋。至於 Rx_In 或是 Cordic_In 這兩個輸入訊號該使用多少個位元，我們利用下圖(2.1.3.3)來說明：

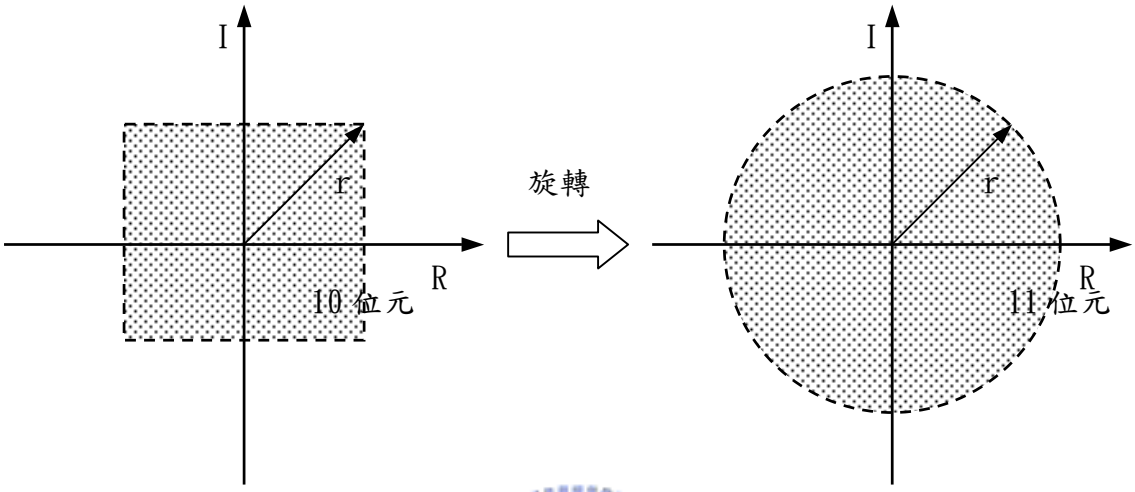


圖 2.1.3.3 字元長度選擇示意圖

在圖(2.1.3.3)中的左邊，網底所表示的是一開始的取樣點可能落在的區域，其中 r 的值是十位元、二的補數所能表示的最大值的根號二倍，所以當我們把取樣點旋轉後，得到的就是 Rx_In 或是 Cordic_In 輸入訊號，其可能的範圍便形成右邊的區域，同時，因為實部和虛部的可能範圍都擴大到 r 的值，所以為了能把最大可能的 Rx_In 或是 Cordic_In 輸入訊號包括在我們的動態範圍(Dynamic Range)內，所以我們需要多一個位元來儲存這些經過旋轉的取樣點，因此 Rx_In 或是 Cordic_In 輸入訊號總共需要十一個字元排線。

接著來看 Reg1 所需的字元長度，之前的章節 2.1.3.2 中已經說過 Reg1 其實就是兩個相對應的取樣點內積的值，所以他的最大值當然是出現在這兩個相對應的取樣點完全一樣的時候，所以他的動態範圍會是原本十位元、二的補數所能表示的最大值的根號二倍的根號二倍，也就是擴大了兩倍，因此我們可以用十一個字元長度來包含住 Reg1 的動態範圍。相同的原理可以用在 Reg2 上面，但是因為他本身就是取絕對值的平方，所以其結果必定為正，因此沒有必要用到二的補數表示法來表示一個必為正的值，所以我們可以用簡單的正數表示法即可，因此

可以再少一個位元，也就是 Reg2 字元長度用十個字元排線就可以了。

再下來，我們來看滑動框架的部分，因為滑動框架其基本就是個累加器，所以我們只需要看累加多少次，就可以判斷出應該增加多少個字元數才可以滿足動態範圍；在我們硬體設計中，我們採用的是累加長度六十四的滑動框架，所以假設在最糟糕的情況下(Worst case)，應該要多增加六個字元數，所以 Reg3 要比 Reg1 多六個字元長度，也就是 Reg3 需要十七個字元排線；同理，Reg4 需要十六個字元排線。

最後，我們要來看的部分是比較器的上、下兩個訊號，也就是 Cn 和 Pn 的絕對值平方；在 Cn 絕對值平方這部分，其實就跟 Reg2 時一樣，原本是需要二十五個字元長度，但是因為其值必為正，所以我們可以用正數表示法來替代二的補樹表示法，以節省一個字元長度，所以比較器上方的訊號需要二十四個字元排線。至於 Pn 絕對值平方這部分，因為他本身就是正數表示法，所以無字元可省略，而後乘的常數 Threshold，因為介於 0.5 和 0.25 之間，所以其最後的動態範圍可以少一個字元來表示，因此比較器下方的訊號只需要二十二個字元排線。

2.2 頻率偏移補償

一般來說傳送機和接收機兩端，都會有各自的石英震盪器，雖然標準裡會規定好整個訊號在傳送時所使用的載波頻率，但是不同的石英震盪器，就是沒有辦法可以同時震盪出兩個同步的時脈訊號；而這兩個不同時脈訊號的差異，我們便稱之為頻率偏移(Frequency Offset)。雖然通常頻率偏移都很小，大約在正負一百個百萬分之一之間，但是因為通常一次傳送的資料量很大，所以很小的頻率偏移都會慢慢的累積造成載波間的互相干擾(Inter-Carrier-Interference，ICI)，這問題在正交分頻多工系統裡面，尤其嚴重。因為當有頻率偏移的時候便會造成載波間的正交性喪失，進而使得訊號雜訊比降低，所以我們必須要做頻率偏移的補償，這也是本節要克服的問題。

2.2.1 頻率偏移補償的演算法

假設我們定義 Δf 是傳送端和接收端兩者的頻率偏移，也就是式(2.2.1.1)

$$f_{Tx,Carrier} = f_{Rx,Carrier} + \Delta f \qquad \text{式(2.2.1.1)}$$

在式中 Tx 和 Rx 分別表示的是傳送端和接收端之意;如果我假設接收端在時間取樣點沒有偏移的話，那接收端的取樣點 $x_R(n_0)$ 和傳送端當初傳送出來的取樣點 $x(n_0)$ 有著式(2.2.1.2)的關係：

$$x_R(n_0) = x(n_0) \cdot e^{-j2\pi\Delta f \cdot n_0 T} \qquad \text{式(2.2.1.2)}$$

其中 T 表示的是取樣時間，而 n_0 則是表示第幾個取樣，所以由這個式子中我們也可以發現，若是不作頻率偏移的補償的話，不僅會有載波間互相干擾的問題，同時也會造成在時域上的取樣點會隨著取樣的時間，而一直在複數平面上旋轉，造成訊號雜訊比大幅降低。

接著我們再來看第($n_0 + D$)個取樣點，如同式(2.2.1.2)一般：

$$x_R(n_0 + D) = x(n_0 + D) \cdot e^{-j2\pi\Delta f \cdot (n_0 + D)T} \qquad \text{式(2.2.1.3)}$$

其中 D 可以是任意一個整數。此時如果我們利用短調整符元的週期性，也就是將 D 取為短調整符元的週期：六十四個取樣點，並且限制了 $x_R(n_0)$ 和 $x_R(n_0 + D)$ 都會落於短調整符元的區間中，那便可以滿足下式(2.2.1.4)：

$$x(n_0 + D) = x(n_0) \qquad \text{式(2.2.1.4)}$$

同時，我們將 $x_R(n_0)$ 和 $x_R(n_0 + D)$ 作內積運算，可得到下式(2.2. 1.5)：

$$x_R^*(n_0) \cdot x_R(n_0 + D) = \left|x(n_0)\right|^2 \cdot e^{-j2\pi\Delta f \cdot DT} \qquad \text{式(2.2.1.5)}$$

因此，如果我們能夠求得在短調整符元內兩個距離週期六十四個取樣點的內積值，則這個內積值所對應的主幅角就是 $2\pi\Delta fDT$ ，因此只要能求得主幅角的值，就是等同於找到了頻率偏移 Δf 。

至於這個內積值該如何求得呢？其實我們早在封包偵測的時候，就已經得

到了，在式(2.1.1)中我們可以發現，Cn 其實就是我們這邊要求的內積值，而 Cn 的主幅角便是我們要求得頻率偏移的關鍵，所以在圖(2.1.3)中，才會有個訊號線 Cn_Output 作輸出訊號，就是為了作頻率偏移的估計與補償之用。

所以，接下來的關鍵就是該如何求得一個複數值的主幅角，如果我們有個硬體能將輸入輸出間的關係成為：輸出的訊號是輸入的複數訊號的主幅角；那我們只需要將 Cn_Output 的輸出訊號線，當這個硬體的輸入，我們就可以估計出頻率偏移的大小，進而作頻率偏移的補償。

一般來說，要求得一個複數的主幅角在硬體的實現上，都是採用查表法(Look Up Table)：先建立好一個主幅角的值和複數的實部、虛部比值的表格，當需要求得一個複數的主幅角時，先求得其實部和虛部比值，然後在表格中，找出其相對應的主幅角值；但這方法缺點很多，最大的問題就是：當越高的精密度有所需求，相對的就需要越大的硬體負擔。試想，若希望能有更精確的主幅角值，那便需要有很龐大的表格，同時，實部和虛部比值也要夠多位的精密度，不然表格建立的在多龐大也是沒用；因此，若要採用查表法，當精密度需求高的時候，相對的要有龐大的記憶體儲存表格，也要有能有高位準的除法器，這些都是遙不可及的，因此我們用數位座標旋轉 (Coordinate Rotation Digital Computer，CORDIC) 的演算法來取代查表法，這將會在下節中作完整的說明。

2.2.2 數位座標旋轉演算法

數位座標旋轉演算法不僅可以用來求一個複數的主幅角，同時也可以用於將一個複數旋轉一個角度，因此當我們用數位座標旋轉演算法求得頻率偏移後，我們同時也可以用數位座標旋轉演算法作頻率偏移的補償；這比單使用查表法求得頻率偏移，之後在使用複數乘法器作補償來得有效率以及整合，這也是我們使用此演算法的原因。

一開始我們先介紹如何使用數位座標旋轉演算法作複數的旋轉。假設我們有個複數向量 (x_0, y_0) 以及一個我們想要旋轉的角度 θ，則經過旋轉的複數向量 (x_n, y_n) 滿足式(2.2.2.1)：

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\theta}} \begin{bmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

式(2.2.2.1)

在這個旋轉的過程中，我們將要旋轉的角度θ拆解成 N 個階段角度α_i，也就是說：原本是一次轉到角度θ，而現在替換成每次旋轉的角度為α_i總共旋轉了 N 次；所以θ和α_i會滿足式(2.2.2.2)：

$$\theta = \sum_{i=0}^{N-1} \sigma_i \alpha_i \qquad \sigma_i \in \{-1,+1\} \qquad \text{式(2.2.2.2)}$$

而其示意圖如下圖(2.2.2.1)：

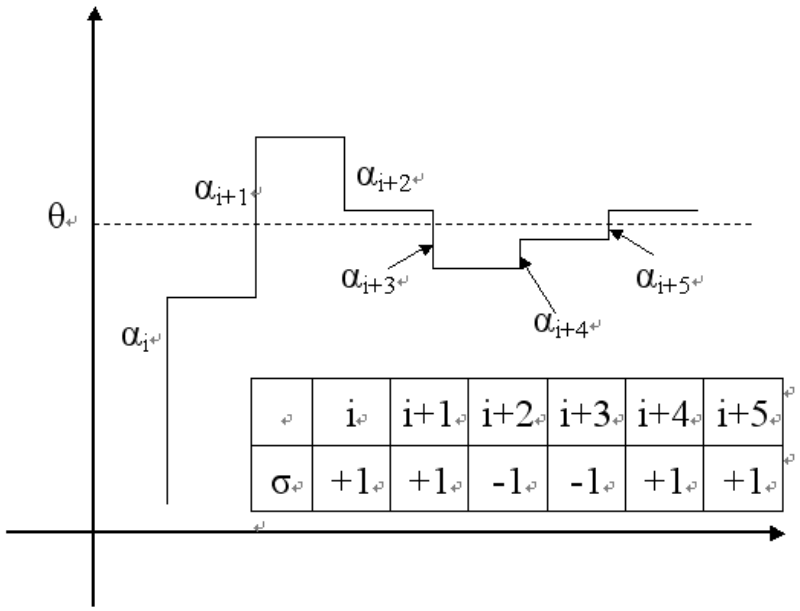


圖 2.2.2.1 旋轉角度和階段角度關係示意圖

但是，不是任意的階段角度α_i都可以讓數學式(2.2.2.2)成立，相反地，階段角度必須要滿足當索引i增加時，α_i的絕對值會持續的減少，如此才有可能在 N 不用很大的情況下，讓階段角度α_i的累加和可以趨近亦或是等同於我們原本要旋轉的角度θ。

可是到目前為止，上面的方法還沒有辦法有效的利用；相反地，在實作上反而是增加硬體的負擔，從原本需要一個複數乘法器以旋轉角度θ的硬體，變成需要N個複數乘法器以旋轉N個階段角度α_i。這其中的關鍵便在於階段角度α_i的

選取，當選取的階段角度有助於硬體的實現，同時又能滿足：α_i的絕對值會隨著索引i增加而減少時；那便能有效取代原本使用複數乘法器以旋轉角度θ的方法。在數位座標旋轉演算法中，我們的階段角度α_i定義如下式(2.2.2.3)：

$$\alpha_i = \tan^{-1}(2^{-i}) \qquad i = 0,\quad 1,.....\quad N-1 \qquad \text{式(2.2.2.3)}$$

如此我們可以將式(2.2.2.1)中的矩陣表示法轉換成式(2.2.2.4)：

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = k_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}, \quad k_i = \frac{1}{\sqrt{1+2^{-2i}}} \qquad \text{式(2.2.2.4)}$$

其中σ_i的正負將可以由下式(2.2.2.5)來求得：

$$\sigma_i = \begin{cases} +1 & z_i \geq 0 \\ -1 & z_i < 0 \end{cases}$$
$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i}) \qquad \text{式(2.2.2.5)}$$

在式子中z_{i+1}表示的是欲旋轉的角度θ和過去i個旋轉過的階梯角度α_i的差，因此其起始條件為$z_0 = \theta$；有了z_0之後便可以求得σ_0，接著利用式(2.2.2.4)便可以求得(x_1, y_1)，然後再回到式(2.2.2.5)以求得z_1，有了z_1後便可以再求得σ_1，接著就可以獲得(x_2, y_2)，如此週而復始下去；當作的次數越多，則正整個旋轉過的階梯角度α_i就會越接近欲旋轉的角度θ，也就是(x_i, y_i)會越趨近於(x_n, y_n)，藉此達到我們要旋轉一個複數向量的目的。或許這樣還不容易看出來在硬體的實現上有何優點，但是當我們將式(2.2.2.4)展開，並且忽略矩陣前的常數k_i後，可以得到式(2.2.2.6)：

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i$$
$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i \qquad \text{式(2.2.2.6)}$$

忽略矩陣前常數k_i的原因，是因為在從(x_0, y_0)開始求得(x_1, y_1)乃至於求得(x_n, y_n)的過程中，這些常數都是連乘的，所以與其在每求得一個(x_i, y_i)時就乘入式(2.2.2.4)中的常數k_i，還不如在求得最後一個(x_i, y_i)時再乘入相對

應的常數，當然，這個相對應的常數已經不是式(2.2.2.4)中的常數，而是之前所有常數k_i的連乘。此外，σ_i不是正一就是負一，所以$\sigma_i 2^{-i} y_i$可以經由很簡單的位移轉換器就可以達成，這也就是數位座標旋轉演算法在旋轉角度上的優點：我們可以將原本要旋轉的角度θ分成i個階梯角度α_i的旋轉，而每個經過階梯角度α_i旋轉的複數向量(x_i, y_i)，我們可以利用簡單的位移轉換器和加法器來求得，藉此降低原本需要使用複數乘法器的硬體負擔。

上面所述，都是數位座標旋轉用於旋轉角度的演算法，至於要用於求主幅角的概念其實是一樣的；假設給了一個複數向量(x_0, y_0)，我們希望能求得其相對應的主幅角，其實等同於我們將這個複數向量(x_0, y_0)旋轉至x軸上時，中間經過i個階梯角度α_i的旋轉的總和；因此我們只需要將σ_i判斷正負的式(2.2.2.5)改為下式(2.2.2.7)：

$$\sigma_i = \begin{cases} -1 & x_i \cdot y_i \geq 0 \\ +1 & x_i \cdot y_i < 0 \end{cases}$$

$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i}) \qquad \text{式(2.2.2.7)}$$

其中z_i表示的是現在複數向量(x_0, y_0)經過i次階梯角度α_i旋轉後，所有旋轉角度的總和，所以其起始條件為$z_0 = 0$，同時因為(x_0, y_0)已知，因此可以利用式(2.2.2.7)求得σ_0和z_1，接著用式(2.2.2.6)獲得(x_1, y_1)，有了(x_1, y_1)後便可以求得σ_1和z_2，如此週而復始下去；當作的次數越多次，就會發現y_i的值會越接近零，也就是說(x_0, y_0)慢慢的被旋轉到x軸上，此時相對應的z_i也就會慢慢的趨近於(x_0, y_0)的主幅角，藉此達到求得主幅角的目的。

同樣地，在使用數位座標旋轉演算法作求主幅角的運算時，我們只需要用間單的位移轉換器和加法器，便可以取代原本需要使用查表法的複雜硬體，而且當主幅角需要有較高的精密度時，我們不需要多增加硬體的負擔，而只需要多作幾次的遞迴即可，這些都是數位座標旋轉演算法的優點所在，也是我們會採用的原因。

2.2.3 數位座標旋轉作求主幅角的硬體架構

在整個硬體的設計當中，硬體的系統時脈頻率我們都是建立在兩倍的資料取樣時脈頻率上，也就是說：當有一個資料取樣點輸入時，我們有兩個系統的時脈可以使用，這個條件不論在本論文中的哪個硬體都是滿足的，因此下圖(2.2.3.1)雖然是作十六個遞迴的數位座標旋轉作求主幅角之用，但因為一個資料取樣點的輸入可以有兩個系統時脈可供硬體的運作，所以我們將此硬體在時間上展開以減少硬體的負擔。

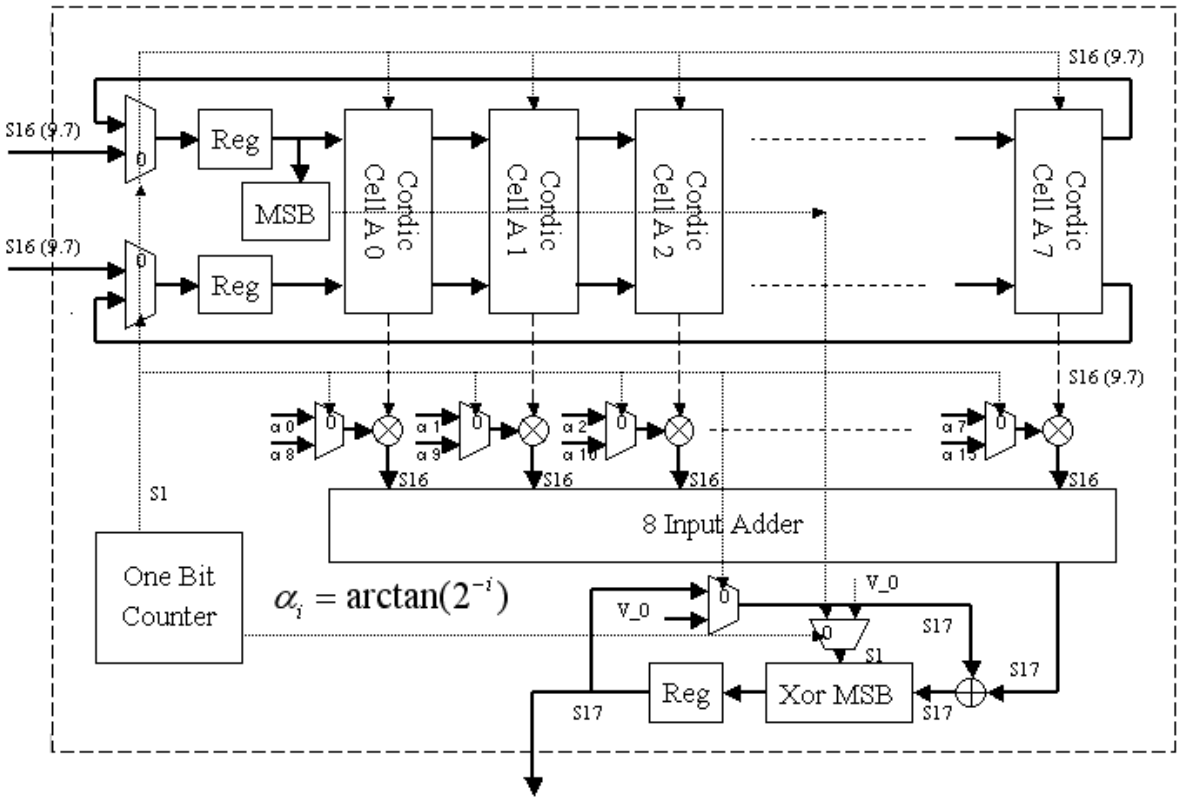


圖 2.2.3.1 數位座標轉換作求主幅角的硬體架構

在此架構中，資料每經過一個 Cordic Cell A N，便表示經過一次的遞迴，也就是利用其輸入的複數向量(x_i, y_i)和數學式(2.2.2.7)，求得σ_i和輸入下一次遞迴的複數向量(x_{i+1}, y_{i+1})；此 Cell 的架構如下圖(2.2.3.2)：

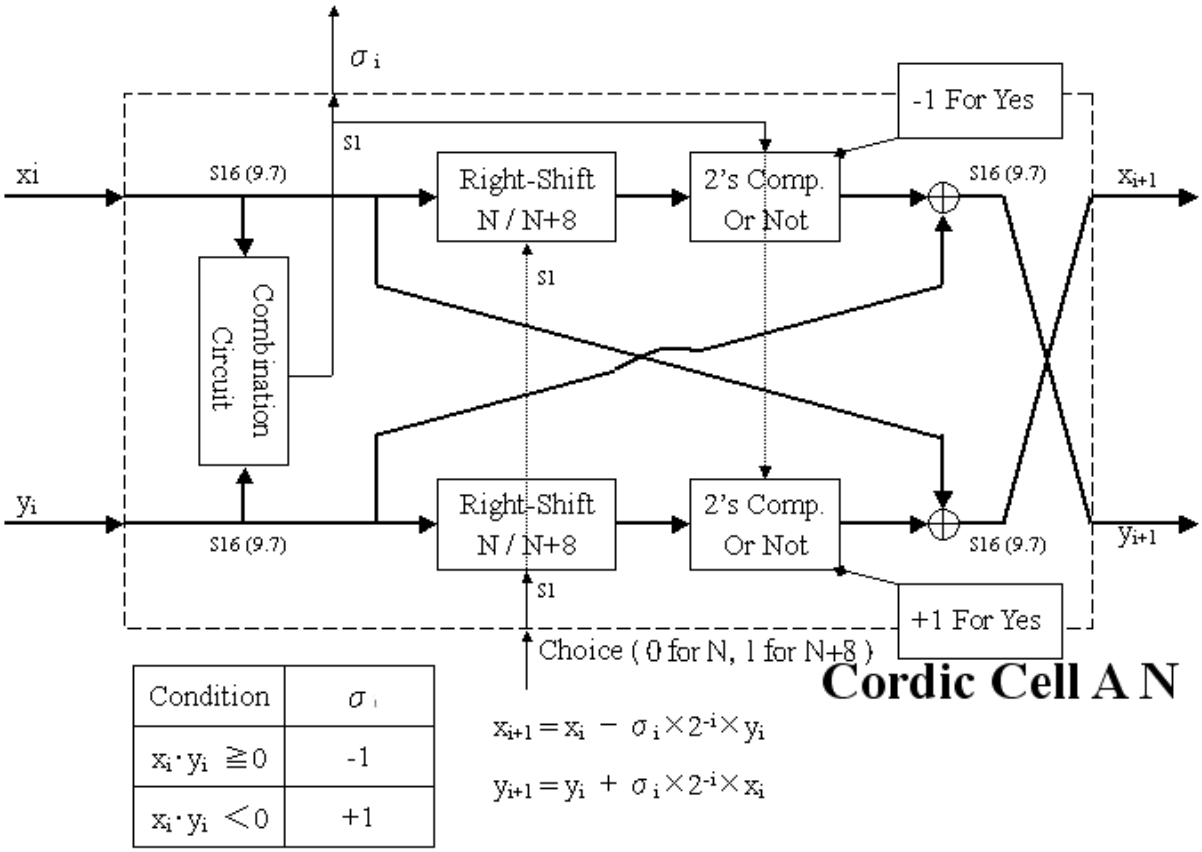


圖 2.2.3.2 Cordic Cell A N 的硬體架構

2.2.3.1 輸入以及輸出訊號

圖(2.2.3.1)中，整個數位座標旋轉作求主幅角的硬體架構：

- Cordic_R：輸入訊號。欲求主幅角的複數向量，其實數部分
- Cordic_I：輸入訊號。欲求主幅角的複數向量，其虛數部分
- Theta_Out：輸出訊號。經由數位座標旋轉演算法求得的主幅角

圖(2.2.3.2)中，Cordic Cell A N 的硬體架構：

- x_i：輸入訊號。經過前一階遞迴，複數向量被旋轉過後的實數部分
- y_i：輸入訊號。經過前一階遞迴，複數向量被旋轉過後的虛數部分
- σ_i：輸出訊號。經由 x_i 和 y_i 的正負所判斷出來此次遞迴階梯角度 α_i 的正負
- x_{i+1}：輸出訊號。經過這一次遞迴，複數向量被旋轉過後的實數部分
- y_{i+1}：輸出訊號。經過這一次遞迴，複數向量被旋轉過後的虛數部分

2.2.3.2 階段角度

如同式(2.2.2.3)，階段角度α_i是已知的，所以我們在硬體的實現時可以當作常數來使用，但是必需要事先求得其實際值，同時還要將其對應到相對應的二進位表示，不然對硬體來說是無法使用的。下表(2.2.3.2)是階段角度α_i、實際值和二進位表示的關係表：

階段角度	實際值	二進位表示	階段角度	實際值	二進位表示
α_0	0.78539	0100000000000000	α_8	0.00390	0000000001010001
α_1	0.46364	0010010111001000	α_9	0.00195	0000000000101001
α_2	0.24497	0001001111110110	α_{10}	0.00097	0000000000010100
α_3	0.12435	0000101000100010	α_{11}	0.00048	0000000000001010
α_4	0.06241	0000010100010110	α_{12}	0.00024	0000000000000101
α_5	0.03123	0000001010001100	α_{13}	0.00012	0000000000000011
α_6	0.01562	0000000101000110	α_{14}	0.00006	0000000000000001
α_7	0.00781	0000000010100011	α_{15}	0.00003	0000000000000001

表 2.2.3.2 階段角度、實際值和二進位表示

2.2.3.3 硬體運作的機制

在 Cordic Cell A N 的硬體架構中，其運作的原理就是形成一次複數平面上角度的旋轉。其輸入是一個複數，然後依照這複數的實部和虛部是否同號，來決定下一個階段角度α_i的正負，這也就是σ_i；然後再利用數學式(2.2.2.6)，利用簡單的位移轉換器和加法器，來求得此複數經過旋轉後的實部和虛部，最後連同σ_i一起作為輸出訊號輸出，供下一階遞迴和整個硬體架構作求主幅角之用。

至於整個硬體架構，就如同我們在本章節 2.2.3 中所述一般：對一個輸入的資料取樣點而言，我們將會有兩個系統的時脈可以使用。所以整個硬體的架構就是建立在兩個系統時脈中，要作完十六次遞迴的基礎之上。也就是說：在一個系統時脈裡，硬體的設計上要經過八次 Cordic Cell A N，這也就是圖(2.2.3.1)裡上半部串接了八個 Cell 的原因；而這八個 Cell 所輸出的σ_i訊號會與由一位

元計數器(One-bit Counter)選擇出相對應的α_i相乘，然後累加。當然在硬體裡，我們為了要節省硬體的負擔，所以不會使用乘法器來實現；相反的，我們打算用多工器來簡化之；使用多工器的主要原因是因為α_i和$-\alpha_i$本身都是已知的常數，所以我們可以用σ_i來做選擇，依σ_i的正負選擇圖(2.2.3.1)中，加法器的輸入中是α_i還是經過二進位補數的$-\alpha_i$，藉此簡化硬體。

最後，在整個硬體設計裡剩下的，就是加法器下方的架構，其本身是一個累加器；如同我們之前說過的，我們把階段角度α_i的累加拆成在兩個系統時脈完成，所以我們也需要一個累加器來把兩次系統時脈裡，求得的階段角度α_i累加值累加，同時這個累加器也需要一個機制能作到歸零的動作，不然角度將會不斷的累加上去，當然這樣的結果不是我們所預期的，所以我們使用一位元計數器來完成此機制；當需要歸零時，回饋的資料就不採用暫存器的訊號線，取而代之的是值為零的常數，藉此作到歸零的動作；相反的，當需要作累加器時，回饋的資料便會採用暫存器的訊號線，達到累加的機制。

在這個累加器中，其實還有個機制，這個機制便是會將累加出來的階梯角度和的第一個字元(MSB)，與當初輸入的複數向量中實部的符號字元(Sign Bit)作 XOR 的動作，我們可以在累加器中的加法器後面，看出硬體在這方面的實現。會有這機制的原因，是因為數位座標旋轉演算法用於求主幅角時，其求出來的角度一定是落於$\pi/2$和$-\pi/2$之間，所以當輸入的複數向量落於第一和第四象限的時候，用數位座標旋轉演算法求出的結果會是正確的，但是當複數向量落於第二第三象限的時候，求出來的角度和實際的主幅角便會有π的角度的差別；若是在第二象限時，數位座標旋轉演算法求出的角度需要加上π值，那才會是真正的主幅角；而要是落在第三象限，則求出的角度需要減去π值，也才會是真正的主幅角；所以我們便利用當初輸入複數向量的實數的符號字元來做判斷的條件，看輸入的複數向量是落在一、四象限，或者是落在二、三象限，然後在依情況增加或是減少π值，但是不論是增加亦或是減少π值，對硬體來說都是將累加出來的階梯角度和的第一個字元與高電壓準位作 XOR 的動作；因此再配合著之前說的判斷條件，便形成了將累加出來的階梯角度和的第一個字元與輸入訊號的實部的

第一個字元作 XOR 的機制了。如此，經過圖(2.2.3.1)的硬體架構後，其輸出的主幅角度不再受限於數位座標旋轉演算法的限制，而必須位在$\pi/2$和$-\pi/2$的範圍之內；相反的，經由此硬體架構後可以將範圍擴充到π和$-\pi$上，讓此演算法更加的完備。

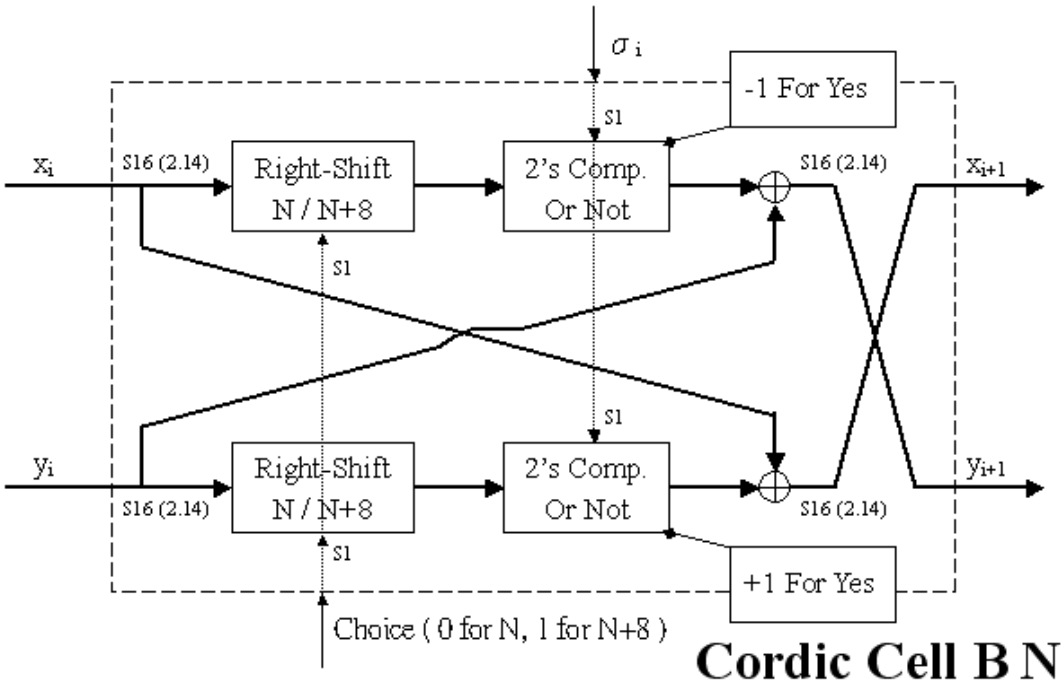
2.2.3.4 字元長度的選擇

接著，我們來看字元長度的選取。如我們之前說的，我們總共需要作十六次的遞迴，而每次的遞迴便會比前一次的遞迴多位移一位元，所以整個硬體中，最大會需要作十六位元的位移，因此我們的資料輸入最少需要十六個字元長度，否則，便會有浪費位移轉換器的精密度的問題。至於階段角度α_i使用了十六個位元表示的原因，主要是因為當數位座標旋轉作旋轉角度之用的時候，其硬體架構也是使用了十六次的遞迴作拆解角度之用，為了不讓硬體的精密度有所浪費，所以我們階段角度α_i使用了十六個位元表示。

至於加法器的輸出訊號線使用十七個字元的原因是因為在表(2.2.3.2)中的α_0，我們是用 0100000000000000 的十六個字元來表示，而在實際的運算上α_0其實就是$\pi/4$，因此若我們使用表(2.2.3.2)轉換的時候，這十六個字元所能表示角度的範圍為$\pi/2$到$-\pi/2$，但是我們在前一章節提過：利用 XOR 累加角度和的第一個字元，我們可以把求得的主幅角度範圍擴大到π和$-\pi$之間；因此我們需要在多一個位元來表示最後主幅角的結果，這就是因為加法器的輸出訊號線需要使用十七個字元的原因。

2.2.4 數位座標旋轉作旋轉角度的硬體架構

如同數位座標旋轉演算法作求主幅角一樣，我們將演算法的遞迴部分，在硬體的實現上分開成 Cell 來設計，這也就是圖(2.2.4.1)中的 Cordic Cell B N，而整個旋轉角度的硬體，我們可以在圖(2.2.4.2)中看到。



$$x_{i+1} = x_i - \sigma_i \times 2^{-i} \times y_i$$

$$y_{i+1} = y_i + \sigma_i \times 2^{-i} \times x_i$$

圖 2.2.4.1 Cordic Cell B N 的硬體架構

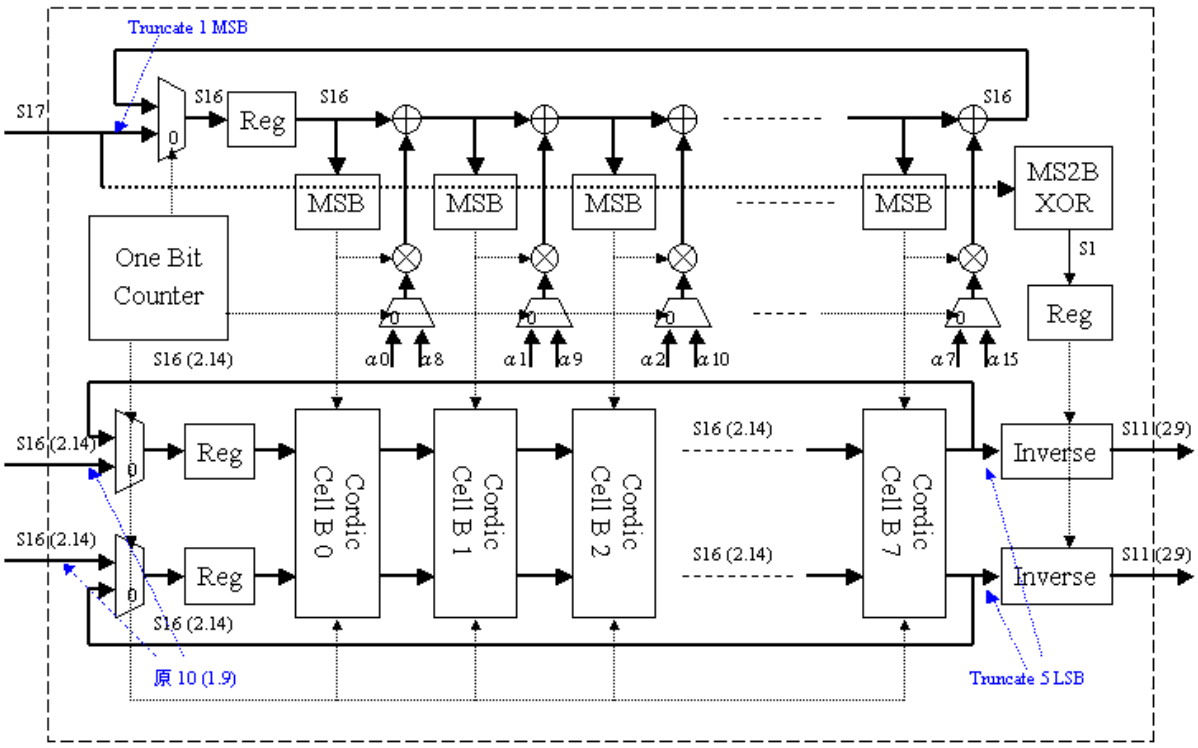


圖 2.2.4.2 數位座標轉換作旋轉角度的硬體架構

2.2.4.1 輸入以及輸出訊號

圖(2.2.4.1)中，Cordic Cell B N 的硬體架構：

- x_i ：輸入訊號。經過前一階遞迴，複數向量被旋轉過後的實數部分
- y_i ：輸入訊號。經過前一階遞迴，複數向量被旋轉過後的虛數部分
- σ_i ：輸入訊號。經由 z_i 的正負所判斷出來此次遞迴階梯角度 α_i 的正負
- x_{i+1} ：輸出訊號。經過這一次遞迴，複數向量被旋轉過後的實數部分
- y_{i+1} ：輸出訊號。經過這一次遞迴，複數向量被旋轉過後的虛數部分

圖(2.2.4.2)中，整個數位座標旋轉作旋轉角度的硬體架構：

- RxIn_R：輸入訊號。欲旋轉複數向量的實數部分
- RxIn_I：輸入訊號。欲旋轉複數向量的虛數部分
- Theta_In：輸入訊號。欲旋轉的角度
- Cordic_OutR：輸出訊號。複數向量經旋轉過後的實數部分
- Cordic_OutI：輸出訊號。複數向量經旋轉過後的虛數部分

2.2.4.2 硬體運作的機制

在 Cordic Cell B N 的硬體架構中，其實現的就是一次複數向量的旋轉，但是和當初 Cordic Cell A N 的硬體架構不同的是，這次旋轉的方向不是由 Cell 的輸入訊號來決定，而是由 Cell 外、整個旋轉角度的硬體中，相對應的角度 z_i 來決定。當相對應的 z_i 大於零的時候，或者是說當 z_i 的第一字元為低電壓準位的時候，表示現在的複數向量應該要往逆時鐘的方向旋轉，而旋轉的角度便是 α_i ，然後將 z_i 減去 α_i 形成 z_{i+1} 的更新，以判別下一次複數向量旋轉的方向，就如同式(2.2.2.5)一樣；上面所說的旋轉方向，其實就是 σ_i 所代表的意義。當 σ_i 為正一時，表示需將複數向量往逆時鐘方向旋轉；相反的，當 σ_i 為負一時，則表示需複數向量往順時鐘方向旋轉。之前所說的似乎是針對當 z_i 大於零的時候，其實當 z_i 小於零也是同樣的道理。

在圖(2.2.4.2)的整個硬體架構當中，其上半部分就是在作 z_i 的更新和求得 σ_i 以供 Cordic Cell B N 作旋轉複數向量之用。判斷旋轉方向的 σ_i ，其實就是

z_i 的第一個字元；當 z_i 大於零的時候，其第一個字元為低電位，相反的當 z_i 小於零的時候，其第一個字元就會為高電位；因此我們可以用 σ_i 的高低電位來決定旋轉的方向是順時鐘或是逆時鐘，有了 σ_i 後便可以更新 z_i 成 z_{i+1}，如此遞迴下去。

而在整個硬體架構圖的下半部分，就是複數向量旋轉的部分；利用上半部分取得的 σ_i 後，送入 Cordic Cell B N 的硬體中作複數向量的旋轉，旋轉出來的複數向量會再送到下一階的 Cell 中，同時上方會更新 σ_i 而得到 σ_{i+1}，配合著剛送入到下一階的複數向量，作下一次的遞迴，如此直到經過了十六次的 Cell 後，最後得到的複數向量就會是我們輸入的複數向量經過我們所要求的旋轉角度後的結果。

最後，在經過 Cordic Cell B 7 旋轉過的複數向量被送出以供其他硬體使用前，還需經過一個變號器(Inverse)，控制這個變號器的條件是當初欲旋轉的角度 Theta_In 最前面的兩個字元互相 XOR 後的結果；若這兩個字元互相 XOR 後為高電位，則經過 Cell 後的複數向量，其實部和虛部需要同時經過變號器的變號後，其結果才會是正確的。反之，若是 XOR 後的結果為低電位，則經過 Cell 後的複數向量可以直接送出供其他硬體使用，而無需使用到變號器。

這個機制的成因是因為數位旋轉座標演算法，在求主幅角或是旋轉角度時有範圍的限制，如同我們之前在本章 2.2.3.3 節所說，其範圍必須落在 $\pi/2$ 和 $-\pi/2$ 之間。因此當我們欲旋轉的角度落於第二或是第三象限時，數位旋轉座標演算法會有不適用的情形；解決的辦法就是將要旋轉的角度先旋轉 π 的角度，如此欲旋轉的角度便會落在 $\pi/2$ 和 $-\pi/2$ 的範圍之間，也就可以使用演算法的硬體架構來求得旋轉後的複數向量；但是這個複數向量和原本我們欲求得的複數向量多旋轉了 π 的角度，所以我們必須要將 π 的角度反轉回來，如此才可以得到正確的結果。這個反轉 π 角度的動作，其實就是將此複數向量的實部與虛部同時變號，這也就是圖(2.2.4.2)的整個硬體架構當中，最後輸出訊號前會有個變號器的原因。

上面所說的是當欲旋轉的角度位於第二或是第三象限時，才需使用的機

制，但若是欲旋轉的角度是位於第一或是第四象限的話，這樣旋轉出來的複數向量將會是錯誤的；因此我們需要個機制去判斷欲旋轉的角度是位於哪個象限，這也就是圖(2.2.4.2)整個硬體中，輸入訊號 Theta_In 取最前面兩個字元互相 XOR 的原因；因為我們在表(2.2.3.2)中 α_i 和十六字元間關係的選取，所以當 Theta_In 的前兩個字元互為不同電壓準位的時候，就表示欲旋轉的角度是位於第二或是第三象限；更精準的來說：當第一個字元為低電壓準位而第二個字元為高電壓準位時，欲旋轉的角度必定會落於第二象限；相反的，當第一個字元為高電壓準位而第二個字元為低電壓準位時，欲旋轉的角度就會落於第三象限。所以我們可以利用這兩個字元互相 XOR 後的結果，作為判斷是否需要反旋轉 π 角度的依據，而綜合了上述所有的機制後，實現出來的便是整個數位座標旋轉演算法作為旋轉角度的硬體架構。

2.2.4.3 字元長度的選擇

接著我們來看字元長度該如何取捨。如同本章 2.2.3.4 節所述，為了要能讓每一次複數向量旋轉的遞迴都可以有效的增加精確度，所以我們在輸入資料的字元長度選擇上，選擇了和遞迴次數相同的大小也就是十六個字元。但是這樣的選擇會有個問題產生，因為數位座標旋轉作旋轉角度時，是用來將類比數位轉換器所取樣到的資料點作頻率偏移的補償，而我們在本章 2.1.3.3 節說過，經由類比數位轉換器取樣到的取樣點會是十個字元大小，這和我們這邊要求輸入為十六字元有所衝突，問題便因此產生了。我們解決的方法是將收到的取樣點作一個符號字元的延伸(Sign-bit Extend)，同時於此取樣點的末位加入五個低電壓準位的字元，如此便形成了十六個字元大小的輸入訊號了。至於只延伸一個符號字元的原因，是因為如同本章 2.1.3.3 節說過：當一個複數向量作旋轉之後，我們需要多使用一個字元來儲存旋轉過後的結果；而這延伸的一個符號字元便是替旋轉後的複數向量，預留一個字元的空間，避免可能產生溢位(Overflow)的問題。

至於欲旋轉角度的訊號線 Theta_In，因為當初數位座標旋轉演算法作求主幅角時，求出來的角度是用十七個字元來儲存，而這求出來的角度就是為了作頻率偏移補償之用，所以欲旋轉角度的訊號線也應該是用十七個字元來儲存。最

後，整個字元長度的選擇便如同圖(2.2.4.2)裡所標示的一樣。