國 立 交 通 大 學
電機與控制工程研究所

碩 士 論 文

Multi-DSP 平台應用於 DOA 與 Beamforming 之即時模擬系統

Real-time DOA and Beamforming Simulation System on a Multi-DSP Platform

研 究 生： 張 家 瑋
指導教授： 胡 竹 生 教授

中 華 民 國 九 十 三 年 七 月


E
S
A
1896

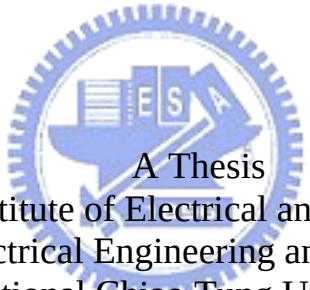
Multi-DSP 平台應用於 DOA 與 Beamforming 之即時模擬系統

Real-time DOA and Beamforming Simulation System on a Multi-DSP Platform

研 究 生：張 家 瑋　　Student：Chia-Wei, Chang
指導教授：胡 竹 生 教授　Advisor：Prof. Jwu-Sheng, Hu

國立交通大學
電機與控制工程學系

碩 士 論 文



A Thesis
Submitted to Institute 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


E
S
A
1896

Multi-DSP 平台應用於 DOA 與 Beamforming 之即時模擬系統

研究生：張　家　瑋　　指導教授：胡　竹　生　教授

國立交通大學電機與控制工程研究所碩士班

摘　要

訊號的抽取，對於陣列天線訊號的傳輸品質來說，是很重要的一個課題。到達角度估測〈DOA〉與波束形成〈Beam forming〉演算法可以有效提高陣列天線訊號接收之訊號雜訊比，得到較佳的通訊品質，但是需要龐大的運算量。本論文以三顆數位訊號處理器為模擬平台核心，藉由分析數位訊號處理器內部運算架構、MUSIC 與 ESPRIT 演算法資料流量分析、個人電腦與數位訊號處理器之間資料快速傳遞方法，建構出適合陣列天線訊號前端處理之即時分散式運算平台。

Real-time DOA and Beamforming Simulation System on a Multi-DSP Platform

Student： Chia-Wei, Chang Advisor： Prof. Jwu-Sheng, Hu

Institute of Electrical and Control Engineering

ABSTRACT

Signal extraction is an important topic for the transmission quality of array signal processing. Direction Of Arrival and Beamforming algorithms improve the signal to noise ratio of antenna array signal, and provide better communication quality. However, it needs huge computation. This thesis takes three DSP as simulation platform kernel. By discussing the internal architecture of DSP, analyzing data flow of both MUSIC and ESPRIT algorithm, and fast data transmission technique between PC and DSP, a real time distributed computation platform suitable for the front-end of antenna array signal processing is realized.

誌謝

研究所短短兩年的時光，來的快，去的也快，雖然一路上跌跌撞撞，但終究也到達了終點。

感謝 905 實驗室的大家長—胡竹生老師，在兩年前一個陽光燦爛的下午，滿面笑容的答應讓我進入實驗室，才有這接下來一連串的點點滴滴。

歡樂 905 的學長同學們，如果沒有你們的鼓勵與支持，恐怕現在我還陷在 matlab 模擬的水深火熱之中：神通廣大無所不知鄭博士、天真白爛重訓天王劉大人、男子氣概擴充天王蘇學長、交大電控機器人權威立偉學長、天線組第一代宇宙無敵霹靂強家銘、承先啟後提攜後輩天線組第二代鍾公青衛兄、永遠的組員造型女王蕭小琪、阿毛春捲機歪仁鳥哥、自言自語古道熱腸無師自通康康、冷鋒颼颼赤子之心順子、實驗室之花便當代言人憶如、永遠的管理員蔡安喬、吃菜也會胖的俊德、少一根經碎碎念的宴榮、木訥寡言的群棋、只喝咖啡的鏗元、身價百萬的興哥、很會游泳的士奇、吃不胖的實驗室一姐。

感謝有情有義的山服人，一同在人生的重要時刻，不要命的挺身而出，只為了一份感動：義氣建仁、胖子照慶、大屁股皓棠、好相處韻璇、慵懶摩卡、固執小龜、捲毛性獸、瘋狂鼓手元元、摧琴手小白、可愛麵包、身手矯健哈帕、中流砥柱倩嫻、瘦排骨郭博、爛漫天真大哥、妖姬品瓜、大臉柚子、負責的 a 達，以及數以萬計的學弟妹們。

感謝我的父母大人、祖母大大人、姑姑、妹妹，從小到大對我的無微不至照顧，讓我有這個機會，坐在這裡打這篇誌謝。也要感謝一直都陪在我身旁的姵汎，不管是誰在照顧誰，終究是互相扶持，互相砥礪。

最後還是要感謝 905 實驗室的大家長—胡竹生老師，在兩年後一個陽光燦爛的下午，滿面笑容的答應讓我離開實驗室，才結束這段風雨飄搖的日子。

回首向來蕭瑟處，歸去，也無風雨也無晴。

目錄

摘　要 ..... **i**
ABSTRACT ..... **ii**
誌謝 ..... **iii**
目錄 ..... **iv**
圖表 ..... **vi**
第一章　緒論 **1**
1.1 研究動機 1
第二章　陣列訊號處理 **4**
2.1 基本概念 5
2.2 陣列天線形狀 6
2.2.1 均勻線性排列天線 6
2.2.2 平面排列天線 7
2.2.3 環形排列天線 8
2.3 空間濾波 9
2.3.1 訊號源角度估測 9
2.3.1.1 MUSIC 9
2.3.1.2 ESPRIT 11
2.3.2 波束形成 14
2.3.2.1 MVDR Beamforming 15
2.4 多重路徑訊號 16
2.4.1 Forward/Backward Smoothing 16
2.5 實際訊號分析 18
2.5.1 Channel Sounder 簡介 19
2.5.2 實際量測資料 20
2.5.3 訊號分析 24
第三章　實驗平台架構 **26**
3.1 硬體簡介 26
3.1.1 DM11 26
3.1.2 DP12 27
3.1.3 ES10 28
3.1.4 6701 DSP 29
3.1.5 PCI Bridge 30
3.1.6 基本架構 31

3.2 軟體架構 31
3.2.1 Host <-> DSP 31
3.2.2 DSP <-> DSP 33
3.2.3 程式開發流程 34
第四章　運算效能分析 36
4.1 6701 DSP 運算原理 36
4.1.1 平行處理的運算單元 36
4.1.2 管線化處理的硬體架構 38
4.1.3 執行指令階段 39
4.2 軟體效能的提昇 41
4.2.1 記憶體指標的影響 41
4.2.2 平均的使用運算單元 42
4.2.3 多個執行週期指令對暫存器的影響 43
4.2.4 硬體資源的衝突 46
4.2.5 資料的相關性 48
4.2.6 線性組合語言 50
4.3 演算法分析 53
4.3.1 MUSIC 53
4.3.2 ESPRIT 54
4.3.3 分散式運算架構 55
第五章　結論 58
5.1 研究結果 58
5.2 未來展望 58
參考文獻 60

圖表

圖 1 遠場、近場示意圖 5
圖 2 均勻線性排列天線 6
圖 3 平面方形陣列 7
圖 4 平面環形陣列 8
圖 5 MUSIC 角度估測能量分佈 11
圖 6 ESPRIT 訊號分群法則示意圖 12
圖 7 波束形成示意圖 14
圖 8 MVDR 波束形成能量分佈圖 16
圖 9 前後空間平滑子矩陣分群法則示意圖 17
圖 10 CHANNEL SOUNDER 的內部方塊圖 18
圖 11 四根天線的切換情形 19
圖 12 交通大學兩側分佈圖 20
圖 13 視線範圍訊號的空間通道響應 21
圖 14 多重路徑訊號的空間通道響應 21
圖 15 視線範圍訊號的到達角度估測 23
圖 16 多重路徑訊號的到達角度估測 23
圖 17 視線訊號範圍的角度估測 24
圖 18 多重路徑訊號的角度估測 24
圖 19 DM11 內部方塊圖 26
圖 20 DP12 內部方塊圖 27
圖 21 實驗板整體內部方塊圖 28
圖 22 ES10 與 DP12 連接示意圖 28
圖 23 TMS320C6701 方塊圖[10] 29
圖 24 PLX 9054 方塊圖[9] 30
圖 25 平台硬體架構圖 31
圖 26 PC 端與 DSP 端溝通架構 33
圖 27 MCBSP 溝通接腳圖 34
圖 28 演算法開發流程圖 34
圖 29 效能分析 36
圖 30 八個運算單元平行處理組語表示 36
圖 31 運算單元指令集分類[11] 37

圖 32 為最佳化的洗衣流程 [9] 38
圖 33 管線化處理後的洗衣流程 [9] 39
圖 34 6701 三階段管線化示意圖 39
圖 35 6701 管線化執行階段硬體架構圖 [11] 40
圖 36 運算單元使用狀況表 43
圖 37 運算單元使用狀況表 44
圖 38 預測管線化的程式執行情況 44
圖 39 執行延遲時間造成的管線化執行效能降低分析表 45
圖 40 運算單元使用狀況表 45
圖 41 四個執行週期的內部迴圈資源使用狀況逐步分析圖 47
圖 42 資料相關程度分析圖 49
圖 43 內部迴圈效能分析步驟 52
圖 44 MUSIC 空間濾波方塊圖 53
圖 45 ESPRIT 空間濾波方塊 54
圖 46 效能分析 55
圖 47 適合空間濾波之分散式數位訊號處理器架構 56
圖 48 效能分析 57

第一章　緒論

1.1 研究動機

隨著無線通訊的迅速發展，社會多媒體訊息交流的急遽增加，人們對於無線通訊服務的品質要求，與日俱增。如何提昇資料傳輸的品質，降低發射功率的消耗，有效率的利用有限的頻寬，已成了無線通訊領域最關注的話題。而智慧型天線〈Smart Antenna〉技術，即為一提升頻譜使用效率、系統容量和通訊品質的有效途徑。

傳統的基地台設計，是以指向性天線或全向性天線，來均勻的分配至基地台四周的空間，由於無法判斷使用者方位，所以無法依照訊號源方向來調整每根天線發射功率，只能以提高所有天線的功率，以確保可以涵蓋整個接收範圍，這樣子的做法，缺點是造成大量的功率耗損，並且對於頻譜的使用來說，是很沒有效率的。而智慧型天線的概念，是透過陣列天線來接收訊號，藉著從接收到的訊號中，以複雜的演算法來計算出使用者所在的位置，作為調整每一根天線訊號發射或接收功率的依據，來達到減少能量的損耗，增加基地台涵蓋區域內頻譜使用率的目的。[19]因此，要實現智慧型天線系統，基本的兩個問題，便是『如何估計訊號方向』以及『如何接收某個方向的訊號』。

估計訊號方向，最典型的演算法，便是 Multiple Signals Classification Method〈MUSIC〉與 Estimation of Signal Parameters via Rotational Invariance Techniques〈ESPRIT〉。這兩種演算法，都是藉著分析訊號空間特徵的差異，來分離出訊號空間與雜訊空間，進而找出想要取得的訊號方向。而空間濾波〈Spatial filtering〉則是接收方向性訊號所常用的方法。Minimum Variance Distortionless Response〈MVDR〉與 Fourier 方法是利用空間特徵的分析，以及已知訊號來源的方向，透過波束形成〈Beam forming〉的技

巧，來抑制非訊號方向的雜訊，得到最佳的接收訊號。在論文中，將針對這些方法做討論與分析。

1.2 研究目標

龐大的運算量，是建此系統時所面臨的最主要問題。在實驗平台已大致架設完成的情況下，論文中，將討論如何將此平台的運算能力，發揮到極限。論文中將從下列觀點切入來做探討：

- 運算架構

本實驗平台的特色是擁有三顆數位訊號處理器。之前畢業學長已規劃出運算架構大致的輪廓，本論文中，將整合兩位學長建立的架構，針對 MUSIC 與 ESPRIT 等空間濾波的演算法，找出最適合的運算架構。[15][16]

- DSP 處理器的運算效能

本實驗平台搭配的三顆數位訊號處理器，是德州儀器公司 TMS320C6701 系列，它與一般微處理器不同之處，在於它分析了數位訊號處理常用的運算，結合德州儀器公司本身先進的硬體設計技術，作為它硬體架構的基礎，使得這顆數位訊號處理器，有強大的運算能力。本論文希望透過演算法的分析與軟體的最佳化，來實現分散式的運算，探討此實驗平台運算的能力。

1.3 論文內容概述

本論文內容共分為五章：

第一章：概論研究動機與研究目標，並說明論文架構。

第二章：介紹空間濾波的理論，包括了陣列訊號處理的基本原理以及方法，並透過 ChannelSounder 量測到的真實訊號，驗證平台演算法。

第三章：說明平台軟硬體的架構。

第四章：經由實作分析平台的運算效能，並探討改善方法。

第五章：結論與展望

第二章　陣列訊號處理

陣列訊號處理，是使用一組陣列訊號接收器，將接收到的訊號做適當的處理，達到空間濾波〈spatial filtering〉的方法。訊號由傳送端發射經過空間介質的傳遞之後，接收端以陣列天線接收，並依據訊號到達每根天線時間不同的性質，來解出相對於天線的訊號來源方向，並利用波束形成的理論，來完成空間濾波。經由空間濾波，可以提高訊號雜訊比〈SNR〉，以達到更好的通訊品質；另一方面，利用空間濾波可以排除空間中不需要的雜訊，讓接收端收到單純來自某個方向的訊號。

陣列訊號理論在通訊上的應用相當廣泛，從軍事雷達系統、無線通訊、視訊會議等，都有涉及。本論文主要探討的部分，屬於無線通訊的應用。在基地台的訊號發射端，使用陣列天線可以帶來的優點有很多：

- 容量增加：智慧型天線可增強有用的接收信號強度，同時降低干擾強度，可以增加話務量，尤其是可調性陣列，它能產生大幅度的改進。
- 通訊範圍增加：在鄉下或人煙稀少的地區，基地台部署的前提是無線電涵蓋區域，而非容量。由於智慧型天線的指向性較傳統的扇形或全方向天線高，故通訊範圍有可能增加。這表示基地台可以隔得更遠，有可能導致更節省成本的部署。與單一原件天線相比，天線增益值增加的量等於天線的元件個數，例如：有八個元件的天線可提供的增益值為八。
- 新型服務：使用智慧型天線時，網路可以使用關於用戶的空間資訊。這項資訊可以用來估算用戶的位置，比現有的網路準確得多。在緊急呼叫及定點帳單通知等服務中可使用定位功能。
- 安全性：使用智慧型天線時，要監聽連線將更為困難。若想成功地監聽連線，侵入者必須與用戶處於同一方向上〈從基地台來看〉。
- 減少多重路徑傳播：在基地台使用狹窄的天線波束，可稍微減少多重路徑傳播。實際減少的程度視情況而定，但不會很大量。

2.1 基本概念

訊號源本身可以被清楚的分辨為某個頻率，稱之為窄頻訊號源〈narrowband source〉。相對於此，一個訊號是由一段頻帶內的頻率所組成，則稱之為寬頻訊號源〈broadband/wideband signal source〉。從物理角度來看，訊號源可能離陣列天線相當遠，也有可能就在天線的附近。前者，是屬於遠場的情況〈far-field〉，陣列天線收到訊號源傳來的訊號，是屬於平行波的形式；而後者是近場的情況〈near-field〉，陣列天線收到的訊號形式，是屬於球形波的形式。論文中所要處理的訊號源，是屬於遠場的窄頻訊號。

訊號之間的關係，可能是不相關〈uncorrelated〉、部分相關〈partially correlated〉、或是完全相關〈completely coherent〉。在自然界中，雜訊與人為的訊號，往往是不相關的，所以我們在天線訊號的處理上，也做了這個假設。[7]

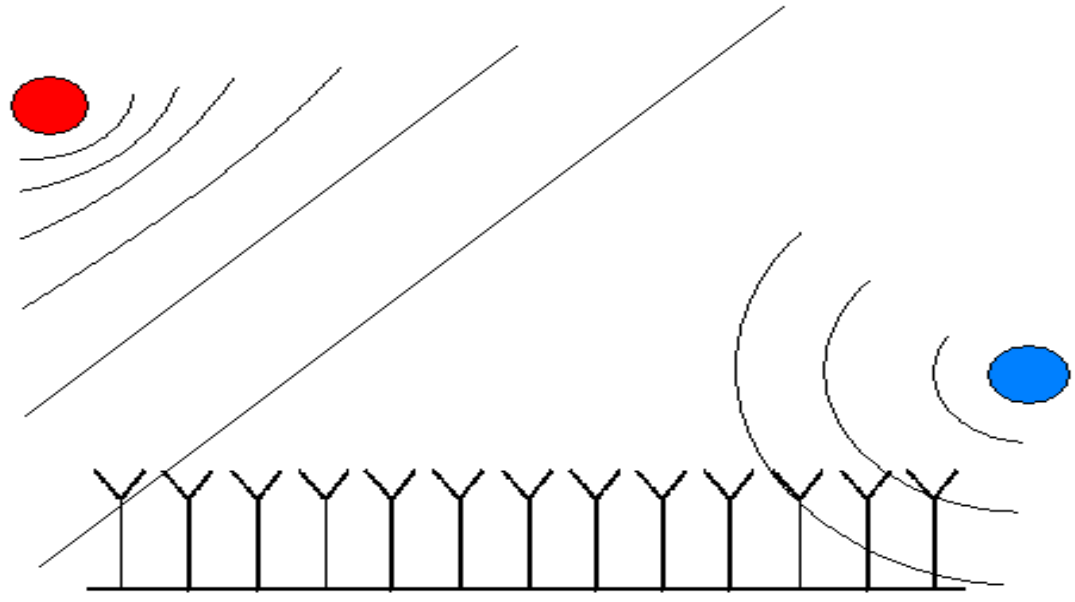


圖 1 遠場、近場示意圖

2.2 陣列天線形狀

一般來說，我們最常用的陣列天線，可分為幾種排列方式：均勻線性排列天線〈Uniform Linear Array〉、平面排列天線〈Planar Array〉以及環形排列天線〈Circular Array〉。

2.2.1 均勻線性排列天線

均勻線性排列天線〈Uniform Linear Array〉，是最常使用的天線形式，它的優點在於容易實現，且數學模型的推導最為簡單。以下是簡單的推導：

X1、X2、X3：代表天線收到的訊號

S：訊號源

r：訊號源到陣列天線的距離

Vc：波速

d：天線之間的固定距離

θ：訊號入射角度

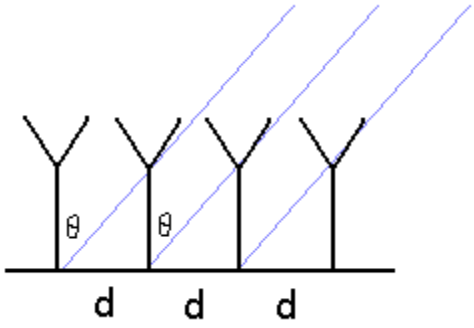


圖 2 均勻線性排列天線

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} S(t-\frac{r}{v_c}) \\ S(t-\frac{r}{v_c}-\frac{d\sin\theta}{v_c}) \\ S(t-\frac{r}{v_c}-\frac{2d\sin\theta}{v_c}) \end{bmatrix} \Leftrightarrow \begin{bmatrix} x_1(e^{jw}) \\ x_2(e^{jw}) \\ x_3(e^{jw}) \end{bmatrix} = \begin{bmatrix} S(e^{jw})e^{-jw\frac{r}{v_c}} \\ S(e^{jw})e^{-jw(\frac{r}{v_c}+\frac{d\sin\theta}{v_c})} \\ S(e^{jw})e^{-jw(\frac{r}{v_c}+\frac{2d\sin\theta}{v_c})} \end{bmatrix}$$

我們以第一根天線作為參考點，則接收到的訊號在時域與頻域分別如以上的表示。以下我們做了兩個假設：一個是遠場的情況〈far-field〉，所以 r 將趨近於無限大；另一個是以第一根天線接收到訊號源的訊號，當作參考訊號。經過兩個假設，表示式簡化為以下形式：

$$\begin{bmatrix} x_1(e^{jw}) \\ x_2(e^{jw}) \\ x_3(e^{jw}) \end{bmatrix} = \begin{bmatrix} x_1(e^{jw}) \\ x_1(e^{jw})e^{-jw(\frac{d\sin\theta}{v_{s1}})} \\ x_1(e^{jw})e^{-jw(\frac{2d\sin\theta}{v_{s1}})} \end{bmatrix} = \begin{bmatrix} 1 \\ e^{-jw(\frac{d\sin\theta}{v_{s1}})} \\ e^{-jw(\frac{2d\sin\theta}{v_{s1}})} \end{bmatrix} \begin{bmatrix} S_{x_1}(e^{jw}) \end{bmatrix}$$

以相同的道理，當訊號源有三個的時候，則表示式可擴充為以下形式：

$$\begin{bmatrix} x_1(e^{jw}) \\ x_2(e^{jw}) \\ x_3(e^{jw}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ e^{-jw(\frac{d\sin\theta_1}{v_{s1}})} & e^{-jw(\frac{d\sin\theta_2}{v_{s2}})} & e^{-jw(\frac{d\sin\theta_3}{v_{s3}})} \\ e^{-jw(\frac{2d\sin\theta_1}{v_{s1}})} & e^{-jw(\frac{2d\sin\theta_2}{v_{s2}})} & e^{-jw(\frac{2d\sin\theta_3}{v_{s3}})} \end{bmatrix} \begin{bmatrix} S_{1x_1}(e^{jw}) \\ S_{2x_1}(e^{jw}) \\ S_{3x_1}(e^{jw}) \end{bmatrix}$$

以上的式子說明了天線端所接收到的訊號，是訊號源經過空間中不同的延遲效應之後，疊加而成。而這個特殊的轉換矩陣，稱之為 manifold matrix[5]，這個矩陣在往後理論介紹當中，會有更進一步的探討。

2.2.2 平面排列天線

均勻線性陣列天線在使用上最大的缺點，在於它只能分析一維的訊號源，當多個訊號源從空間中不同的角落入射的時候，則均勻線性陣列天線就不敷使用了。為了改進這個問題，有人提出了平面排列天線〈Planer Array〉。我們先設定左下角的天線作為參考點，x 軸方向有 M 根天線，y 軸方向有 N 根天線。接下來的推導與均勻線性排列天線的推導原理相同，只是平面方形陣列增加了一個維度。以一個訊號源推導出來的 manifold vector 是以下的形式：

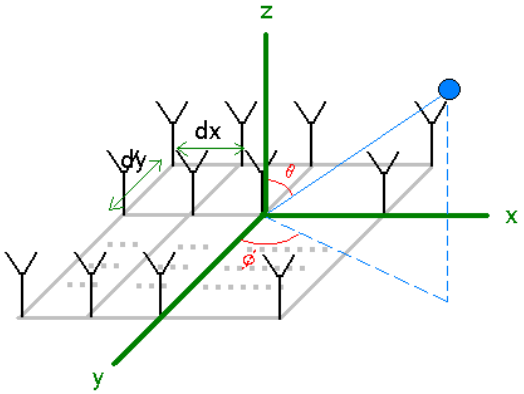


圖 3 平面方形陣列

$$a^T(\theta,\phi) = [\ 1 \quad e^{-jw\frac{d_x \sin\theta\sin\phi}{Vc}} \quad \dots \quad e^{-jw\frac{(M-1)d_x \sin\theta\sin\phi}{Vc}}$$
$$e^{-jw\frac{d_y \sin\theta\cos\phi}{Vc}} \quad \dots \quad e^{-jw\frac{\sin\theta(d_x\sin\phi + d_y\cos\phi)}{Vc}}$$
$$\dots \quad e^{-jw\frac{\sin\theta((M-1)d_x\sin\phi + (N-1)d_y\cos\phi)}{Vc}}\ ]$$

2.2.3 環形排列天線

除了平面排列天線之外，環形排列天線也是改良均勻線性陣列的一種，它使用了球形座標，所以讓最後得到的 manifold vector 表示法較平面排列天線來的簡單。

圓形的排列，我們改以圓心作為參考點，整個圓上的天線數目共有 M 個，推導得到的 manifold vector 為以下的形式：

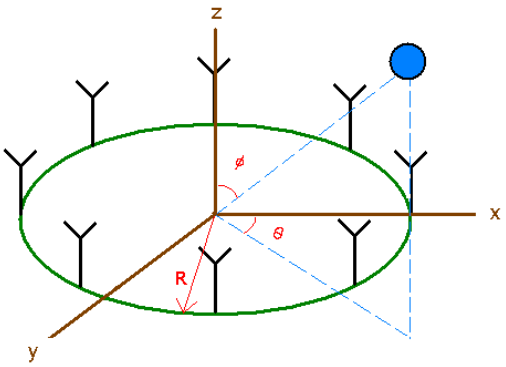


圖 4 平面環形陣列

$$a^T(\theta,\phi) = \left[e^{-jw\frac{R\sin\phi\cos\theta}{Vc}} \quad e^{-jw\frac{R\sin\phi\cos(\theta-\frac{2\pi}{M})}{Vc}} \quad \dots \quad e^{-jw\frac{R\sin\phi\cos(\theta-\frac{2(M-1)\pi}{M})}{Vc}} \right]$$

2.3 空間濾波

對於陣列訊號處理來說，最重要的問題就是我們該如何從收到的訊號中，抽取出我們想要的訊號。空間濾波的方法，在早期的做法，以相關矩陣為基礎〈covariance-based〉的方法，直接以相關矩陣來做運算，達到空間濾波的目的，例如：『Fourier method』或是『MVDR method』，這種方法的優點是演算法運算較為簡單，易於實現，缺點是要使用較多的天線，才能得到效果。比較新的做法，是以特徵向量為基礎〈eigenvector-based〉的演算法，這類的方法，並不直接使用相關矩陣，而是利用分析相關矩陣的特徵，來作為理論分析的基礎。這一類的方法，對於天線數目的要求較低，效果也比較好，不過，運算也比較複雜。論文中使用的演算法，是屬於後者，以下將針對特徵向量為基礎的兩種演算法做說明。

2.3.1 訊號源角度估測

2.3.1.1 MUSIC

MUSIC 演算法的全名，是 Multiple Signals Classification Method。以下是這種方法的推導：

在陣列天線接收端收到的訊號中，除了訊號源發射的訊號以外，還混雜著雜訊在裡面，所以收到的訊號可以表示成以下的形式：

$$X(n) = \sum_{i=1}^{M} a(\theta_i) S_i(n) + N(n)$$
$$= AS_i(n) + N(n)$$

其中 A 矩陣，就是先前推導出來的 manifold matrix。在訊號處理上，想針對一段訊號有更清楚的了解，最常用的方法就是去分析協方差〈covariance〉。首先，我們針對接收到的陣列訊號，做 auto-covariance：

$$R_{XX} = E\left(x, x^H\right) = E\left(AS + N, (AS + N)^H\right)$$
$$= AR_{SS}A^H + \sigma_N^2 I$$

其中，訊號與雜訊 cross-variance 是零的原因，是基於前面的基本假設。因為雜訊的能量遠小於訊號，所以以上的式子可以簡化為

$$R_{xx} \approx AR_{SS}A^H$$
$$\Rightarrow \quad V\Lambda^2 V^H = AR_{SS}A^H$$
$$\Rightarrow \quad [V_1\, V_2 \ldots V_M]\begin{bmatrix} \lambda_1^2 & 0 & \ldots & 0 \\ 0 & \lambda_2^2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \lambda_M^2 \end{bmatrix}[V_1\, V_2 \ldots V_M]^H = AR_{SS}A^H$$

在我們對訊號的 covariance matrix 做特徵值分解之後，我們就可以更深入到訊號空間來分析。首先，R_{ss} 與 R_{xx} 的關係，是透過 manifold matrix 的相似轉換，所以，向量空間 V_i 與 manifold matrix 的行向量，是對應到相同的空間。λ_1 到 λ_M 分別對應 V_1 到 V_M 的向量空間，從能量的觀點來看，訊號的能量會遠大於雜訊，所以在陣列天線數目大於訊號源的數目時，從 λ_1^2 到 λ_M^2 的大小，就可以判斷出何者為訊號，何者為雜訊。MUSIC 演算法另一個基本假設，是訊號源彼此之間是不相關〈uncorrelated〉，所以不同訊號源對應的訊號空間 V_i 彼此之間是互相正交的〈orthogonal〉。藉由此正交的關係，我們可以由雜訊空間來反求出訊號的方向來。[1][5]

$$MAX\left(\left\|V_{Noise}A(\theta)\right\|\right) \quad , \quad \theta = -90^0 \sim 90^0$$

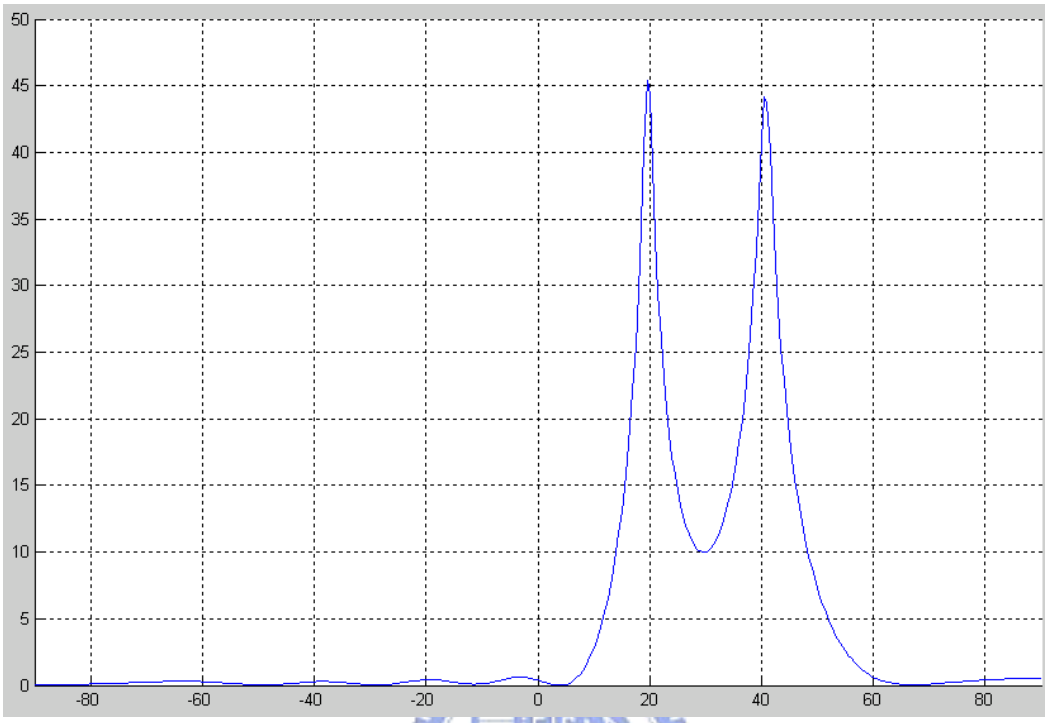


圖 5 MUSIC 角度估測能量分佈

以上是透過 MUSIC 演算法估測訊號源角度，上圖表示了 $\left\|V_{Noise}A(\theta)\right\|$ ， $\theta=-90^{0}\sim 90^{0}$ 的能量分佈，可以清楚的看出來在 20 度與 40 度有訊號入射。然而，在找出訊號源之後，想要擷取出訊號來，還必須再對訊號做波束形成〈Beamforming〉。波束形成的理論，將在後面做介紹。

2.3.1.2 ESPRIT

ESPRIT 演算法是另一種常用的訊號源角度估測的方法，全名是 Estimation of Signal Parameters via Rotational Invariance Techniques，推導如下：

ESPRIT 在均勻線性陣列天線的處理方式，是先將矩陣接收到的訊號，如下圖分先成 X1、X2 兩群，再重新組合，至於原因，接下來會說明。

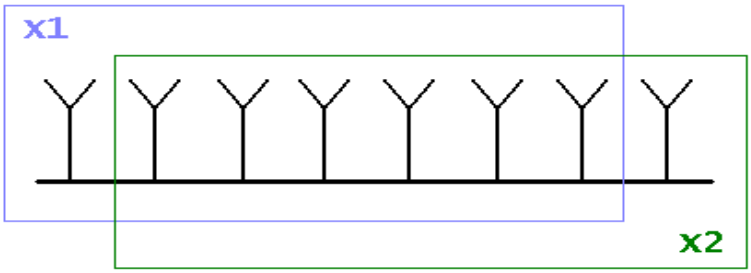


圖 6 ESPRIT 訊號分群法則示意圖

X1 與 X2 這兩群天線所收到的訊號，事實上只有相差一個相位的延遲，所以我們以ϕ來表示時間的延遲，它的形式是

$$\begin{cases} X1 = AS + n_{x1} \\ X2 = A\phi S + n_{x2} \end{cases}$$

$$\Rightarrow X = \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} A \\ A\phi \end{bmatrix} S + \begin{bmatrix} n_{x1} \\ n_{x2} \end{bmatrix} = A_z S + n_z$$

其中ϕ是一個延遲矩陣，而 A 則是如前面定義的 manifold matrix。

$$\phi = \begin{bmatrix} e^{-jw(\frac{d\sin\theta_1}{v_{s1}})} & 0 & \cdots & 0 \\ 0 & e^{-jw(\frac{d\sin\theta_2}{v_{s2}})} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & e^{-jw(\frac{2d\sin\theta_M}{v_{sM}})} \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 1 \\ e^{-jw(\frac{d\sin\theta_1}{v_{s1}})} & e^{-jw(\frac{d\sin\theta_2}{v_{s2}})} & \cdots e^{-jw(\frac{d\sin\theta_M}{v_{sM}})} \\ e^{-jw(\frac{2d\sin\theta_1}{v_{s1}})} & e^{-jw(\frac{2d\sin\theta_2}{v_{s2}})} & \cdots e^{-jw(\frac{2d\sin\theta_M}{v_{sM}})} \end{bmatrix}$$

為了分析訊號的特性，我們對 X 做 auto-covareance，

$$E\left(x, x^H\right) = E\left(A_z S + n_z, (A_z S + n_z)^H\right)$$

$$\Rightarrow \quad R_{xx} = A_z R_{SS} A_z^H + \sigma_n^2 I$$

$$\Rightarrow \quad V \Lambda V^H \approx A_z R_{SS} A_z^H$$

因為 V 與 A_z 的行向量對應到相同的子空間，所以 V 與 A_z 之間，必定存在一個矩陣 T ，使得

$$V = A_z T$$

$$\Rightarrow \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} AT \\ A\phi T \end{bmatrix}$$

我們直接對特徵向量矩陣分成 V_1 、 V_2 ，對應到 AT 與 $A\phi T$ 。

$$V_1^{-1} V_2 = \left(AT\right)^{-1} \left(A\phi T\right)$$

$$= T^{-1} A^{-1} A \phi T$$

$$= T^{-1} \phi T$$

由上式中，可以看出我們所要找的訊號源角度資訊，可以從 $V_1^{-1} V_2$ 的特徵值分解直接找到。[5][6]這個方法最大的優點，在於它省去了 MUSIC 演算法在尋找訊號源角度時，掃描各個入射角能量的龐大運算。

在找到訊號源角度之後，接下來也是要做波束形成。

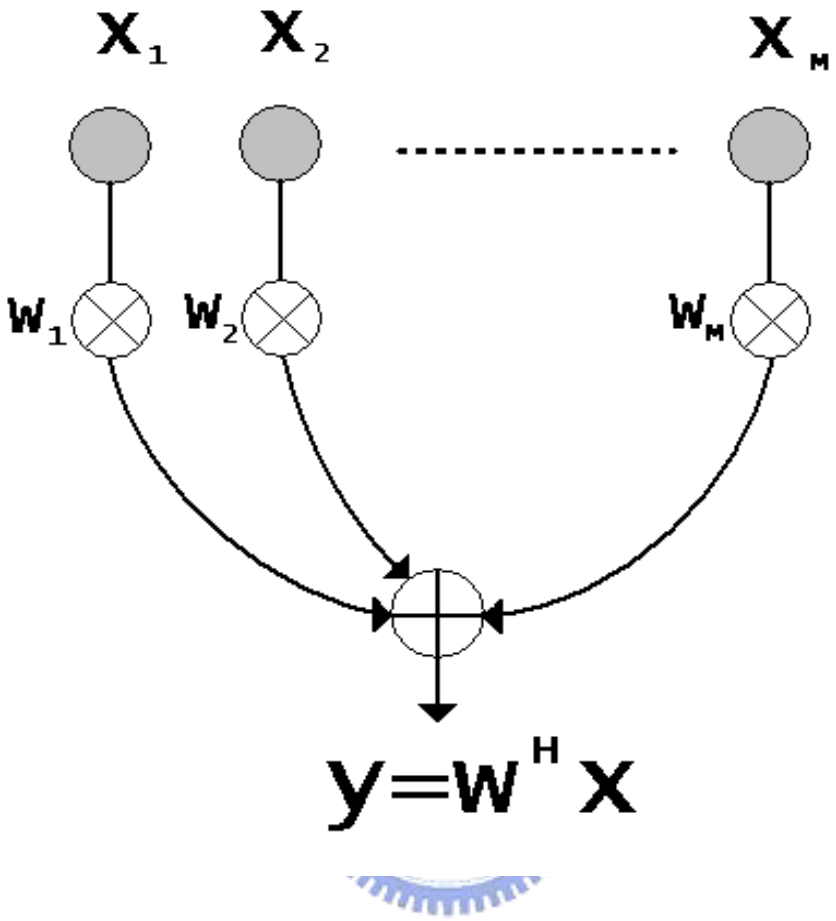


圖 7 波束形成示意圖

2.3.2 波束形成

波束形成的方法中，以 Fourier Beam Forming 與 MVDR Beam Forming 最為容易實現，也最為常用。在論文中，選擇了 MVDR，原因是 MVDR Beam Forming 演算法除了使用訊號源的入射角度之外，還考慮了相關矩陣提供的資訊，不但保有了訊號源角度的訊號，更有效的壓抑了其他方向入射雜訊的能量，讓接收到的訊號，有更好的品質；而 Fourier Beam Forming 則是指考慮了訊號源的入射角度的資訊，單單只對入射角度的訊號加強，並無對其他的雜訊做處理，所以擷取出來的訊號品質比較差。

2.3.2.1 MVDR Beamforming

波束形成的基本構想，是希望對每根陣列天線收到的訊號，乘以不同的權重，使每跟訊號加總起來的訊號，可以有濾波的效果。所以我們假設經由空間濾波後的訊號，可以表示成 $y = W^H X$ 。MVDR 演算法的基本假設有兩個：第一個是將訊號源入射角度的訊號能量維持在 0dB，第二個是要讓收到訊號的能量降到最低。這兩個假設組合在一起，就達到了濾除雜訊的效果。收到訊號的能量，可以表示為

$$E\left\{\left|y\right|^2\right\} = E\left\{\left|W^H X\right|^2\right\} = W^H R_{xx} W$$

為了同時滿足兩個假設，使用了 Lagrange Multiplier

$$\begin{cases} \nabla_W [W^H R_{xx} W] - \lambda \nabla_W [W^H a(\theta_s)] = 0 \\ W^H a(\theta_s) = 1 \end{cases}$$

可解出

$$\lambda = W^H R_{xx} W = \frac{1}{a^H(\theta_s) R_{xx}^{-1} A(\theta_s)}$$

$$\Rightarrow \quad W = \frac{1}{a^H(\theta_s) R_{xx}^{-1} A(\theta_s)} R_{ss}^{-1} A(\theta_s)$$

而 W 即為每根天線的權重值，$y = W^H X$ 可濾出想要的訊號。

下圖是 MVDR 演算法對於 $\left\|W^H a(\theta)\right\|$ ， $\theta = -90^0 \sim 90^0$ 所掃出的能量圖。我們可以明顯的看出，此時 MVDR 所要接收的訊號是來自 40 度入射的訊號源，並且也將來自 20 度的訊號源，做了將近 35dB 的壓抑，其他角度的雜訊，也都有明顯的壓制。[5]

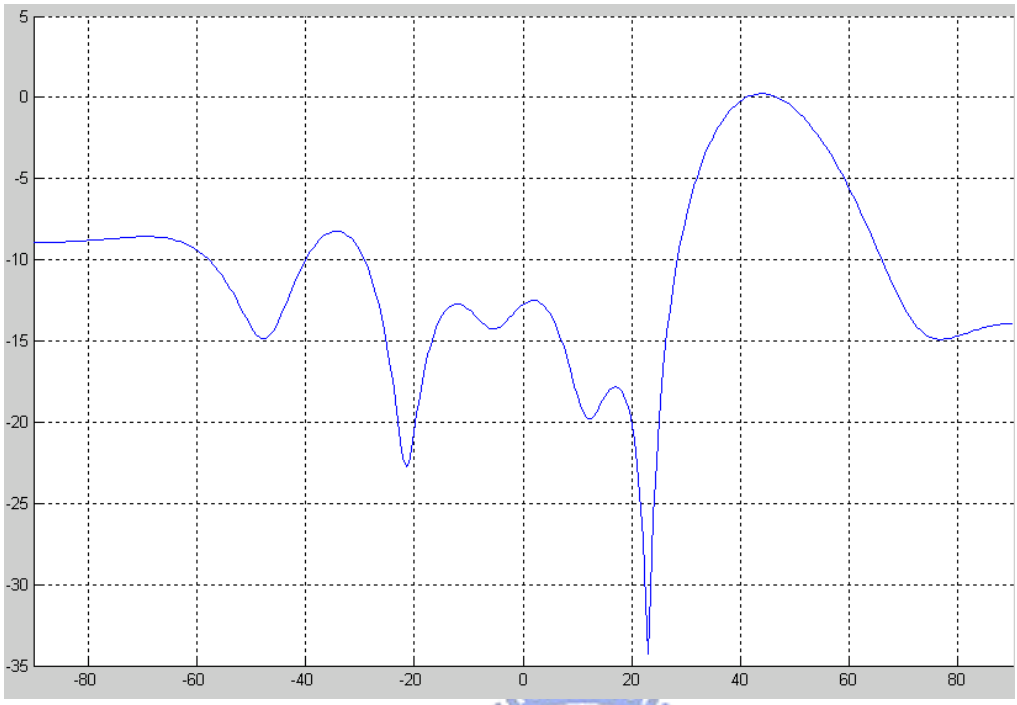


圖 8 MVDR 波束形成能量分佈圖

2.4 多重路徑訊號

之前討論過，訊號可以分成完全相關、部分相關以及不相關，而前面介紹的訊號源入射角度估測方法，都是建立在訊號彼此之間是不相關的理論基礎上，然而，在實際應用上，卻是有可能發生訊號源彼此之間存在某種程度的相關性。在這種情況發生的時候，我們之前的 eigenvector-based 訊號角度估測方法，就會因此讓 covariance matrix 變成奇異矩陣，無法做 EVD 分解，造成估測結果產生相程度的誤差。為了克服這種情況的發生，我們將利用一種相關性消除的演算法，來維持接收訊號的品質。

2.4.1 Forward/Backward Smoothing

這個演算法的目的，是消除訊號相關性所造成的相關矩陣無法滿秩的情況，以讓 eigenvector-based 的訊號入射角度估計演算法，得以順利進行。以下是它的做法：

對於均勻線性陣列天線而言，必須先將陣列訊號類似 ESPRIT 的做法，分成幾群，

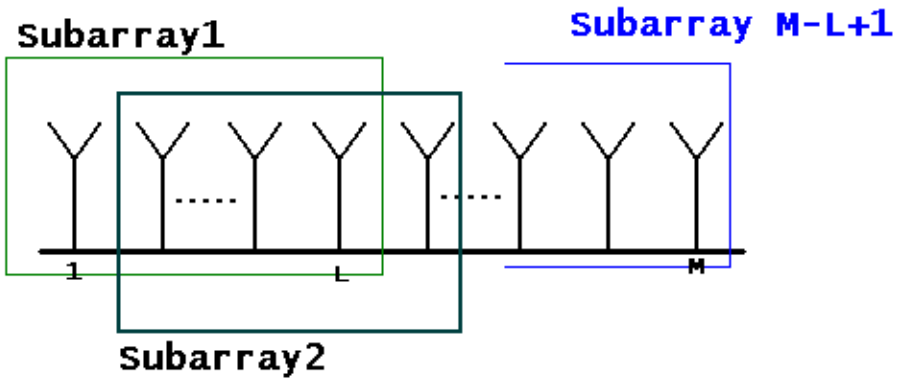


圖 9 前後空間平滑子矩陣分群法則示意圖

則每一個子陣列〈Subarray〉的訊號以及相關矩陣可表示為

$$X_i = A\phi^{i-1}S + N_i$$

$$R_{xx}^{(i)} = E\left\{X_i, X_i^H\right\}$$

$$= A\phi^{i-1}R_{ss}\left(\phi^{i-1}\right)^H A^H + \sigma_n^2 I$$

其中延遲矩陣的定義

$$\phi = \begin{bmatrix} e^{-jw\left(\frac{d\sin\theta_1}{v_{s1}}\right)} & 0 & \cdots & 0 \\ 0 & e^{-jw\left(\frac{d\sin\theta_2}{v_{s2}}\right)} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & e^{-jw\left(\frac{2d\sin\theta_L}{v_{sL}}\right)} \end{bmatrix}$$

經過前後空間平滑修正後的相關矩陣，定義如下：

$$R_{ave} \equiv \frac{1}{2}\left(R_F + R_B\right)$$

$$= \frac{1}{2}\left(R + \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \overline{R} \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \cdots & 0 & 0 \end{bmatrix}\right)$$

$$R \equiv \frac{1}{M - L + 1} \sum_{i=1}^{M-L+1} R_{xx}^{(i)}$$

透過以上的修正，得到的 R_{ave} 就是滿秩的相關矩陣，就可以拿來進行訊號源入射角度估計了。 [2][3][4]

2.5 實際訊號分析

接下來是利用實際量測到的訊號，來驗證演算法的效果。這裡所使用的訊號，是由 Channel Sounder 實際量測所獲得的空間訊號。

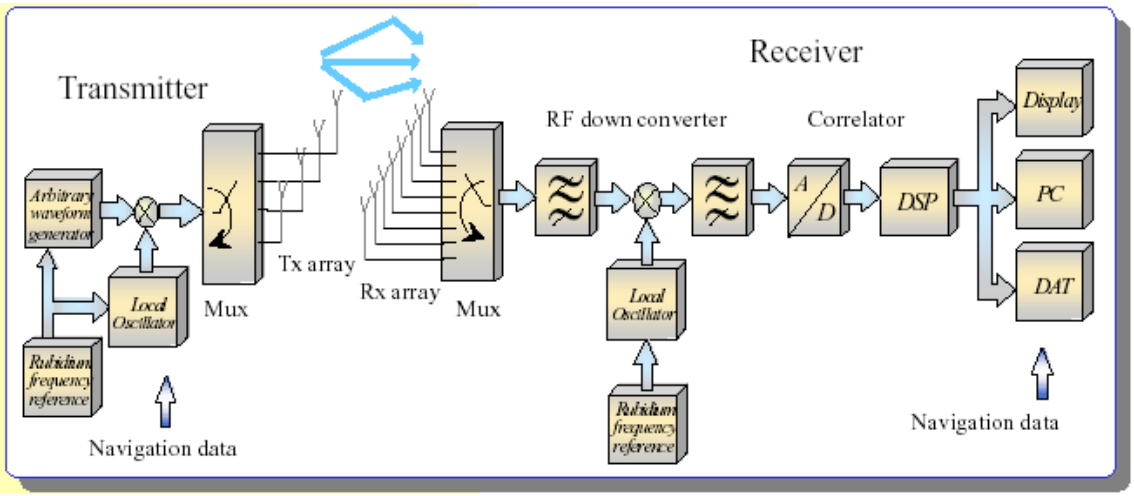


圖 10 Channel Sounder 的內部方塊圖

2.5.1 Channel Sounder 簡介

Channel Sounder 的接收端有十根天線，兩根是輔助天線，其餘八根是接收訊號用的，傳送端的射頻可到達 5.2G，頻寬是 120MHz 也就是說解析度可到 8.3ns，脈衝響應的長度有 0.8、1.6、3.2、6.4、12.8、25.6 μs，Channel Sounder 的前端是採用類似多工器的架構，為了節省硬體的費用，利用一個高速的切換器，一次只接收一個天線的訊號，掃完八根天線為一個週期，接著將收到的訊號降頻濾波到中頻 80MHz，利用 320MHz、8bits 的類比數位轉換器將類比的中頻訊號轉換成數位訊號，然後以每秒 320MBytes 的資料量即時的送進 DSP(TMS320C40)再作一次降頻轉換到基頻，同時把資料利用快速傅立葉轉換從時域轉換到頻域，並計算資料的相關矩陣後，才把資料秀在螢幕上或是儲存在硬碟裡。

以下是舉例說明當有四個天線個數時，前端高速的切換器的運作情形，x 軸是時間、y 軸是天線數，其中 t_p 定義為切換器在某個天線收資料的時間，當切換器要切換到下一根天線的時候，會等 t_p 時間之後，所以假設有 M 根天線，切換器完整的掃完 M 根天線所需要花的時間定義為 $t_s = (2M-1)t_p$，雖然資料不是同時八根天線一起接收，但由於傳送端會一直週期性的發射同一個 pattern，而且天線全部掃完一次頻率可到達 20kHz 以上，所以可看成好像是同時到達，t_r 定義為接收第一根天線到下次接收同一根訊號所間距的時間。

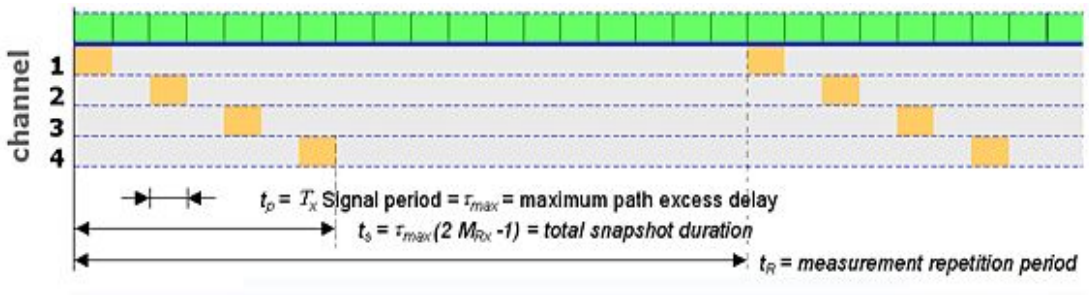


圖 11 四根天線的切換情形

2.5.2 實際量測資料

這邊採用的實際訊號是在交通大學的校園內量測到的，基地台設定在工程四館的頂樓，發射端則在中正堂附近，Channel Sounder 的參數設定射頻頻率是 2.4GHz，訊號頻寬是 120MHz，t_p 取 6.4μs，兩個量測週期時間差 t_r 為 1ms，採用兩個實際情況的資料作分析，一個是發射端可以直接看到基地台的情況，可作視線範圍(Line of Sight)的測試；另一個較複雜的情況是發射端看不到基地台，基地台收到的訊號是由發射訊號與建築物反射，訊號會在不同時間到達接收端，強度也會不同，較接近實際應用情形。

圖 12 交通大學量測分佈圖

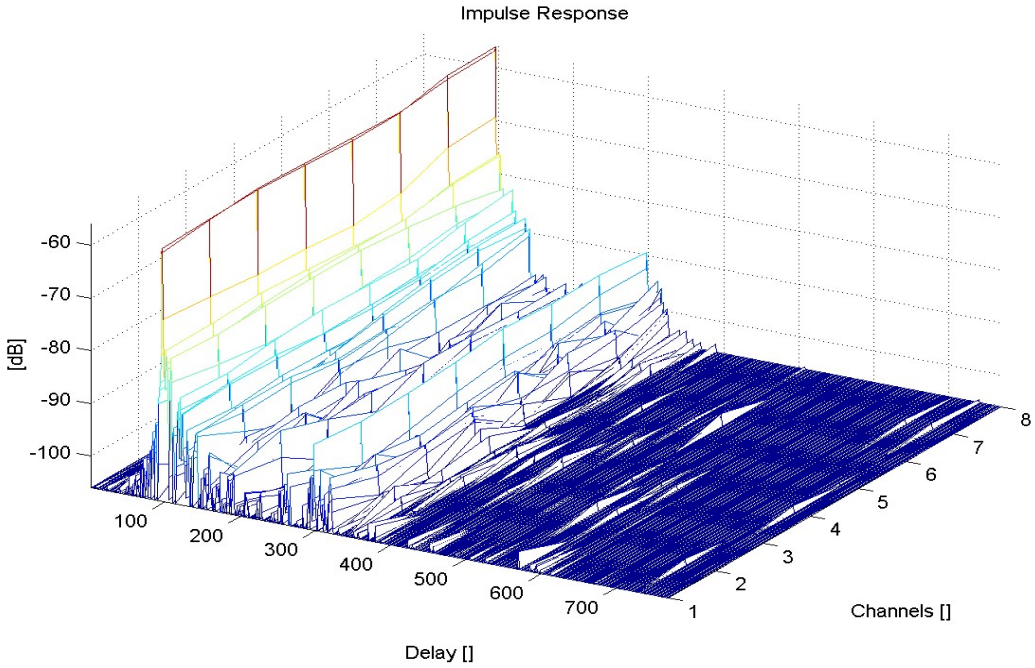


圖 13 視線範圍訊號的空間通道響應

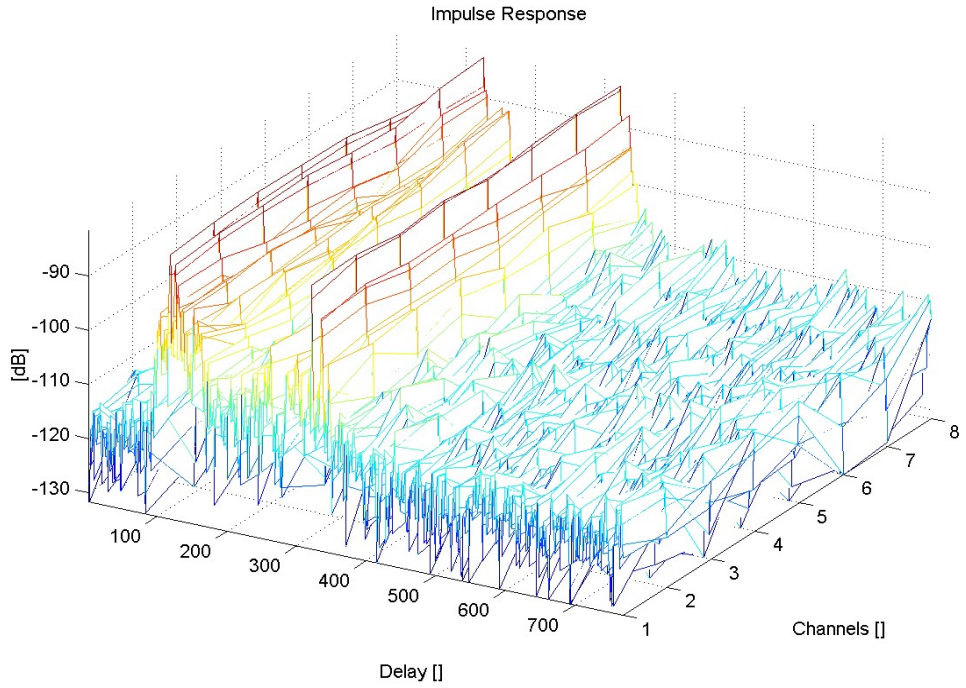


圖 14 多重路徑訊號的空間通道響應

視線範圍訊號通道響應圖會以三維表示，分別以 delay、channel、dB 表示，dB 表示訊號的大小，channel 表示是第幾根天線的訊號，delay 是用來表示訊號經過不同路徑到達的時間，delay 這裡取 6.4μs，是表示訊號即使經過最遠的路徑在 6.4μs 內一定會到達接收端，且因為頻寬是 120MHz，因此 delay 是由 768 個點表示。圖 13 是將發射端設置在中正堂與行政大樓中間以步行的速度移動發射端，位置如圖 12 LOS 所示。若直接由圖 12 的位置分佈圖來看，從中正堂往工程四館的方向看去，會被計網中心與工程三館所給擋住，實際上在交大校園內工程四館興建的遠比計網中心與工程三館高，所以發射端放在圖 12 內的位置是可以直接看到工程四館接收端的基地站台，可以進行視線範圍訊號的採樣。

圖 14 是將發射端靜止放置在中正堂背後接近校門口的地方，位置如圖 12Multipath 所示，是沒有辦法直接看到工程四館頂樓的接收端，這樣設定方式接收端所接收到的訊號都是發射端經由建築物反射獲得的，在圖 14 中可以看出接收端接收到有兩條較強的訊號經由空間中不同路經抵達。

Channel Sounder 內附有的 MATSYS 軟體是使用 MATLAB 語言寫成的分析軟體，是套功能很強大的軟體，可將 Channel Sounder 採樣的的訊號轉換到各種不同的表示域，藉以作空間通道資料分析，在 MATSYS 內支援 ESPRIT 的演算法作到達角度的分析，圖 15 與圖 16 是分別對圖 13 與圖 14 的資料作完角度分析的結果。圖 15 是視線範圍資料的角度估測，所以可以很準確的測出發射端發送的訊號到達接收端的入射角度，圖 16 是分析多重路徑入射的訊號，在圖上可以找出兩條路徑擁有較強入射訊號的角度。

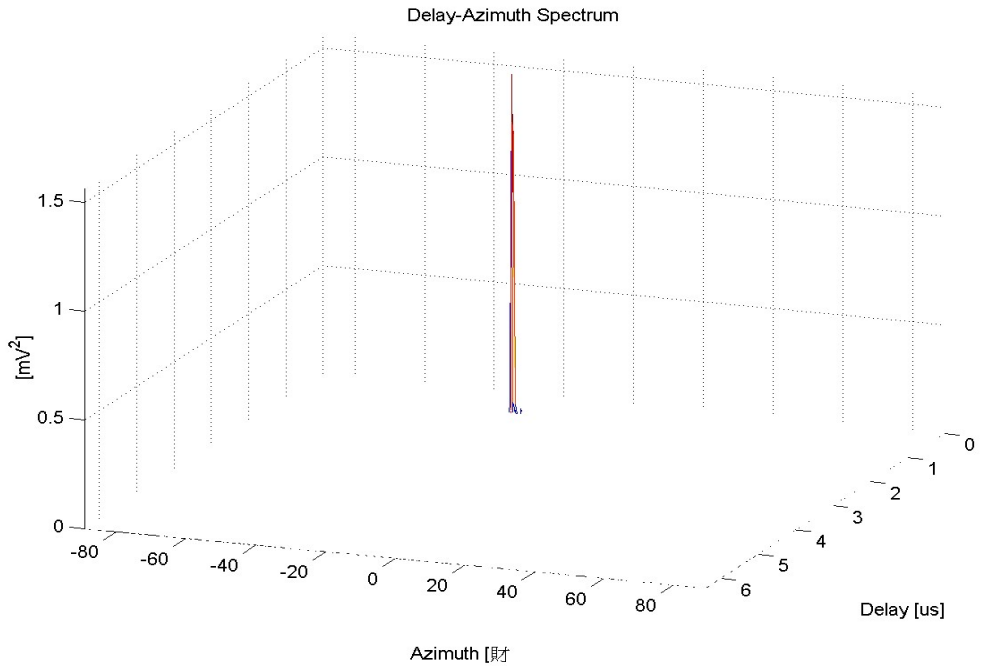


圖 15 視線範圍訊號的到達角度估測

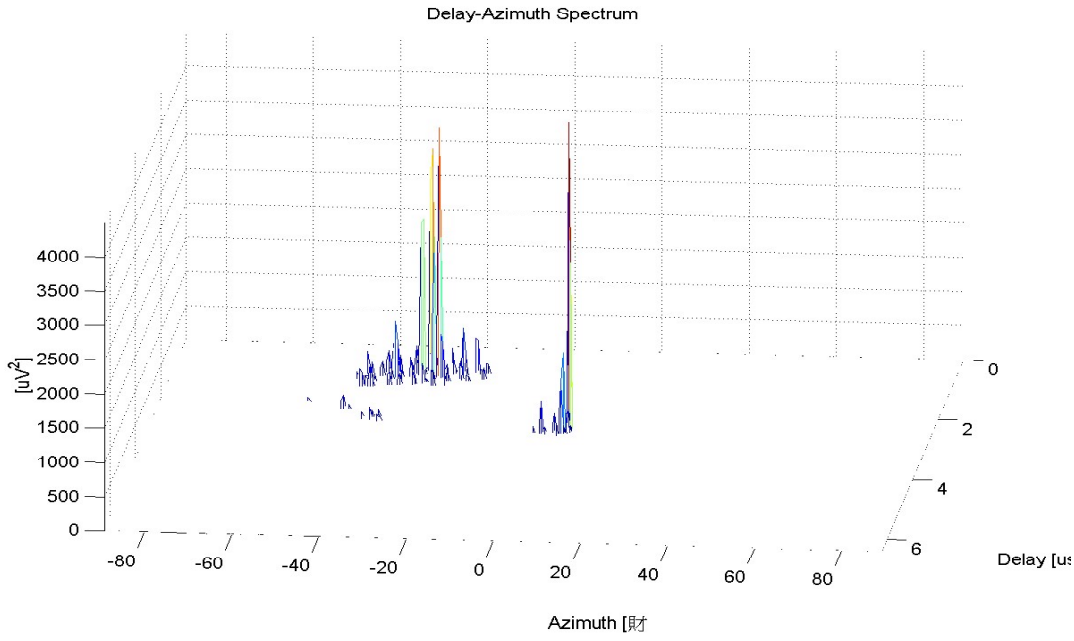


圖 16 多重路徑訊號的到達角度估測

2.5.3 訊號分析

將實際接收到的訊號，送入 DSP 計算訊號入射角度，以下是執行 MUSIC 演算法的結果。

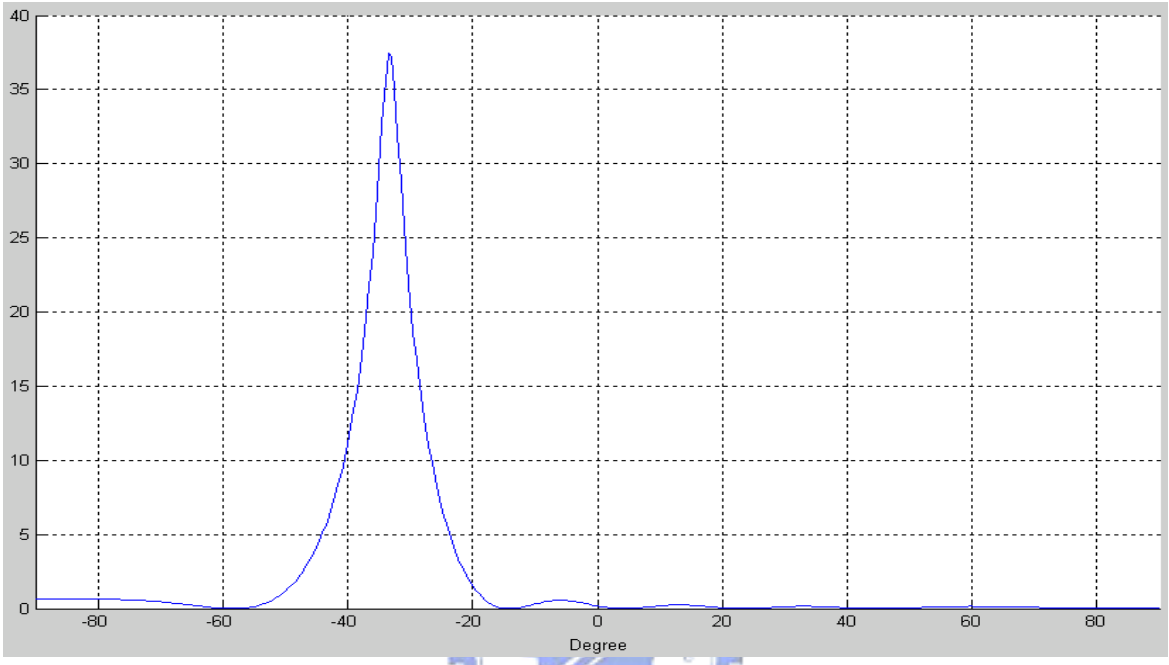


圖 17 視線訊號範圍的角度估測

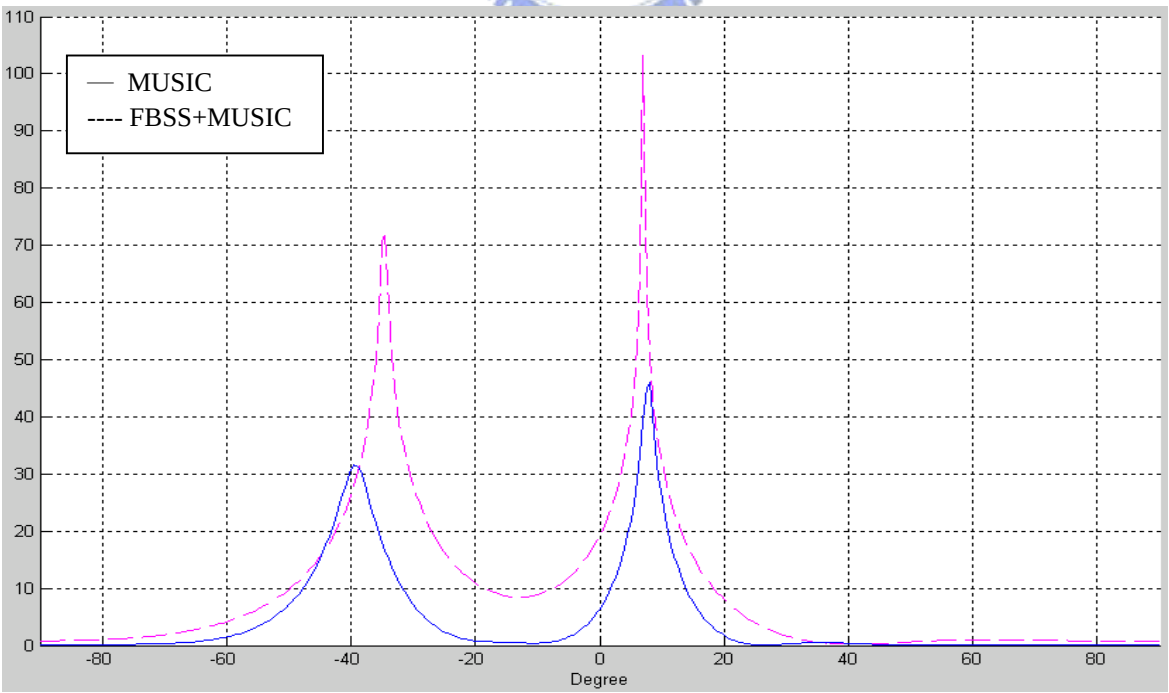


圖 18 多重路徑訊號的角度估測

由以上的分析中可以看出，在視線範圍入射的訊號，由實驗平台解出的訊號到達角度為 33 度，與 Channel Sounder 所量測出來的相符合；而在多重路徑的訊號，實驗平台解出來的結果，大致上也與 Channel Sounder 分析的結果相同。其中，我們可以注意到，MUSIC 演算法在多重路徑的環境下，訊號雜訊比有下降的情況；在加入前後空間平滑的前處理之後，這種情況有明顯的改善，並且解出的角度：7 度、34 度，也較傳統的 MUSIC 演算法所解出的角度，更接近 Channel Sounder 所分析出來的訊號入射角度。

第三章　實驗平台架構

實驗平台的架構，我們將從硬體架構與軟體架構兩方面來說明。

3.1 硬體簡介

實驗平台是由兩塊實驗版搭配而成，皆是由英國 AG Electronics 公司製造。

3.1.1 DM11

- 一顆德州儀器公司 C6701 系列的數位訊號處理器
- 32M Bytes SDRAM
- 1M Bytes SBSRAM
- 512K Bytes flash EPROM
- 一顆 Xilinx Virtex XVC100 FPGA
- 一顆 PCI controller

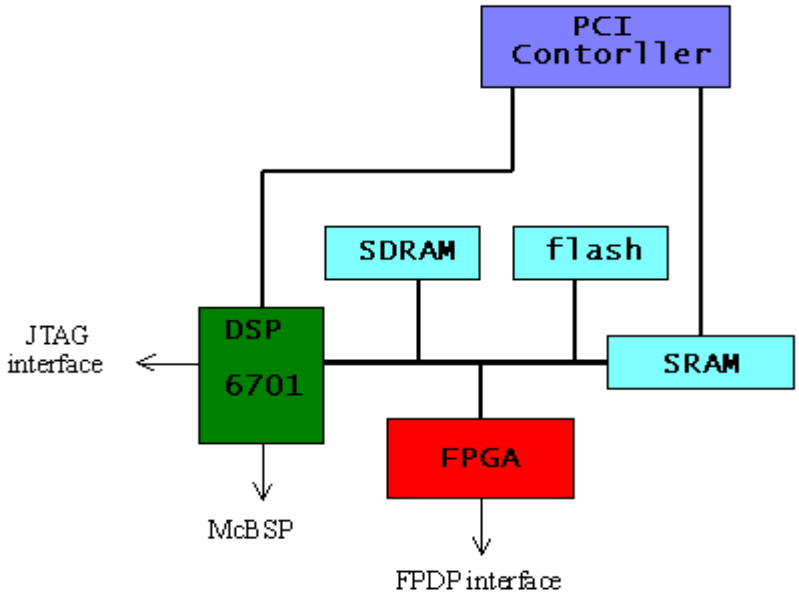


圖 19 DM11 內部方塊圖

這塊版子的特色在於擁有一塊用來連接 FPDP〈Front Panel Data Port〉高速輸入埠。此傳輸規格是以 32bits 同步傳輸，與許資料傳輸速度最快可達每秒鐘 160Mbytes。與 DP12 實驗板的溝通，可透過 6701 DSP 內建的全雙工序列傳輸埠 McBSP，直接與另外兩顆處理器做溝通；或是透過 PCI controller 的協助，以 local PCI bus 資料途徑，與其他顆 DSP node 配備的 PCI controller 交換資料，進而完成訊息傳遞的目的。至於 JTAG 低速傳輸的介面，主要是當作 debug 之用，並可與 Matlab 整合，實現即時資料傳輸〈Real-time Data Exchange〉。

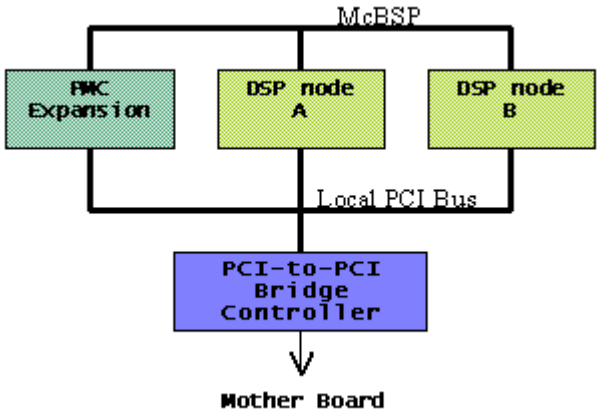


圖 20 DP12 內部方塊圖

3.1.2 DP12

- 64M Bytes SDRAM
- 2M Bytes SBSRAM
- 兩顆 PCI controller
- 1024K Bytes flash EPROM
- 兩顆德州儀器公司 C6701 系列的數位訊號處理器
- 一顆 PLX 9054 PCI-to-PCI Bridge controller

主要的架構如上圖。PMC Expansion 插槽，是用來與 DM11 相連接的擴充埠，可經由 McBSP 或 local PCI bus 的介面與 DP12 上的兩個 DSP node 做連結。DSP node A 與 DSP node B 是與 DM11 完全相同的硬體架構，但是只有 DM11 有搭配 FPGA，擁有前端資料處理的通道。三顆數位訊號處理器，可以透過 PCI Bridge Controller 的協助，將資料導入主機板上的 PCI 插槽，與個人電腦溝通。以下是整個架構詳細的方塊圖：

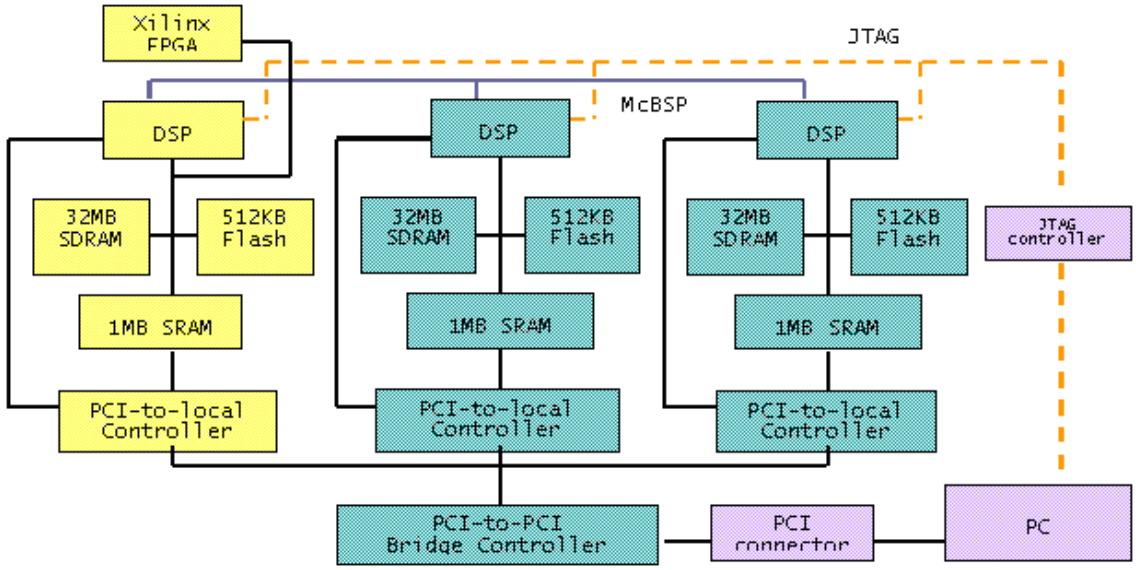


圖 21 實驗板整體內部方塊圖

3.1.3 ES10

ES10 這塊版子擁有 JTAG 仿真器〈JTAG emulator〉，可以與德州儀器公司 6000 系列的數位訊號處理器完全相容。功能是輔助軟體開發整合介面—Code Composer Studio〈德州儀器公司開發之專屬數位訊號處理器軟體開發平台〉，實現即時監控 DSP 狀態的功能。

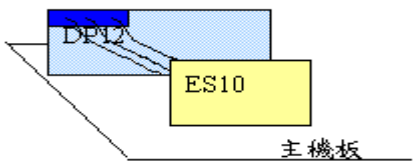


圖 22 ES10 與 DP12 連接示意圖

3.1.4 6701 DSP

TMS320C6701 是德州儀器公司所開發的數位訊號處理器，為 32 位元浮點運算的處理器，具有特殊的硬體架構與指令集，使它比一般的 CPU 更適合數位訊號處理的工作。以下是幾個重要的特性：

- CPU 架構採用先進的超長指令集〈Very Long Instruction Word，VLIW〉設計，並且具有六個算數邏輯運算單元〈ALU〉以及兩個浮點數乘法器，可同時運算，實現平行化處理。
- 支援管線化〈pipeline〉架構，指令執行階段最深可達 10 個時相〈phase〉。
- 支援 8、16、32 位元資料存取格式。
- 內部時脈為 166MHz，運算量可達 1G FLOPS。
- 支援 IEEE single/double precision 的浮點數運算。
- 內建 64Kbytes 的程式記憶體與 64Kbytes 的資料記憶體。
- 擁有 4 個 DMA 控制器，兩個 McBSP 序列傳輸通道控制器以及兩個計時器。

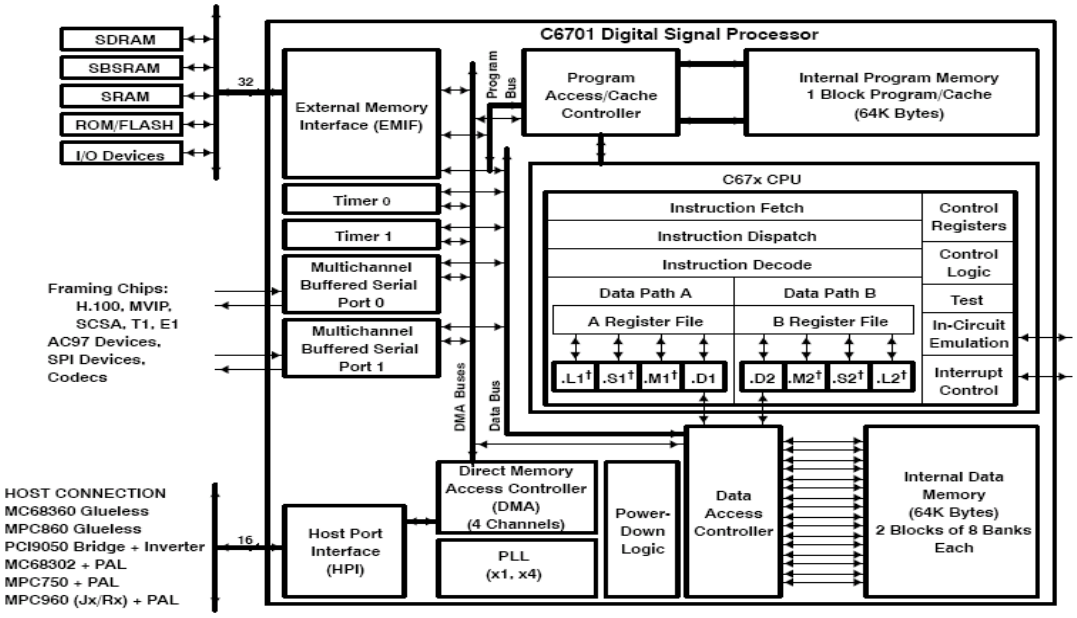


圖 23 TMS320C6701 方塊圖[10]

3.1.5 PCI Bridge

為了溝通 DSP 與主機板，AG 公司使用 PLX Technology 的 PCI 9054 橋接晶片，來橋接主機板的 PCI bus 與實驗板上的 local PCI bus。PCI 9054 提供了 DSP 以 Direct Master〈PCI Initiator〉模式來存取來自 PC 端的資料，以及存取其他顆 DSP。以下是它內部的方塊圖：

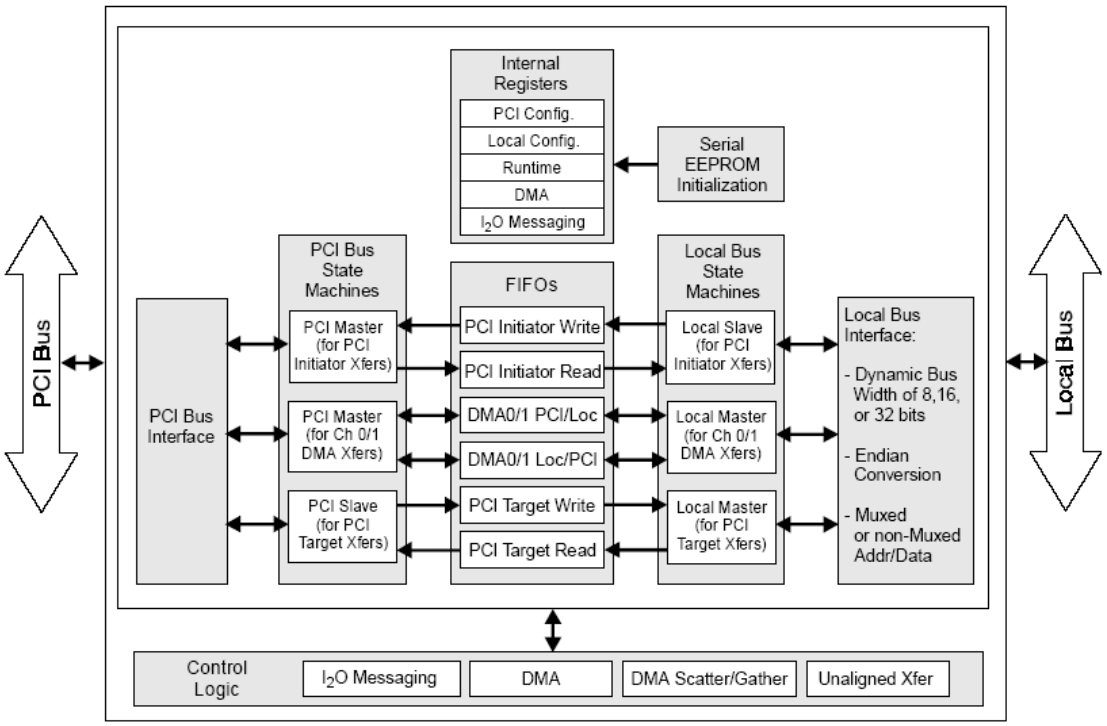


圖 24 PLX 9054 方塊圖[9]

3.1.6 基本架構

據畢業學長加名提出的架構，再搭配青衛建立的 McBSP 溝通管道，平台的架構已達完備，以下便是平台的架構。

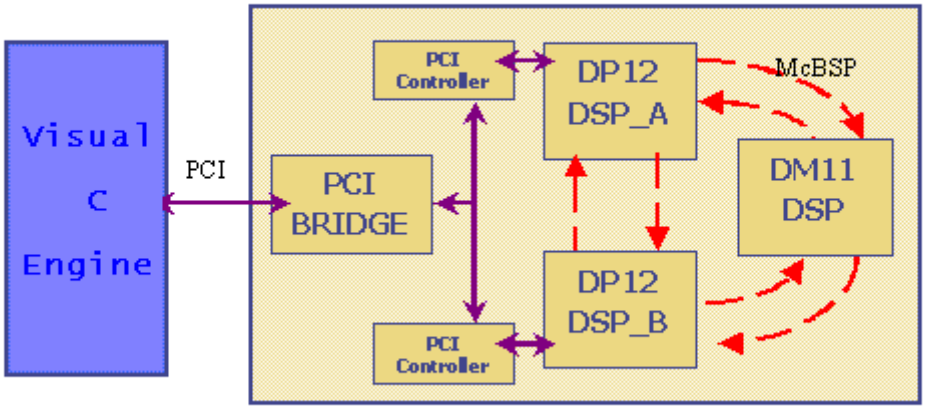


圖 25 平台硬體架構圖

3.2 軟體架構

本平台的軟體介面，主要可分成兩個部分：一是 HOST 端與 DSP 溝通的介面，二是 DSP 之間的 McBSP 溝通介面。

3.2.1 Host <-> DSP

此平台在 HOST 端的程式，是以 Visual C 來完成，藉由 PCI 9054 橋接器的資料交換，達到溝通 DSP 的目的。在此過程中，HOST 端是處於被動的角色，只有 DSP 可以主動存取 Visual C 執行程式內的資料。

在 DSP 端傳送的指令格式，我們設定為以下形式：

```
Hostlink_Control Structure
{
    unsigned   command;
    unsigned   args[6];
    volatile   unsigned   done;
}
```

command 部分是由使用者自行設定需要的指令。Args[6]的部分，表示從 DSP 傳到 HOST 端，或是由 HOST 端回傳給 DSP 的參數數目，最多可以達到六個。而 done 的設計，是為了讓 DSP 知道 HOST 端是否正確的接收到命令。因為這個指令傳送的過程，是利用橋接器裡面 I2O 的通道來傳遞，而 I2O 通道是一個 FIFO 通道，在前面的資料會因為前一次的通訊尚未完成而未取走，造成通道阻塞的狀況，所以必須要有確認的機制，才能確定傳送與接收的資料是正確的。

在 VisualC 內，也必須先行定義可以被 DSP 存取的函數〈可視為 DSP 與 HOST 溝通窗口〉，並透過 GPCI_command_descriptor 資料結構來記錄每一個函數的位址〈可視為一個函數位址對應表〉；在 DSP 端則利用內部記憶體前端預留的 16 個 WORD 空間，來存放對應於 Visual C 內 GPCI_command_descriptor 的順序〈0X80000000～0x80000010〉，經由 PCI 9054 傳遞轉換資料，則 PC 端即可知道被呼叫的函數以及傳進來的參數。

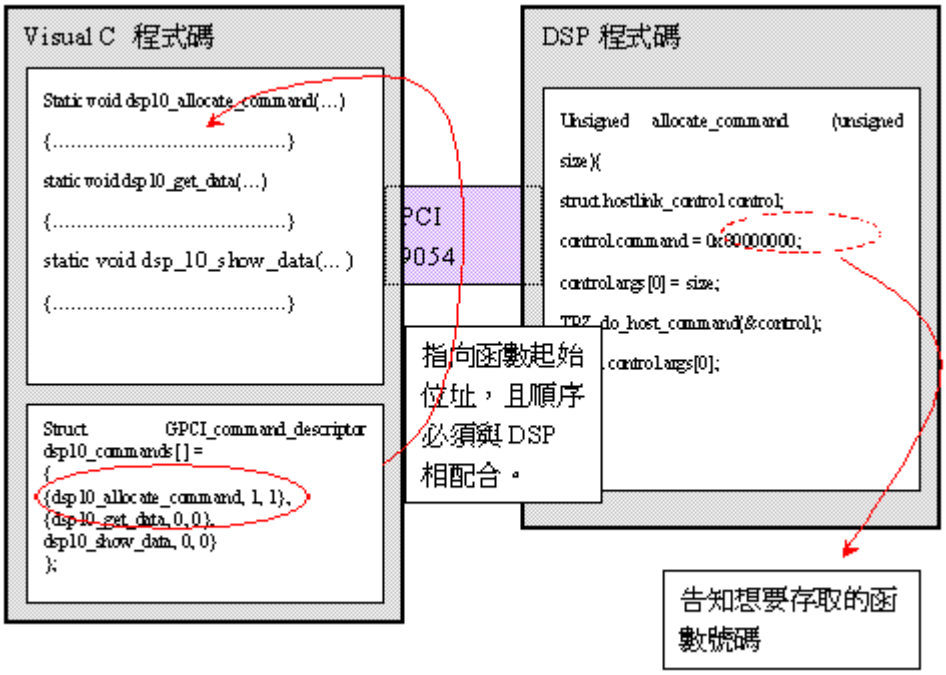


圖 26 PC 端與 DSP 端溝通架構

此傳送過程中，DSP 必須先向 HOST 端要求一塊 PCI buffer，作為資料在 PCI bus 上傳輸的空間，而這塊空間，Visual C 與 DSP 都必須互相知道，才能夠順利的完成資料的傳輸。

3.2.2 DSP <-> DSP

DSP 之間的溝通，是利用內建的 McBSP〈Multichannel Buffered SerialPorts〉介面來溝通。這種介面的特色，是以全雙工〈FullDuplex〉的模式，來與其他 DSP 單元交換資料，速度為 3.125Mbytes/sec，並且最多可以擴充至傳收 128 個不同的 channel 資料。傳輸訊號的時脈，可以由外部輸入或是 DSP 內部產生，增加了此介面的彈性。[10]

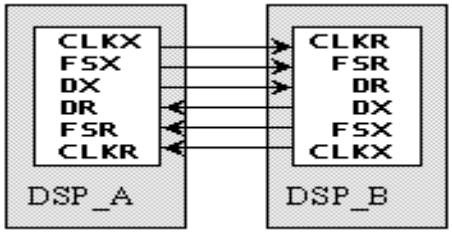


圖 27 McBSP 溝通接腳圖

因為是高速串列傳輸，為了確保資料的正確性，除了兩條資料傳輸線之外，還有使用了四根的同步訊號〈兩條資料線、兩條 clock 線，以及兩條同步訊號線〉，來達到全雙工的目的。在使用上，可以搭配 DMA 使用，直接將接收到的資料，導入記憶體中，而不必透過 DSP 來搬運，可以減輕 DSP 的負擔。

在撰寫程式的過程中，需要特別注意的是，當資料傳送完畢時，傳送端不能馬上關閉通道〈channel〉，因為此時接收端可能尚未接收完資料，如果太快關閉的話，會導致接收端的資料不穩定，造成最後幾筆接收到的資料發生錯誤。

3.2.3 程式開發流程

陣列訊號的演算法，是以矩陣運算為基礎。MATLAB 是針對處理矩陣運算而設計的程式開發軟體，提供了大量複雜的數學函數運算功能，例如『解矩陣特徵向量』、『解高階方程式』等等，並擁有強大且易於使用的繪圖功能，所以在選擇了演算法之後，我們選擇在 MATLAB 先做驗證，以縮短驗證的時程。

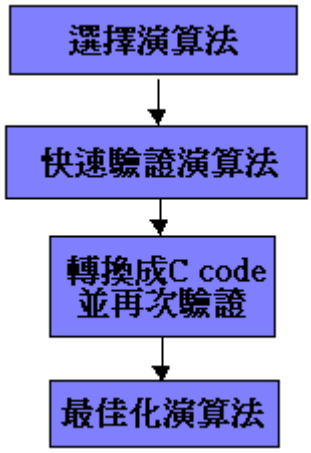


圖 28 演算法開發流程圖

在完成演算法的驗證之後，我們選擇先在 Visual C 再次驗證演算法，而非直接開始寫入 DSP，主要有兩個原因：第一是原因是雖然 Visual C 與 DSP 都是寫 C code，但是 Visual C 的 debug 介面功能較為強大，在驗證完畢之後，可以直接移植到 DSP，因此做此選則。第二是 Visual C 可以透過內建的函式庫，呼叫 MATLAB engine，直皆存取 MATLAB，可以避免掉在 C 裡面撰寫複雜的繪圖程式，浪費時間。

在完成 Visual C 的驗證之後，我們便可以著手開始撰寫 DSP 程式。德州儀器公司針對 6000 系列的 DSP，提供了軟體開發工具—CCS〈Code Composer Studio〉，擁有 C compiler，可以讓使用者直階以高階的 C 語言來開發程式，這跟傳統的組合語言撰寫相比較，程式設計者能夠以更快的速度來撰寫出複雜的程式碼，且在 debug 程式的時候，也會更加的容易。由於 DSP 是一個嵌入式系統，在硬體資源上，無法像在個人電腦上獲得龐大的支援，最明顯的就是 DSP 的資料記憶體只有 16K Bytes，完全無法與個人電腦動輒 256K 以上的記憶體相提並論，這個限制，直接的衝擊了 DSP 程式執行的效能，也考驗了程式設計師程式撰寫的功力。DSP 程式碼的最佳化，將在下一個章節有詳細的討論。

第四章　運算效能分析

運算平台	Intel Pentium 800MHz		TI 6701 166MHz	
	L1 64K	L2 256K	L1 16K	L2 32K
程式執行時	7ms		112ms	

圖 29 效能分析

以上的數據，是針對 8 根天線，128 個 snapshot 的資料量，來進行 MUSIC 演算法處理。DSP 部分的程式碼，是由實驗室畢業學長青衛開發的程式碼測得的數據[18]；PC 部分是我在 Visual C 量測的結果。

大部分的人都認為，處理器處理的速度，記憶體的大小，就直接決定了工作平台執行的效能，但其實，這兩者之間，還隱藏著軟體設計的技巧。從接下來的分析當中，我們將慢慢看出一些端倪。

4.1 6701 DSP 運算原理

在編譯器〈compiler〉效能日益精進的年代，大部分著手於高階語言程式設計的人，都將程式碼的最佳化，交給 compiler 來處理，而逐漸淡忘了軟硬體相互配合的重要性。以下我們從硬體的分析開始。

4.1.1 平行處理的運算單元

6701 DSP 內部擁有 8 個處理單元〈functional unit〉，依照功能來做區分，可分為.D、.S、.L、.M 四種，可用來處理 6701 所有形式的指令。而這八個處理單元，如果在指令

```
   [A2] SUB     .S1     A2,1,A2
||      SUBSP   .L1     A0,A7,A0
||      ADDSP   .L2     B6,B8,B9
||      MPYSP   .M1     A5,A3,A7
||      MPYSP   .M2X    A10,B7,B8
||      LDW     .D1T1   *+A8(4),A3
||      LDW     .D2T2   *+B4(4),B7
```

圖 30 八個運算單元平行處理組語表示

的編排上有規劃好，他們是可以在同一個 clockcycle 裡面一起執行指令。所以，如果我們可以將指令平均的運用，分散在這八個處理單元，則我們可以提昇程式處理的速度到達 8 倍。上圖就是八個指令平行運算，可以在同一個 clock 執行。

.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits **Byte shifts** **Data packing/unpacking** **5-bit constant generation** **Dual 16-bit arithmetic operations** **Quad 8-bit arithmetic operations** **Dual 16-bit min/max operations** **Quad 8-bit min/max operations**	Arithmetic operations DP → SP, INT → DP, INT → SP conversion operations	.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) **Byte shifts** **Data packing/unpacking** **Dual 16-bit compare operations** **Quad 8-bit compare operations** **Dual 16-bit shift operations** **Dual 16-bit saturated arithmetic operations** **Quad 8-bit saturated arithmetic operations**	Compare Reciprocal and reciprocal square–root operations Absolute value operations SP → DP conversion operations
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit constant offset Loads and stores with 15-bit constant offset (.D2 only) **Dual 16–bit arithmetic operations** **Load and store double words with 5-bit constant** **Load and store non-aligned words and double words** **5-bit constant generation** **32-bit logical operations**	Load doubleword with 5–bit constant offset	.M unit (.M1, .M2)	16 x 16 multiply operations **16 x 32 multiply operations** **Quad 8 x 8 multiply operations** **Dual 16 x 16 multiply operations** **Dual 16 x 16 multiply with add/subtract operations** **Quad 8 x 8 multiply with add operation** **Bit expansion** **Bit interleaving/de-interleaving** **Variable shift operations** **Rotation** **Galois Field Multiply**	32 X 32–bit fixed–point multiply operations Floating–point multiply operations

圖 31 運算單元指令集分類[11]

除了這八個處理單元平行處理的特色之外，6701 DSP 另一個特色就是他支援管線化〈Pipelining〉的硬體。

4.1.2 管線化處理的硬體架構

我們都知道，邏輯運算單元是由順序性的邏輯電路〈sequential logic〉所組成，對於傳統的處理器而言，程式碼執行的方法是一行指令送進去邏輯運算單元，等到結果出來以後，再送下一行指令進去。這樣子的方法，對於設計複雜、功能強大的邏輯運算單元來說，就並不是那麼有效率。

我們用一個簡單的例子來說明。下圖，是我們洗衣服的流程：1.洗衣服 2.烘乾衣服 3.將衣服摺疊整齊 4.收進衣櫃。一顆功能強大的處理器，可視為是一次可以做完這四樣東西的機器，但是做一次需要兩個小時的時間，然後才能進行下一次工作。但是，我們都知道，洗衣機洗完衣服之後，就空出來了，我們沒有必要等到衣服收進櫃子裡才開始洗下一次的衣服。

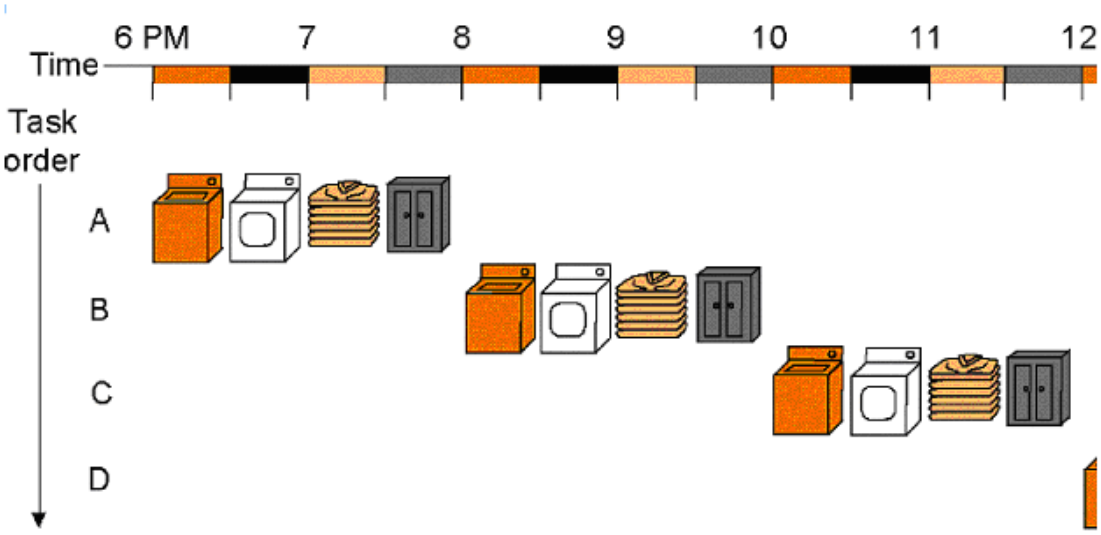


圖 32 為最佳化的洗衣流程 [9]

下圖就是改進後的洗衣流程，原本要兩個小時時間才能完成一次洗衣的流程，經過把工作步驟分割的更細，現在只要半個小時就可以完成一次洗衣流程。Pipelining 的概念就是如此，我們將一個指令分成好幾個部分來執行，則硬體資源就可以被充分的利用，進而提昇程式的效能。

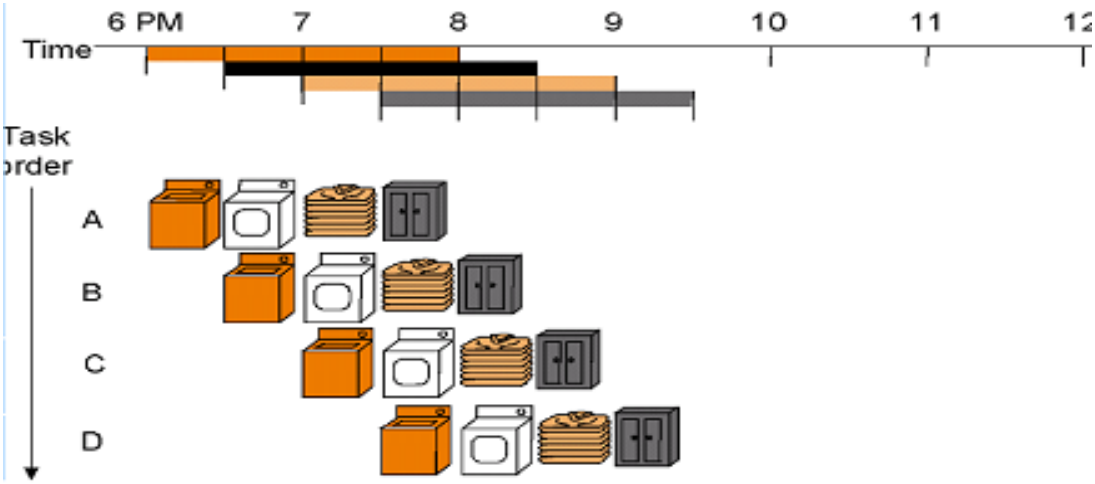


圖 33 管線化處理後的洗衣流程 [9]

6701 DSP 不只針對指令的執行應用了 pipelining 的技術，它在指令擷取〈instruction fetch〉，指令解碼〈instruction decode〉的階段也都應用了這種技術，這也是這一顆 DSP 最核心的技術。下圖即為 6701 DSP 執行指令的流程，括弧內的數字，代表了 pipeline 劃分的層級數，其中，執行階段的層級數，視指令的不同而有所變化。[11][12]

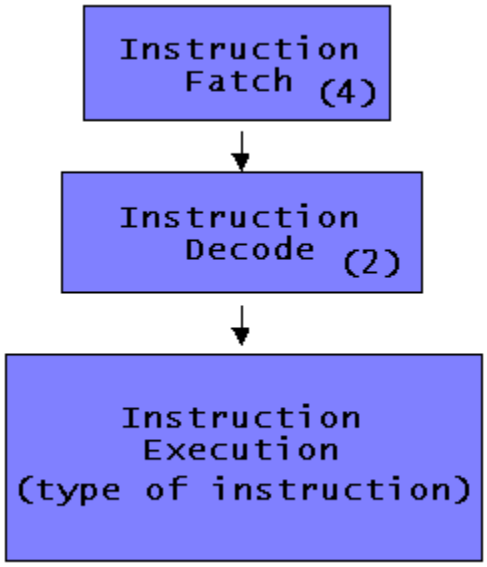


圖 34 6701 三階段管線化示意圖

4.1.3 執行指令階段

接下來，我們更進一步的深入執行階層，來探討資料的執行流程。下圖是八個運算單元與兩組暫存器、內部記憶體之間的關係圖，也隱含了組合語

言運算的過程。對於任何形式的處理器來說，資料來源一定是先搬進暫存器，然後才做運算，這是因為存取暫存器的速度遠比存取內部記憶體來的快。6701 內部，提供了兩組資料途徑，用來存取內部記憶體，所以在同一個時間週期內〈clockcycle〉，可以同時存取兩筆記憶體資料；32 個 32 位元的暫存器，分別連接在左右兩組運算單元，並且各自提供了一條讓另一側運算單元存取的資料途徑〈crosspath〉，但是在同一個時間週期之內，這兩個資料途徑只能各自讓一個運算單元做存取。內部記憶體，則是劃分成兩塊，每一塊又細分為八個區塊〈bank〉。每個區塊與內部記憶體控制器〈internal memory controller〉只有一條資料路徑相連接，所以同一個時間週期之內，一個內部記憶體區塊內的資料，只能被一個運算單元存取。

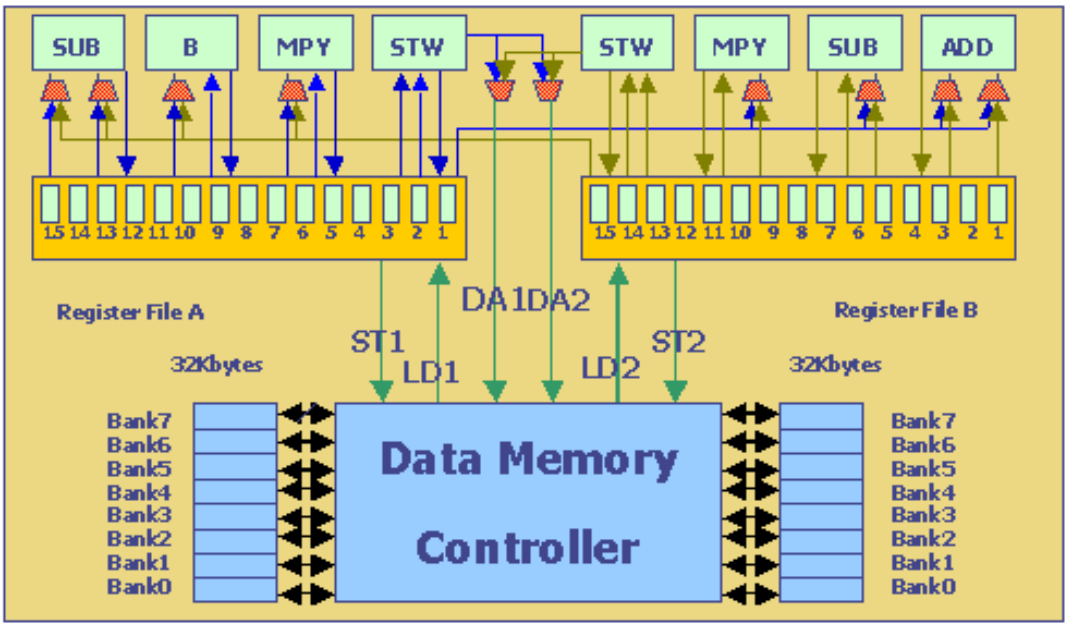


圖 35 6701 管線化執行階段硬體架構圖 [11]

4.2 軟體效能的提昇

在了解了主要的核心技術之後，我們便可以開始著手於程式碼的分析了。

4.2.1 記憶體指標的影響

處理大量的矩陣運算，為了提昇效能，我們通常以指標傳遞，來降低函數呼叫時所需要額外的運算量〈overhead〉。以下是一個 8x8 矩陣相加的例子：

```
void matrix_mov ( float * dst, float * src)
{   int i;
    for( i=0;i<64;i++)
        dst[i]= src [i];
}
```

很普通的一個程式，只有幾行指令，乍看之下，沒什麼問題，一個 load，一個 store，一個紀錄迴圈次數的 add，一個 branch，再簡單不過了，但是對於 compiler 而言，卻非如此。指標傳進來的位址，通常是 base address，可以透過加上 offset 指到任意的位址。對於 compiler 而言，它無法判斷以 dst、src 為 base 的指標，是否會發生同時存取到同一個位址的情況，一但發生，就會造成運算上的錯誤，所以，compiler 為了安全起見，任何以這兩個指標存取的動作，都會等到 execution stage 結束之後，才進行下一個指令的執行，無法運用 pipelining 的方法來運算，造成效能嚴重的降低。浮點數的 load 需要 5 個時間週期才能完成，store 需要一個週期，如果有 pipelining，需要 64+5 個時間週期來執行，但是如果沒有 pipelining，就需要(1+5)x64 個時間週期，相差了五倍的時間。

4.2.2 平均的使用運算單元

前面有提到，每一個指令都對應到適合的運算單元，但是一個時間週期〈clock cycle〉之內，只有八個運算單元可以使用。接下來，我要探討的是如何在有限的運算單元中，發揮最大的效率。在解特徵值的時候，矩陣需要大量的列運算〈row operation〉，下面是一個典型的例子。

```
void row_op (float *restrict dst, float *restrict src1, float *restrict src2, float scalar)
{    int i;
     for( i=0; i<128; i++ )
          dst[i] = src1[i] + src2[i]*scalar;
}
```

程式中『restrict』關鍵字，是用來告知編譯器這兩個指標在接下來的運算當中，並不會發生記憶體存取的衝突，也就是避免了前一個例子問題的發生。這個程式執行的是兩個 load，一個 mpy，兩個 add，一個 store，一個 branch。其中，乘法運算是對應到『.M』運算單元，浮點數加法對應到『.L』，定點數加法以及跳躍指令對應到『.S』，載入與儲存都對應到『.D』，所以說，在迴圈當中，每次需要一個.M、兩個.S、一個.L 以及三個.D。但是在同一個時間週期內，只有兩個.D 運算單元，所以，表示了這個迴圈必須要花兩個時間週期才能完成一筆資料運算，所以在執行 128 次迴圈之後可以完成所有的運算。如果我們不做任何改變，則這個迴圈需要 2x128+(5+4+4)個時間週期。但是如果我們將迴圈改寫成為下面的形式，那麼迴圈內的運算單元需要量，就改變為兩個.M、兩個.L、兩個.S 與六個.D，平均分配在八個運算單元的話，就是一次迴圈需要執行三個時間週期，完成兩筆資料的運算，在 64 個迴圈內執

```
for( i=0; i<64; i += 2 )
{    dst[i] = src1[i] + src2[i]*scalar;
     dst[i+1] = src1[i+1] + src2[i+1]*scalar;
}
```

行完所有的運算。所以總共花費的時間是 3x64+(5+4+4)，執行效能提昇了 50%。下圖描述了一個迴圈執行，所需用到的運算單元個數，其中.X 代表了 cross path 使用的個數，II 表示一個迴圈執行所需要的時間週期〈clock cycle〉，而『*』標示出限制時間執行週期的運算單元。

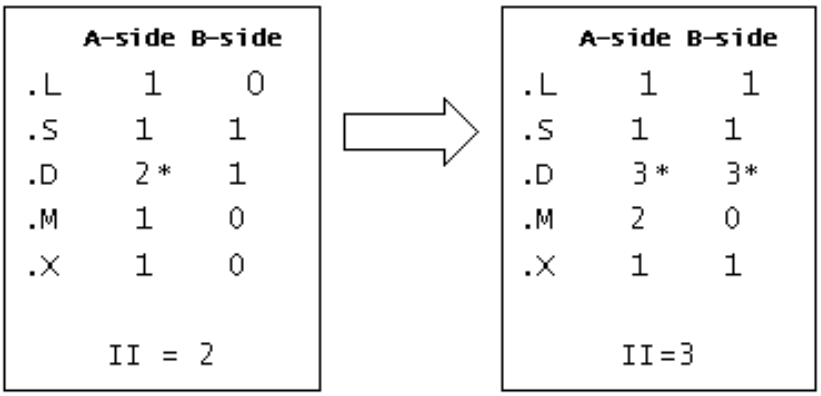


圖 36 運算單元使用狀況表

4.2.3 多個執行週期指令對暫存器的影響

管線化的處理，是 6701 DSP 的特色，編譯器會把你寫的程式碼，依照先前介紹的概念處理，以發揮出高執行效率。但是，如果軟體撰寫的不夠好，經過管線化處理的效能，會大打折扣。

```
void square_sum ( float * restrict dst, float * restrict src ,int n)
{    int i;    float sum=0;
     for ( i =0; i < n; i ++)
          sum + = src[i]*src[i] ;
     *dst = sum ;
}
```

MUSIC 演算法完成後，畫出 spectrum 的時候，我們都會對它的強度取對數來表示，這牽扯到對複數向量取平方和的動作，如以上程式所示。迴圈內所需要的運算單元，經過分析可以發現只用了一邊的運算單元，有一半沒有使用到；但是除此之外，還有一個隱藏的問題。浮點數的加法，需要四個時間週期才能執行完畢；對於管線化處理的架構而言，它的流程如下：

	A-side	B-side
.L	1	0
.S	1	1
.D	1	0
.M	1	0
.X	0	0

圖 37 運算單元使用狀況表

『*』的各數，表示迴圈的數目。理想的狀況下，迴圈從第零個時間週期開始執行，一直到第 9 個時間週期的時候，管線化的效能達到最高，每一個時間週期都可以平行執行 5 個指令，並且在第 13 個時間週期開始，每個時間週期都可以得到一個 sum 的運算結果〈因為浮點數的加法，需要四個時間週期才能執行完畢〉。

	0	1	2	3	4	5	6	7	8	9	10
.D1	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src	LDW src
.D2											
.S1					B	B	B	B	B	B	B
.S2				SUB	SUB	SUB	SUB	SUB	SUB	SUB	SUB
.L1										ADDSP	ADDSP
.L2											
.M1						MPYSP	MPYSP	MPYSP	MPYSP	MPYSP	MPYSP
.M2											

圖 38 預測管線化的程式執行情況

但事實上，編譯器在處理迴圈指令的時候，會以一個迴圈為單位，讓累加器內的資料不會被交錯〈stagger〉，然後才會開始第二個迴圈，也就是說，因為 ADDSP 必須執行四個時間週期，所以實際上管線化的結果是

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
.D1	LDW src				* LDW src				** LDW src				*** LDW src	
.S1					B				* B				** B	
.S2				SUB				* SUB				** SUB		
.L1										ADDSP				* ADDSP
.M1						MPYSP				* MPYSP				** MPYSP

圖 39 執行延遲時間造成的管線化執行效能降低分析表

ADDSP 的三個執行延遲時間〈delay slot〉造成每個迴圈多花了三個執行時間，迴圈數越多的話，造成時間浪費也就越多。解決的方法，就是把迴圈展開成四次，那麼就可以把空出的時間補滿。如果執行的迴圈數是 n 的話，那麼執行的時間，就回從原本的 4xn+(5+4)，降為 n+(5+4)，等於是增加了 4 倍的效能。

	A-side	B-side
.L	4	0
.S	1	1
.D	4	0
.M	4	0
.X	0	0

圖 40 運算單元使用狀況表

再回到運算單元的使用情況來看，每個時間週期只用了一半的資源，為了達到充分的利用，我們在將程式改寫成為

```
int main( void)
{     square_sum ( dst , src1 , &src[n/2] , n);
}
```

```
void square_sum ( float * restrict dst, float * restrict src1, float * restrict src2 ,int n)
{     int i;     float sum=0;
      for ( i =0; i < (n/2); i +=4)
      {sum1 + = src1[i]*src1[i] + src1[i+1]*src1[i+1] + src1[i+2]*src1[i+2] + src1[i+3]*src1[i+3];
       sum2 + = src2[i]*src2[i] + src2[i+1]*src2[i+1] + src2[i+2]*src2[i+2] + src2[i+3]*src1[i+3];
      }
      *dst = sum1+sum2;
}
```

如此一來，我們將每個迴圈內的運算單元使用發揮到極限，並且，運算的速度可再提升兩倍。

4.2.4 硬體資源的衝突

```
void mat_mul_cplx(float*restrict x, int r1, int c1, float* restrict y, int c2, float* restrict r)
{
     float real, imag;              int i, j, k;
     for(i = 0; i < r1; i++)
     {     for(j = 0; j < c2; j++)
           {     real=0;
                 imag=0;
                 for(k = 0; k < c1; k++)
                 {     real += (x[i * 2 * c1 + 2 * k] * y[k * 2 * c2 + 2 * j]
                             - x[i * 2 * c1 + 2 * k + 1] * y[k * 2 * c2 + 2 * j + 1]);
                       imag += (x[i * 2 * c1 + 2 * k] * y[k * 2 * c2 + 2 * j + 1] +
                             x[i * 2 * c1 + 2 * k + 1] * y[k * 2 * c2 + 2 * j]);
                 }
                 r[i * 2 * c2 + 2 * j] = real;
                 r[i * 2 * c2 + 2 * j + 1] = imag;
           }
     }
}
```

複數矩陣的乘法，是處理輸入訊號 correlation 的主要運算，以上是一個典型的實現方法。從編譯器的角度來分析，內部的迴圈，使用了 4 個 load、4 個 mpy、4 個浮點數加減法，2 個定點數的加減法〈一個是 loop counter，另一個是 y 矩陣 index 的調整〉，以及一個 branch，乍看之下，如果平均分配於兩側的運算單元，則可以在兩個時間週期之內完成一個迴圈的運算，再考慮前面說的浮點數加法延遲的影響，將內部迴圈展開成兩

次，就可以得到最大的效能；不過在實際的情況，並不如我們想像的單純，以下是我們的分析：

		0	4	8	12	16
T1	.D1	LDW XO				
T2	.D2	LDW YO				
	.S1					
	.S2					
X	.L1				SUBSP 6_M2,7_M1	
	.L2					ADDSP 11_L2,imag1
	.M1			MPYSP X2,Y3		
X	.M2			MPYSP X2,Y2		
		1	5	9	13	17
T2	.D1	LDW X1				
T1	.D2	LDW Y1				
	.S1					
	.S2	ADD inx				
	.L1					ADDSP 12_L1,real1
X	.L2				ADDSP 8_M1,9_M2	
X	.M1			MPYSP X3,Y3		
	.M2			MPYSP X3,Y2		

		2	6	10	14	18
T1	.D1	LDW X2				
T2	.D2	LDW Y2				
	.S1				SUB loop_count	
	.S2					
X	.L1				SUBSP 8_M2,9_M1	
	.L2					ADDSP 13_L2,imag2
	.M1		MPYSP XO,Y1			
X	.M2		MPYSP XO,YO			
		3	7	11	15	19
T2	.D1	LDW X3				
T1	.D2	LDW Y3				
	.S1				B	
	.S2	ADD inx				
	.L1					ADDSP 14_L1,real2
X	.L2			ADDSP 6_M1,7_M2		
X	.M1		MPYSP X1,Y1			
	.M2		MPYSP X1,YO			

圖 41 四個執行週期的內部迴圈資源使用狀況逐步分析圖

因為 addsp 有四個延遲的時間，所以，我們以四個浮點數加法來填滿運算單元停滯的時間。四個浮點數運算，也就對應了四個複數值的運算，經過分析，我們發現不論在運算單元，或是跨暫存器資料途徑，都沒有發生資源衝突的問題，並且，在實際程式驗證上，效能的確獲得將進一倍的提昇。

另一個在迴圈內經常遇到的問題，就是計算的編排沒有考慮到重複的讀取，造成效能的降低。

4.2.5 資料的相關性

```
void   iir (float * restrict x, float * restrict y, float c1, float c2, float c3,int n)
{    int i;
     for( i = 0; i < n; i ++)
          y[i+1] = c1 *x[i] + c2 *x[i+1] + c3 *y[i];
}
```

IIR filter 是一個常見的運算單元，以上是最簡單的形式。透過下面的圖，我們來分析迴圈內部的資料流。

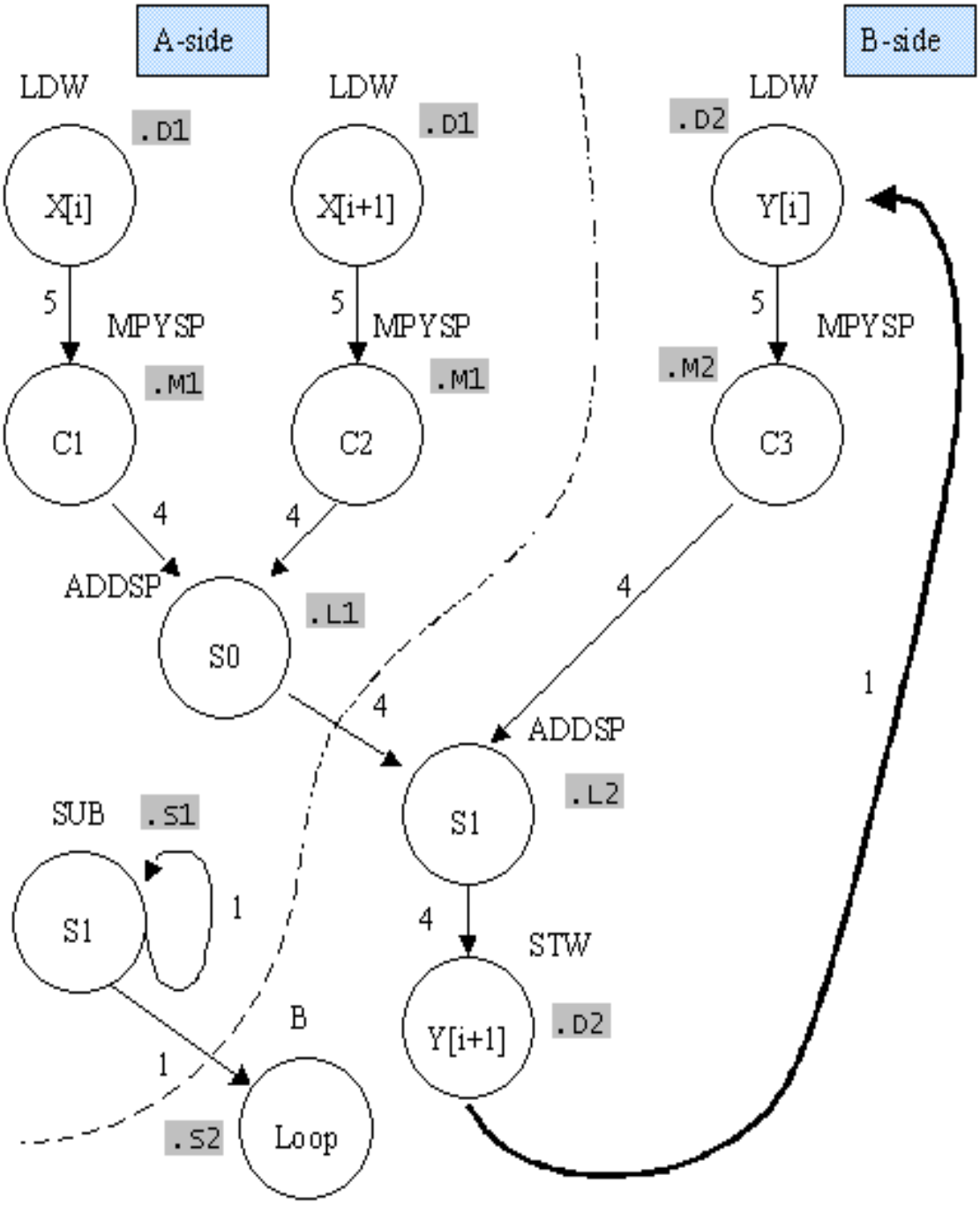


圖 42 資料相關程度分析圖

因為第一個迴圈計算出來的 y[i+1]，就是第二個迴圈運算的 y[i]，所以第二個迴圈的開始，必須要等到第一個迴圈內所有的運算都結束之後，才能開始，換句話說，就是執行每一個迴圈所需的時間，應該是 14 個時間週期，這是 iir 運算先天無可避免的缺點，這種現象稱之為迴圈相依〈loop carry dependency〉。不果仔細分析這 14 個時間週期，可以發現其實第二個迴圈的 y[i]，在執行完『ADDSP　S1』就已經產生，並且可以直接傳給『MPYSP C3』，如此一來，我們就可以將原本需要 14 個週期才能完成的迴圈，減為 8 個週期，整體的效能提昇 75%。以下是修改過後的程式結構。

```
void   iir (float * restrict x, float * restrict y, float c1, float c2, float c3,int n)
{    int i;
     float k = y[0];
     for( i = 0; i < n; i ++)
     {    k = c1 *x[i] + c2 *x[i+1] + c3*k;
          y[i+1] =k;
     }
}
```

4.2.6 線性組合語言

在部分的狀況下，編譯器所產生的組合語言並非如我們所想像，6701 DSP 提供了一種較高階的組合語言—線性組合語言〈linear assembly〉，讓程式設計師，可以搭配著 C 語言使用。它具有以下不同於傳統組合語言的特色：

- 程式設計師可以自行宣告暫存器名稱，方便程式碼的開發與維護。〈組譯器〈assembler〉會幫忙做轉換〉

- 程式設計師可以決定每一行指令由哪一個運算單元處理，讓組譯器更清楚的了解設計師所描述的程式架構，產生貼切的程式語言。〈亦可由組譯器自行決定〉
- 程式撰寫時，不需以 nop 告訴組譯器每一行指令執行的時間，讓程式碼看起來更乾淨易懂。〈組譯器會以管線化的觀點來分析，產生高效率的程式碼。〉

以下是將前面的矩陣搬移函數，改寫成為線性組合語言的程式碼。

```
             .global      _matrix_mov
_matrix_mov  .cproc    dst,src
             .reg      i,r1

             MVk       64,i
loop:        LDW    .D1    *src++,r1
             STW    .D2    r1,*dst++
          [i]SUB    .S1    i,1,i
          [i]B      .S2    loop
             .endproc
```

經過以上的最佳化技巧[11][12][13][14][15]，我們將原本需要執行 112ms 的 MUSIC 運算，提升到 4.77ms。

歸納前面的分析，整理出最簡單的加速內部迴圈運算效能的步驟：

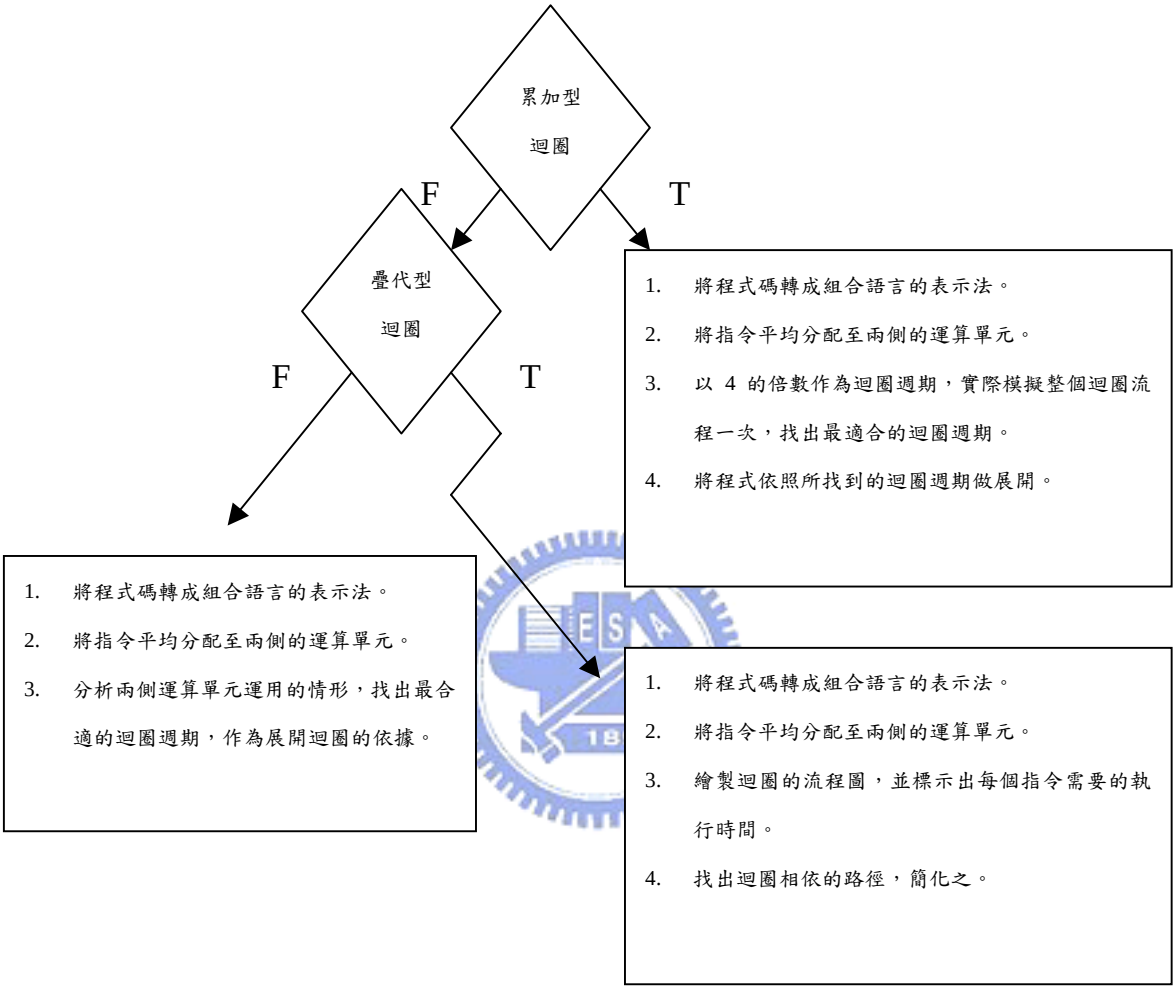


圖 43 內部迴圈效能分析步驟

4.3 演算法分析

MUSIC 與 ESPRIT 兩個演算法，都是屬於利用特徵空間來找出訊號源的方法，以下兩頁的圖是簡要的流程以及運算瓶頸的分析。

4.3.1 MUSIC

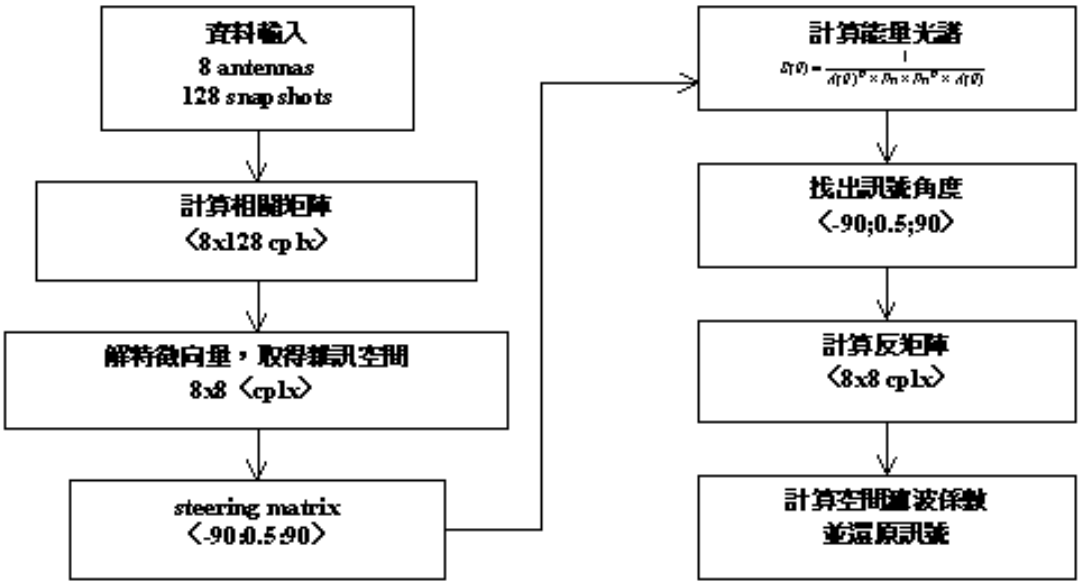


圖 44 MUSIC 空間濾波方塊圖

經過了程式碼的最佳化之後，在這個演算法，發現有五個主要的瓶頸，分別是：

- 解 8x8 複數矩陣特徵向量：特徵向量的解法，是透過 Householder transformation、QR iteration、以及 back transformation 三個矩陣轉換的運算來完成，需要大量的疊代計算，並且在疊代過程中，需要用到『除法』、『平方根』等較複雜的運算方法，花了約 325us。
- 建立 steering matrix：steering matrix 本身是一個 exponential 加一個 sin 函數的運算，在 C 語言的實現上，我將 exponential 函數分為實部虛部的運算，也就一個 sin 以及一個 cos 函數。從-90 度到 90 度，每隔 0.5 度計算一次，所以總共需要 361 次

的 cos 以及 722 次的 sin 預算，實際測量，發現共花了 3.53ms，這是相當可觀的。

- 計算能量光譜：這個部分的計算，也是需要-90 到 90 度去掃描出 $S(\theta) = \frac{1}{A(\theta)^H \times Pn \times Pn^H \times A(\theta)}$ 公式計算的結果，並且取 log，以方便我們觀察。361 次的『6x8 與 8x361 複數矩陣運算』加上 361 次的 log 運算，總共需要 765us。
- 計算 8x8 反矩陣:解 8X8 複數反矩陣的運算,為了節省程式記憶體空間,我利用了之前解特徵向量的程式得到特徵向量以及特徵方程式，利用 $A^{-1} = V(E^{-1})V^H$ 來求出反矩陣，所需要的時間，是 322us。
- 計算 8x8 相關矩陣：這個部分需要一次 8x128 複數矩陣複製，加上一次[8x128]x[8x128]複數矩陣乘法。共需要 150us

4.3.2 ESPRIT

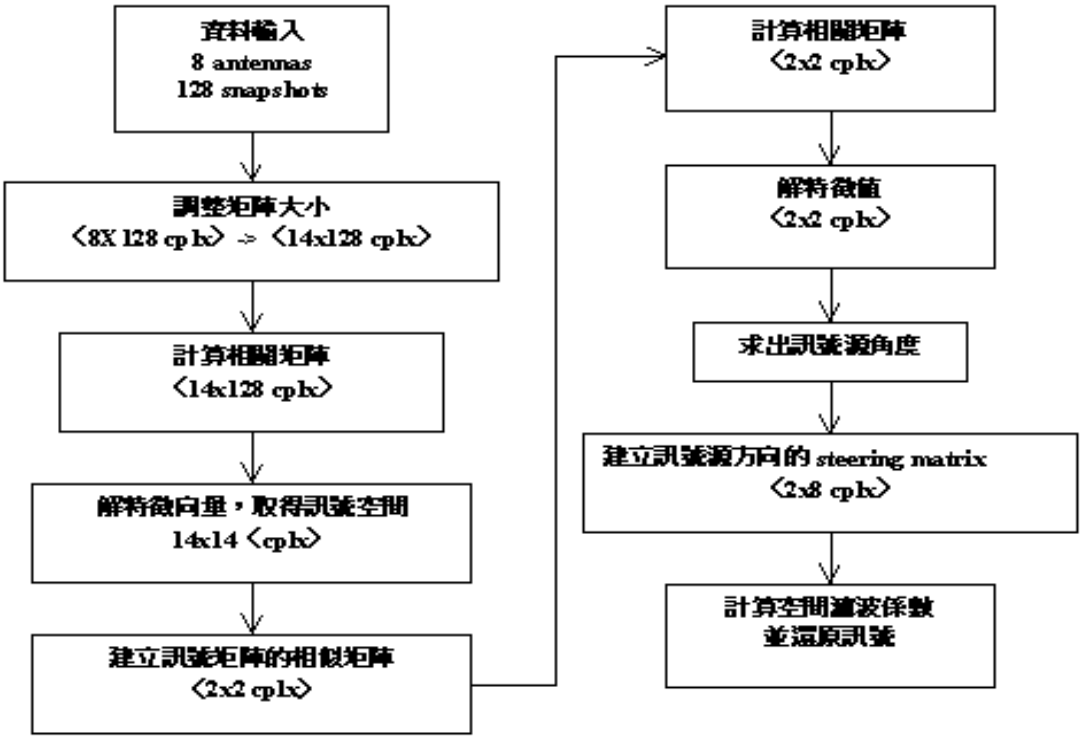


圖 45 ESPRIT 空間濾波方塊

分析最佳化過後的 ESPRIT 演算法，發現有兩個瓶頸：

- 計算相關矩陣：ESPRIT 演算法將矩陣的大小擴充至 14x128，所以增加了相關矩陣計算量，成為[14x128]x[14x128]，花費了 310us 的時間。
- 解特徵向量：我們解特徵向量的演算法，在前面有說明，是利用疊代的運算，一直到收斂算出答案來，所以矩陣階數越高，所需要的時間也就越長。14x14 的複數矩陣，一共花費了 750us 的時間。

以上概略的描述了兩個演算法，在經過程式碼的最佳化之後，剩下來的瓶頸。可以看出以 MUSIC 來做空間濾波，計算量要比 ESPRIT 大的多。

一顆 DSP 架構		
演算法	MUSIC MVDR	ESPRIT
程式執行時	4.77ms	1.2ms

圖 46 效能分析

4.3.3 分散式運算架構

接下來我們想要做的，是利用三顆 DSP 的協同運算，以提昇計算的效能。首先，我想利用三顆 DSP 平均分配運算量，以降低執行速度至 1/3，不過，在實際的評估之後，發現三顆 DSP 之間資料的傳遞，會造成另一個時間上的瓶頸。因為 DSP 之間的資料傳輸途徑 McBSP 速度只有 3.125M Bytes/sec，並不適合用來傳遞大量的訊號。

考慮實際的情況，訊號源的移動速度，假設是 120Km/hr，也就是 3.33cm/ms，所以每毫秒角度的變化量，並不會差太多，從接收訊號者的角度，可以每隔一段時間，做一次的角度更新，是不會影響接收訊號的品質。

以此觀點為基礎，則我們可以將訊號源角度估測的運算，移到後級的兩顆 DSP，最前端的 DSP 就專司空間濾波的任務，等這後級的 DSP 做完角度估測之後，再去更新前端 DSP 的訊號源角度，如此一來，平台運算的效能，可以再提高。其架構圖就如下：

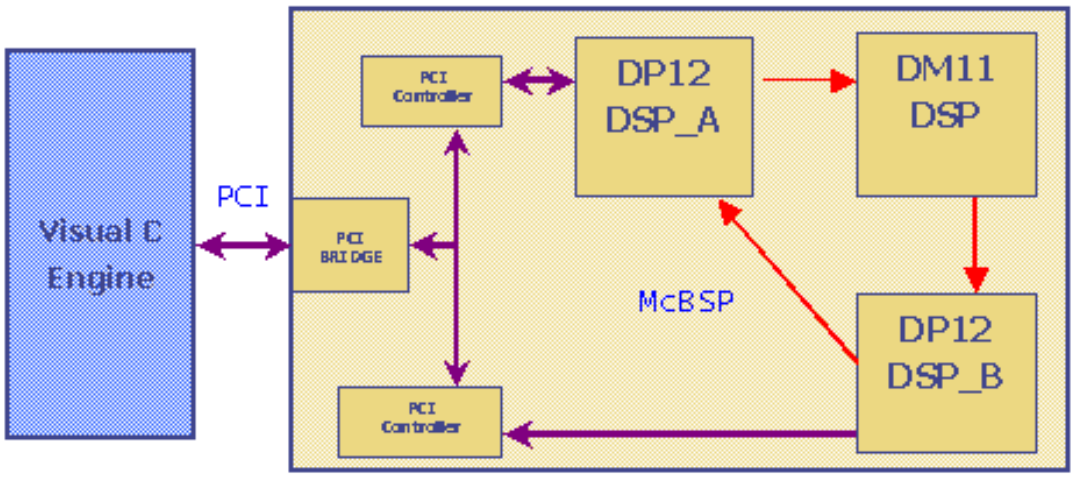


圖 47 適合空間濾波之分散式數位訊號處理器架構

在此架構中，DSP_A 是前端的 DSP，負責做空間濾波，將訊號分離出來；而 DSP_B 與 DM11 的 DSP，負責計算訊號源的角度。DSP_B 的另一個任務，是將產生的 MUSIC spectrum 與 MVDR Beam pattern 傳給 Visual C 做圖形的繪製，如此一來，就完全不會影響到空間濾波的速度了。

對於 MUSIC+MVDR 來說，只有『計算 8x8 相關矩陣』與『計算 8x8 反矩陣』運算與空間濾波有直接關聯，所以簡化後的架構，每 480us 即可完成一次空間濾波；而角度估測的部分，我們發現『建立 steeringmatrix』產生的資料，是一筆 8x361x2 words 的常數，在內部資料記憶體空間允許的狀況下，我們可以預存起來，節省重複運算的時間，如此一來，則每隔大約 1ms 的時間，就可以估測出新的訊號來源。

至於 ESPRIT，以相同的想法，第一顆 DSP 每隔約 420us 可以完成一次空間濾波；約 782us 可以更新一次訊號源的角度。

另外，我們如果以空間平滑〈Spatial Smoothing〉來作為 MUSIC 演算法的前處理，則第一顆 DSP 上『計算 8x8 反矩陣』的部分，階數降為 4x4，運算量也大幅的降低，空間濾波時間可再往下修正為 239us；而且在角度估測的部分，計算 8x8 矩陣的特徵向量，也會變為 4x4 的矩陣，運算量也會減少，訊號源角度更新的時間，可從原本約 1ms，提昇至 448us 即可完成一次運算，如此的處理速度，相當於可處理 535KHz 的基頻訊號，已達到即時的要求。以下是數據的整理：

測試資料：

- 天線數目：8
- 資料量：128 snapshot

三顆 DSP 架構			
演算法	MUSIC MVDR	ESPRIT	MUSIC_FBSS MVDR
空間濾波時間	480us	412us	239us
更新訊號源角度時間	1ms	782us	448us

圖 48 效能分析

第五章　結論

5.1 研究結果

本論文的目標分為兩部分，一是硬體平台資料傳輸介面的整合，二是測試平台運算的極限。

在硬體平台資料傳輸的部分，主要是將家銘學長所建立的 PCI Base 的資料傳輸架構，與青衛學長建立的 McBSP+RTDX 傳輸架構，加以整合。目前 DSP 之間的運算資料傳輸是以 McBSP 來實現，三顆 DSP 之間可以互相全雙工傳輸，組成一個強而有利的 DSP 運算單元；與 PC 端的溝通，則是透過 PCI 介面的傳輸，存取 PC 端的 Visual C slave server 內的資料，並透過此 slave server，再去呼叫存取 PC 上的其他資源，展現強大的資源擴充性。論文中是透過 VisualCslaveserver 來呼叫 Matlabengine，來產生模擬天線訊號資料與即時繪圖分析的功能。

平台運算極限的部分，分成演算法的最佳化與資料流分析兩部分。論文中分析 6701 DSP 內部硬體運算的算架與原理，撰寫出適合此數位處理器運算的程式碼，來提昇 MUSIC 與 ESPRIT 演算法的運算效能。藉由分析演算法每個步驟資料量與資料流的狀況，規劃了三顆 DSP 分散處理的架構，將可處理的訊號頻寬，提昇至 500KHz 左右。

5.2 未來展望

平台硬體的架設，已經到達相當完備的程度，而軟體開發的部分，不論是 PC 端的 slave server 架構，或是 DSP 端的撰寫流程與技巧，也有了充分的說明。以下就此平台在未來的應用，提出幾點可以再加強的部分：

- DSP 之間的傳輸速度：之前在分析平台資料量與資料流的時候，發現 McBSP 每秒鐘 3.125 Mbytes 的傳輸速度，會是一個潛在的瓶頸。雖然說我們以巧妙的規劃三顆 DSP 運算工作的方式，來解決這個問題，但

是如果換了其他演算法之後，這可能會成為降低效能的原因之一，所以建議內部資料的傳輸，改以 local PCI 的方式，會比較適合。

- 訊號擷取的速度：目前平台前端處理訊號輸入的 FPGA 損壞，導致訊號無法透過高速資料埠〈FPDP，160 Mbytes/sec〉連接至類比/數位轉換器，直接收取天線訊號，而改以由 Matlab 產生模擬訊號。在這個問題解決之後，則可以對天線訊號做更深入的研究。

參考文獻

[1]Petre Stoica and Torsten Soderstrom, Statistical analysis of MUSIC and ESPRIT estimates of sinusoidal frequencies, *International Conference, Acoustics, Speech, and Signal Processing*, 14-17 April 1991.

[2]John S. Thompson, Peter M. Grant, and Bernard Mulgrew, Performance of Spatial Smoothing Algorithms for Correlated Sources, *IEEE Trans. Signal Processing*, Vol. 44, No. 4, April 1996.

[3]Hongyi Wang, 2-D Spatial Smoothing for Multi-path Coherent Signal Separation, *IEEE Trans. Aerospace And Electronic Systems*, Vol. 34, No. 2 April 1998.

[4]H.Wang and K.J.R.Liu, "Coherent Signal Processing Using Arrays of Arbitrary Geometry", *ISR Technical Research Report*, T.R.94-34

[5]Ta-Sung Lee, *Array Signal Processing*, course sheet, 2002.

[6] Alle-Jan van der Veen," Direction Estimation Using Esprit", *Signal Processing for Communications,* course sheet, 2001.

[7]S.U. Pillai, *Array Signal Processing*, Springer-Verlag, 1989.

[8]H.T.Lau., *A Numerical Library in C for Scientist and Engineers*, Boca Raton, CRC Press, 1995.

[9]Patterson Hennesey, *Computer Organization & Design The Hardware/Software Interface*, Morgan Kaufmann, 2nd edition

[10]*TMS320C6000 Peripherals Reference Guide*, Texas Instruments, 2001.

[11]*TMS320C6000 Programmer's Guide*, Texas Instruments, 2000.

[12]*TMS320C6X Optimizing C Compiler User's Guide*, Texas Instruments,1998.

[13]*TMS320C6X Assembly Language Tools User's Guide*, Texas Instruments, 1998.

[14]*TMS320C67x FastRTS Library Programmer's Reference*, Texas Instruments, 2002.

[15]*TMS320C67x DSP Library Programmer's Reference Guide*, Texas Instruments, 2003.

[16]*PCI 9054 Data Book,* PLX Technology, April, 1999.

[17]林家銘, *以數位訊號處理器為架構隻智慧型天線接收機雛形實現*, 交大碩士論文,2002.

[18]鍾青衛, *智慧型天線接收機之 DOA 與 Beamforming 演算法即時實現*, 交大碩士論文,2003.

[19]梁開泰,"智慧型天線及其應用", *通訊雜誌*,第 69 期,10 月號,1999.

[20]Shaku Anjanaiah and Vassos Soteriou, *Using the TMS320C6000 McBSP as a High Speed Communication,* Texas Instruments, SPRA455A, August, 2001.

[21]Shaku Anjannaiah and Brad Cobb, *TMS320C6000 McBSP Initialization*, Texas Instruments, SPRA488B, April, 2002.

[22]Bavid Bell, *TMS320C6000 DMA Example Applications*, Texas Instruments, SPRA529A, April, 2002.

[23]Rusk Channel Sounder Information [Online] Available : http://www.channelsounder.de/