國 立 交 通 大 學
電機與控制工程學系

碩 士 論 文

即時 DSP 嵌入式 Linux 系統開發
及其於腦波訊號處理之應用

The Development of Real-Time DSP Embedded Linux System and Its Application for Brain Signal Processing

研 究 生： 黃 冠 智
指導教授： 林 進 燈 教授

中 華 民 國 九 十 三 年 七 月

即時 DSP 嵌入式 Linux 系統開發及其於腦波訊號處理之應用

The Development of Real-Time DSP Embedded Linux System and Its Application for Brain Signal Processing

研 究 生：黃 冠 智　　Student：Kuan-Chih, Huang
指導教授：林 進 燈 教授　　Advisor：Prof. Chin-Teng, Lin

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

授　權　書
（碩士論文）

本授權書所授權之論文為本人在 國立交通 大學 電資 學院 電機與控制工程 系所 電機控制 組 九十二 學年度第 二 學期取得 碩 士學位之論文。

論文名稱：即時 DSP 嵌入式 Linux 系統開發及其於腦波訊號處理之應用
The Development of Real-Time DSP Embedded Linux System and Its Application for Brian Signal Processing

1. ☑同意　　☐不同意

本人具有著作財產權之論文全文資料，授予行政院國家科學委員會科學技術資料中心、國家圖書館及本人畢業學校圖書館，得不限地域、時間與次數以微縮、光碟或數位元化等各種方式重製後散佈發行或上載網路。本論文為本人向經濟部智慧財產局申請專利的附件之一，請將全文資料延後兩年後再公開。
(請註明文號：　　)

2. ☑同意　　☐不同意

本人具有著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鉤選，本人同意視同授權。

指導教授姓名：　林　進　燈　教授

研 究 生簽名：　　學號：　9112516
（親筆正楷）　　（務必填寫）
日　　期：　民國　九十三　年　七　月　十六　日

---

1. 本授權書請以黑筆撰寫並影印裝訂於書名頁之次頁。
2. 授權第一項者，所繳的論文本將由註冊組彙總寄交國科會科學技術資料中心。
3. 本授權書已於民國 85 年 4 月 10 日送請內政部著作權委員會（現為經濟部智慧財產局）修正定稿。
4. 本案依據教育部國家圖書館 85.4.19 台(85)圖編字第 712 號函辦理。

國立交通大學

論文口試委員會審定書

本校 電機與控制工程 學系碩士班 黃冠智 君

所提論文 即時 DSP 嵌入式 Linux 系統開發及其於腦波訊號處理之應用
The Development of Real-Time DSP Embedded Linux System and Its Application for Brain Signal Processing

合於碩士資格標準、業經本委員會評審認可。

口試委員：

指導教授： 教授

系 主 任： 教授

中 華 民 國 九十三 年 七 月 日

即時 DSP 嵌入式 Linux 系統開發及其於腦波訊號處理之應用

研究生：黃　冠　智　　指導教授：林　進　燈　教授

國立交通大學電機與控制工程學系碩士班

摘　　要

在人們的生活中，嵌入式系統應用於訊號處理上十分廣泛，如多媒體語音與影像資訊處理等。為了讓訊號得到較佳的處理，往往需要大量的數學運算，但有鑑於嵌入式系統在運算能力不足，所以本論文採用同時具有 ARM 和 DSP 微處理器之 OMAP 來彌補這個問題。OMAP 是由 TI 針對嵌入式系統所開發的一個具有優秀運算能力的微處理器，使用 OMAP 架構可以使數學運算上更為迅速有效率，且 ARM 和 DSP 可以同步處理各自不同的行程。

本論文有三大發展主軸，分別為「OMAP 系統環境」、「OMAP 內部通訊」及應用於「腦波訊號處理系統」，主旨在於呈現如何在 OMAP 下有效運用其 ARM 和 DSP 的資源。其中 OMAP 運用以含有 DSP 模組的 DSPLinux 做為其系統環境，並利用 DSP Gateway 讓 OMAP 中 ARM 和 DSP 部分能相互溝通。本論文以 DSP Gateway 的機制將 ICA 演算法交由 ARM 和 DSP 分工處理，並進行系統分析與設計到落實系統設計，建構一個腦波訊號處理系統，經由腦波訊號處理系統中 ICA 演算法將訊號做實際的特徵擷取。此外，本論文亦分別對 OMAP 和 ARM 處理器做系統效能測試比較，由測試結果中可明白顯示出 OMAP 高效能的數學運算能力。

關鍵字：OMAP，ARM，DSP，DSPLinux，Linux，DSP Gateway
獨立成分分析(ICA)，腦波訊號處理(BSP)

The Development of Real-Time DSP Embedded Linux System and Its Application for Brain Signal Processing

Student： Kuan-Chih, Huang Advisor： Prof. Chin-Teng, Lin

Institute of Electrical and Control Engineering

ABSTRACT

The application of embedded system in signal processing is very wide in our life, such as Multimedia information processing on speech and image, and so on. In order to get better signal processing results, it always needs a large amount of computation. Because the capability of embedded system to compute is not enough, we use a microprocessor OMAP which has both ARM and DSP microprocessors to solve this problem. OMAP is a outstanding computational ability microprocessor which TI develop on embedded system. It is more efficient to compute by using OMAP, and ARM and DSP can synchronize to handle differenct processes.

In this thesis, we build DSPLinux OS on OMAP platform and utilize the resource of ARM and DSP effectively by DSP Gateway. DSP Gateway is a interface which make ARM and DSP communicate in OMAP. The thesis accomplish ICA algorithm which ARM and DSP handle cooperatively by DSP Gateway mechanism. It includes system analysis, design and realization. Then, we apply OMAP system to brain signal processing. By the ICA algorithm, we can successfully get signal feature from signals. Besides, we also proceed some tests of the system efficiency on OMAP and ARM architectures, and compare them. Finally, from the experimental results, we can find that OMAP have better computation ability than ARM.

KEYWORD：OMAP, ARM, DSP, DSPLinux, Linux, DSP Gateway,
Independent Component Analysis(ICA), Brain Signal Processing(BSP)

誌　　謝

本論文的完成，首先要感謝我的指導教授林進燈博士在過去兩年研究期間提供豐富的實驗資源和環境並且從旁指導協助，讓我能有高度的自主性決定自我的研究方向。

其次，我要感謝智慧型控制實驗室的全體成員，沒有他們也就沒有我個人的成就。特別感謝已畢業的陳建宏學長和王耀聰學長給予我在各方面的指導，無論是研究上疑難的解答、研究方法、寫作方式、經驗分享以及生活上壓力調適等惠我良多，很慶幸在研究生涯上能遇到如此的學長。另外要感謝黃騰毅同學，在過去兩年研究生活中同甘共苦，相互扶持。此外，也要感謝陳玉潔學姊在研究上的幫助以及生活中的扶持，還有感謝高士政、陳俞傑、董行偉以及李欣泓學弟，在過去這一年中的相伴。感謝實驗室助理在事務上的幫忙。

另外，感謝許多朋友的鼓勵，許駿飛博士、鐘翊方博士夫婦、林世賢小姐、周高平同學以及廖志昇學長，讓我在研究路途上站穩腳步，還有感謝我的一些好友，讓我的研究生活中，增填許多色彩。

最後，我最想感謝的是我的母親張鳳琴女士，父親黃世長先生，由於他們長久的辛勞和細心的照顧，才有今天的我，在此我要說一聲，我好愛你們，真的非常地謝謝你們。還有我的家人，我的親人，謝謝你們的鼓勵，謹在此將本論文獻給我的家人，共享這份榮耀與喜悅。

目　　錄

表 目 錄

圖 目 錄

縮寫對照

縮寫	原文
API	Application Program Interface
BCI	Brain-Computer Interface
CISC	Complex Instruction Set Computer
DARAM	Dual Access Memory
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FSF	Free Software Foundation
GPL	General Public License
HAL	Hardware Abstraction Layer
ICA	Independent Component Analysis
LART	Linux Advanced Radio Terminal
MPU	Micro Processor Unit
OMAP	Open Multimedia Architecture Platform
PDROM	Program and Data ROM
PCA	Principal Component Analysis
RISC	Reduce Instruction Set Computer
SARAM	Single Access Memory
SBC	Signal-Board Computer

一、 緒論

在多媒體發達的時代，數位處理技術發展快速，透過數位訊號處理可針對一些不同的訊號分析出許多的資訊，再利用這些資訊做許多不同的應用。舉例來說，可用於聲音通訊、影像甚至於在生理訊號的分析，發展相當多元化。隨著處理技術的進步，人們對於訊號處理的要求愈來愈精確快速，但精確的處理在計算上仍需花費不少時間。雖然目前在一般個人電腦上，在訊號處理上已有相當長足的進步。但對於現在強調微小且省電的嵌入式系統而言，在運算能力上便略顯不足。有鑑於此，考量加入 DSP 處理器於一般 ARM 的嵌入式微處理器做系統做整合。本論文主旨著重於 OMAP 微處理器系統應用的探討，針對兩種微處理器技術的整合，以及將其應用於訊號分析處理。以腦波生理訊號為處理對象，建構一個以腦波訊號分析計算為例的系統，以下分就「研究動機」、「研究背景」、「研究方法」及「相關研究」四部分加以說明。

1.1 研究動機

隨著多媒體與無線通訊的發展，訊號處理演算法不斷更新。為了追求更為真實的感受，在多媒體系統上必需做出更為精確的運算。對於腦波生理訊號處理應用，在生理訊號分析技術層面上，運算處理上要求更為嚴苛。就軟體的層面而言，需要一個好的演算法與良好的計算程序；就硬體的層面而言，處理器運算之能力亦是關鍵。現今軟硬體發展迅速，雖然大型主機對於多媒體的應用已不是個問題，但是大型主機，空間大且昂貴，為了縮小體積，嵌入式系統的發展勢必是個趨勢，強調小與省電。一般較有名的嵌入式系統的微處理器架構有 ARM、MIPS 等。雖然這些微處理器相當省電、運算能力不錯且支援多種作業系統，但處理大量的數位訊號資訊上，還是稍嫌不足，為了使系統運算能力更為強大，部分廠商引入了 DSP 微處理器來加強嵌入式微處理器的運算能力。為了提高在訊號處理的處理速度，採用此一類型之嵌入式微處理器有其必要性，除了一些精簡指令集外，還擁有 DSP 指令集，來提升系統運算之能力。

1.2 研究背景

有關本研究的背景，以下分成「舊發展系統」與「發展中新系統」分別論述。

1.2.1 舊發展系統

實驗室早期將既有虛擬實境發展之技術，已經由工業電腦逐漸轉至嵌入式單板電腦(SBC，Signal Board Computer)的應用。本實驗室最早採用 DSP 控制卡，來取代工業電腦，雖然具有優越的計算能力，硬體精簡，高效率，可即時處理大量訊號等特性，但不足的是缺乏作業系統的支援，且 DSP 系統軟體程式移植不易，故實驗室近年來改採用 LART 嵌入式單板做為開發環境。

LART(Linux Advanced Radio Terminal)是以 StrongARM 為微處理器的嵌入式單板，具有不錯的執行效率，體積小，支援多種作業系統，同時具備低成本的優點，採用這種精簡的嵌入式硬體單板可以取代原來的電腦和 DSP 控制卡。就其作業系統而言，是採用 ARM 架構的 ARMLinux，具有 Linux 作業系統開放原始碼的特性，在許多資料整合上，都能得到協助。但隨著演算法複雜度的增加，在運算處理上，以 ARM 架構為主的 StrongARM 微處理器，處理速度漸漸不合所需。整個舊發展控制系統衍進示意如圖 1-1：

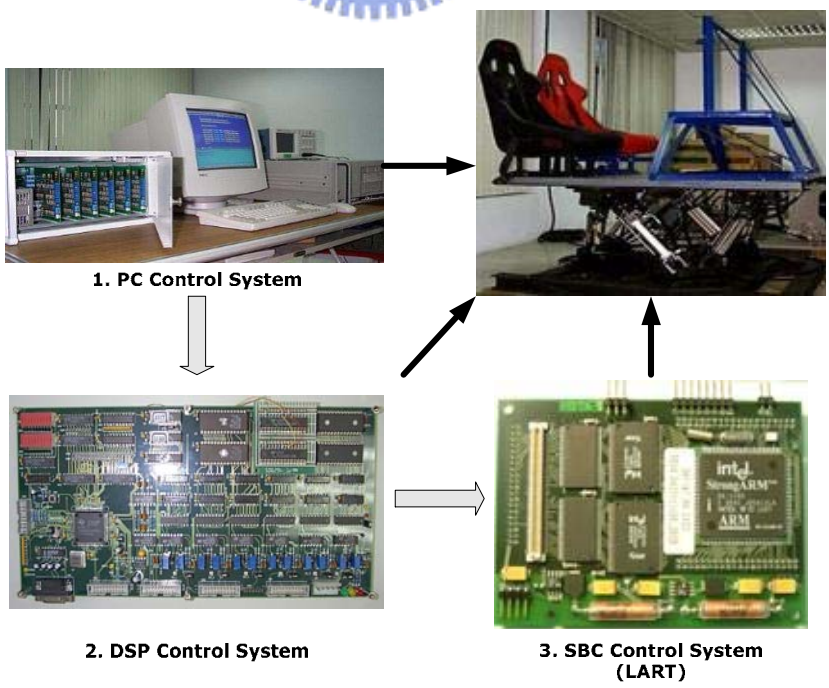


圖 1-1 實驗室單板技術衍進

1.2.2 發展中新系統

目前為了處理日益複雜的訊號處理的演算法，嵌入式單板勢必面臨大量的特殊運算，但一方面希望在嵌入式單板上，保有 ARM 處理器的移植性，發展性和作業系統的支援，另一方面，又希望加入 DSP 所擁有的優秀計算能力，因此，採用德州儀器(TI)所發展出的新處理器 OMAP。OMAP 結合了 ARM 和 DSP 的長處，支援多種作業系統，並且在運算上能使用 TI 所開發的 DSP 技術。

就嵌入式的作業系統而言，以原先的 ARMLinux 為藍本，加入對 TI DSP 系統的支援，成 DSPLinux。裡面包含了對於 OMAP 處理器上 DSP 部分的驅動，和 ARM 與 DSP 溝通的通道。於此，使得整個系統依然可以擁有開放原始碼的特性進行整合，容易維護開發，並且提升在運算上之能力等優點。

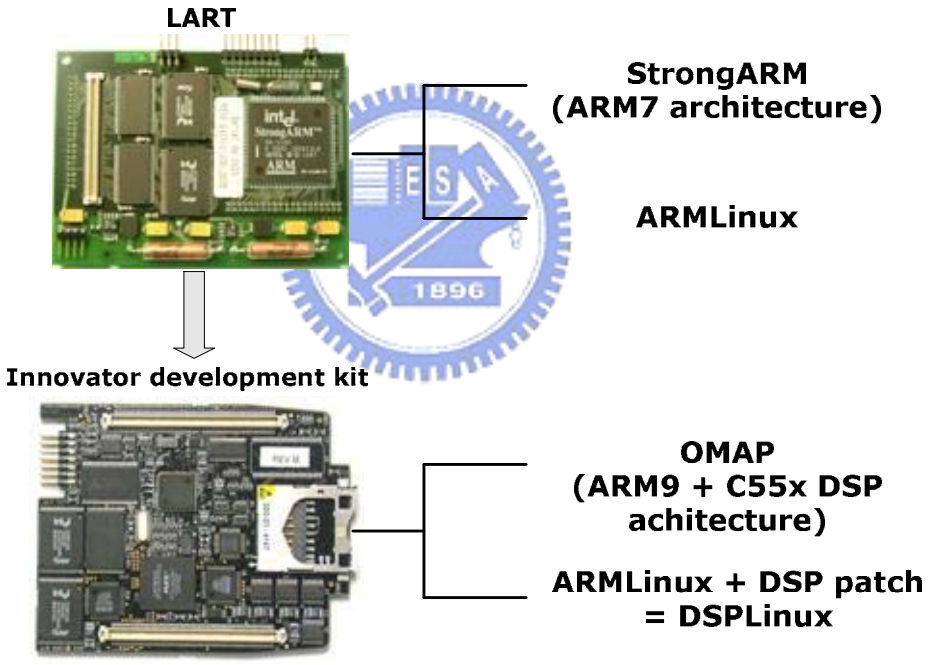


圖 1-2　發展中系統沿革及軟硬體之比較

1.3 研究方法

對於研究的動機與背景皆明確之後，還需要研究定位之方向，規劃之後必需進行之方式。本節將對「研究主軸」和「研究目標」兩個部分加以說明。

1. 研究方法

嵌入式系統的研究繁多，定一個明確的研究方向，確認主軸是必要的。本論文主要分為三個部分，分別為「OMAP 嵌入式系統」、「DSP Gateway-ARM/DSP 通訊」、「腦波訊號處理的實現與應用」。其中 OMAP 嵌入式系統，是在描述關於使用 OMAP 單板的發展環境，硬體和軟體特性，其中利用 DSP 部分 DSP/BIOS 的特性，建立 ARM/DSP 之間通訊的基石。至於 DSPLinux 環境建構之方式將於附錄說明。DSP Gateway-ARM/DSP 通訊描述在 DSPLinux 作業系統下，負責扮演作業系統主控者的 ARM 微處理器部分如何與 DSP 部分通訊，並將複雜運算之部分交由 DSP。這部分主要在描述兩顆處理器之間的流程分析及設計方式。腦波訊號處理的實現與應用在描述實驗室現階段對於分析腦波的方式，如何在 OMAP 的單板上實現，包含有快速傅立葉轉換(FFT)、獨立元素分析(ICA)及一些腦波分類的方法等。本論文將針對其中 ICA 演算法實現於 OMAP 單板上。在 OMAP 上的程式都將會包括 ARM 與 DSP 兩個部分，最後將整個分析的系統流程完整地表現出來。關於三大研究主軸的詳細內容，本論文將分別於第二、三、四章逐一解說。

2. 研究目標

有鑑於複雜的訊號處理應用需要使用到大量的運算，對於改善運算能力較差的嵌入式處理器，希望擁有較高計算能力的處理能力。尤其在腦波資料運算時，演算法過程複雜。故本論文最終目標是希望實現『即時腦波處理系統』，使用以 DSPLinux 為作業系統的 OMAP 嵌入式單板做為發展平台。建立一個運算能力強，快速、可靠度、精確度高的系統。

1.4 相關研究

雖然 OMAP 目前還在開發階段，OMAP 已被許多大廠做為開發研究平台。如 Nokia、SONY 及 Ericsson 等，就目前已經有幾家大廠推出以 OMAP 為架構的手機，國內也有相當多的廠商漸漸投入 OMAP 開發應用中。以下就針對 OMAP 列舉一些相關研究：

■ 官方資源：

(1) TI(Texas Instruments)OMAP 平台發展的官方資源
http://focus.ti.com/omap/docs/omaphomepage.tsp

(2) PSI(Productivity Systems Inc.)為是一家獨立 OMAP 技術中心，負責 TI 在 OMAP 上開發平台 Innovator Development Kit 的技術支援。除此之外，PSI 自己也開發由 OMAP 為核心的平台 Minno O5。
http://www.prodsys.com/

■ 業界開發：

(1) Nokia 60 系列使用 Innovator Development Kit 做為應用開發，使新的 2.5G 和 3G 的應用移植到具備高性能、高效率應用處理器的 OMAP 家族中，其中包括 OMAP710 和 OMAP1510。

(2) MontaVista 軟體公司在 TI 的 OMAP 上開發 Linux 平台
http://www.mvista.com/

1.5 論文架構

本論文共分為五個章節，第一章旨在說明本論文的動機、主軸、目標、研究背景及相關研究的介紹。第二章則針對第一項主軸「OMAP 嵌入式系統」加以介紹，第三章則介紹如何在 OMAP 內 ARM 和 DSP 的利用和溝通，主要是說明 DSP Gateway 的架構和設計方式。於第四章將說明本論文所實現的腦波分析系統的架構，設計與實現，並且將系統做實際的測試討論。最後於第五章中，則是對本論文作一個總結，展示腦波分析的結果，並且做整合性的探討，並對於系統給予總結。各章節之間的關係，如圖 1-3：

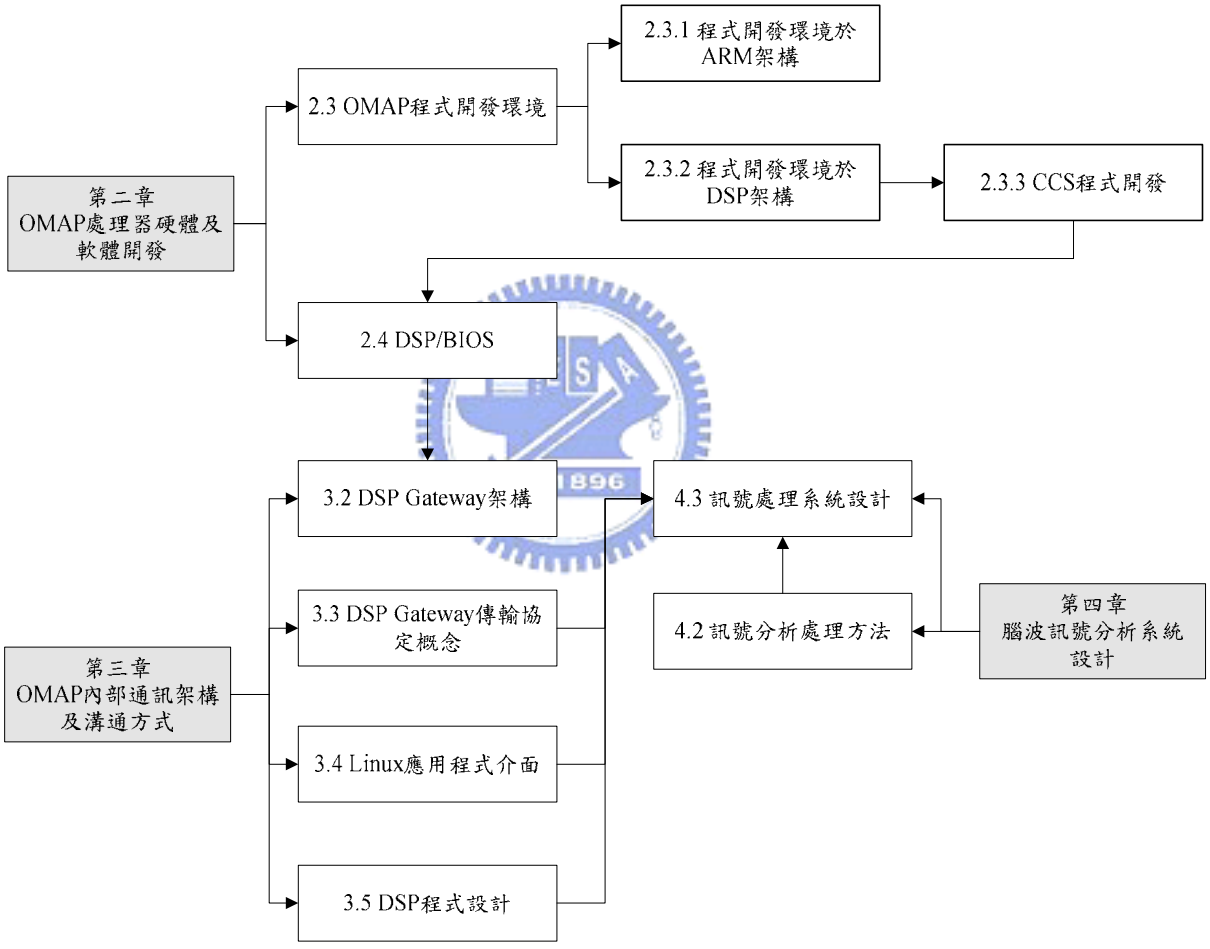


圖 1-3　論文章節關係圖

二、 OMAP 處理器硬體及軟體開發環境

本論文主要使用以 OMAP1510 微處理器為核心，並利用德州儀器(Texas Instruments)所開發的嵌入式單板 Innovator Development Kit 為開發環境。所以本章節將對 OMAP1510 微處理器系統做個探討，主要分成二部分加以描述，並且加以比較。首先，將介紹 OMAP1510 微處理器的硬體架構進行規格說明。再來，將比較其他常見的嵌入式微處理器 Xscale 和 x86 的平台做整體性比較。以及單板之開發環境 Linux 和 Win CE 做簡單的說明比較，將採用何種作業環境做為研究平台的系統。最後將對 OMAP 程式開發環境和特性做個說明，在程式開發上，主要有 Linux 平台下的跨平台環境(Cross Compiler)開發與 ARM 微處理器相容的應用程式。DSP 端的程式開發，主要是以 TI's CCS(Code Composer Studio) 做為開發環境，並針對 DSP 多執行緒的作業核心—DSP/BIOS 所提供的功能來做說明。

2.1 OMAP1510 處理器

OMAP 為 Open Multimedia Architecture Platform 的縮寫，OMAP1510 是一個有雙核心的微處理器，使用 TI-enhanced ARM925 微處理器並結合 TMS320C55x DSP 核心。OMAP 具有高效能平衡以及低功率消耗的能力，且擁有極佳的數值能力。本小節將簡述 OMAP 的背景及其架構。

2.1.1 OMAP 開發背景

德州儀器公司(Texas Instruments，TI)以 DSP 微處理器的優越性能著名，近年來更推出一項先進的技術由 RISC 整合 DSP 的 SOC 嵌入式系統微處理器：OMAP。OMAP 最大的特點在整合了 TI DSP 核心及 ARM 的 RISC 架構和各種週邊控制器的設計，其應用定位於即時的多媒體影音資料處理、語音辨識系統、網際網路通訊及無線通訊等訊號分析應用。OMAP 採用的 TI 的 TMS320C55X DSP，適合處理大量即時多媒體資料，比如說 MPEG1、MPEG2、MPEG4 或是其他音訊資料等。此外 OMAP 本身就是一個以 ARM925 RISC 為主體的微處理器，可以作為嵌入式作業系統的控制核心，用於處理人

機介面等系統功能主控的相關運算傳輸。另外 OMAP 功率消耗方面也有驚人的表現，可以在很低的功率下完成資料處理的動作。

OMAP 開放式的架構，並提供了一套標準界面，協助廠商發展新的應用軟體或是增加新的功能。OMAP 架構可移植性高，並且相容於大多數的作業系統，例如：Symbian OS™、Linux、Microsoft® Windows CE 3.0 和.NET 以及 Palm OSO 等。廠商可利用 OMAP 微處理器做出效能最高、電力消耗低的 2.5G 與 3G 無線裝置，並讓軟體工程師輕易運用即時 DSP 功能。此外 TI 發展出 DSP/BIOS Bridge 架構，設計人員利用最佳的方式，把資料分析計算工作適當地分配給 ARM RISC 和 DSP 微處理器，使得系統呈現出最佳的運算效能，並能節省功率的消耗(Power Manager)。

在軟體方面，面對新的硬體平台與新的產品需求，嵌入式軟體技術也勢必要大大提升，基本上 OMAP 架構微處理器內部的 DSP 微處理器核心與 ARM RISC 微處理器核心分別由兩個不同的系統所控制，DSP 微處理器核心是採用 TI 所自己研發的微核心（Micro Kernel）多工即時的系統，稱為 DSP/BIOS。在 DSP/BIOS 的架構上面可以提供軟體工程師方便開發符合即時運算效率的軟體元件工作。在 ARM 部分，一般常看到的嵌入式作業系統，Linux 和 WinCE 等都有支援，並且可以用來控制整個系統的運作。由於兩個部分分屬於不同的系統所管理，為了發揮由 ARM 操作和 DSP 運算的分工效能，兩個處理器之間的資料流通就成為研發重點，TI 針對這點提出 DSP/BIOS Bridge 架構，能夠讓應用程式開發人員在雙處理器架構下撰寫程式，就有如在單一處理器的感覺一樣方便，在下一個章節，所討論到的 DSP Gateway 便是以其為基準，所提出的溝通方式。

2.1.2 RISC 加入 DSP 處理器理由

為了支援多媒體應用聲音訊號處理以及其他應用，例如本論文中的腦波訊號分析，數位訊號技術顯得重要。但是做數位訊號處理，其中運算一般而言是相當大量的，已經超出一般 RISC 微處理器所能負荷的，雖然嵌入式微處理器的指令處理時脈速度一直在進步，但是這速度提升有限，最重要的根本方法還是需要對於運算處理有特別設計電路，增加常用的訊號處理運算指令集，TI 系列的 DSP 微處理器，便是具有這方面的特性。本論文中，在處理腦波訊號時，如同視訊與音訊的播放都是一種信號處理工作，同樣是要用到大量的運算。TI 自己的優勢：DSP 微處理器，就是支援信號處理運算，而且相較於 RISC 處理器，DSP 每個時脈週期內消耗比 RISC 更少電力。DSP 用更少的指令完成一個重複大量數學運算的演算法，並在一個時脈週期內執行更多的指令。當一個應用程式執行時，一顆 RISC 處理器，執行其他程式時受到影響，加上缺乏 DSP 指令集，受限於 RISC CPU 本身信號處理能力的限制，效能往往不佳。有鑑於此，TI 在 ARM 的架構上再加入 DSP 的微處理器，讓整個系統保有 RISC 微處理器，支援多種作業系統的平台特性，同時具有強力運算能力的 DSP 微處理器。OMAP 架構能讓 DSP 與 RISC 處理器並行工作，程式設計工程師可使用 DSP 功能做計算上處理，RISC 執行擅長的命令與控制功能。

2.1.3 OMAP1510 硬體架構

TI 的 OMAP 1510 硬體結構圖，如圖 2-1所示，包含 175MHz 的 ARM925 和 200MHz 的 TMS320c55x，整合了多功能的周邊控制元件，例如 LCD 控制器、記憶體擴充介面、紅外線介面、觸控式面版擴充介面及 USB 介面等，是一顆高度整合性的多功能嵌入式系統微處理器。目前有許多世界性的大廠宣佈將以 OMAP 作為新一代無線通訊的新資訊家電產品核心，如 Nokia、SONY 及 Ericsson 等。以下是 OMAP1510 的特性，分別就 OMAP 整體及 DSP 和 ARM 部分的探討[17]：

- 低功率(Low power), 高效能(High-Performance) 的 CMOS 技術

1. TMS320C55x DSP Core：

C55x DSP 內有 5 組數據匯流排，在一個周期內允許三次讀取作業和兩次寫入作業。C55x 具備雙 MAC 結構，內部具有一個硬體圖形加速器。綜上所述，C55x DSP 是高度複雜功能強大的，專為基於多媒體的即時應用而設計低功耗元件。DSP 用於處理所有多媒體應用。以下為 TMS320C55x 介紹

- 可達 200MHz(最大處理頻率)，且電壓在 1.5v(一般情況)
- 一個循環(cycle)可執行一到二個命令
- 32K x 16-bit on-chip dual-access RAM (DARAM) (64 KB)
- 48K x 16-bit on-chip single-access RAM (SARAM) (96 KB)
- 16 KB I-cache, 8 KB D-cache
- Video hardware accelerators for DCT, iDCT, pixel interpolation, and motion estimation for video compression

2. TI925T ARM9TDMI Core：

ARM 既支援 32 位元也支援 16 位元(Thumb 模式)指令集，ARM925 用於執行作業系統(OS)。以下為 ARM925 的介紹:

- 可達 175MHz(最大處理頻率)，且電壓在 1.5v(一般情況)
- 16KB I-cache; 8KB D-cache
- 192-KB of shared internal SRAM - frame buffer
- Support for 32-bit and 16-bit (Thumb mode) instruction sets
- Data and program MMUs
- Two 64-entry translation look-aside buffers (TLBs) for MMUs

■ 17-word write buffer

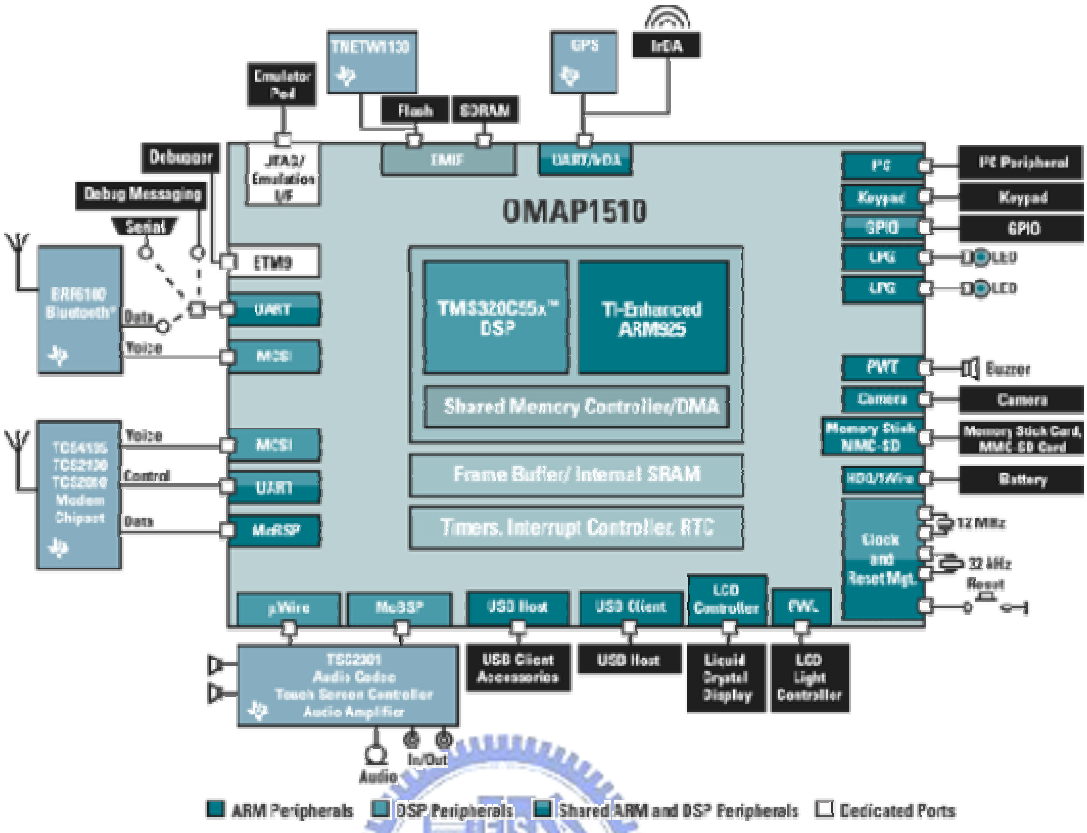


圖 2-1 OMAP1510 微處理器架構圖

資料來源：Texas Instruments http://www.ti.com/[17]

2.1.4 OMAP1510 DSP 內部記憶體

在前面 OMAP1510 的規格中，在 DSP 核心部分的內部記憶體，列出有 DARAM 和 SARAM.。事實上 OMAP1510 內部有三種不同的內部記憶體給 DSP 使用，分別是 DARAM(Dual Access Memory)，SARAM(Single Access Memory)和 PDROM(Program and Data ROM)。一般可以使用的為 DARAM 和 SARAM，而 PDROM 在 DSP Gateway(ARM 與 DSP 溝通的橋梁，在第三章將會詳細說明)並沒有使用。表 2-1是這三個記憶體在使用時，DSP 和 ARM 相對於 MPU 實體記憶體的對應位置，圖 2-2則用圖形的方式表示出記憶體圖的對應關係。

表 2-1　OMAP1510 內部記憶體位置對應表

	DSP Byte Address	MPU Physical Address	Linux Virtual Address	Size
DARAM	0x000000 | 0x00ffff	0xe0000000 | 0xe000ffff	0xe0000000 | 0xe000ffff	64kB
SARAM	0x01000 | 0x027fff	0xe0010000 | 0xe0027fff	0xe0010000 | 0xe0027fff	96 kB
PDROM	0xff8000 | 0xffffff	0xe0ff8000 | 0xe00ffffff	(not used)	32 kB

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

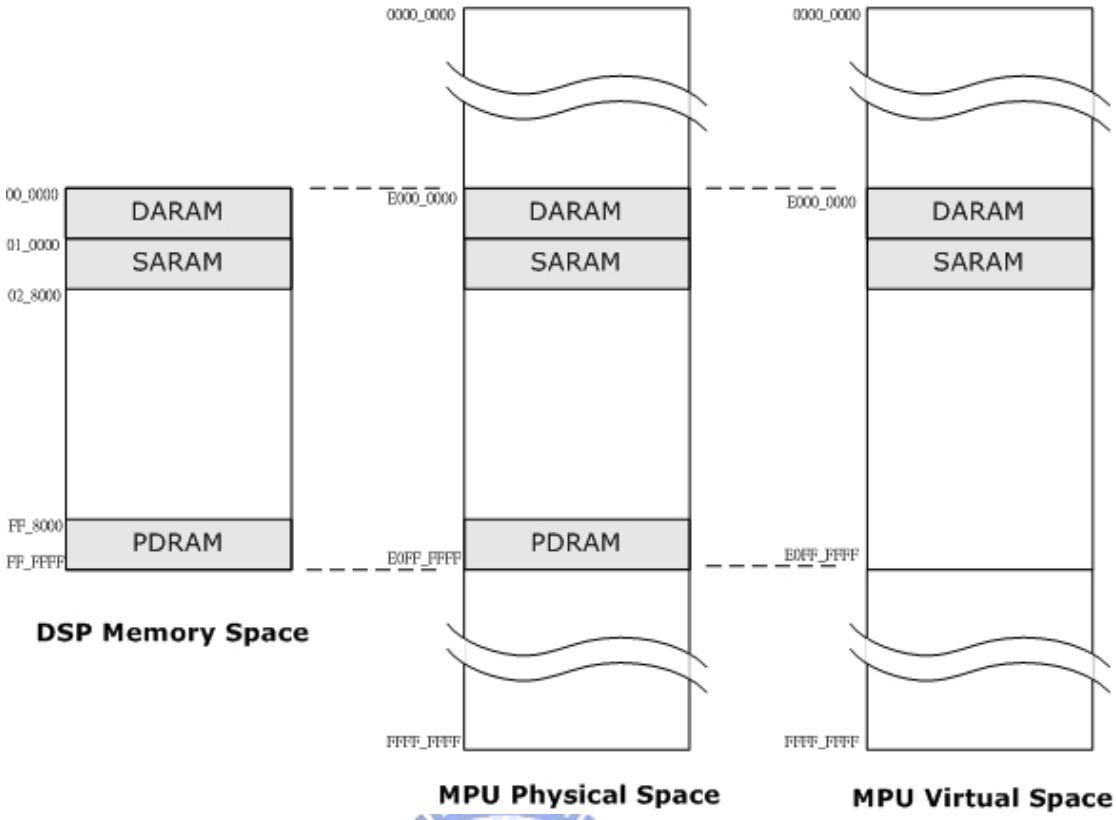


圖 2-2　OMAP1510 內部記憶體堆疊圖

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

2.2 微處理器架構比較

一般常見的嵌入式，常見微處理器架構，有 Xscale、x86，以及 OMAP。本節就以這三種不同的架構來做比較。

2.2.1 x86 與 OMAP、Xscale 硬體比較

x86、OMAP 與 Xscale 大致上可以分成兩類，一種是 x86 系列的，一種是 ARM 架構系列的。Linux 對 x86 系列的處理器支援，從 Intel 推出 386 以來，一直涵蓋到現在，早年來以 SOC 技術整合 386 系列處理器的 CPU 核心（CPU core）和傳統 PC 週邊機能，加入嵌入式的市場。而另一類 ARM(Advanced RISC Machine)，是 ARM Holdings Ltd.所維護推動的處理器系列，與其他製造商不同，ARM Holdings 不會製造處理器，ARM 會依據 ARM core 為客戶設計 CPU 核心。雖然沒有自己的製造的處理器，但是只要是使用他們架構的核心，都有一致的特性，就是共同享有 ARM 指令集（instruction set），讓不同版本的 ARM 處理器在軟體上完全相容。目前，ARM 微處理器的製造商包括 Intel、Toshiba 等許多其他廠商，ARM 的架構在許多應用中都相當受歡迎。而 TI，更將 ARM 和 DSP 核心結合成 OMAP。一方面可以讓其嵌入式微處理器對作業系統支援度高，且具有一般 ARM 微處理器所沒有的 DSP 運算指令集。下面將 x86 架構加以說明，再說明另一款相當受歡迎的以 ARM 架構為主體的微處理器 Xscale。最後以表 2-2將三個不同點做個比較。

(1) x86

x86 架構即為一般個人電腦(PC)所相容的 CPU 架構，就機能和程式上的編程而言，大多數的 x86 嵌入式系統都與其非常類似或都完全相同。使用 x86 的 pc 架構應該是文件最普及的架構，書籍及網路資料相當豐富，文件相當完整，所有作業系統與應用軟體，都以其為測試平台，但儘管是 x86 是受歡迎且廣為人知的系統，但也只能代表嵌入式系統的一部分，因為 x86 是 CISC 非 RISC 指令集，故在電路設計上較為複雜，通常基於複雜度和成本的考量，一般設計者大都使用 ARM、MIPS 和 PowerPC 而不使用 x86。且在嵌入式系統相當重要的功率消耗，較擁有 ARM 架構的微處理器消耗較高是其缺點。因為有許多嵌入式系統如 Mobile phone 和 PDA，強調能攜帶方便，如果能量消耗過高，將不利於使用的時間長度。

(2) Xscale

Intel 針對資訊發展的大量需求之下，發展高工作效率及更低功率消耗的第二代嵌入式微處理器 Xscale 架構。Xscale 是 StrongARM 的新一代微處理器架構，同樣是以 ARM 架構為其 CPU 核心，但 Xscale 微處理器不包含 ARM RISC 微處理器所擁有的浮點運算(Floating Point)硬體線路，Xscale 微處理器只擁有定位點運算(Fixed Point)的線路架構。Xscale 的 RISC 微處理器核心作為符合 ASSP 標準(Applications Specific Standard Products)嵌入式系統控制核心，Xscale 在工作頻率上表現相當優秀，使得大部分的嵌入式系統微處理器工作頻率上只能夠望其項背，加上 Xscale 微處理器採用了 Intel Pentium III 微處理器中 superpipeline 技術， Xscale 只需要 1.75W 功率消耗下，擁有 1270MIPS 的執行效率。但是對於訊號處理運算上，由於無 DSP 支援，相較於 OMAP 而言，這點是 Xscale 較 OMAP 所不足的地方。

表 2-2　三種不同微處理器架構的比較

處理器 (microprocessor)	OMAP	Xscale	x86
指令集 (Instruction Set)	ARM(RISC)、DSP	ARM(RISC)	x86(CISC)
訊號處理運算 (DSP)	有(優勢)	無	無
電力消耗 Power Consumption	低	低	高(缺點)
作業系統 (Operating System)	支援多種作業系統	支援多種作業系統	支援多種作業系統

2.2.2 Linux 與 Windows 工作平台

開發者選用 Linux 的最重要理由，就是開放原始碼軟體優於私利軟體的地方，事實上有了原碼不僅可以讓作系統變得容易理解，也可以避免依賴單一作業系統廠商的情況。除此之外，選用 Linux 理由還有對程式碼的品質與可靠度等優點。一般而言，Linux 核心及系統上大部分計劃的程式都具有以下幾個特性：模組結構，容易修改，可擴充且

可設定高，可預測以及從錯誤中復原之能力，並具有長期執行的能力。使用者可以無限制地使用 Linux 的原碼及所有建立工具，包括核心在內，都是在 GNU 通用公共授權書(GNU General Public License，簡稱 GPL)的保護下發行。取用元件的原碼是每個使用者的權利。

Linux 對硬體的支援度廣泛，意味著 Linux 支援各種不同類型的硬體平台裝置。儘管仍有些廠商尚不提供 Linux 驅動程式，但預計會有越來越多的廠商加入，而且大部分的驅動程式是靠 Linux 社群自己維護，可以安心使用，不用擔心廠商一旦停止該生產線。沒有其他作業系統可以提供如此的移植性。成本是 Linux 的最大的優勢，Linux 的軟體幾乎都是免費的，不需使用權(License)，沒有使用人數上限，且使用 PC 比一般 Unix 機器便宜，即可達到不錯的效能。相反地，Windows 系統即需要每台機器買一個 License，連線人數受到控制。綜合以上 Linux 不管是效能及穩定性的優點，再加上成本低廉，使得 Linux 在許多平台，皆漸漸成為主流。

表 2-3 嵌入式作業系統開發平台比較

平台	Linux	Windows
原始碼	公開	不公開
程式發展工具	有，免費(Ex：gcc)	有，需費用(Ex：Visual C++)
維護	Linux 社群維護居多	廠商提供服務
應用程式	相當多，具免費	多，大多需要費用
平台移植性	高	廠商需與 Microsoft 合作
費用	免費	相當高

綜合以上所言，關於 Linux 與 Windows 的比較如表 2-3所示，由表上可知選擇 Linux 為開發環境可減少成本，並且修改維護自由，不依賴定廠商。由於 Linux 在嵌入式系統市場上有發展之潛力，促使許多公司開始在這個領域推廣 Linux，一般常看到的廠商有 Red Hat、MontaVista[1]、WinRiver、LynuxWorks 等。

[1] MontaVista 和 WinRiver 為專門為 Embedded 開發 Linux 系統的廠商

2.3 OMAP 程式開發環境

在前面將 OMAP 微處理器硬體和環境選擇簡單的說明之後，接下來要進入軟體應用的開發環境部分。因為 OMAP 是由 ARM 和 DSP 兩種架構所組成的，如果只有一種程式發展平台是不夠的，必需要兩種發展平台，對於這一點，本論文的規劃是：

(1) ARM code 使用 Linux 跨平台程式編譯 arm-linux-gcc(Cross Compiler)。

(2) DSP code 使用 TI 的 CCS(TI proprietary Integrated Development Environment)來發展。

在本節將會針對 ARM 和 DSP 的程式開發環境做個說明。

2.3.1 程式開發環境於 ARM 架構

在一般 PC 的 Linux 的環境下開發程式，最常使用到的編譯器，莫過於 gcc(GNU Compiler Collection)，然而嵌入式系統開發也需要用到編譯器(Compiler)、連結器(Linker)、組譯器(Assembler)，整合開發環境。以 ARM 而言，不同於 x86 架構，在 x86 環境下的產生的執行檔，並不能用於 ARM 架構的環境中，若想要在 x86 環境下開發 ARM 程式需要建立一個跨平台開發的工具（cross-platform development tool，或稱為 cross development tool）。對於 ARM 程式的開發，採用 GNU 的工具鏈（toolchain），對 ARM 平台的 cross-compiler(交叉編譯)—arm-linux-gcc 來進行程式的編譯。arm-linux-gcc 將程式碼針對 ARM 的架構進行編譯，產生出來的執行檔便可以於 ARM 的架構上執行。

2.3.2 程式開發環境於 DSP 架構

CCS (Code Composer Studio)是德州儀器所提供用開發 DSP 程式的軟體，包含了跨平台的開發工具，CCS 提供一個完善的整合發展環境(IDE，Integrated Development Environment)，對於多種處理器、多使用專案，並且是第一個提供 DSP(TMS320C2000、TMS320C5000、 TMS320C6000) 與 OMAP 應用程式開發的環境。CCS 包含了程式編譯器、模擬器以及程式除錯器，藉由這些工具可以很輕易地開發 OMAP 的應用程式，尤其在撰寫 DSP 的應用元件時，透過 CCS 可以很容易在 DSP/BIOS 上整合支援影像及音訊資料處理的套件。

CCS 以一致的環境來整合所有 host 與 target 工具，包括 TI 的 DSP/BIOS kernel、code-generation tools、fast simulators、debugger、與 Real-Time Data Exchange (RTDX) 技術，簡化應用程式的開發。

2.3.3 CCS 程式開發

在使用 CCS 做為 DSP 程式開發之前，先了解程式開發流程，將有助於使用 CCS 的元件。在一個程式設計前，對於要設計的程式先有個概念，再開始程式碼的編寫，在過程中遇到錯誤時需要除錯時，CCS 具有相當大的功能，可以對程式編碼做除錯，以及即時的分析，程式執行過程，都可以追蹤。CCS 的程式發展流程如圖 2-3：

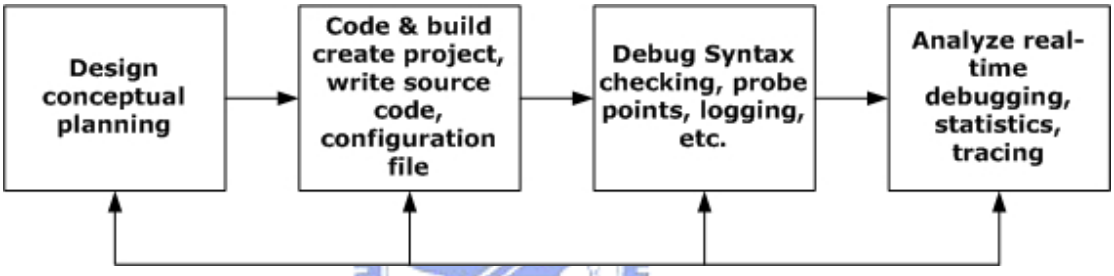


圖 2-3　CCS 程式發展設計流程

資料來源：Code Composer Studio Getting Started Guide Rev. C[6]

在使用 CCS 開發程式時，要先根據開發的系統 ARM 和 DSP 做好系統的配置，如此才能跨平台開發程式，而且 CCS 具有在不同硬體或模擬目標(target)上開發的環境。DSP 程式碼的編寫前，需要先設定 CCS 編程的環境，包括環境記憶體的配置等。CCS 的 Code Generation Tool 包含能將程式碼做最佳化的 C/C++編譯器(Compiler)、組譯器(Assembler)、連結器(Linker)，和功能安排，最後產生一個 DSP 的可執行二元檔(binary code)，整個程式編譯的流程，可以圖 2-4表示：

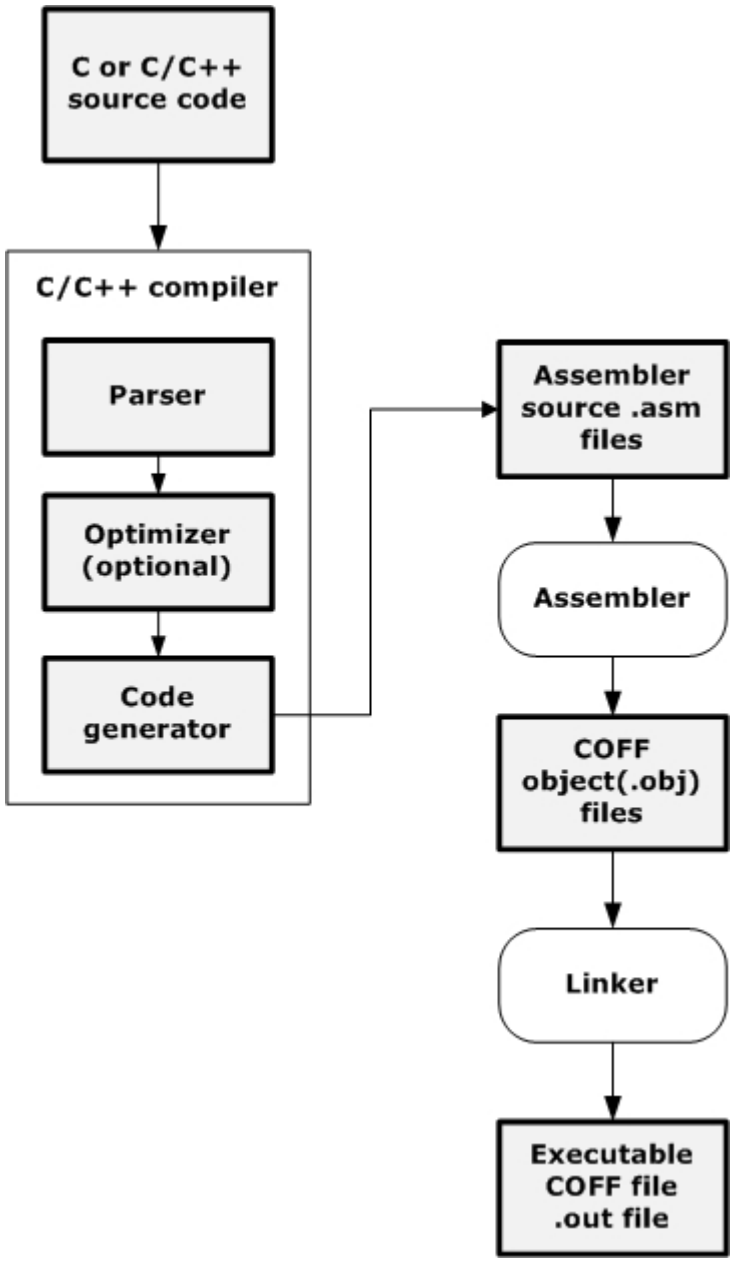


圖 2-4　DSP 程式編譯流程圖

圖 2-4表示，在產生 DSP 的執行檔中，由 C/C++的原始碼經過 C/C++的編譯器(C/C++ compiler)可產生組合語言原碼，再經過組譯器(Assembler)產生物件(Object)檔，經由連結器(Linker)，產生最後的 Executable COFF(Common Object File Format)檔(以.out 為副檔名)。在後面的章節，將會說明有關於 DSP 的動作，都是先將原始碼，經 CCS 編譯之後，產生 COFF 檔，再將 COFF 檔在 Linux 的環境下載入，便具有此 DSP 碼裡面所寫的功能，也就是後面第三章將會說明的 DSP 工作(DSP task)。

CCS IDE 提供圖形化的操作介面。開發一個 DSP 應用程式時，大致上的流程是，首先建立一個專案(Project)，當專案建立時，系統會自動產生 Make 檔。使用都只需要寫 C/C++或組合語言原始檔，加入專案裡面，裡面包含的資料有：

- 原始檔和函式庫的名字
- 編譯器、組譯器和連結器的設定
- 所有相關的檔案(例如：自訂的標頭檔、DSP 的配置檔)

經過圖 2-4的編譯過程，就可以產生所要的 COFF 檔。

2.3.4 CCS 除錯工具

在上一小節中 曾經提到程式開發中，程式除錯的過程，CCS IDE 提供數種不同的除錯工具，讓程式開發人員能對發展的程式除錯，CCS 並非所有的硬體都支援，表 2-4，表示 CCS 所支援的硬體以及除錯工具所提供的功能。其中 OMAP 微處理器是屬於 ARM9 和 C55x(OMAP DSP Core)兩種硬體。

表 2-4　CCS 除錯工具以及支援硬體

Debug Tools	C62x	C64x	C67x	C54x	C55x	C28x	470R1x (ARM7)	470R2x (ARM9)
Breakpoints	V	V	V	V	V	V	V	V
Watch Window	V	V	V	V	V	V	V	V
Probe Points	V	V	V	V	V	V	V	V
Simulator Analysis	V		V	V				
Emulator Analysis (Call Advanced Features for ARM)	V		V	V	V	V	V	V
Event Trigging	V		V				V	V
Graphs	V	V	V	V	V	V	V	V
Symbol Browser	V	V	V	V	V	V	V	V
GEL	V	V	V	V	V	V	V	V
Command Window	V	V	V	V	V	V	V	V
Pin Connect	V			V	V	V	V	V
Port Connect	V		V		V	V	V	V
Data Converter	V		V	V				
EMT Tool							V	V
OS Selector							V	V

資料來源：Code Composer Studio Getting Started Guide Rev. C[6]

2.4 DSP/BIOS

具有強大運算功能的 DSP 處理器，原本就是設計用來製作較為複雜的 DSP 系統。但在系統中，如何有效地利用 DSP 的資源，在一複雜的系統中，如想做到即時多工處理，必定面臨的一個問題。就此一問題，TI 的 DSP/BIOS 發展環境的核心，就提供了一個多執行緒(Multi-thread)的管理系統，讓使用者能在 DSP 上發展出一套嵌入式即時系統(Embedded real-time system)。CCS IDE 裡面 DSP/BIOS 具有即時核心(Real-time kernel)和即時分析(real-time analysis)的特性。可以快速且簡便地設計發展出複雜的應用，且具有除錯的功能。這一小節，將說明 TI 所提供的即時核心：DSP/BIOS 核心，以及其多執行緒(multi-thread)內部的定義。下個章節的 DSP Gateway 的 DSP 應用部分便是使用 DSP/BIOS 做為其主要程式的核心所開發出來的 APIs。

2.4.1 DSP/BIOS 核心

DSP/BIOS 核心是一個即時核心，主要被設計出來針對某些應用層面，其中包含：即時排程(Real-Time scheduling)、同步(Synchronization)，以及 HOST 和 TARGET 之間的通訊(Communication)。DSP/BIOS 核心提供了(1)強制性(Preemptive)的多執行緒(Multi-Thread)，也就是在 DSP 中有許多工作(Task)時，高優先權的工作可以將正在執行的低優先權的工作的中斷，並且搶得執行權。(2)硬體抽象化(Hardware Abstraction)，系統核心和硬體間的硬體處理層，系統的硬體中斷完全由硬體抽象層所控制。使用者只要針對硬體抽象層給予的硬體描述，便可以控制硬體，(3)即時分析(Real-Time Analysis)的功能。DSP/BIOS 核心模組是可以連結於一應用程式內。除此之外，DSP/BIOS Configuration 工具可將程式的大小，最佳化處理。DSP/BIOS 還能探查追蹤一些有關於執行時，DSP 內部的資料，並且即時的呈現出來，圖 2-5中表示了 DSP/BIOS 在 CCS 上所扮演的角色。

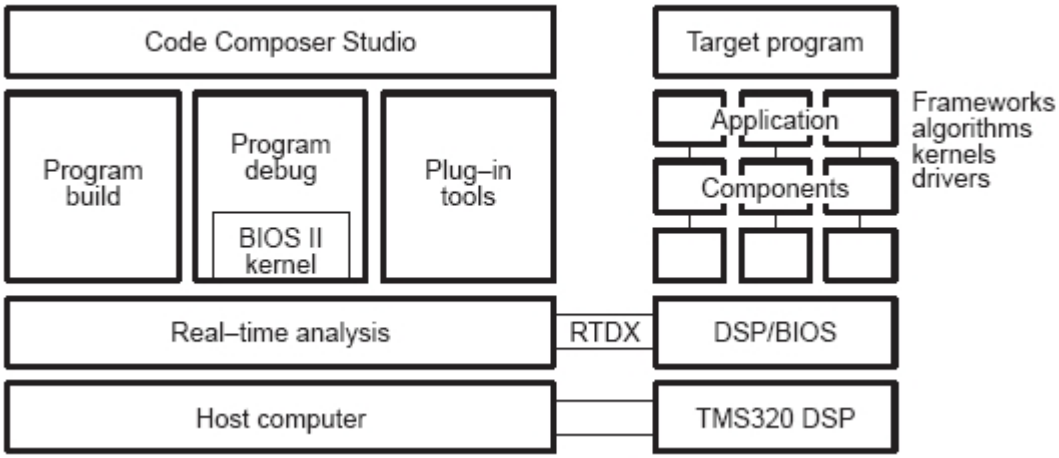


圖 2-5　CCS 整合發展環境

資料來源：Code Composer Studio Getting Started Guide Rev. C[6]

DSP 的應用程式使用 DSP/BIOS 核心，DSP/BIOS 核心是一個事件驅動(Event-Driven)的架構，透過 DSP/BIOS Configure 對於 DSP/BIOS 內的機制作設定，DSP/BIOS 系統配置工具是用來幫助開發者在撰寫事件處理函式時，設定事件的特性與參數，如 HWI、SWI、TSK、SEM、TIMER 等來使用 DSP/BIOS 的多執行緒能力。

2.4.2 DSP/BIOS 系統配置工具(Configuration Tool)

DSP/BIOS 系統配置工具是一圖形介面式編輯工具，用來輔助使用者設定一個即時作業核心以及各項參數。程式設計者可以由系統內建的 DSP/BIOS 系統配置樣本(Template)檔中選擇一個編輯，如圖 2-6。隨後在第三章的 ARM 與 DSP 通訊所使用的 DSP Gateway APIs，因為是以 DSP/BIOS 為核心，故亦需要對 DSP/BIOS 參數設定。

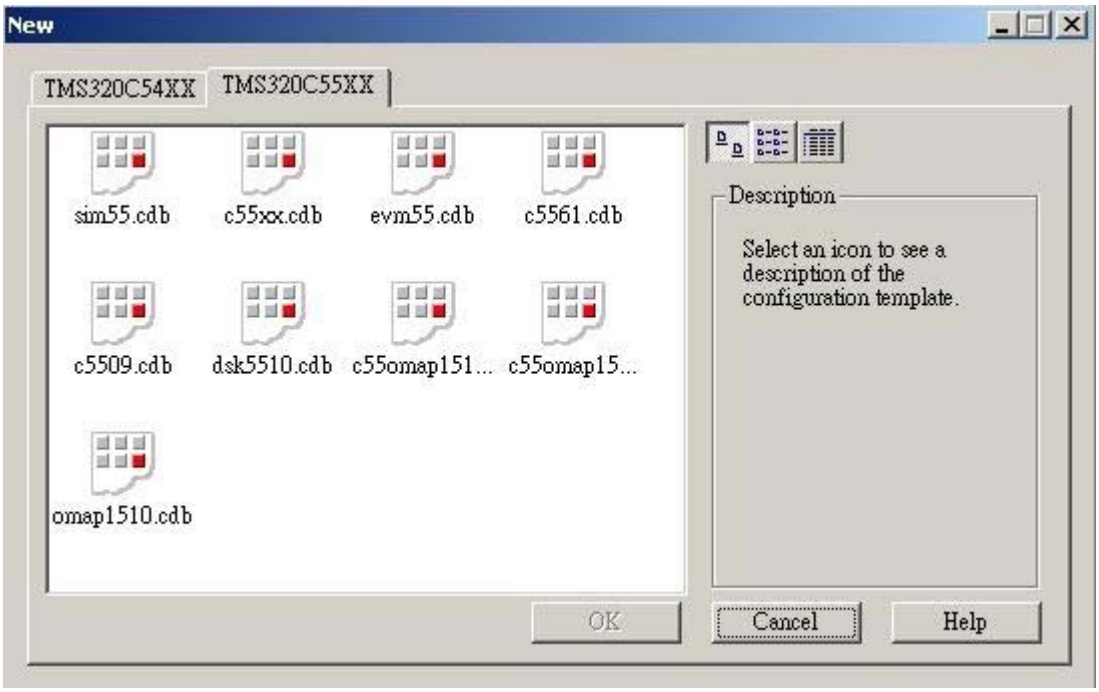


圖 2-6　DSP/BIOS 系統配置工具

2.4.3 DSP/BIOS 的執行緒(Thread)

DSP/BIOS 核心實行了執行時(Run-Time)由 DSP/BIOS APIs 所建構的應用程式。圖 2-7表示，DSP/BIOS 執行緒的關係，執行緒之間是各自獨立的 DSP 的命令，可以是中斷要求(ISR，Interrupt Service Request)、副函式(Subroutine)或者是一個函式呼叫(Function Call)。例如，當一個硬體發出中斷，就是一個執行緒，這時會做出一個 ISR。

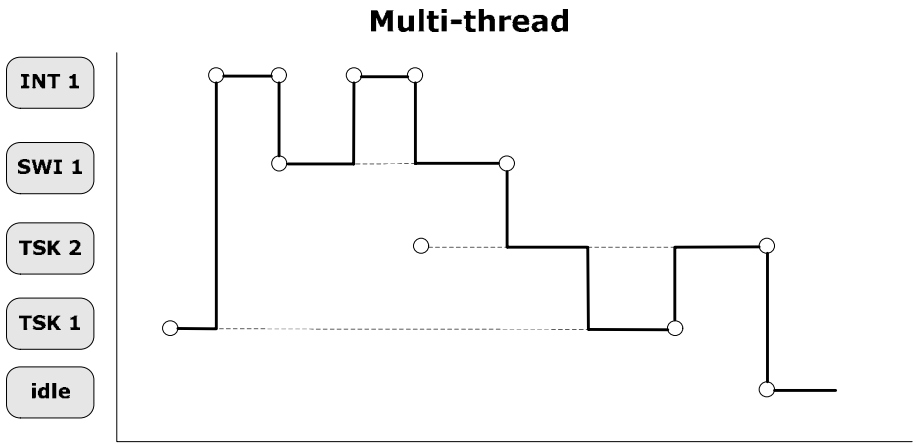


圖 2-7　DSP/BIOS 執行緒表示

資料來源：TMS320 DSP/BIOS User's Guide Rev B[4]

單一處理器系統經由優先權(Priority)的安排，高優先權的執行緒，比低優先權的先處理而達到多執行緒的應用。DSP/BIOS 提供了 30 個層級的權先權順序並分成四個明確的類型，圖 2-8 表示。依優先順序可分成：

(1) 硬體中斷(Hardware interrupt，HWI)：在 DSP 的環境下產生外部非同步的事件觸發的回應。HWI 函式(或稱為一個 ISR，interrupt service routine)是在硬體發生中斷之後在一定的時間內執行的一個 Critical Task。硬體中斷具有最高的優先權。

(2) 軟體中斷(Software interrupt，SWI)：提供硬體中斷與一般執行緒間另一種優先順序的選擇，與硬體中斷一樣，必須將中斷函式執行完畢才會將執行權釋出。通常軟體中斷，排程會在 100 微秒(microseconds)或更久，才會執行。

(3) 一般工作(task，TSK)：一般工作可以暫停(Suspend)，等待一些必要資源齊全後再繼續執行，DSP/BIOS 亦提供特殊結構，以利於一般工作間的通訊(inter-communication)及同步功能，這些結構包括：Atomic Queue，QUE 模組、旗號(Semaphores，SEM 模組)及 Mailbox，MBX 模組。優先權較 Background thread 來的高。

(4) 背景執行緒(Background thread)：執行一個不工作的迴圈(IDle Loop，IDL)，在 DSP/BIOS 內是最低優先權。

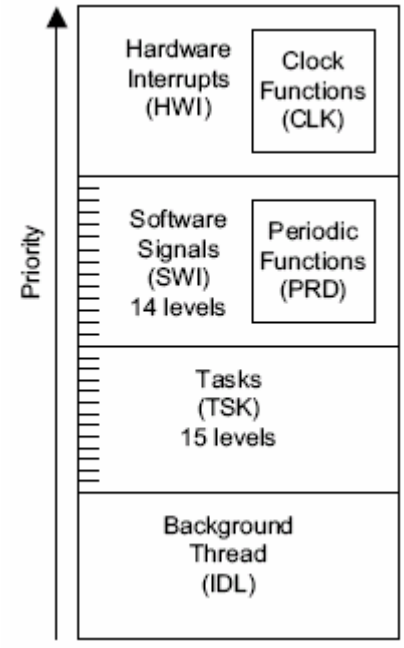


圖 2-8　DSP/BIOS 各種執行緒之優先順序

資料來源：TMS320 DSP/BIOS User's Guide Rev B[4]

表 2-5是針對四種類型的特性做比較，表 2-6 則是強制性的比較說明。圖 2-9以表 2-6為參表，針對不同類型做一個時序上的呈現表示。

表 2-5　執行緒的特性比較

特性	HWI	SWI	TSK	IDL
優先權(priority)	最高	次高	次低	最低
優先權層級的數目	和 DSP 有關	15，工作排程器是最低的層級	16(包含1的IDL)	1.
能否被暫停 Can yield and pend	不能，一旦執行時，要完整結束，才交出執行權	不能，一旦執行時，要完整結束，才交出執行權	能	必定不能；會阻礙 PC 收到目標的資訊
執行狀態 Execution states	不執行，準備，執行中	不執行，準備，執行中	準備，執行中，推延、結束	準備、執行
排程如何停止	HWI_disable	SWI_dspable	TSK_disable	Program exit
如何輪詢或準備執行	中斷產生	SWI_post, SWI_andn, SWI_dec,SWI_INC, SWI_or	TSK_create	當進入主程式時，只要沒有其他程式在執行
使用何種堆疊 Stack used	系統堆疊 System stack (1 per program)	系統堆疊 System stack (1 per program)	工作堆疊 Task stack (1 per task)	預設工作堆疊 Task stack used by default
本文儲存 Context saved	客製化 customizable	將暫存器存在系統堆疊	將全部的內容存在工作堆疊	-Not Applicable-
執行緒資料共享	資料流 Stream，佇列 queue，pipes，全域變數 global variables	資料流 Stream，佇列 queue，pipes，全域變數 global variables	資料流 Stream，佇列 queue，pipes，locks、mailbox，全域變數 global variables	資料流 Stream，佇列 queue，pipes，全域變數 global variables
執行緒同步	-Not Applicable-	SWI mailbox	Semaphores，mailbox	-Not Applicable-
Dynamic creation	Yes	Yes	Yes	No
動態改變優先權	No	Yes	Yes	Yes

1：結束一般工作管理對談通訊，IDL 執行緒才會去使用系統堆疊。

資料來源：TMS320 DSP/BIOS User's Guide Rev B[4]

表 2-6　執行緒的相互強制性(Thread Preemption)

執行緒發佈(Thread Posted)	執行緒執行(Thread running)			
	硬體中斷	軟體中斷	一般工作	不工作迴圈
開始硬體中斷	可強制執行	可強制執行	可強制執行	可強制執行
結束硬體中斷	等待回應	等待回應	等待回應	等待回應
開始高優先權的軟體中斷	-----	可強制執行	可強制執行	可強制執行
結束軟體中斷	等待	等待回應	等待回應	等待回應
低優先權的軟體中斷	等待	等待	-----	-----
開始高優先權的一般工作	-----	-----	可強制執行	可強制執行
結束一般工作	等待	等待	等待回應	等待回應
低優先權的一般工作	等待	等待	等待	-----

資料來源：TMS320 DSP/BIOS User's Guide Rev B[4]

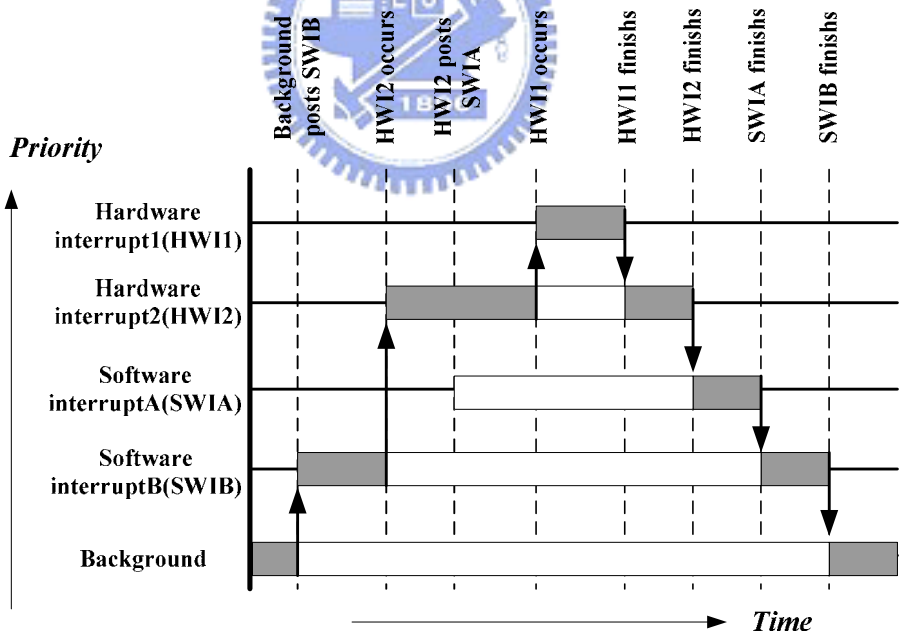


圖 2-9　Preemption Scenario

資料來源：TMS320 DSP/BIOS User's Guide Rev B[4]

三、 OMAP 內部通訊架構及溝通方式

OMAP1510 微處理器結合 ARM925T 和 TMS320C55x DSP 處理器，但是要如何將這兩個處理器的功能相互結合應用，是一個問題所在，DSP Gateway 提供了一個解決的管道。DSP Gateway 是一個機制，能使 OMAP1510 微處理器能在 ARM 和 DSP 之間溝通，使得系統能同時使用 ARM 以及 DSP 的資源。如此一來能在 ARM 的架構上建立一個 Linux 作業系統，同時又可以使用 DSP 的運算能力。本章將針對『DSP Gateway 架構』、『DSP Gateway Linux API』、『DSP Program』等部分，將 ARM 和 DSP 溝通的機制一步一步的說明。

3.1 DSP Gateway 簡介

ARMLinux 是一個相當受歡迎的嵌入式作業系統。從網路上，可以找到相當多有關於 ARMLinux 的資料。然而，為了使 DSP 能在 ARMLinux 架構下使用，對於整個系統必須做個小變動，讓程式設計人員使用 ARM 的同時，亦可以使用 DSP。

DSPGateway 是一個機制，能幫助程式設計人員在設計應用程式時可同時使用 ARM 和 DSP 部分。DSPGateway 主體包含了兩個部分，一個部分是在 ARM 上的 Linux 的裝置驅動程式：DSP 核心韌體，另一部分為 ARM 和 DSP 通訊的應用程式介面(API)。ARM 和 DSP 的通訊，驅動程式提供了常見的 Linux 裝置檔案操作(File Operations)介面，程式設計人員欲使用 DSP 時，在 ARM 部分，可以透過一般的系統呼叫(System Calls)，像是 read 和 write 與 Linux 根目錄下 dev 目錄內的 DSP 工作裝置檔案(Task Device File)進行溝通。DSP 部分，可使用 DSPgateway 所提供的 APIs 和 DSP/BIOS 的 APIs 與 ARM 做同步的程序處理。圖 3-1為 ARM 和 DSP 溝通的狀態表示：

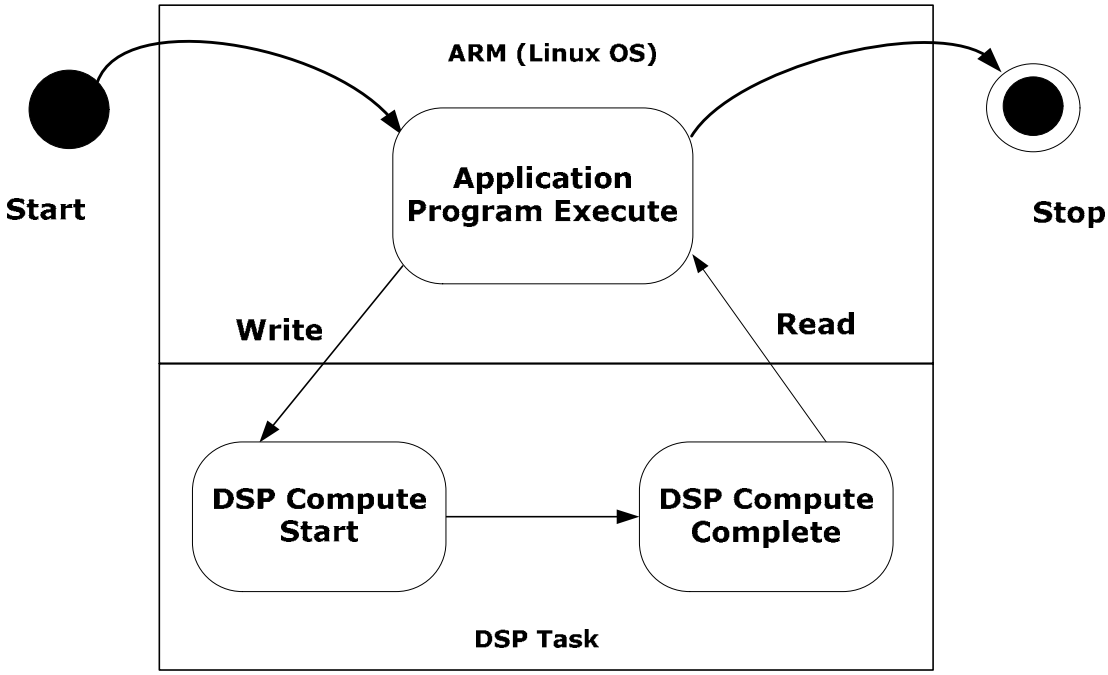


圖 3-1　ARM 和 DSP 呼叫狀態圖

在程式的開發設計功能上，ARM 的部分，可以用於設計使用者介面，系統資源管理等方面的工作。在 DSP 部分，透過 DSP Gateway 的機制將應用程式中複雜的計算交予 DSP 核心計算，善用 DSP 的計算能力，達到最佳的處理效能。必需要注意的是，在使用 DSP Gateway 做 ARM 和 DSP 溝通時，從 ARM 或 DSP 傳輸及接收資料的形式大小、傳輸方式，都需要被明確的定義，在這一部分在本章節中將會逐一探討說明。

3.2 DSP Gateway 架構 (DSP Gateway architecture)

DSP Gateway 主要可分成 DSP Gateway 架構和程式編寫的方式，兩個部分來探討之。其中包括『DSP Driver 與 Linux API』、『Mailbox 傳輸命令機制』與『DSP Gateway BIOS(tokliBIOS)與 DSP APIs』。DSP Driver 與 Linux APIs 包含 ARM 的環境下提供與 DSP 溝通操作的方式及介面。Mailbox 機制，將 ARM 與 DSP 之間操作動作所傳送的 Mailbox 命令做解譯，並且傳送資訊。DSP Gateway BIOS 與 DSP APIs，主要是與 DSP 溝通和 DSP 系統管理機制有關，建構於 TI 的 DSP/BIOS 提供的即時多工的核心和 APIs 所建構而成的。圖 3-2可用來表示 DSP Gateway 整個架構的關係圖。

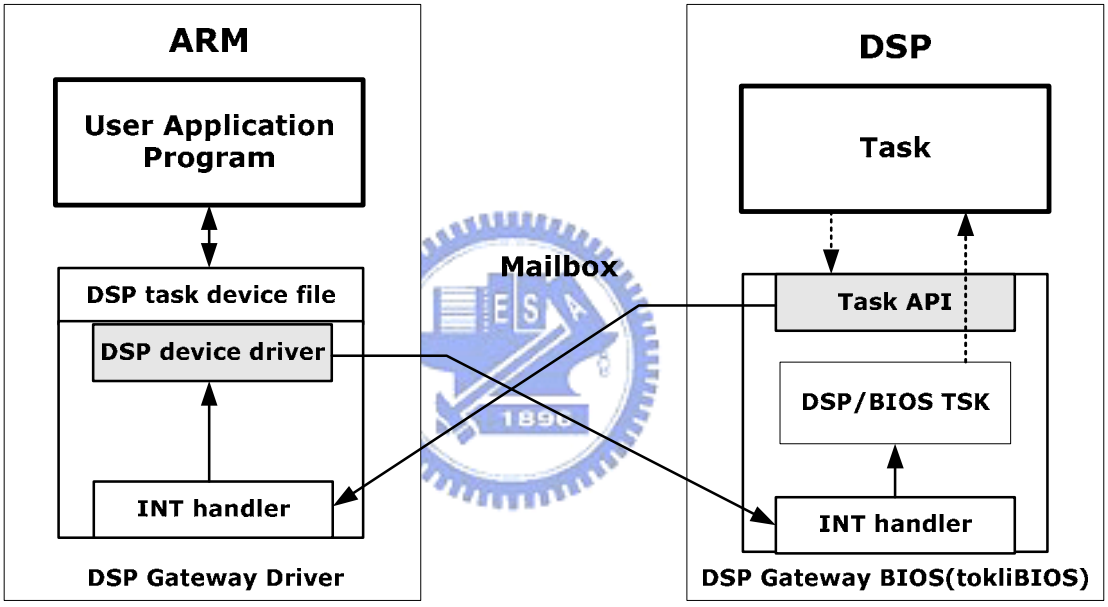


圖 3-2 DSP Gateway 系統架構圖

在之後每小節，將一步一步說明圖 3-2的整個溝通架構和溝通流程方式。首先述說，DSP Gateway 架構的基礎，此部分包含 DSP Gateway BIOS 和 DSP Gateway Driver 和 Mailbox 三部分，介紹 DSP Gateway 的整體概念。其次再進入系統程式及應用程式的設計的部分，將分成 Linux API 與 DSP program(DSP Task)二個部分說明。

3.2.1 DSP Gateway BIOS

DSP Gateway BIOS 是以 DSP/BIOS 為主體的處理介面，圖 3-2右側 DSP Gateway BIOS 繼承了 DSP/BIOS 即時多工核心所提供的 APIs，並且會將『系統初始化』，還有處理『Mailbox 中斷』的工作。圖 3-3表示 DSP Gateway BIOS 組成的架構，DSP Gateway BIOS 配置 DSP/BIOS 的動作。

初始化動作，主要包含：

- 配置 DSP 1-cache，其形式為 2-way set associative cache。
- 啟動 mailbox 中斷以及 watchdog timer 中斷
- 設定 Timer1 每 10ms 中斷一次
- 初始化 Timer Queue
- 初始化所有使用者的 tasks 和 supertask

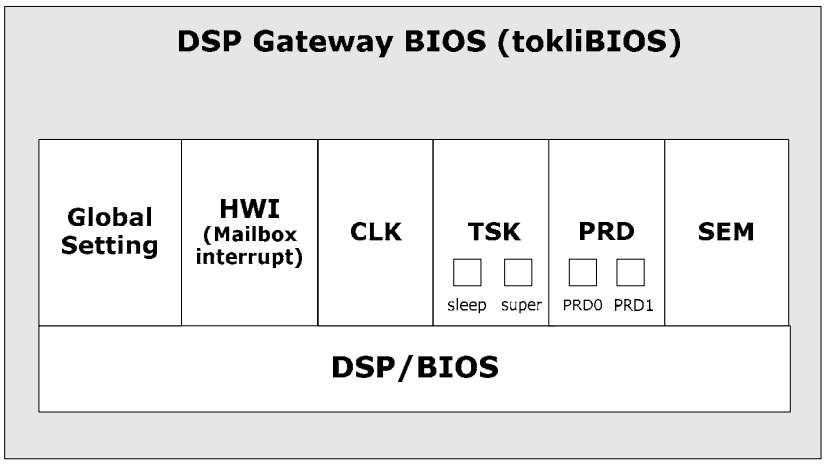


圖 3-3　DSP Gateway 與 DSP/BIOS 關係圖

DSP Gateway 對於 DSP/BIOS 的配置功能，可以歸納如下：

- 系統—整體設定
- 硬體中斷—HWI_INT5 (Mailbox interrupt)
- 硬體中斷—HWI_INT13 (Watchdog Timer interrupt)
- 時序函式(Clock function, CLK)
- 週期函式(Periodic function, PRD)—PRD0(每 10 ms 執行一次)
- 週期函式(Periodic function, PRD)—PRD1(每 1s 執行一次)
- 工作—TSK_sleep(優先權最低的工作函式)

- 工作—TSK_super(優先權最高的工作函式)
- 旗號—SEM_super

3.2.2 DSP Gateway Driver

在 Linux 的核心中，每個裝置的操作都有其檔案操作的結構，而系統則提供一個共同的系統呼叫介面(System Call Interface)給使用者空間(User Space)的應用程式來對裝置進行存取動作。作業系統核心所扮演的必需是根據不同的裝置，將系統呼叫介面與驅動程式的檔案操作進行連結對應，如圖 3-4：

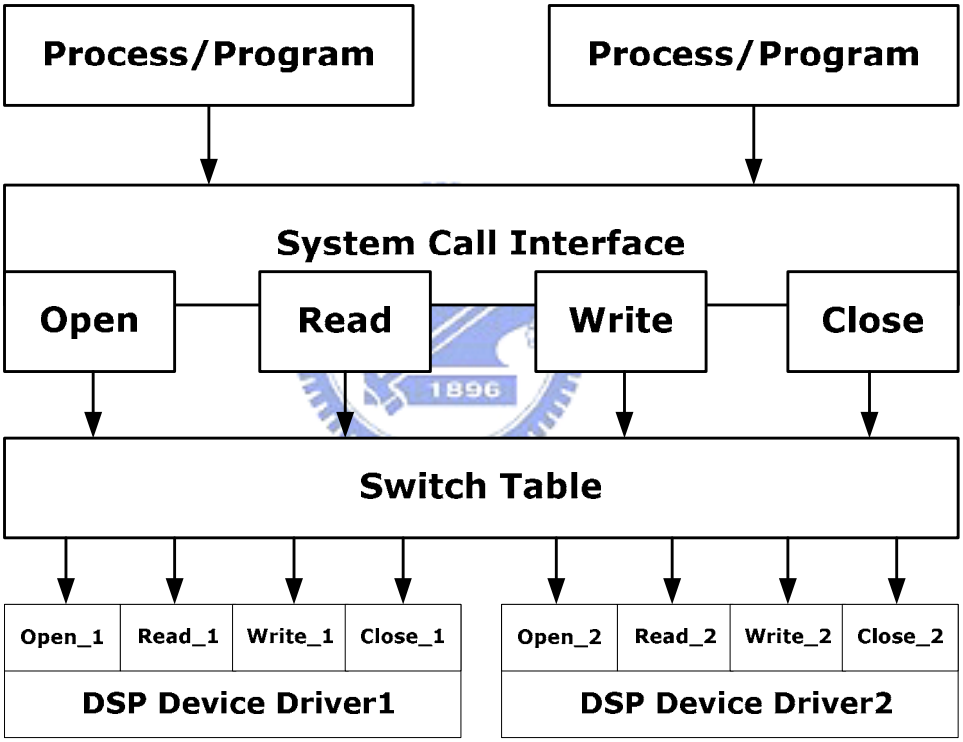


圖 3-4　DSP 驅動程式檔案操作介面

圖 3-4可用來表示 DSP Gateway Driver 內部的基本架構，其中 DSP Device，是 DSP Gateway driver 使用 devfs 新的裝置檔案系統產生的，可以藉由存取 DSP 核心資源的裝置檔案，來傳送資料和命令給 DSP 核心，這些裝置檔案包括有：DSP task device、DSP task watch device、DSP control device、DSP watchdog device 及 DSP exmem device 五種裝置。DSP Device Driver 內亦包括檔案操作(file operations)可以用於存取 DSP 裝置。關於 DSP 的裝置檔案，在後面的小節，將會詳細說明。

3.2.3 Mailbox

圖 3-2中間部分為，在 OMAP1510 處理器內，ARM 和 DSP 處理器內部通訊是使用了 DSP Gateway 的 Mailbox 的機制。Mailbox 有三個暫存區，一個是用於微處理器單元(MPU，Micro Processor Unit)送資訊和分給一個中斷(INT5)給 DSP。其於兩個是 DSP 送中斷要求(IRQ10/IRQ11)給 MPU 要回送資料，通常用到 IRQ10。每個暫存器包括是兩個 16-bit 暫存器和 1-bit 旗標暫存器(flag register)。圖 3-5為 ARM-和 DSP 通訊的 Mailbox 機制，圖 3-6為暫存器的 bit 定義表示，表 3-1為 Mailbox 命令的定義。

圖 3-5整個通訊的流程，可以分成兩點來表示：

(1) MPU 送一個訊息和發佈一個中斷(INT5：Interrupt 5)給 DSP。

(2) DSP 回送一個訊息和發佈一個中斷(IRQ10/11)給 ARM。需要注意的是，DSP 傳訊號到 ARM，同時，僅能有一個 mailbox 能被使用。

INT5 的中斷服務函式(ISR)，Mailbox 中斷所做的工作是讀取命令和將 ARM 端傳送的 mailbox 命令做註冊。一般而言，分成兩個中斷情況，一種是一般工作，另一種是系統工作。假如接收到要給使用者工作(user task)命令。它會被註冊在目標工作中且增加 task 中 MBQ(Mailbox Queue)的旗號(semaphore)。然後，這個命令將會被註冊到將要被處理的每個 task 的 MBQ。如果這個命令是對於系統工作的，它將會被註冊到 supertask 的 MBQ。交由 supertask 來處理這些命令。假如接收到的命令需要用到 Global IPBUF(例如：BKSND)，除了要對 MBQ 做註冊外。Global IPBUF balancing process 在這個函式中亦要被執行。

圖 3-6說明了 Mailbox command 的封包格式，每個 Mailbox 命令都是以這個結構傳遞訊息，對照表 3-1，可以了解清楚知道到每個封包分部和在不同的數值下，所表示的動作涵義。

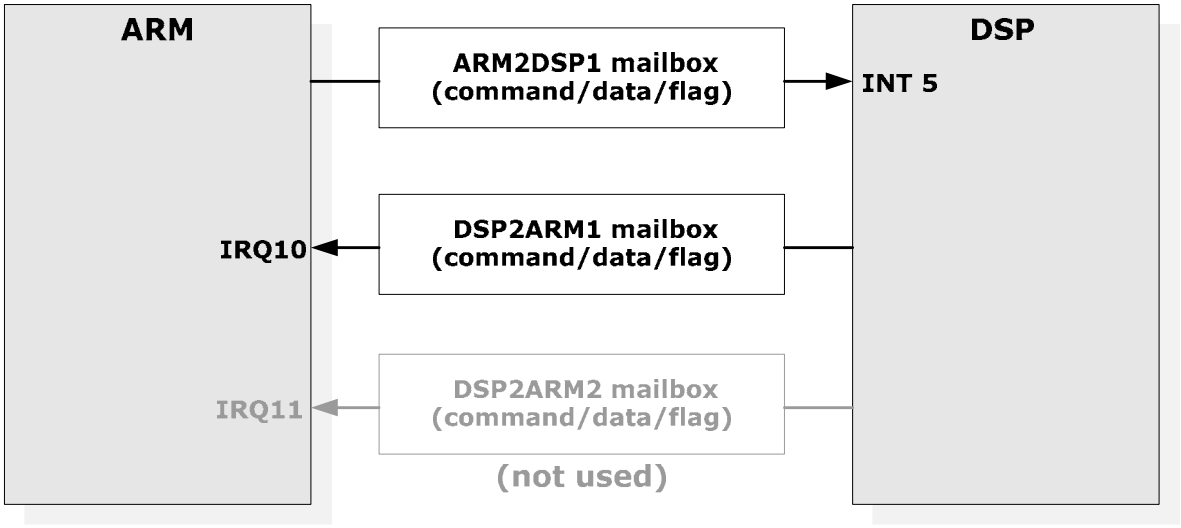


圖 3-5　Mailbox 機制

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

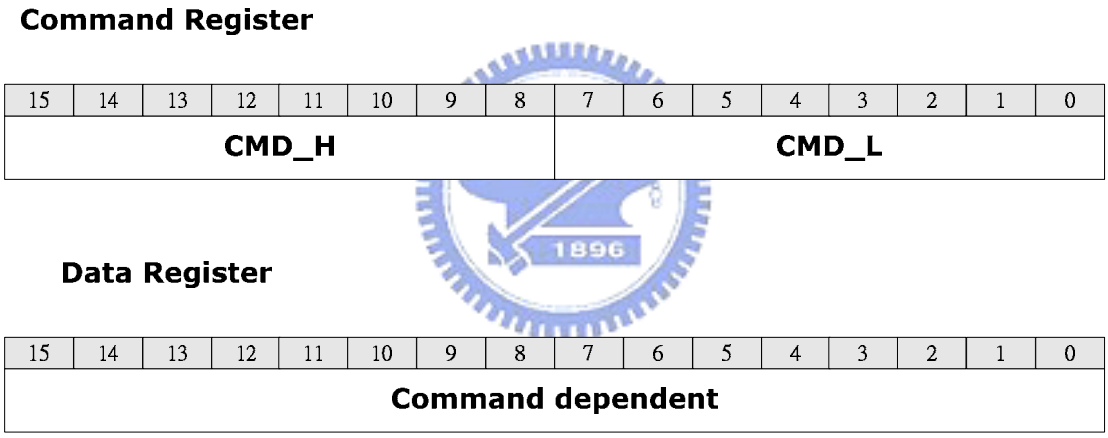


圖 3-6　Mailbox 暫存器配置

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

表 3-1　Mailbox 命令定義

Command	CMD_H	CMD_L	Data register	Short Description
WDSND	0x10	TID	word data to send	Word Send
WDREQ	0x11	TID	-	Word Request
BKSND	0x20	TID	BID	Block Send
BKREQ	0x21	TID	requesting count	Block Request
BKYLD	0x23	-	BID	Block Yield
BKSNDP	0x24	TID	-	Block Send Private
BKREQP	0x25	TID	-	Block Request Private
TCTL	0x30	TID	-	Task Control
WDT	0xd0	-	notification value	Watchdog Timer Notification
TCFG	0xe0	TID	configuration data	Task Configuration
TADD	0xe2	TID	BID	Task Add
TDEL	0xe3	TID	-	Task Delete
TSTOP	0xe5	-	-	Task Stop
DSPCFG	0xf0	CFGTYP	configuration data	System Configuration
REGRW	0xf2	TYPE	address/data	Register Read/Write
GETVAR	0xf4	VARID	data	Get Variable
SETVAR	0xf5	VARID	data	Set Variable
ERR	0xf8	EID	command caused the error	Error Information
DBG	0xf9	TID	-	Debug

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

3.3 DSP Gateway 傳輸協定概念(DSP Gateway Protocol concept)

前面，說明了 DSP Gateway 架構組成的三個部分之後，知道了 Mailbox 可以傳遞命令外，另外在整個資料傳輸過程中，還需要傳輸資料的方式。

DSP 處理器，利用給予不同的 TID(Task ID)來區分不同的工作，達到處理多工(Multi-Tasks)的效果。由表 3-1 可以知道在傳送命令時包含 TID(包含在 Command Low 的部分)，如此 MPU 便能處理混合 TID 的資料。處理器內部 ARM 和 DSP 資料傳輸是透過內部 buffer(IPBUF，InterProcessor Buffers)。利用 IPBUF 做處理器內部 ARM 和 DSP 之間區塊(block)資料傳輸的 buffter 時，IPBUF 必須先將 IDs(BID)先定義出來，且 IPBUF 不同於一般的區塊，僅有處理器內部，才可以使用。當使用 IPBUF 傳輸資料時，IPBUF 的所有權亦會隨著資料轉移，跟著轉移至接收者。IPBUF 的 BID 資料亦被定義於 Mailbox 中資料暫存器內，會隨著 Mailbox 傳到 DSP，然後 DSP 便知道要到那一個 IPBUF 去取得，所需要的處理資料。圖 3-7表示，Mailbox 和 IPBUF 在 ARM 和 DSP 之間的關係。

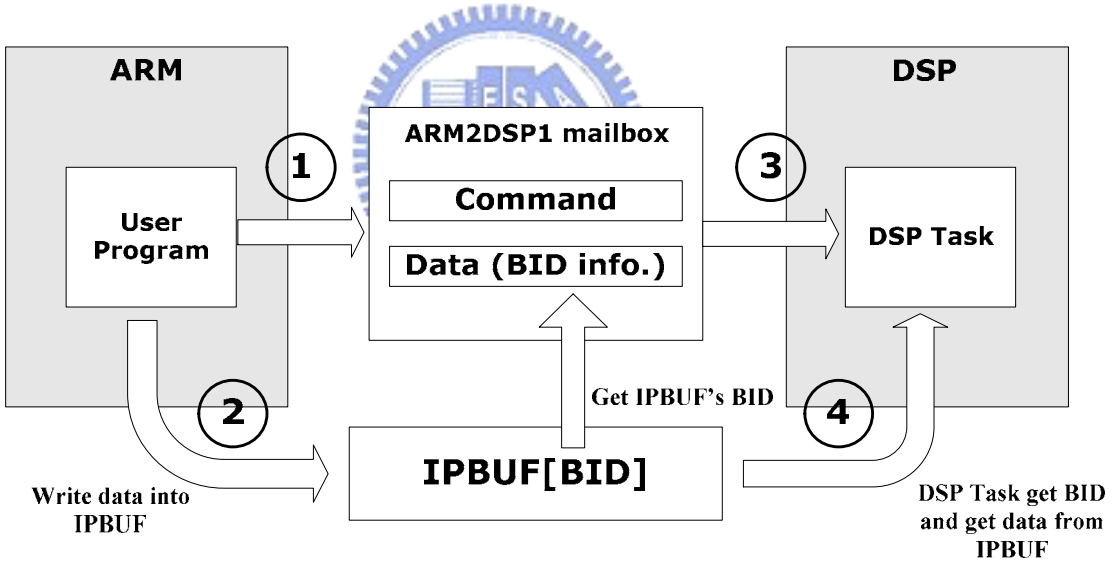


圖 3-7　ARM 與 DSP 命令及 IPBUF 傳輸

圖 3-7中表示了資料傳輸之間的關係外，還可以看出其流程，ARM 端的 user program 要將資料交給 DSP task 處理時，也要將資料傳給 DSP 端。傳輸過程中，一開始要傳送 Mailbox command 和 data 的資訊(BID)給 DSP，同時將要傳送處理的資料寫入 IPBUF，這時在傳送的 data 資訊中將知道 IPBUF 的 BID，並且和命令一起傳送給 DSP task，這時 DSP task 便會知道要去那一個 BID 取得資料來進行處理。

3.3.1 DSP Gateway 傳輸 buffer

前面提到，處理器內部的資料是由 IPBUF 來傳輸。而 IPBUF 分成兩種，一種是 Global IPBUF，另一種是 Private IPBUF。兩種 IPBUF 各有不同的特性，接下來，將說明這兩個 IPBUF 的特性，以及操作的方式。

(1) Global IPBUF

Global IPBUFs 是用於 ARM 和 DSP 之間區塊資料的傳輸時，以 BIDs 來區分不同的 IPBUF。是所有的行程都可以使用 IPBUF，並沒有特殊的限制，IPBUF 的最大為 64kwords(128kbyte)。IPBUF 的格式如圖 3-9：

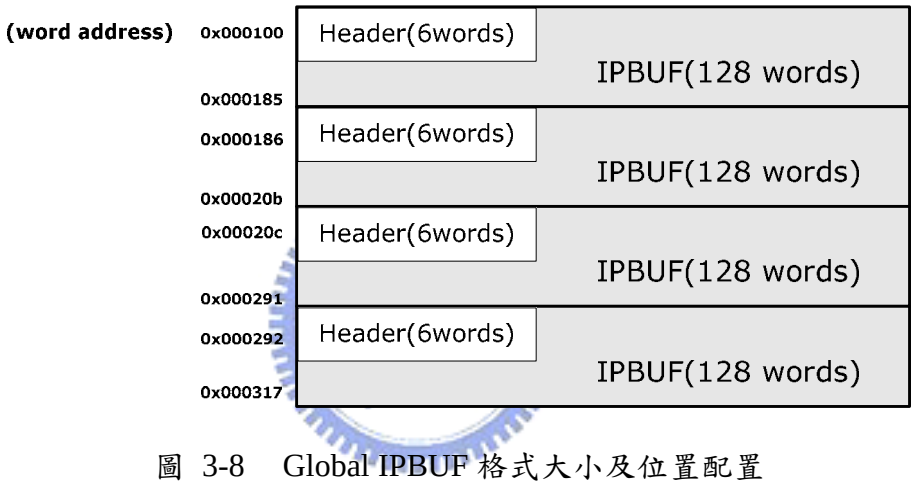


圖 3-8 Global IPBUF 格式大小及位置配置

資料來源：Linux DSP Gateway Specification Rev 2.0[1]

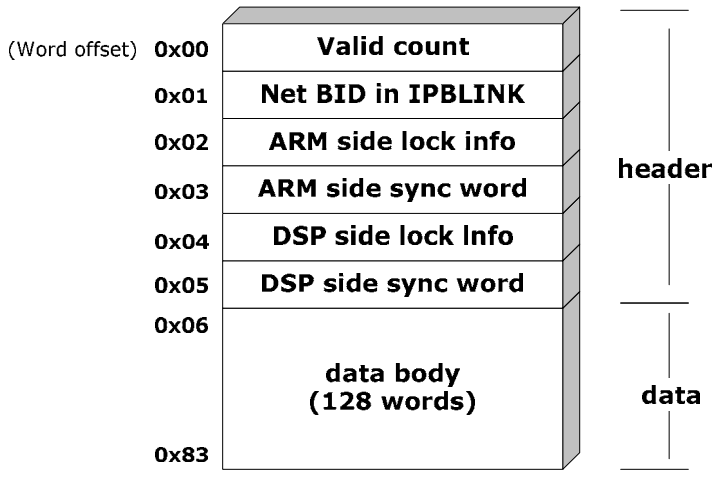


圖 3-9 Global IPBUF 結構圖

資料來源：Linux DSP Gateway Specification Rev2.0[1]

表 3-2　Global IPBUF 結構定義

Structure	*Data member*	*Data type*	*Description*
ipbuf	c	unsigned short	ipbuf 有效的 word 數
	next	unsigned short	下一個 IPBLINK 的進入點
	la	unsigned short	當 ARM 在使用時，鎖住 ipbuf 的所有權
	sa	unsigned short	當 DSP 在使用時，鎖住 ipbuf 的所有權
	ld	unsigned short	確保在 ipbuf 通訊時，接收者必需等待資料完全寫入 IPBUF 之後才可以進行讀取的工作
	sd	unsigned short	
	d[0]	unsigned short	傳送資料

資料來源：Linux DSP Gateway Specification Rev2.0[1]

圖 3-9中每個 IPBUF 有 6 個 words 標頭(header)，裡面包括，valid count、next BID in IPBLINK、ARM 和 DSP 的 side lock info 和 side sync word，最後才是要傳輸的主體資料，表 3-2 表示 IPBUF 內部結構的定義。其中 side lock info，表示，當 ipbuf 被 ARM 或 DSP 使用時，此 IPBUF 的所有權將會被暫時的鎖住，直到此 IPBUF 的所有權被 ARM 或 DSP 釋放出來，才可以再使用。Sync word 則表示，當 ARM 和 DSP 利用 IPBUF 做溝通時，用來確保寫完之後讀入(read-after-write)的機制，動作是當某一方需要另一方的資料時，它將會一直等待到另一方資料將資料完全寫入 IPBUF 之後才進行讀取的動作。圖 3-10表示 Sync word 的工作流程。Global IPBUF 是由 IPBLINK 所管理的，這個機制是用於 free pool 中連接 ipbuf 的 link，它會試著讓 ARM 和 DSP 處理器，維持所持有的 buffer(IPBUF)平衡，如圖 3-11所示。

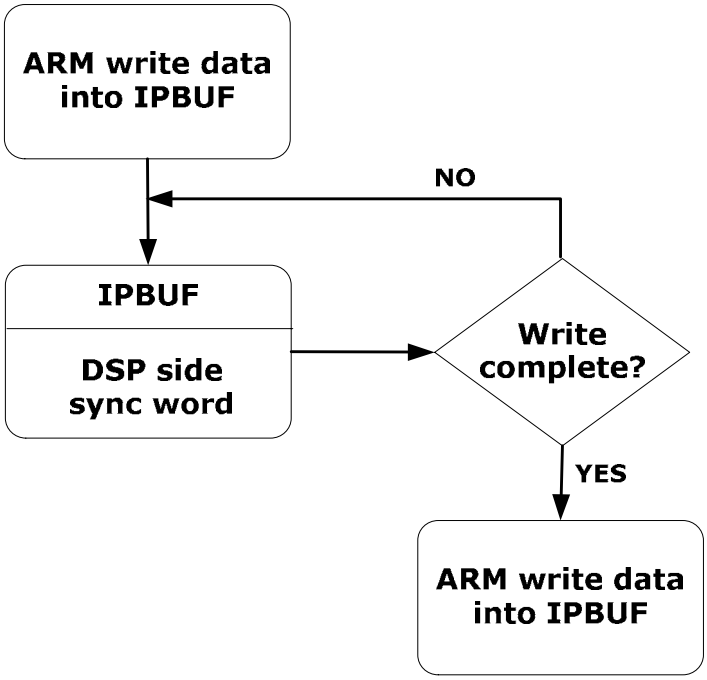


圖 3-10 Sync word 工作流程

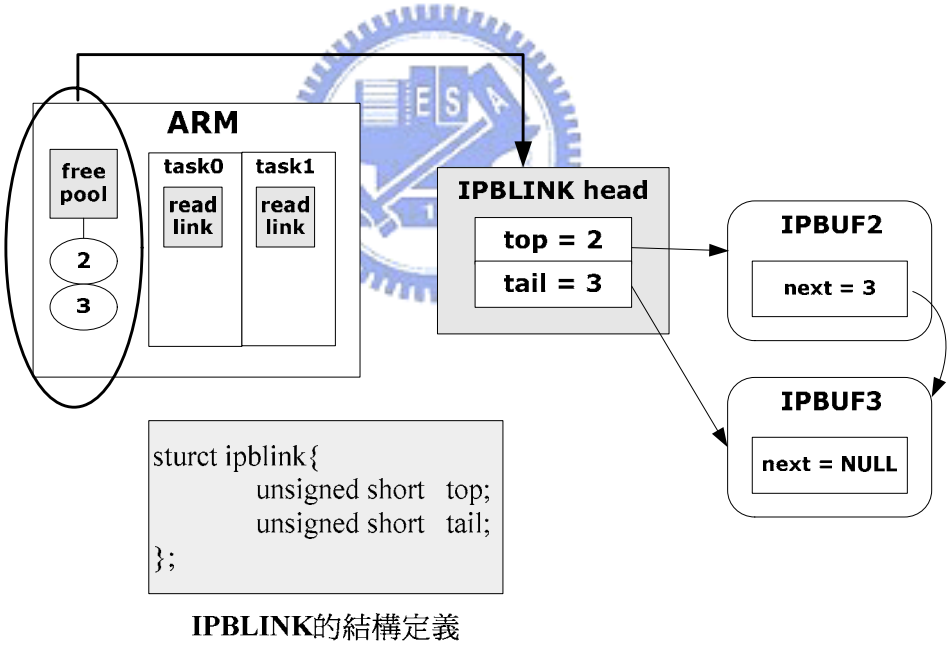
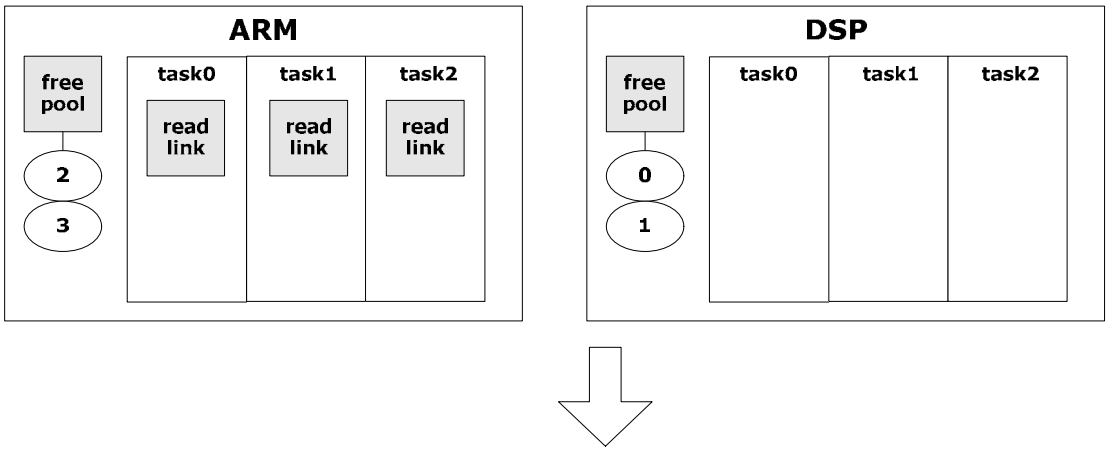


圖 3-11 IPBLINK structure

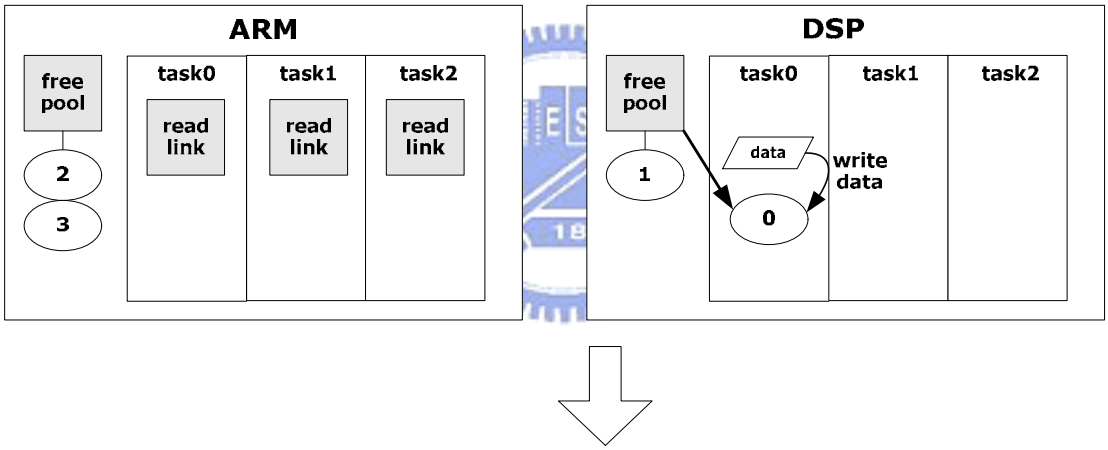
資料來源：Linux DSP Gateway Specification Rev2.0[1]

圖 3-12為一個 ARM 和 DSP 資料傳輸的例子，其中包含 buffer 所有權的轉移的流程：

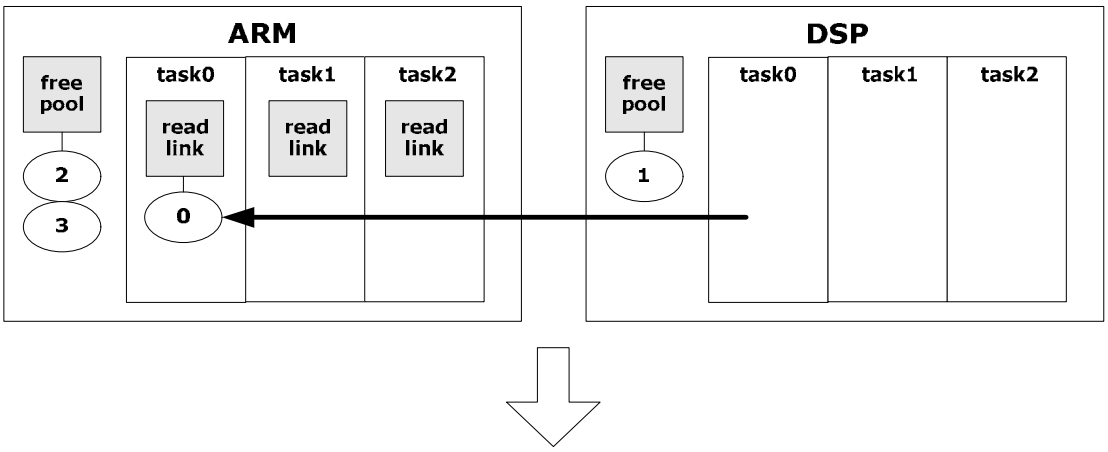
(1)初始化狀態



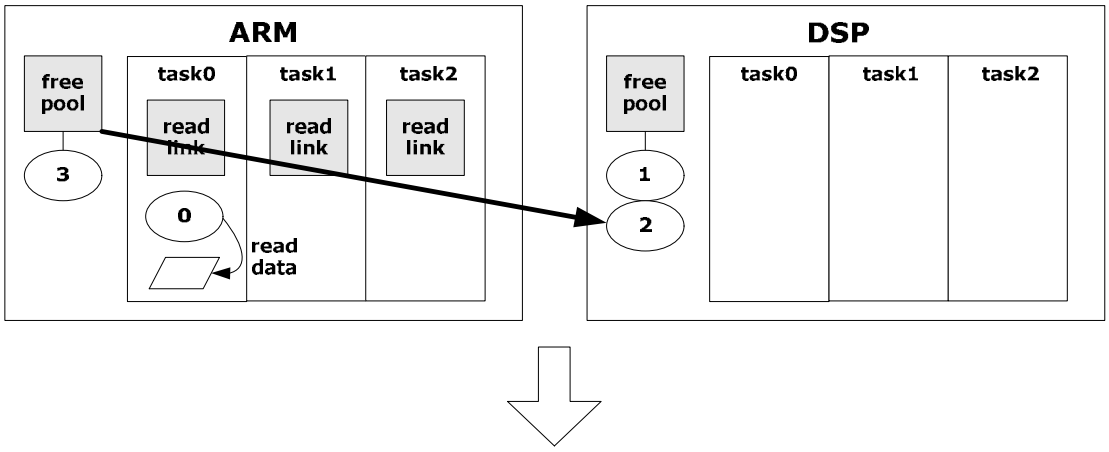
(2)DSP 的 task0 從 pool 中取 buffer0，並且寫入資料



(3)DSP 送 buffer0 的資料給 ARM，這時 ARM Task0 的 read link 與 buffer0 連結



(4)ARM 從 buffer0 讀取資料，並且讓出 buffer2 給 DSP



(5)buffer0 回到 free pool 中，此時又回復到平衡狀態

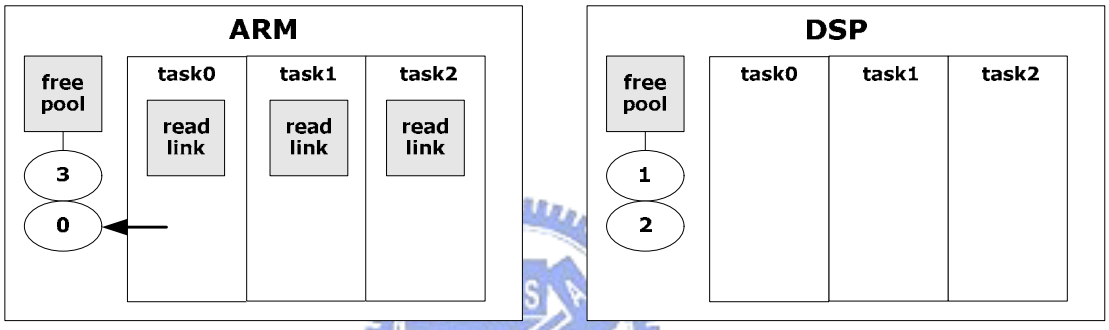


圖 3-12 IPBUF 轉移示意圖

資料來源：Linux DSP Gateway Specification Rev2.0[1]

(2) Private IPBUF

除了 Global IPBUF 外，一般工作也可以擁有 Private IPBUF。Private IPBUF 不同於 Global IPBUF 的架構。IPBUF 資料的主體位置在每次傳資料傳輸時都會改變，且只有執行中的『接收工作』可以有 Private IPBUF 從 ARM 端接收資料。資料從 ARM 送給 DSP 之前，必需取得資料 buffer(IPBUF)的位置才可以接收。Private IPBUF 的格式，如圖 3-13：

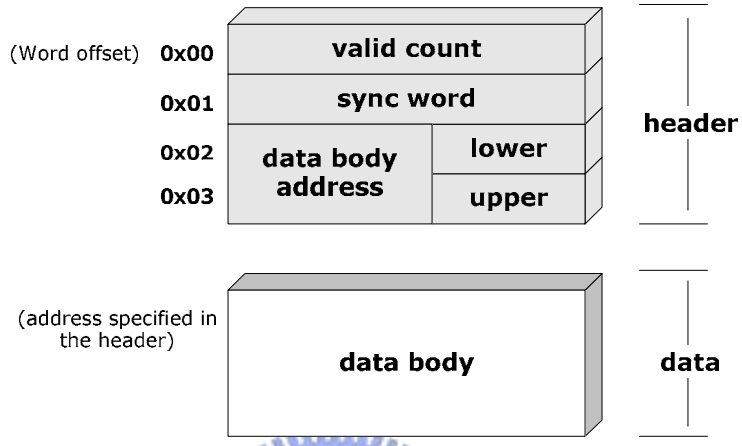


圖 3-13 Private IPBUF 結構圖

資料來源：Linux DSP Gateway Specification Rev2.0[1]

Private IPBUF 和 Global IPBUF 不同，並沒有預先設定位置的問題，所以在 header 中，並沒有所有權相關標示的參數，在 Private IPBUF 中 sync word 和 Global IPBUF 一樣，是用來確定在處理器中寫之後讀取(read-after-write)操作。送資料的一端在寫入資料到資料區後給定 TID，接收資料的一端等到 sync word 變成 TID 時才開始讀取資料。在讀取完畢之後，會寫入 0xff 代表 buffer 被釋放出來，buffer 可再重複被送資料的一端使用。表 3-3，為 Private IPBUF 的結構定義。

表 3-3 Private IPBUF 結構定義

Structure	***Data member***	***Data type***	***Description***
ipbuf_p	c	unsigned short	ipbuf 有效的 word 數
	s	unsigned short	確保在 ipbuf 通訊時，接收者必需等待直到所要的數值傳來
	a	**unsigned long**	資料主體的位置，以一個 long 來表示

資料來源：Linux DSP Gateway Specification Rev2.0[1]

3.4 Linux API (Application Program Interface)

在前面圖 3-2 的架構圖中的 DSP Gateway driver 的部分，提到了存取 DSP 核心資源的裝置檔案系統。DSP Gateway 共有五種裝置的介面-DSP task device、DSP task watch device、DSP control device、DSP watching device 和 DSP exmem device。一般而言，使用者想要使用 DSP 工作時只需要存取 DSP task device。其他裝置只是使用在特殊用途的程式、Dynamic Loader Daemon、wdt Daemon 和 dspctl 利用。且 DSP Gateway 的五種 DSP device 是利用 devfs 的機制，在使用時才動態產生的系統。在建立 DSPLinux 系統時，不需要特別建立這些裝置檔。

3.4.1 DSP 裝置檔案系統建立

較新版的 Linux 核心特地為裝置節點提供了一種新的特殊檔案系統，稱為 devfs(2.3.46 之後才正式納入核心)，devfs 的優勢為：

- 在/dev 目錄下的裝置節點，由裝置初始化程序產生，在移除裝置時移除。
- 驅動程式可以指定裝置名稱，所有權、權限位元。不過，user-space 的程式仍然可以修改所有權與權限位元。
- 不需要為目標裝置擇定一個主編號(major number)，也不必處理次編號(minor number)。

有 devfs 的功能，驅動程式能控管自己的裝置檔。一般而言，要建立和移除置節點，驅動程式需要 include 標頭檔：linux/devfs_fs_kernel.h，以及使用 devfs_register()和 devfs_unregister，devfs_register，可以新建一個裝置檔，並且設定其位置、名稱、主編號、次編號、存取模式等設定值。而 DSP Gateway 便是使用 devfs 的機制，來產生存取控制 DSP 核心的裝置節點。接下來將逐一介紹這些裝置檔。

3.4.2 DSP 工作裝置(Task Device)

DSP Task Device 為 DSP Gateway driver 為 Linux 應用提供一個與 DSP 溝通介面，這些檔案使用 devfs 機制，當 DSP task 被配置載入時會自動產生在/dev/dsptask 目錄中。

```
crw-r--r--    1 root     root      97,   0 Jan  1 00:00 /dev/dsptask/task0
crw-r--r--    1 root     root      97,   1 Jan  1 00:00 /dev/dsptask/task1
```

當讀取或寫入這些裝置，便是代表從 DSP 工作接收或傳送資料。DSP 工作有兩個型式，一個是靜態工作(static task)和請求工作(on-demand task)。靜態工作是指連結 tokliBIOS 的函式庫，同時負荷於 DSP 記憶體。經過 DSP 配置後，便可以使用，一直到解除對 DSP 的配置。與靜態工作(Static task)不同，請求工作(On-demand task)不連結 tokliBIOS 的函式庫，但是會做負載和動態連結。要使用請求工作，Linux 一端需要使用 MKDEV ioctl 的命令產生工作裝置(task device)檔案，給/dev/dspctl，之後負載工作程式給 DSP 記憶體，而且分派 TADD ioctl 命令給 DSP 控制裝置。串列的行程將會被監控由 DSP Dynamic Loader Daemon。本論文中，所使用的方式是靜態工作(Static task)，做為 DSP 工作的型態。

3.4.3 輸入輸出函式

有了和 DSP 溝通的裝置檔之後，還需要溝通函式來與之溝通。在前面的小節中，在 DSP Gateway 的架構中，曾經說明有關於 DSP Gateway driver 的內容，也明確指出對裝置的溝通是透過作業系統的系統呼叫(System Call)，透過驅動程式，參考裡面的檔案操作(File Operations)定義，來對特定的裝置做輸入輸出 I/O。這些輸入輸出的函式，一般較常用的包括有 open、close、write、read、poll、selet 和 ioctl。其中 open、close、write 和 read 最為重要，是一個驅動程式內最基本的輸入輸出函式。以下將針對 open、close、write 和 read 做個說明。

(1) open 裝置開啟

open 作業方式提供驅動程式為後續作業進行任何必要的初始化準備工作，通常會遞增目標裝置的「用量計次」，以免模組在檔案關閉之前被卸載。讓驅動程式保證每個裝置僅有一個程序能打開。使用 open 應該會有的執行動作如下：

- 遞增用量計次
- 檢查 DSP 工作(task)裝置是否錯誤，有錯會傳會 error
- DSP 工作裝置進行初始化動作

驅動程式中的 open 作業方法，如果是由 MKDEV ioctl 產生的(例如：DSP 工作是一個請求工作型式)，而且 DSP 工作還未與裝置建立關係時，則 open 將會呼叫這個區塊，

核心會叫醒 DSP Dynamic Loader Daemon。這個 Daemon 會載入適合的工作物件且分派 TADD ioctl 給 DSP control device，此時裝置便可以使用，open 的呼叫工作完成。

(2) close 裝置關閉

減少 open 的計數，當呼叫 close 的函式時，將會把已經開啟的裝置，但是沒有 task 在使用的裝置資料釋放出來。使用 close 應用會有的動作如下：

- 釋放 open 函式所配置的任何東西
- 關閉時將 DSP 目標的工作(task)裝置關掉。
- 遞減用量計次

(3) read 資料讀取

接收來自 DSP 工作的資料。對於一個被動傳輸的工作，ARM 送出一個要求給 DSP，並且等待 DSP 的回應。對於一個主動傳輸的工作，假如 DSP 已經送一些資料給 ARM，read 能立即回應。假如沒有資料在 buffer 內，則 read 會被鎖住。

只有當裝置被開啟時，驅動程式才會接收從 DSP 讀取的資料，甚至從主動傳輸工作得到。如果在資料送達時，裝置還未被開啟，則資料將會被拋棄。應用程式開發人員必需注意 read buffer 的位置(address)，它被定義在 read 這個系統呼叫(System Call)的第二個變數。函式必需確認它，如果沒有確認，就會回傳錯誤。圖 3-14~圖 3-16表示各種 ARM 讀取 DSP 資料的狀態。圖 3-14表示一個被動傳輸讀取工作，圖 3-15和圖 3-16則分別表示主動傳輸工作所遇到的情況。圖 3-15中，當在 DSP 傳送資料前 ARM 讀取裝置，則 ARM 會被 block，直到能讀取資料。圖 3-16則因為 ARM 讀取時，DSP 早以完成資料的傳輸，故並不會有 block 的情況發生。

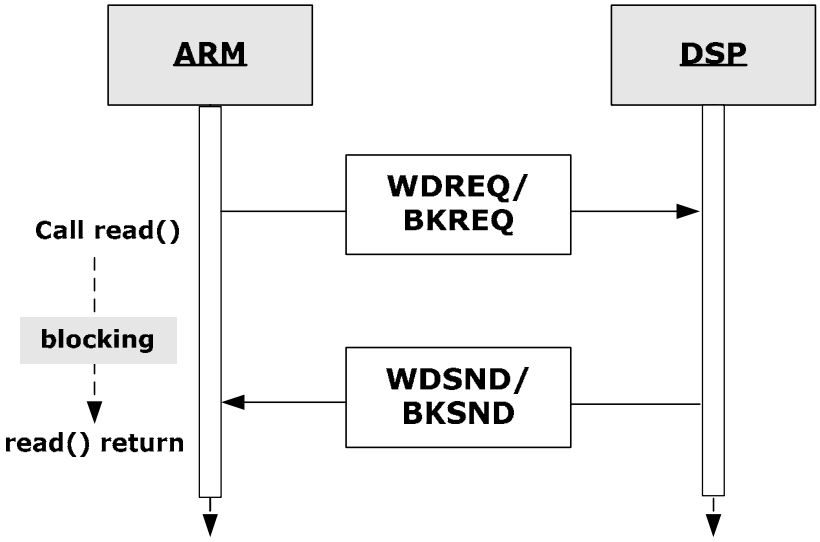


圖 3-14 ARM 從 DSP 讀取資料之被動傳輸模式

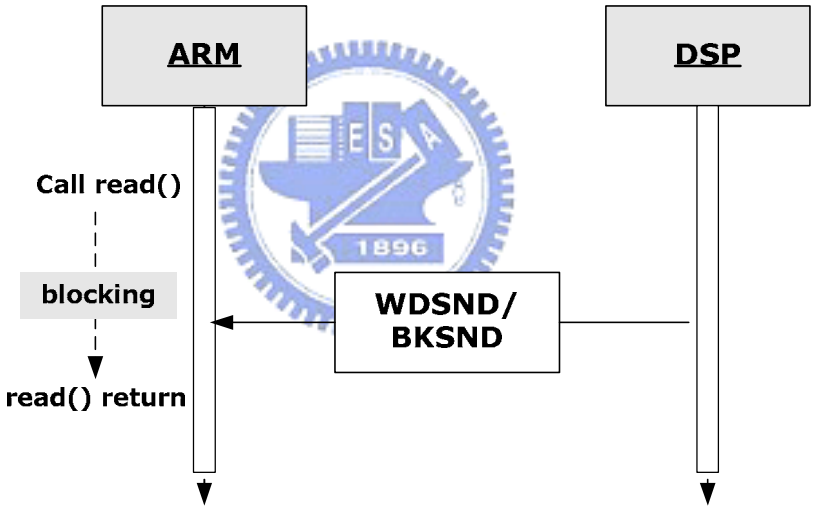


圖 3-15 ARM 在 DSP 送資料前做讀取之主動傳輸模式

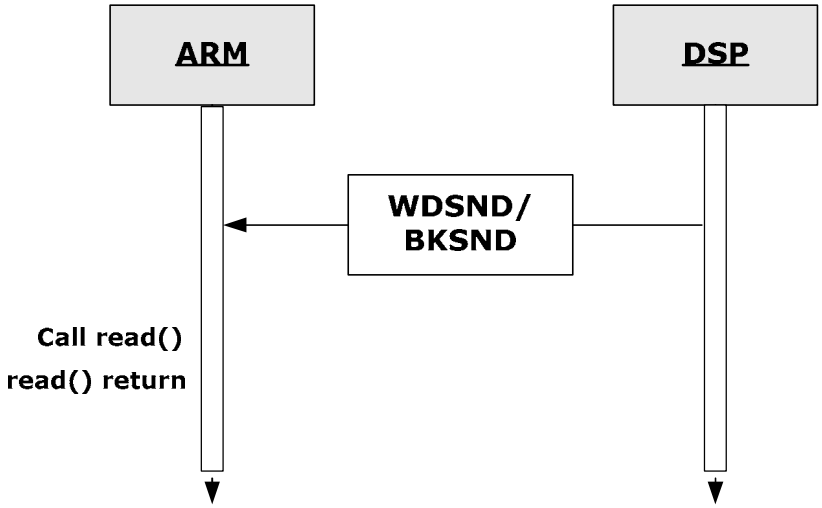


圖 3-16 ARM 在 DSP 送資料後做讀取之主動傳輸模式

(4) write 資料寫入

傳送資料給 DSP 工作。對於一個被動傳輸工作，write 會立即的送資料給 DSP。對於一個主動傳輸的工作，假如 DSP task 還未需要從 ARM 接收任何的資料，則 write 會被 block，假如有 DSP 主動要求(pending)，則 write 會立即送資料給 DSP。

應用程式開發人員必需注意 write data 的位置(address)，它被定義在 write 這個系統呼叫(System Call)的第二個變數。函式必需確認它，如果沒有確認，就會回傳錯誤。圖 3-17~圖 3-19表示各種 ARM 傳輸資料給 DSP 的狀態。圖 3-17表示一個被動傳輸寫入工作，圖 3-18和圖 3-19則分別表示主動傳輸工作所遇到的情況。圖 3-18中，當在 DSP 需要資料前 ARM 已寫入裝置，則 ARM 會被 block，直到 DSP 做需要資料要求。圖 3-19 則因為 ARM 寫入裝置時，DSP 已發出需要資料之要求，故並不會有 block 的情況發生。

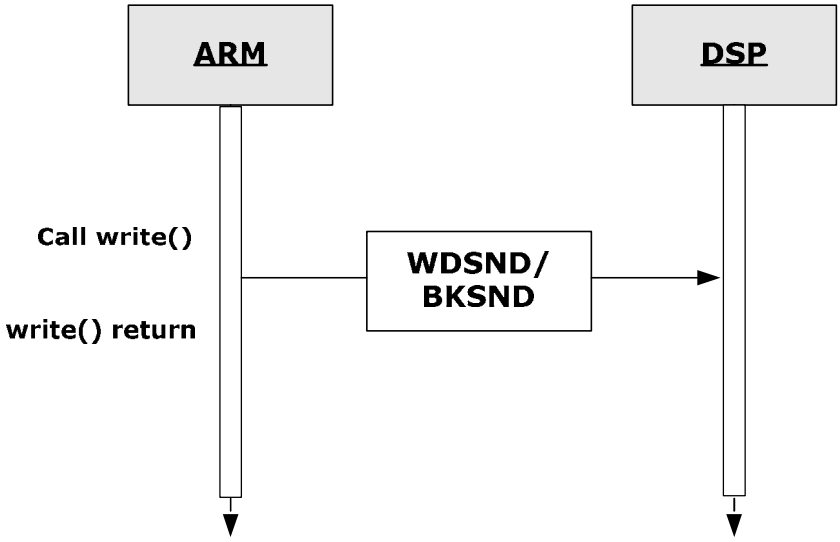


圖 3-17 ARM 傳送資料給 DSP 之被動傳輸模式

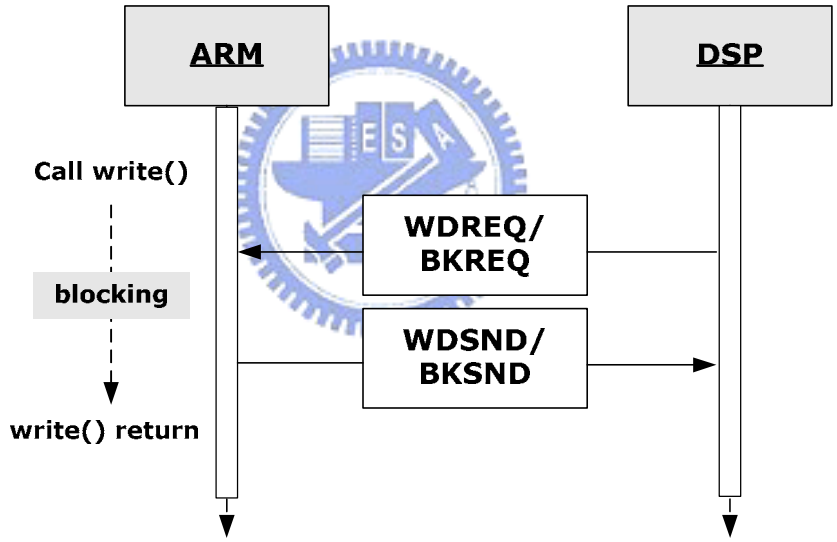


圖 3-18 ARM 在 DSP 要求資料前寫入之主動傳輸模式

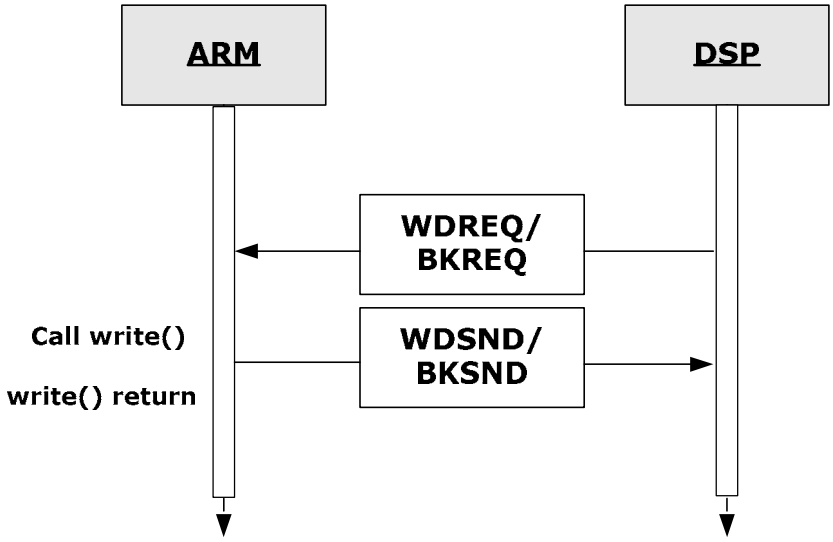


圖 3-19 ARM 在 DSP 要求資料後寫入之主動傳輸模式

(5) poll、select 輪詢機制

圖 3-14~圖 3-19有許多情況下是當 read 啟動時發生資料還未到，而需要等候，系統被 block 的情況。如果不想系統被 block，則可以使用 nonblocking 的 I/O 方式，也就是 poll 和 select 的系統呼叫。例如：在 ARM 和 DSP 溝通的狀況可以用在 Active Receiving tasks 使用 poll 和 check 在輸入核對。如此，ARM 在資料還不能讀取或寫入時，依然可以執行其他工作。

(6) ioctl 裝置控制

ioctl 系統呼叫為驅動程式提供一個給裝置下達特有命令的方式。隨硬體裝置而定，可以讓應用程式存取有特定的功能，而 DSP Gateway 中，當使用這個函式時，其命令包括有 Task initialize(工作初始化)、Task Enable、Task Disable、Reserved 等，關於 DSP Task 的命令，其作業方法，於圖 3-21表示。其中包含了一個 switch 的敘述，可以依據 cmd(command)的引數來選擇處理程序。

3.4.4 DSP 其他相關裝置檔案

在前面，說明了 DSP 中最重要的裝置，DSP Task Device 之後，在 DSP 中還提供了其他裝置做一些特殊的利用。以下將對這些裝置做一個簡單的描述：

(1)DSP Control Device

DSP control device 提供 DSP 控制 Linux 使用的 APIs。Linux 應用能控制 DSP 重新設定，讀取 DSP 配置，產生 DSP task device 給請求工作(on-demand tasks)。Map/unmap SDRAM 記憶體給 DSP 空間等。透過這個裝置檔案。

(2)DSP Exmem Device

DSP exmem device 提供存取 DSP external 記憶體的功能給 Linux，用來載入 DSP 的應用程式。

(3)DSP Task Watch Device

DSP Task Watch device 提供 DSP 動態載入精靈(Dynamic Loader Daemon)所需要的功能。

(3) DSP Watchdog Device

(4) DSP Watchdog device 被用來偵測 DSP watchdog timer[2]的結束

[2] Watchdog timer：每隔一個固定時間就會檢查程式狀態，如果軟體無法正常繼續執行時，watchdog timer 計數週期結束，整個系統會自動重新啟動。

3.5 DSP programming

在前面說明 DSP Gateway 的架構和 Linux 端程式的操作之後，這一節，將說明圖 3-2 中 DSP 程式(DSP Task)的部分。在這之前，先對前面說明過的 DSP Gateway 架構和 Linux APIs 與 DSP Task 之間的動作結合做個流程說明。當一個 Linux 的使用者欲存取使用 DSP task device 時，驅動程式接收 ARM 的操作命令，產生一個 Mailbox command 給 DSP。經過 Mailbox interrupt handler，再傳送至 DSP 端，DSP 系統的 BIOS，DSP Gateway BIOS(tokliBIOS)，接收 Mailbox command，並在合適的 DSP/BIOS TSK 中佇列註冊這項工作。DSP/BIOS TSK 使用正確的工作函式來處理在佇列中的命令，第四章中的圖 4-6 和圖 4-7為一實際資料傳輸的順序圖。其中工作函式是設計 DSP 應用的程式者用於實現 DSP 的函式，而且它可以回送一個 Mailbox command 給 ARM 利用 tokliBIOS 函式庫。圖 3-20表示 DSP Gateway 資料傳輸的架構。

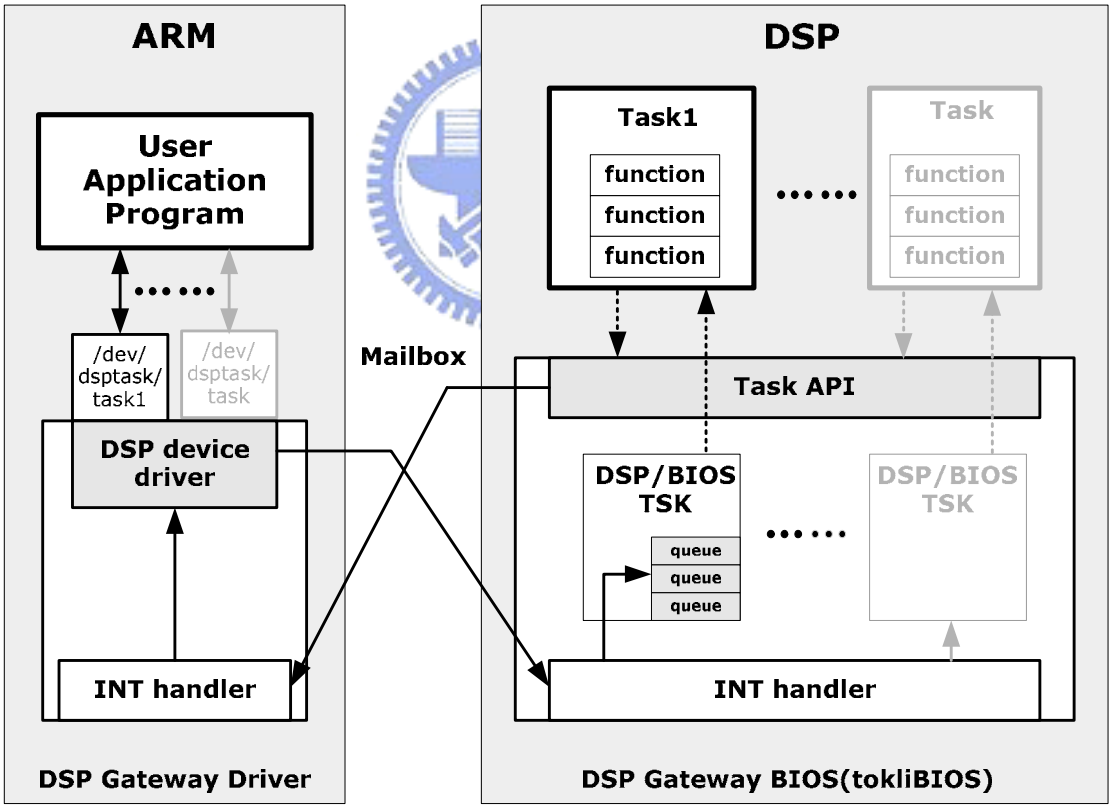


圖 3-20 DSP Gateway 資料傳輸架構圖

3.5.1 DSP Task Programming

DSP 程式設計者在設計 DSP 工作程式時，必須先對工作的型式做個定義，表 3-4 表示了一個 DSP 工作所需要定義的結構。有工作 ID，工作名稱，工作型式，和一些有關於 DSP I/O 及 DSP 功能的函式使用定義。其中 DSP 最大的工作數能到 126(TID 0x00~0xfd)，因為 TID 0xfe 和 0xff 是接收 IPBUF 所有者的資料特別用途。

表 3-4　dsptask 結構

Structure	***Data member***	***Data type***	***Description***
dsptask	tid	unsigned short	Task ID。DSP 工作的 ID。當工作初始化時，DSP Gateway BIOS 設定 TID。
	name	String	Task name(工作名稱)，亦為 device file 的名稱
	ttyp	unsigned short	Task type，工作的型式設定，例如：BKMD(block transfer for MPU→DSP)
	*rcv_snd()	unsigned short	Task function。DSP Gateway 會呼叫這些函式，回應 ARM 的 Mailbox command。
	*rcv_req()	unsigned short	
	*rcv_tctl()	unsigned short	
	*task_attrs	struct TSK_Attrs	DSP/BIOS TSK 特性。如果為 NULL，則為預設值。

資料來源：Linux DSP Gateway Specification Rev2.0[1]

工作型式(Task type)，則定義了 DSP TSK 不同的資料傳輸方式，表 3-5表示 ttyp 參數的定義及意義。在下一小節中將針對 task 中的 DSP I/O 及功能函式做個說明。

表 3-5　dsptask 工作型式

ttyp	***Task type***	***Script***
ASND	Active Sending	不需系統呼叫的 trigger 便會自動傳送資料
PVMD	Passive Sending	需要系統 I/O 函式才會有傳送資料的動作
AREQ	Active Receiving	Task 送出 WDREQ/BKREQ/BKREQP 命令給 ARM，當 Task 需要資料時
PVDM	Passive Receiving	Task 從不送 WDREQ/BKREQ 命令給 ARM

BKMD	Block Sending	Task 使用 BKSND/BKSNDP 命令
BKDM	Block Receiving	Task 送資料從 DSP 給 ARM 的命令

資料來源：Linux DSP Gateway Specification Rev2.0[1]

task_attrs 則是一些對於 DSP/BIOS 的特性設定，其中比較重要的設定為 task 優先權的設定。在規劃一個即時多工程式處理系統時，在 task 的安排上相當重要，這時可以由 task_attrs 來設定每個 task 中優先權的屬性。

3.5.2 DSP Task 應用程式介面

DSP 工作程式設計者能使用一些 Task API 函式處理在工作函式中的資料。這些工作函式包括有 rcv_snd、rcv_req 和 rcv_tctl，這些工作函式會被 DSP/BIOS 的工作所呼叫執行，而且可加入 DSP/BIOS APIs 使用，沒有任何特別的限制。以下將針對這些函式，一一的說明之。

(1) Uns (*rcv_snd)

這個函數定義於當 DSP 工作函式接收到 Mailbox command 中含有 WDSND、BKSND 或 BKSNDP 命令時的動作。表 3-6表示每種命令所對應資料接收的方式。

表 3-6 BKSND、BKSNDP 及 WDSND 命令說明

Receive Command type	***Parameter***	***Script***
WDSND	struct dsptask *task, Uns data	接收 word 資料
BKSND	struct dsptask *task, Uns bid, Uns cnt	Block 傳資料時，利用 Global IPBUF。所以在引數中，需要有 BID 的資料，以及被傳送資料的 Count 數。
BKSNDP	struct dsptask *task, Uns cnt	Block 傳資料時，利用 Private IPBUF，需要被傳送資料的 Count 數。

(2) Uns (*rcv_req)

這個函數定義於當接收到 WDREQ 或 BKREQ 命令時動作。表 3-7表示每種命令所對應的資料要求型態。

表 3-7 BKREQ 及 WDREQ 命令說明

Send Command type	***Parameter***	***Script***
WDREQ	struct dsptask *task, Uns data	接收 word 資料
BKREQ	struct dsptask *task, Uns bid, Uns cnt	Block 傳資料時，利用 Global IPBUF。傳送資料的 Count 數。

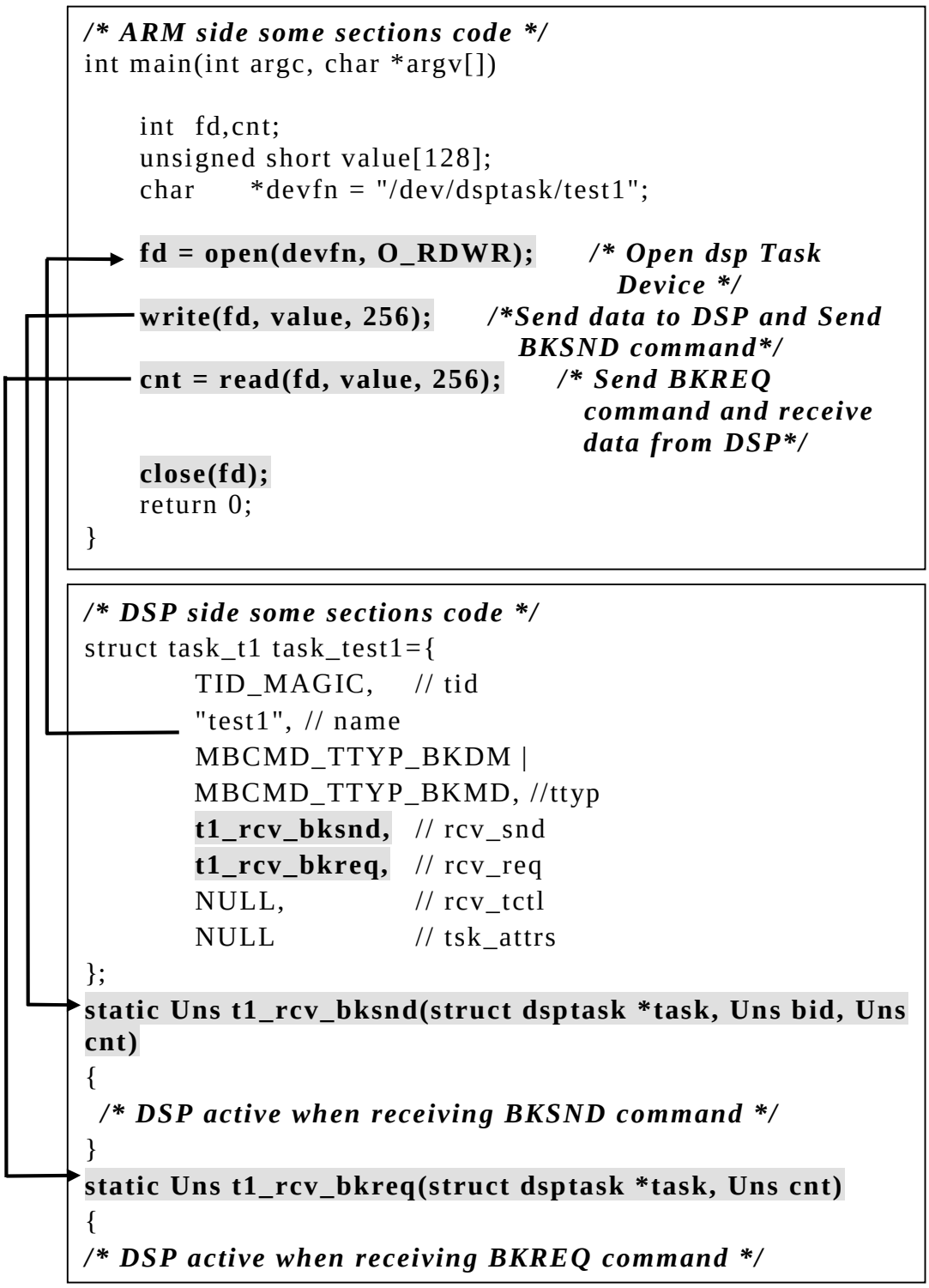
(3) Uns (*rcv_tctl)

這個函數定義於當接收到 TCTL 命令時動作。圖 3-21表示 tctl 函式動作的程式架構

```
Uns t1_rcv_tctl ( struct dsptask *task, Uns ctlcmd )
{
    switch ( ctlcmd ) {
        case MBCMD_TCTL_TINIT:
            return 0;
        case USER_DEFINED_TCTL_COMMAND:
            /*
                user program here
            */
        default:
            return MBCMD_EID_BADTCTL;
    }
}
```

圖 3-21 TCTL 命令作業方式

綜合上述的 DSP 工作函式及 DSP 的工作定義，便可以和 ARM 微處理器做工作上的分工。圖 3-22表示一個 ARM 和 DSP 程式中傳輸相關的部分。



```
/* ARM side some sections code */
int main(int argc, char *argv[])

    int  fd,cnt;
    unsigned short value[128];
    char     *devfn = "/dev/dsptask/test1";

    fd = open(devfn, O_RDWR);      /* Open dsp Task
                                      Device */
    write(fd, value, 256);      /*Send data to DSP and Send
                                  BKSND command*/
    cnt = read(fd, value, 256);     /* Send BKREQ
                                     command and receive
                                     data from DSP*/

    close(fd);
    return 0;
}
```

```
/* DSP side some sections code */
struct task_t1 task_test1={
        TID_MAGIC,    // tid
        "test1", // name
        MBCMD_TTYP_BKDM |
        MBCMD_TTYP_BKMD, //ttyp
        t1_rcv_bksnd,  // rcv_snd
        t1_rcv_bkreq,  // rcv_req
        NULL,          // rcv_tctl
        NULL           // tsk_attrs
};
static Uns t1_rcv_bksnd(struct dsptask *task, Uns bid, Uns
cnt)
{
 /* DSP active when receiving BKSND command */
}
static Uns t1_rcv_bkreq(struct dsptask *task, Uns cnt)
{
/* DSP active when receiving BKREQ command */
```

圖 3-22 ARM 及 DSP 程式基本架構

四、 腦波訊號處理嵌入式系統之設計

論文的最終目標是利用『OMAP 嵌入式單板：Innovator Development Kit』建構一個『腦波訊號處理系統』，用來分析量測到的腦波資料。在腦波訊號取得上，主要利用虛擬實境的技術，在不同的場景刺激下，經由腦波量測儀器測得受測者的腦波訊號，接著做腦波訊號處理分析，擷取訊號特徵，將特徵訊號加以分類，得到不同的輸出。由於整個分析運算複雜，故使用以 TI 的 OMAP 嵌入式單板 Innovator Development Kit 為發展平台，設計一腦波訊號處理系統。以下依系統設計與分析，說明整體架構之流程，並且使用前面章節所述的 DSP Gateway 結合 ARM 和 DSP 的方式加以實現之。

4.1 系統架構分析與系統規格

實驗目標是建構『即時腦波訊號處理系統』於 OMAP 嵌入式單板上。本研究結合實驗室以往之發展，透過虛擬實境之場景，給予人的感官刺激，經過量測擷取腦波訊號，將訊號透過單板運算，可以找出訊號特徵，並對於不同的訊號分類出結果。以下將針對『腦波訊號處理系統』的架構和功能規格做個分析說明。

4.1.1 系統架構分析

本論文是在將訊號分析處理建構於一個即時嵌入式單板，將系統以及演算式建構於實驗單板上。根據實驗室既有技術和現有的裝置，可以將整個系統整合。其中，大致上分有三個部分「虛擬實境六軸平台和場景」、「腦波量測儀器」及「即時腦波訊號處理系統」。「虛擬實境六軸平台和場景」這個部分是為了給受測者有個刺激反應，受測者受到刺激反應之後，腦波會產生一些刺激訊號。之後由「腦波量測儀器」從電極帽接收腦波訊號，並且將其記錄起來。最後將腦波之資料傳送到「腦波訊號處理系統」做訊號處理，腦波訊號處理的方面，又可細分成三個主題。這三個主題是「訊號頻譜分析轉換」、「腦波訊號特徵擷取」及「腦波訊號分類」，並將三個主題相互應用，得到腦波之分析資料。一般而言，訊號頻譜轉換最常使用到的即是快速傅立葉轉換(FFT，Fast Fourier Transfer)。腦波訊號特徵擷取，是使用主要成分分析法(PCA，Principal Component Analysis)

和近年的獨立成分分析法(ICA，Independent Component Ananlysis)，本論文是採用 ICA 的演算方式來做訊號特徵擷取的處理工作。腦波訊號分類是將不同精神狀態下的腦波訊號分類出來，之後便可以將分類出來的特徵模型，做腦機介面(BCI，Brain and Computer Interface)的控制與發展。

4.1.2 設計功能規格與目的

本實驗未來目標設計是用於將腦波訊號處理的儀器微小化，甚至於攜於受測者身上使受測者本身即可隨時讀取個人的腦波資料，亦或是將這個系統裝置結合於不同的嵌入式系統產品中，使得產品具有辨視腦波之能力，更進一步達到 BCI 由人腦直接控制電腦的理想。故本論文動機便是將 BCI 腦波分析技術的核心，在嵌入式系統上實現，將來有朝一日，便可直接將嵌入式系統上的 BCI 技術直接移植到各式各樣需要 BCI 的其他嵌入式系統上。

本論文中的系統，根據目前分析的需求在其內部加入快速傅立葉轉換(FFT)與獨立成分分析(ICA)，最後將會加入腦波之分類與控制溝通之機制，做為分析之人機介面。本論文是以將 ICA 在嵌入式單板實現為其應用，為了證實演算之正確性，之後將以較為常見之 Guasian 和 Sub-Guasian 的訊號做為測試系統之對象。最後再將腦波訊號做實際分析測試。本論文採用的為 TI 的「Innovator Development Kit」，一個以 OMAP1510 為微處理器的嵌入式開發單板。其發展需要下列幾項重要的軟硬體設備：

- Innovator Development Kit (OMAP Platform)
- Host PC：Linux Operation System
- Minicom (connect Host PC and Innovator Development Kit by serial port)
- Cross-Compiler (for ARM code development)
- CCS (for DSP code development)

4.2 訊號分析處理方法

定清楚最終目標和功能規格之後，便是要實踐其內容。前述，要將快速傅立葉轉換(FFT，Fast Fourier Transform)和獨立成分分析法(ICA，Independent Component Ananlysis)加入嵌入式單板之功能，將演算法在嵌入式單板上實現之前，以下將針對 FFT 和 ICA 這兩個方法做較詳細的說明，並決定採用之演算法。

4.2.1 快速傅立葉轉換(Fast Fourier Trasnform)

傅立葉轉換(Fourier Transfer)是在訊號分析上不可或缺的工具之一，對於處理非週期性的訊號相當重要。Fourier Transfer 分析和處理訊號時，往往需要花費許多時間才能處理分析少量的訊號，而由 Cooler-Tukey 提出的 FFT，改善傅立葉轉換的計算時間，在訊號處理的領域上佔相當垂要的地位。現今 FFT 在各書籍和網路上有相當大量的參考資料。在本論文中，加入了基本 FFT 的能力於腦波分析的嵌入式單板中，在腦波分析前便可利用 FFT 觀察腦波頻率的分佈。

4.2.2 獨立成分分析法(Independent Component Analysis)[3]

訊號分析，意謂著解釋訊號代表的意義，然而為了找出訊號的涵義，科學家嘗試著用不同的數學模型去解釋訊號。最基本的傅立葉轉換，使得訊號除了時域(Time-Domain)的資訊外，更進一步的有頻域(Frequency-Domain)的資訊。當然這樣是還不夠的，為了能更精確合理的解釋訊號，開始有了各種利用數學機率統計的方式找出這些訊號的特徵，傳統上常見的有主要成分分析法(PCA，Principal Component Analysis)，常用於聲音、影像等訊號；獨立成分分析法(ICA，Independent Component Analysis)，原先即是為了腦波訊號分析而發展出來的擷取訊號特徵之方式，由於分析效果顯著，近年來也被應用於聲音、影像之分析。本論文是以腦波分析為出發點，主要採用獨立成分分析法作為腦波訊號特徵擷取的方法。

一個狹義的線性獨立成分分析模型基本定義，是使用統計上的潛在變數模型(latent variables model)，假設 **x** 為可觀察到的 n 個獨立成分線性組成的 $x_1, x_2 ....., x_n$，如(4-1)式，是線性獨立隨機變數 s_j 的線性組合

[3] 有關於 ICA 理論部分是參考擷錄自參考文獻[13]

$$x_j = a_{j1}s_1 + a_{j2}s_2 + ... + a_{jn}s_n \quad for\ all\ j \tag{4-1}$$

ICA 演算法就是要盡可能的找到一組 **x = As** 的解，使成分 s_i 之間盡可能的互為獨立。其中

x = As (4-2)

- **x**：觀察到的隨機變數 $\{x_1, x_2 ....., x_n\}$
- **s**：潛在變數(latent variables)成分 $\{s_1, ..., s_n\}$，s_j 互為獨立
- **A**：$m \times n$ 常數 a_j 所構成的混合矩陣(mixing matrix)。

獨立成分是潛在變數，無法直接由觀察得知，其中 A 矩陣也是未知，故 ICA 演算法便是利用觀察到的隨機向量 **x**，將 **A** 和 **s** 估測出來。ICA 演算除了一些統計獨立的假設之外，在使用時有一些限制需要注意：

(1) 所有獨立成分 s_j 都是非高斯分佈，這主要是根據中央極限定理(central limit theorem)，非高斯分佈且互為獨立的隨機變數之和，其分佈會比任一獨立隨機變數的分佈更接近於高斯分佈。ICA 利用此一定理做逆向思考，找出最不是高斯分佈的成分，就最有可能是要找的獨立成分。

(2) 觀察到的混合訊號，其數目 m 最少要和獨立成分的數目 n 相同($m \geq n$)。在 $m < n$ 之情形，**A** 的反矩陣不存在，而現今的 ICA 演算法 $m < n$ 在之下都不成立。

(3) **A** 的行必須有完整秩(full column rank)，可使矩陣 **A** 為非奇異，而 **x = As** 必有解。

4.2.3 獨立成分分析演算法(ICA Algorithms)

ICA 性質與 ICA 的目標函數以及演算法有關。目標函數需有一致性(consistency)，不管初值不何，最後都會收斂於同一點，且誤差最小。演算法必需考慮到收斂速度穩定度。找出目標函數的方法很多，例如以高次統計為基礎的峰態(kurtosis)，及資訊理論(Information theory)中相互資訊(Mutual Information)的觀念。ICA 利用資訊理論，將獨立成分之間相互資訊最小化為目標，與找非高斯分佈的意義相當。為了找出目標函數，將

此一觀念具體化，利用上面資訊理論的相互資訊概念：相互資訊是個別變數不確定性和所有變數不確定性兩者之義，而不確定性即為熵(Entropy)，定義熵為 $H(x)$：

$$H(p_1, p_2 ....., p_M) = -c\sum_{i=1}^{M} p_i \log_2 p_i \qquad c \text{ is usually } 1 \tag{4-3}$$

一般而言，常用的方式有：資訊最大法(information maximization approach)和快速獨立分析法(FastICA)。在嵌入式單板中的實現，選用 FastICA，做為在單板上實現的演算方式。

4.2.4 訊號前處理

ICA 演算法中，在分析資料前會先做前處理使 ICA 的運算簡化，對於資料的處理將能更快且更好。首先，會先對資料做置中(centering)，即是減去 **x** 的平均值向量(mean vector: $\mu = E\{\mathbf{x}\}$)，使得 x_i 成為平均值為零的變數。另一個處理為白化(whitening)，就是將觀察到的訊號轉換成白雜訊(white noise)，可以去除資料之間的相關性，使觀察到的資料互不相關，變異數為 1。由 **x** 隨機向量，其期望值和變積矩陣為

$$E\{\mathbf{x}\} = \mu \tag{4-4}$$

$$E\left\{(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\right\} = \mathbf{C} \tag{4-5}$$

藉由特徵值分解，可以得到正交矩陣

$$E\left\{(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\right\} = \mathbf{C} = \mathbf{EDE}^T \tag{4-6}$$

上式中 **C** 為 **x** 的變積矩陣，**E** 為 $E\left\{(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\right\}$ 的特徵向量以欄向量的形式構成的正交矩陣，**D** 為 $E\left\{(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\right\}$ 的特徵值。如此，可以將白化表示成

$$\tilde{\mathbf{x}} = \mathbf{ED}^{-1/2}\mathbf{E}^T\mathbf{x} \tag{4-7}$$

其中 **D** 為對角矩陣，由 D 的對角元素平方根倒數求得反矩陣 $\mathbf{D}^{1/2}$

$$\mathbf{D}^{-1/2} = \text{diag}\left(d_1^{-1/2}, d_2^{-1/2}, ..., d_n^{-1/2}\right) \tag{4-8}$$

由(4-6)及(4-7)式代入(4-9)式中左式可得到：

$$E\left\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\right\} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T E\left\{\mathbf{x}\mathbf{x}^T\right\}\mathbf{E}\mathbf{D}^{1/2}\mathbf{E}^T$$
$$= \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{E}\mathbf{D}\mathbf{E}^T\mathbf{E}\mathbf{D}^{1/2}\mathbf{E}^T \quad (4\text{-}9)$$
$$= \mathbf{I}$$

由此可以容易的達到白化前處理的目的。另一方面，白化可以將(4-2)式矩陣 A，變成一個計算量更少的正交矩陣 $\tilde{\mathbf{A}}$：

$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{A}\mathbf{s} = \tilde{\mathbf{A}}\mathbf{s} \quad (4\text{-}10)$$

$$E\left\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\right\} = \tilde{\mathbf{A}}E\left\{\mathbf{s}\mathbf{s}^T\right\}\tilde{\mathbf{A}}^T = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^T = \mathbf{I} \quad (4\text{-}11)$$

可以使 $\mathbf{A}$ 由原先需估計自由度減少，使得運算變的簡單許多。

4.2.5 快速獨立成分分析法(FastICA)

熵(亂度)是表示隨機變數觀察值的不確定性，越亂就是越無法預測，熵越大。熵的功用舉個例，在許多分佈中，Gaussian 具有最大的熵，是最亂的一種分佈。Super-Gaussian，其觀測值非常集中在某些特定分佈，熵會較小。由此可以熵可以用來做 non-Gaussian 的量測。將熵做小修改，定義負熵(negentropy)，$J(y)$，在 Gaussian 分佈時為零，恆為正值。

$$J(y) = H\left(y_{gauss}\right) - H(y) \quad (4\text{-}12)$$

用負熵來表示相互資訊(Mutual Information)時，可以以下面的形式：

$$I(y_1, y_2, ..., y_n) = C - \sum_i J(y_i) \quad \textit{Where C is a constant} \quad (4\text{-}13)$$

每個獨立成分的負熵其和為最大方向，也就是找出使相互資訊最小化，FastICA 便是利用負熵為目標函數。但負熵最大的問題在於計算困難，故需要找一個近似函數，且具有統計特性，又不需事前對成分做估計假設的簡單演算法。為此利用古典的方法高次動差(higher-order moment)做為近似函數。

$$J(y) \propto \left[E\left\{G(y)\right\} - E\left\{G(y)\right\}\right]^2 \quad (4\text{-}14)$$

G 可以為任意非二次函數，G 的挑選方式將影響到 ICA 方式的結果。

FastICA 基於定點疊代的運算由負熵最大化來找獨立成分。它能由牛頓法疊代的方式來近似。負熵的 G，以 g 來表示，可以選用下面的例子，在本論文中，是選用 g_1 做為負熵的 G。

$$g_1(u) = \tanh(a_1 u) \tag{4-15}$$

$$g_2(u) = u\exp\left(-u^2/2\right) \tag{4-16}$$

a_1 為大於 1 小於 2 的常數，一般而言是 $a_1 = 1$，而 FastICA 演算法，可以用下面的步驟來表示

Step1 選擇一個初始值(例如：亂數選取)的權重向量 **w**

Step2 使得 $\mathbf{w}^+ = E\left\{\mathbf{x}g\left(\mathbf{w}^T\mathbf{x}\right)\right\} - E\left\{g'\left(\mathbf{w}^T\mathbf{x}\right)\right\}\mathbf{w}$

Step3 使得 $\mathbf{w} = \mathbf{w}^+ / \left\|\mathbf{w}^+\right\|$

Step4 如果還未收斂，則回到 ***Step2***

在運算過程中，收斂代表的意思是舊的向量 **w** 和新的向量 $\mathbf{w}^+$ 互為平行，也就是內積會等於 1。但是不會指向同一點，因為 **w** 和 −**w** 是定義為同一方向。需要注意的是所有的資料在處理之前，都已經先白化過(prewhiten)。

負熵的最大值，可由 $E\left\{G\left(\mathbf{w}^T\mathbf{x}\right)\right\}$ 的最佳化得到。根據 Kuhn-Tucker 情形，在 $E\left\{(\mathbf{w}^T\mathbf{x})^2\right\} = \|\mathbf{w}\|^2 = 1$ 的限制之下，$E\left\{G\left(\mathbf{w}^T\mathbf{x}\right)\right\}$ 的最佳化點會在

$$E\left\{\mathbf{x}g\left(\mathbf{w}^T\mathbf{x}\right)\right\} - \beta\mathbf{w} = 0 \tag{4-17}$$

使用牛頓法解(4-17)式，F 代表左式，而 F 的 Jacobian 以 $JF(\mathbf{w})$ 表示：

$$JF(\mathbf{w}) = E\left\{\mathbf{x}\mathbf{x}^T g'\left(\mathbf{w}^T\mathbf{x}\right)\right\} - \beta\mathbf{I} \tag{4-18}$$

為了簡化(4-18)式的反矩陣，將(4-18)式第一項做近似，近似結果為：

$$E\left\{\mathbf{x}\mathbf{x}^T g'\left(\mathbf{w}^T\mathbf{x}\right)\right\} \approx E\left\{\mathbf{x}\mathbf{x}^T\right\} E\left\{g'\left(\mathbf{w}^T\mathbf{x}\right)\right\} = E\left\{g'\left(\mathbf{w}^T\mathbf{x}\right)\right\}\mathbf{I} \tag{4-19}$$

如此 Jacobian 矩陣就能被對角化，很輕易的反轉。最後利用牛頓法疊代，可得到(4-20)式如下：

$$\mathbf{w}^+ = \mathbf{w} - \left[E\left\{\mathbf{x}g\left(\mathbf{w}^T\mathbf{x}\right)\right\} - \beta\mathbf{w}\right] \Big/ \left[E\left\{g'\left(\mathbf{w}^T\mathbf{x}\right)\right\} - \beta\right] \tag{4-20}$$

假如將(4-20)式寫成矩陣形式，可以得到 FastICA 為：

$$\mathbf{W}^+ = \mathbf{W} + \Gamma\left[\mathrm{diag}\left(-\beta_i\right) + E\left\{g(y)y^T\right\}\right]\mathbf{W} \tag{4-21}$$

其中 $\mathbf{y} = \mathbf{W}\mathbf{x}$, $\beta_i = E\left\{y_i g\left(y_i\right)\right\}$ and $\Gamma = \mathrm{diag}\left(1\Big/\left(\beta_i - E\left\{g'\left(y_i\right)\right\}\right)\right)$。矩陣 **W** 經過這此步驟後會被正交化。在嵌入式單板中的 FastICA 演算法，便是使用矩陣型式的 FastICA(4-21)式。

4.3 訊號分析系統設計

經過系統分析與系統規格的決定之後，以及前面小節針對 ICA 的內容的說明，加上第三章所提到的 ARM 和 DSP 通訊的方式，綜合這些步驟，便可應用於腦波訊號(或是其他訊號)利用 ICA 分析，得到訊號的特徵。本章節主要將針對「系統狀態」與「系統架構設計」兩個部分加以說明。

4.3.1 系統狀態

在整個腦波分析實驗中，在主要是以 ICA 為分析理論，找尋訊號的特徵，將訊號原來的樣子還原。根據 ICA 所需的工作，將系統分成兩種處理狀態，分別是 ICA training，訓練出 ICA 所要求的權重向量(Weight)的值。還有做資料的 testing，將混合的資料乘上所求得的矩陣，便可得訊號的特徵組成。圖 4-1以中間的決策點來分成這兩種處理狀態流程。

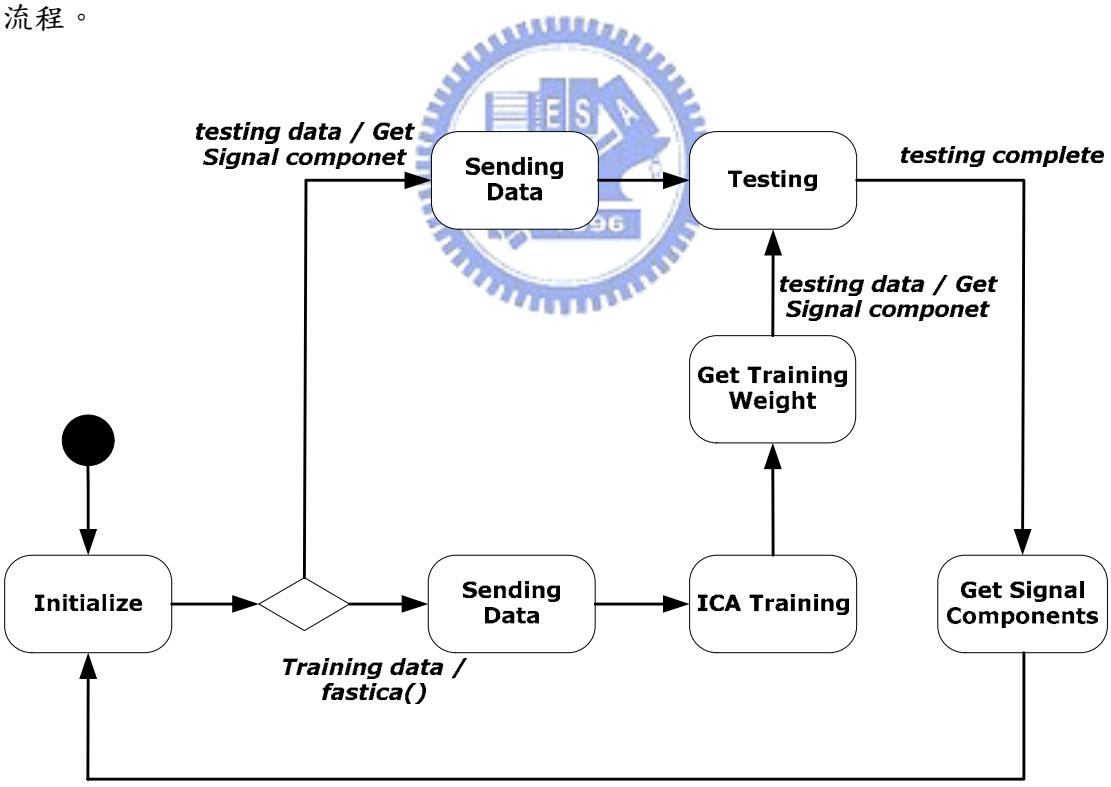


圖 4-1　腦波分析系統狀態圖

1. ICA Training：

在這個模式下，目的是前面在 ICA 理論中的權重向量(Weight)試驗出來，做為訊號轉換的矩陣。在這個架構中，一開始，要先將工作先配置，以進行運算，且會對系統做個初始化的動作(詳見第三章)。系統初始於等待的狀態，當使用者是進入 Training，此時系統將的做動作命令和資料的傳送給處理工作，進行 ICA Training，經過 Training 的過程之後，會得到權重向量(Weight)。這時候會進入 Testing 模式，將原始的資料和 Weight 相乘，做 Testing 的動作，得到訊號的成分。

2. Testing：

這個部分的動作，是將原始資料和所求得的 Weight 相乘，以得到所要求的訊號成分。若訊號已經 Training 完成，或是已有 Training 過後的權重向量(Weight)時，都會進入此模式。在圖 4-1的決策點，如果使用者給 Testing 的命令，則會直接做 Testing 的動作，使用已有的權重向量(Weight)，而不會經過 Training。在工作完成之後，將會得到訊號的成分，回復到原來的狀態，等待新的資料。

以圖 4-1的狀態做為設計主軸，做為系統架構設計前的認知。接下來，將針對每個部分的工作在 ARM 和 DSP 兩個微處理器之間做詳細的分工。

4.3.2 系統架構設計

經過系統狀態的表示之後，接下來便要對系統架構做個規劃。包括，接收提供訊號原始資料，配置系統初始狀態，到 ICA Training 和求得訊號資料的成分等工作。基於使用 OMAP 微處理器，同時擁有 ARM 和 DSP 的兩個架構，接下來便是將工作適當的分配給兩個處理器使用。在工作的安排上，依運算的需要分成兩個部分，一個部分是不需要大量運算，主要是做做資料接收傳送，以及操作介面的安排，另一部分則是，只做運算功能的部分。依照不同的特性，給予不同的工作。所做的安排方式為：

- ICA 的演算法部分，需要大量的持續的運算，被安排於 DSP 的部分，將 ICA 寫成的個 DSP 的 task 功能，供 ARM 微處理器的 process 呼叫。
- 網路、資料取得和檔案存取相關工作不需要大量的數學運算，但是需要操作控制介

面，這個部分安排於 ARM 微處理器的 process。

參考圖 3-1，建立 ICA 訊號分析處理的關係，如圖 4-2所示。從裡面可以看出大致上的工作分配關係。

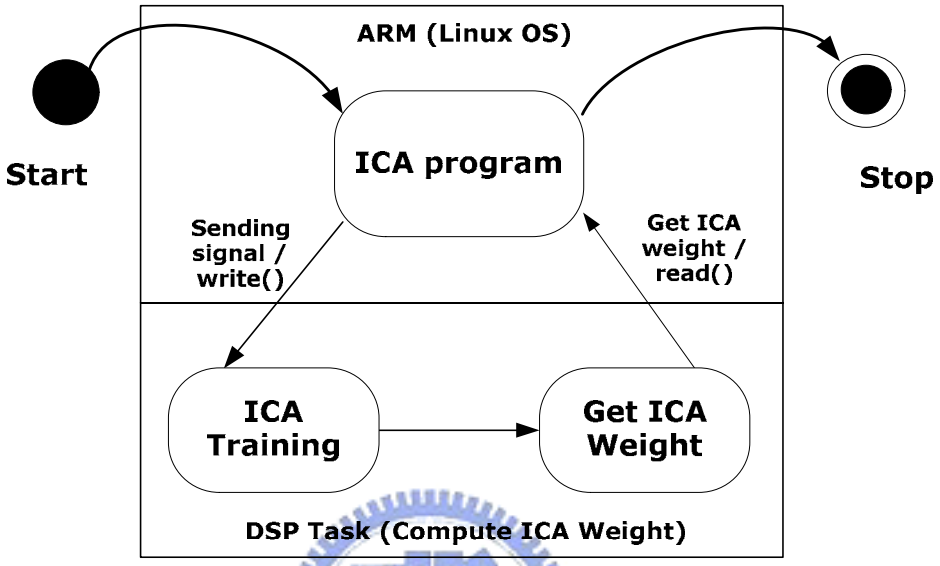


圖 4-2　ICA 資料傳輸處理狀態圖

ARM 微處理器部分，主要是將訊號資料接收，或是讀取和寫入資料於檔案。這個部分可以和其他硬體做一個結合操作。將訊號做 ICA 的運算這個部分，由於需要用到大量的數學函式，主要是交由 DSP 微處理器處理。其中 ARM 和 DSP 的傳輸則是利用到前面章節所說的 DSP Gateway 的機制。表 4-1可將 ARM 和 DSP 的工作做個詳細的細分如下。

表 4-1　ARM 和 DSP 工作分配表

微處理器	工作
ARM	從外部接收傳送訊號
	資料位元置換
	透過 IPBUF 給 DSP task 傳送資料，或 DSP task 接收資料
DSP	從 IPBUF 接收資料，和傳送資料
	將資料給做 ICA 運算的 DSP task 內的 Queue 等待處理
	FastICA 運算

4.3.3 DSP task 載入(ICA task 載入)

上一章節曾經說明 DSP task 的程式開發是用 TI 的 CCS(Code Composer Studio)的環境所編寫的。在整個 ICA 的工作中，前面依處理器特性的不同將整個工作分段分配給不同的處理器處理，在 ICA 訊號分析系統中，有關於 ICA 的演算法部分交予擅長於計算的 DSP 微處理器，有利於整體的效能。將 ICA 演算法的部分編寫成 DSP code，再由 CCS 編譯成 COFF 的執行檔（.out）。編譯出來的 out 檔可以視為一個 DSP 工作模組，而現在有 ICA 的工作模組，接下來的工作，便要要將模組載入 DSP 核心中，讓 DSP 具有 ICA 的運算功能。

在 DSPLinux 下，欲將 DSP task 載入系統中利用，需要載入的工具，而 DSP Gateway 便提供了一個工具：dspctl(DSP Control Utility Program)。在 DSP Gateway 中，可以找到 dspctl 的原始檔，將其內容經過交叉編譯（cross compiler）之後，便可在 ARM 的環境下使用 dspctl 的功能。dspctl 是一個可以將 DSP 的工作模組載入系統，把 DSP task 的環境加以配置，使得 DSP 具有此 task(例如：ICA)的處理功能。

Dspctl，可以提供使用者，載入 DSP 的工作模組外，並且能控制 DSP 的重新設定，啟動。DSP 的記憶體位置配置等功能。圖 4-3表示將 DSP 工作模組載入配置時的情形。

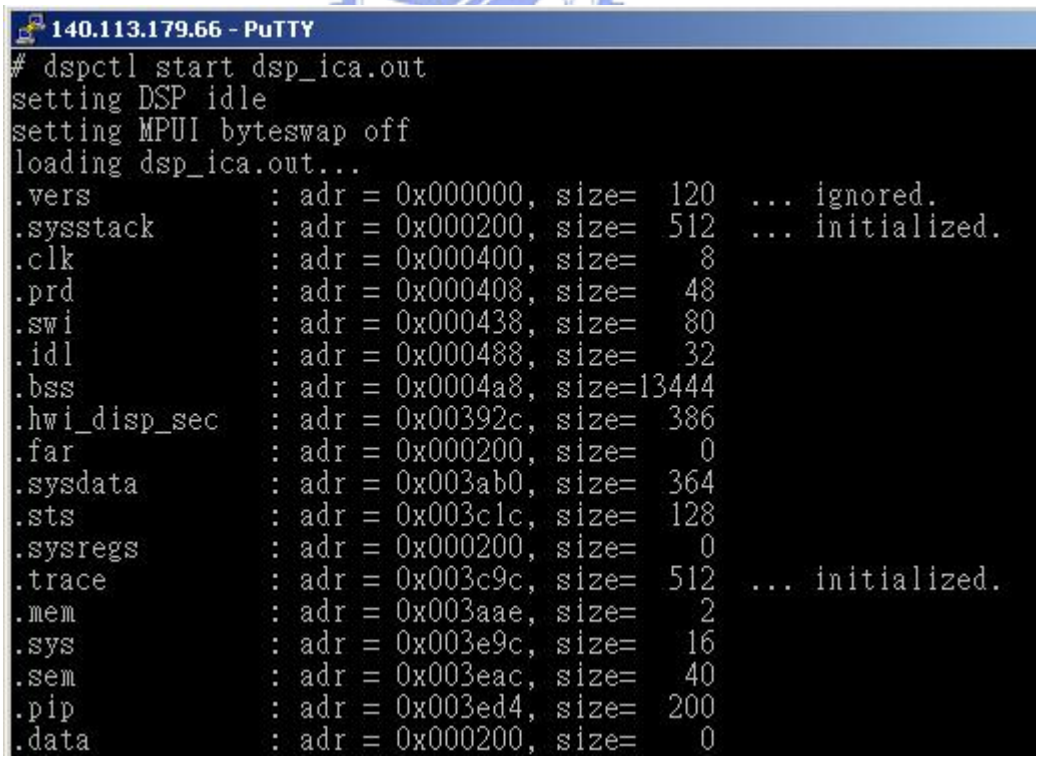


圖 4-3 DSP task 載入

4.3.4 訊號資料傳送

ARM 和 DSP 分工中，一項重要的部分便是 ARM 和 DSP 的資料傳輸。ARM 和 DSP 的內部資料傳輸，主要是利用 DSP Gateway 所定義的 IPBUF，IPBUF 是利用 OMAP 內部的 share memory(詳見圖 2-1)。資料傳輸的型式，所採取的方式是 block send 的方式，ARM 在傳送資料時的 Mailbox command 中在 CMH 的部分給 0x20，表示 BKSND 的命令，並且傳送 BID(buffer ID)給 DSP。ARM 在傳送資料時，會先傳送 Mailbox command，這時會有 Mailbox interrupt(INT5)產生，接收這個命令，經過 interrupt handler 的處理 DSP task(ICA task)收到接收資料的命令，並會到 Mailbox command 裡面指定的 IPBUF 內取得所要處理的資料，經過在 DSP 中 ICA 運算，將得到結果存在記憶體內。同樣的 ARM 這時會送接收資料的命令過來，DSP 將得到的結果送到 IPBUF 中，ARM 可從 IPBUF 取得 ICA 權重向量的資料，經簡單的矩陣運算，即可得訊號的成分。圖 4-6和圖 4-7分別表示整個資料傳送和接收的順序圖。

但是在傳輸時 IPBUF 的大小有限，一個 IPBUF 僅有 128words 的大小(1 words=2 bytes)，也就是說，會有資料無法一次傳輸，需要分批傳輸。將資料分一段一段傳輸，每次傳輸時需要知道傳輸的位置，例如一個 128 點的資料，每次傳 32 點，便要傳四次，而且需要告訴 DSP 現在是傳輸是資料的第幾點開始，第幾點結束。如果一次傳多筆資料的話，需要定義正在傳第幾筆資料，另外，還定義一個資料傳送結束的旗標，告訴 DSP task 在接收完資料之後，便可以開始工作，不需要等到 ARM 的部分送出回傳資料的要求時，再做資料的運算，這樣子，才能發揮 ARM 和 DSP 同步執行的效能，並且節省運算時間。就上述的傳輸方式，定義一個傳輸的封包格式如表 4-2：

表 4-2　資料傳送封包結構

Structure name	***Data member***	***Data type***	***Description***
DATA_T	data_startnum	unsigned short	傳輸資料的起啟點
	data_endnum	unsigned short	傳輸資料的結束點
	signalnum	unsigned short	第幾筆訊號資料
	finishflag	unsigned short	是否傳輸完成
	signaldata[]	float	訊號資料

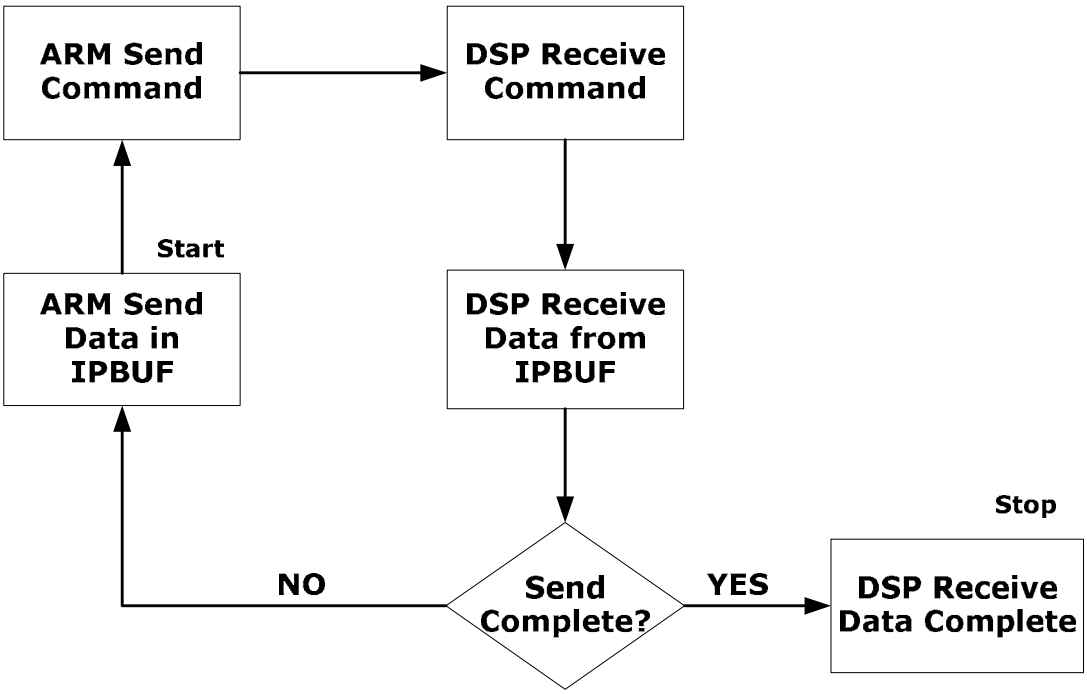


圖 4-4　資料傳送流程圖

傳輸時需要注意一點就是，使用完 IPBUF 時要將其釋放出來，不然 IPBUF 會一直被 task 給佔用，圖 4-5表示了 IPBUF 轉移的過程：

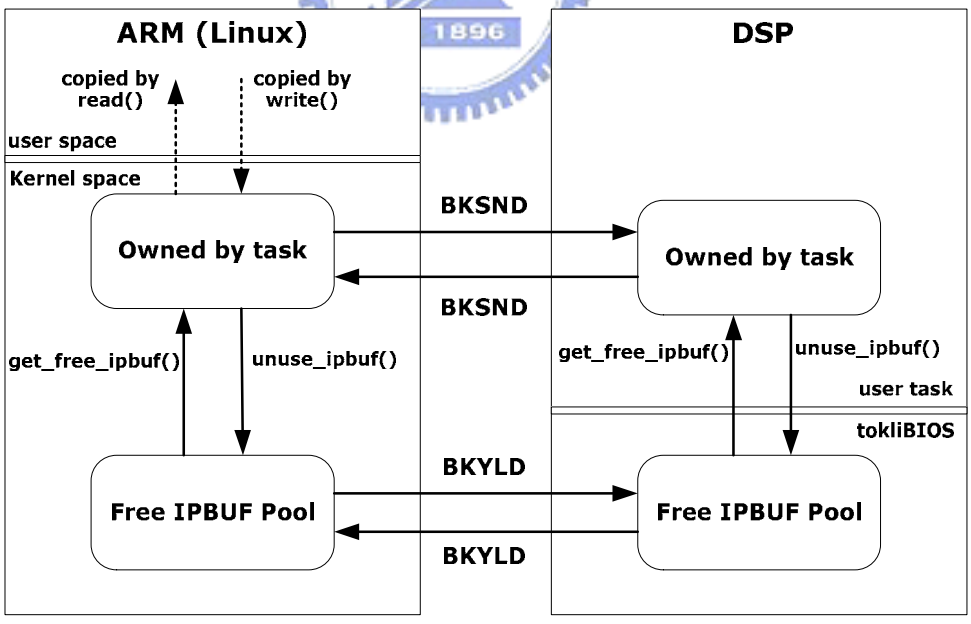


圖 4-5　IPBUF 所有權的轉移

ARM 使用 read 和 write 兩個系統呼叫(System Call)做傳輸的動作，搭配 get_free_ipbuf 和 unuse_ipbuf 兩個函式來取得和使用 IPBUF，有效掌控傳輸的資源。

ARM 傳資料給 DSP 了傳輸要分批傳之外，另外，需要注意的一點是 ARM 和 DSP 溝通時如果使用 32bit 大小以上的資料區塊，需要先經過資料位元的置換，這是因為 ARM 和 DSP 資料的堆疊方式不同。位元組順序，DSP 和 ARM 有所不同。對於多位元組值，x86 PC 將低位元組放在前(高位元組)，這種順序稱為 little endian(LSB 在前面)。但是大部分的位元組是採用另一種順序，big endian(MSB 在前面)。有的處理器甚至同時支援兩種模式。一般而言，都比較偏好於 big endian 模式。德州儀器(TI)的 DSP 為了支援於 Intel-base 的 x86 系列溝通，在設計時是採用 little endian 的模式，但是 ARM-base 是屬於 big endian 的順序，和 DSP 順序不同。例如，一個數值 0x01020304，以 big endian 表示會是「0x01 0x02 0x03 0x04」的順序排列，而 little endian 則會是「0x04 0x03 0x02 0x01」。

解決的方式是盡可能的不需要用到位元組順序，但如果需要多個位元組表示的話，那就要特別注意順序的問題了。在使用之前要把順序調換過來。例如，當在填寫網路封包的標頭時，或是處理一個不同位元組順序的週邊(ARM 和 DSP 可視為這種情況)，要先確定資料是以那一種順序排序，在傳輸時，要記得調整位元組之間的位置。

ARM 與 DSP 溝通，已經知道位元組排列順序不同，所以在傳輸時要考慮到此一問題，不然處理的資料都會有所錯誤。DSP Gateway 針對此一問題，在 DSP Gateway driver 中加入了處理此一問題的做個修正，在傳輸時，會將資料順序重新排列。DSP Gateway 提供 16-bit 的處理，也就是 2-byte 的處理。但在處理 ICA 資料時，為了精確的表示 ICA 的訊號資料，一般都是一 32-bit 來表示。例如，如果一個數值為 0x0102，在 little-endian 下表示，會是「0x02 0x01」，DSP Gateway 會將其位元修正為「0x01 0x02」的 big-endian 表示。處理 ICA 時，以 32-bit 表示一個數值為 0x01020304，以 DSP Gateway 處理後，只能變成「0x0304 0x0102」，所以在程式中，還需要將 32-bit 中的高位元的 16-bit 和低位元的 16-bit 交換才能讓資料正確。在訊號的 raw data 從 ARM 傳入 DSP 之前，需經過 data word swap 的動作，才能確保資料的正確性。綜合前面，可以將整個系統架構和資料傳輸的設計，表示於圖 4-8。圖 4-8中，整個順序分別以數字標號表示，1~6 的流程表示，單板收到訊號資料，執行 ICA 程序，將 ICA 運算交由 DSP 處理，DSP 再將處理完的資料更新至 IPBUF 中，7~9 的流程表示，ARM 的部分給予一個接收結果的 Mailbox 命令，然後從 IPBUF 中接收傳回的命令，並且在 10 的地方得到 ICA 分離出的成分。

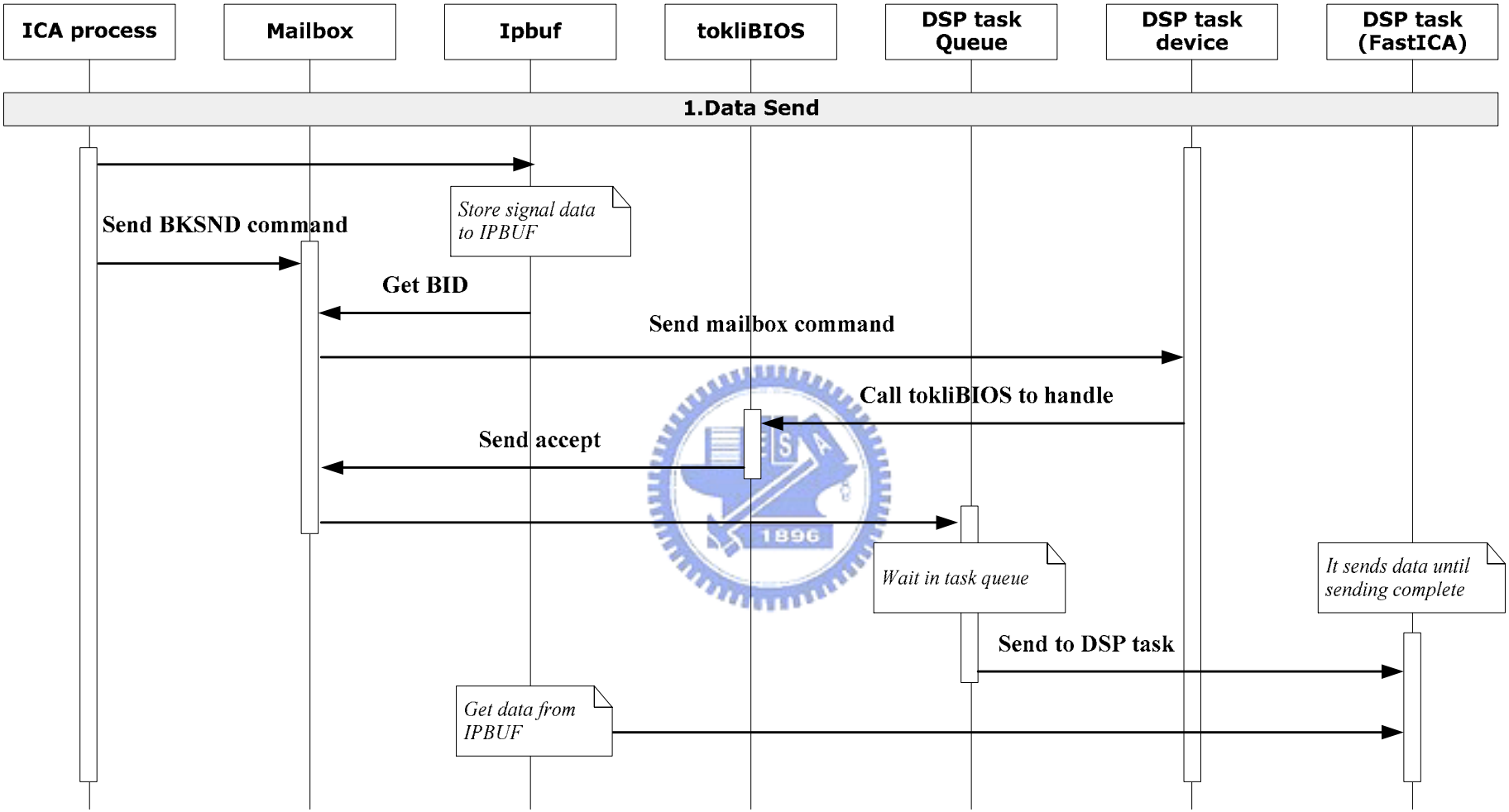

ICA process
Mailbox
Ipbuf
tokliBIOS
DSP task Queue
DSP task device
DSP task (FastICA)
1.Data Send
Store signal data to IPBUF
Send BKSND command
Get BID
Send mailbox command
Call tokliBIOS to handle
Send accept
Wait in task queue
It sends data until sending complete
Send to DSP task
Get data from IPBUF

圖 4-6　資料傳送順序圖

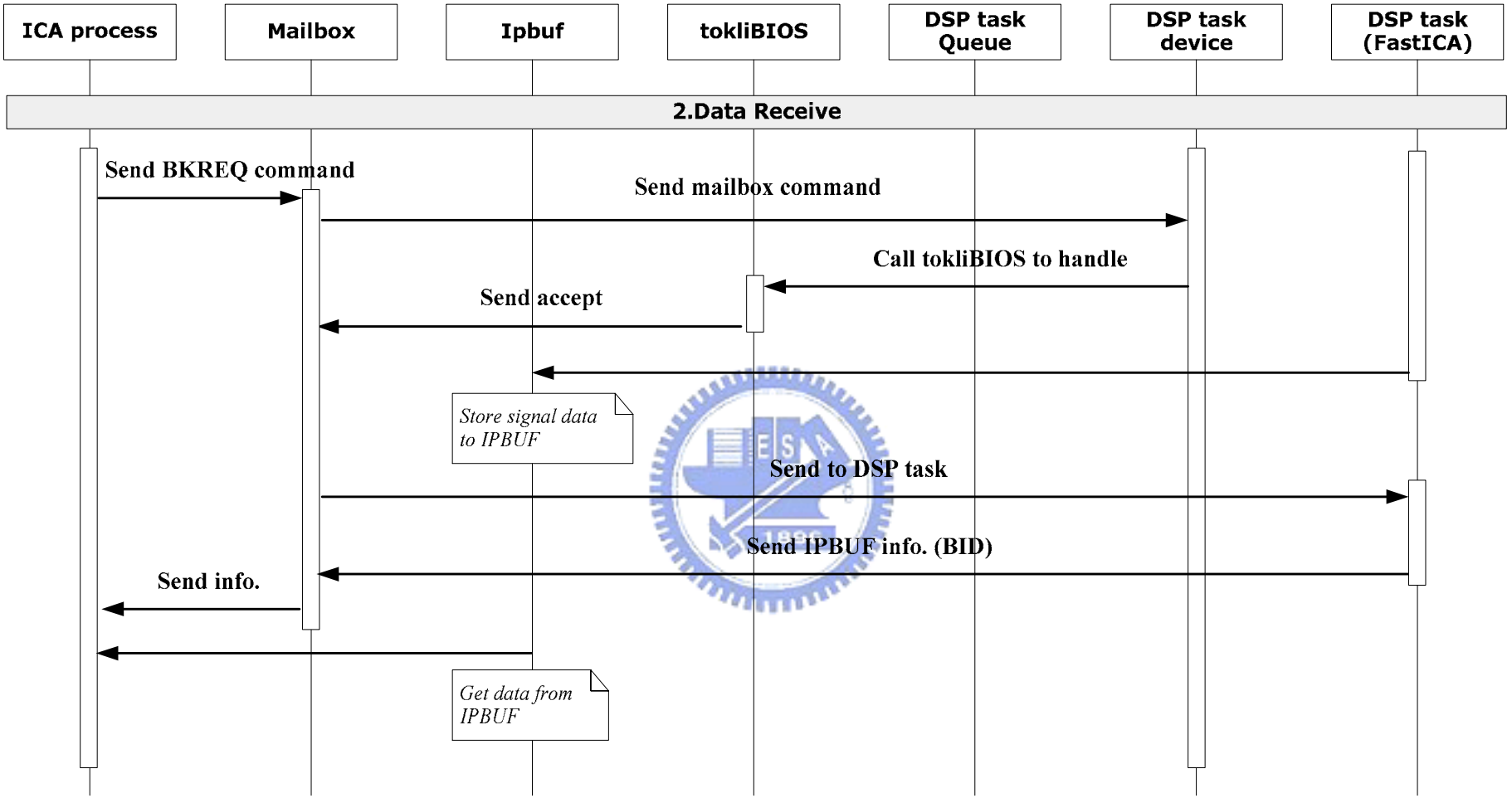

ICA process
Mailbox
Ipbuf
tokliBIOS
DSP task Queue
DSP task device
DSP task (FastICA)
2.Data Receive
Send BKREQ command
Send mailbox command
Call tokliBIOS to handle
Send accept
Store signal data to IPBUF
Send to DSP task
Send IPBUF info. (BID)
Send info.
Get data from IPBUF

圖 4-7　資料接收順序圖

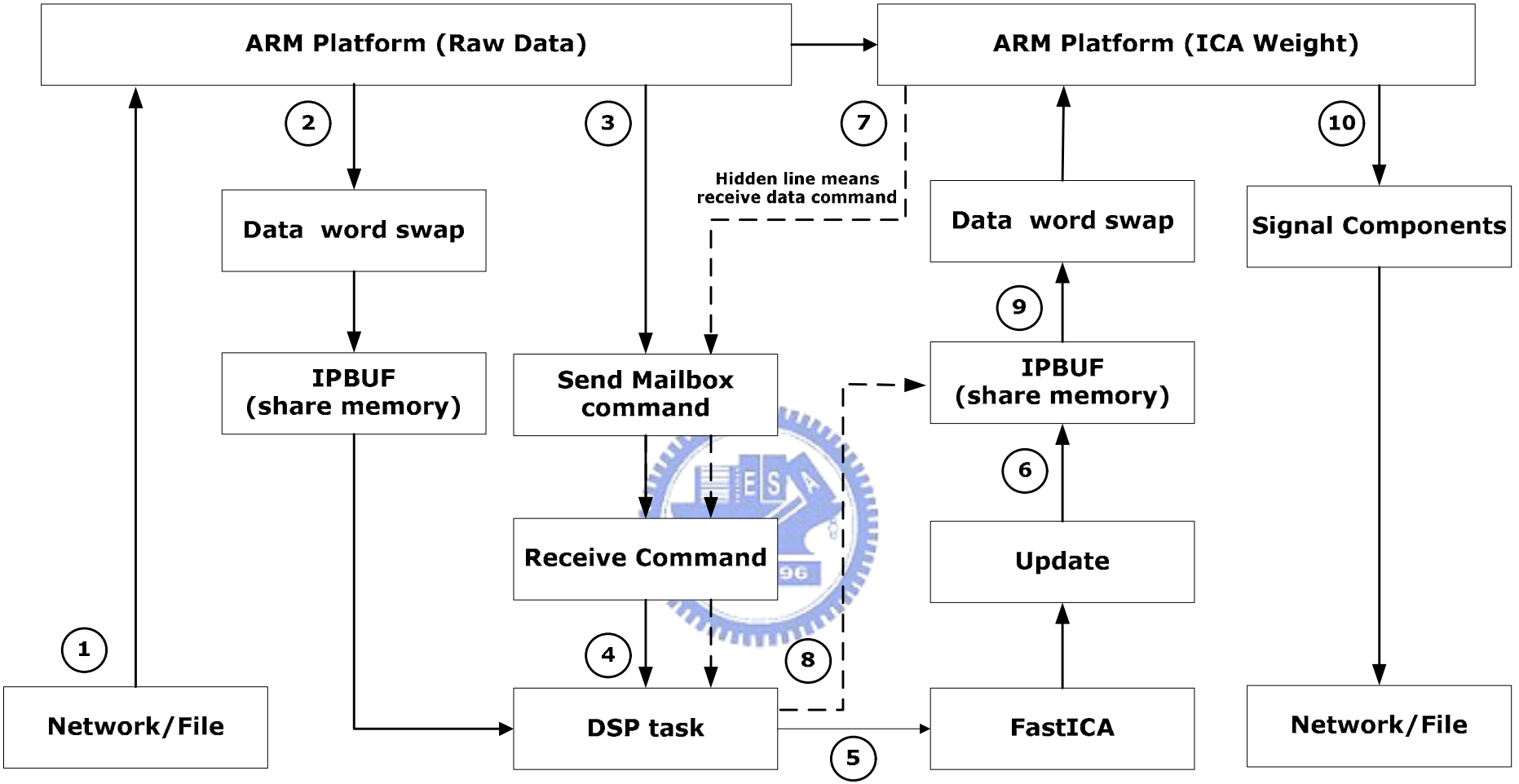


圖 4-8　腦波處理系統資料傳輸及 ICA 處理架構圖

4.4 系統正確性測試實驗

前面完成了整個訊號處理系統的架構分析設計與實現後，為了證實 ICA 在 OMAP 下的正確性，本節將對於 ICA 訊號處理系統做測試。其測試的方式使用將含有 Gaussian、Super-Gaussian 與 Sub-Gaussian 的函數，做為測試對象。先產生一組 Component 訊號，再將這些訊號混合成一不同的訊號，一般經過 ICA 的分析是幾筆資料會產生幾個 Component。這裡測試使用兩種不同訊號的 Component，並將其混合成兩筆要輸入的測試訊號。經過 ICA 測試，將會得到兩個 Component，再將 Component 和原先設定的兩個 Component 比較，是否特性一致。

(1) 實驗說明及實驗方法：

在這個部分，本實驗具有兩個實驗意義，一個是測試訊號的正確性，另外一個是測試分析之效果。將實驗分成實驗組和對照組兩組來比較運算數值的正確性。下面實驗組和對照組實驗平台的硬體和軟體的規劃：

實驗組：硬體—OMAP 平台，ARM 和 DSP 兩個部分並用。

軟體—C 語言的 FastICA，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA。

對照組：(1)硬體—PC 平台，ARM 平台。

軟體—C 語言的 FastICA，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA。

(2)硬體—PC 平台。

軟體—使用 MATLAB 編寫的 FastICA，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA。

實驗訊號主要是以含有 Gaussian、Sub-Gaussian 及 Super-Gaussian 成分的訊號為測試的對象。

圖 4-9為實驗步驟的流程示意圖

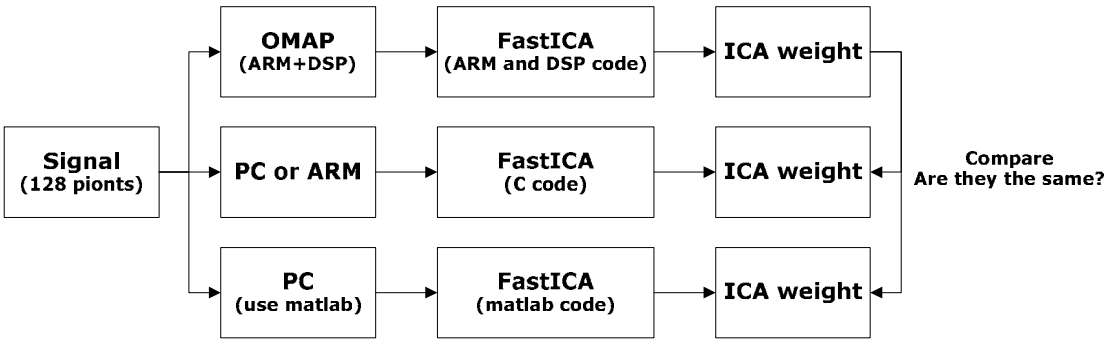


圖 4-9　實驗步驟流程

輸入含有 Gaussian、Sub-Gaussian 和 Super-Gaussian 的訊號，經過圖 4-8 不同的平台計算的方式，再比較 ICA 正確性，最後將資料繪製出來表示。

(2)　實驗結果：

實驗測試：以一個隨機訊號和含有 Super-Gaussian 的訊號混合，再利用 ICA 將成分分離

原始訊號

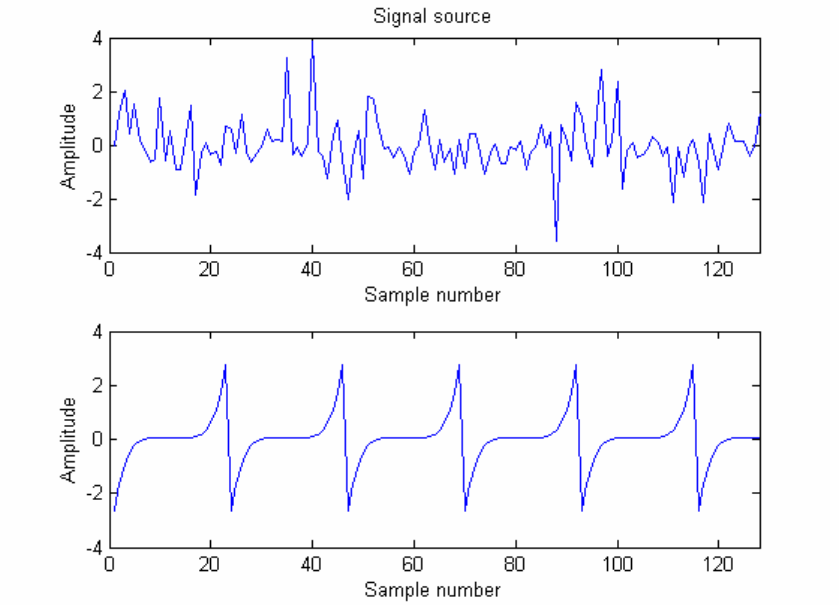


圖 4-10　測訊訊號的原始訊號

ICA 輸入訊號：由原始測試訊號混合之後，再給予 ICA 做分離測試

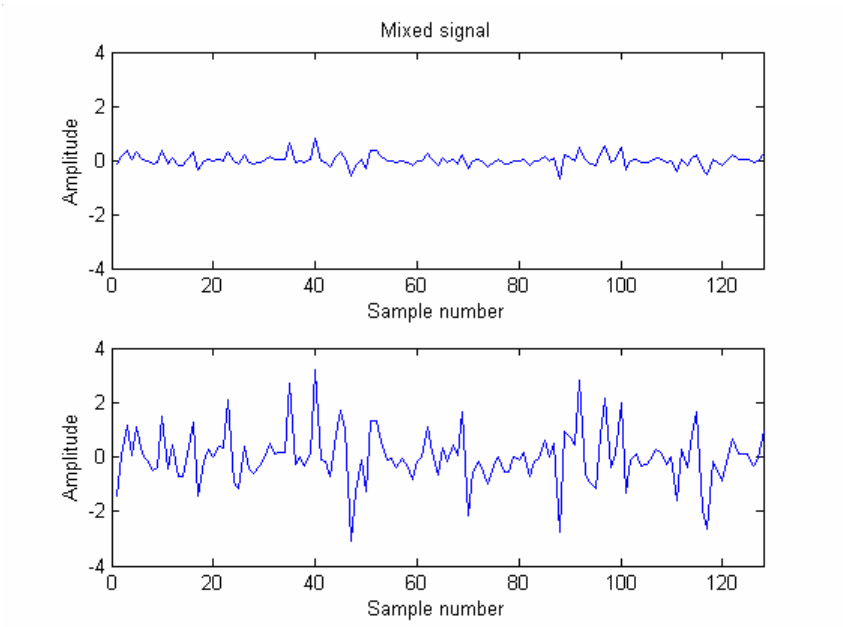


圖 4-11 測試訊號的原始訊號混合

1. 實驗組 FastICA 分離結果

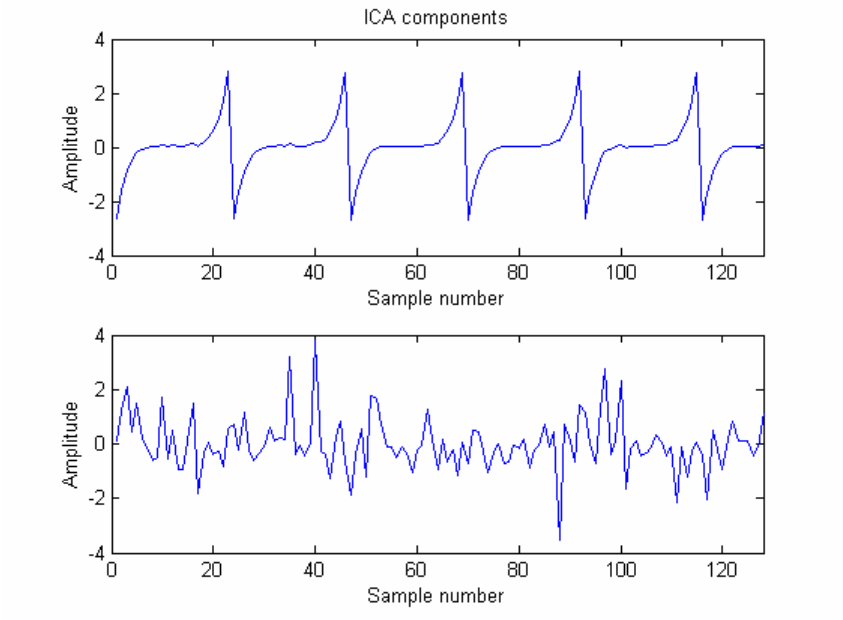


圖 4-12 利用 OMAP 的 ICA 分離結果

2. 對照組 FastICA 分離結果

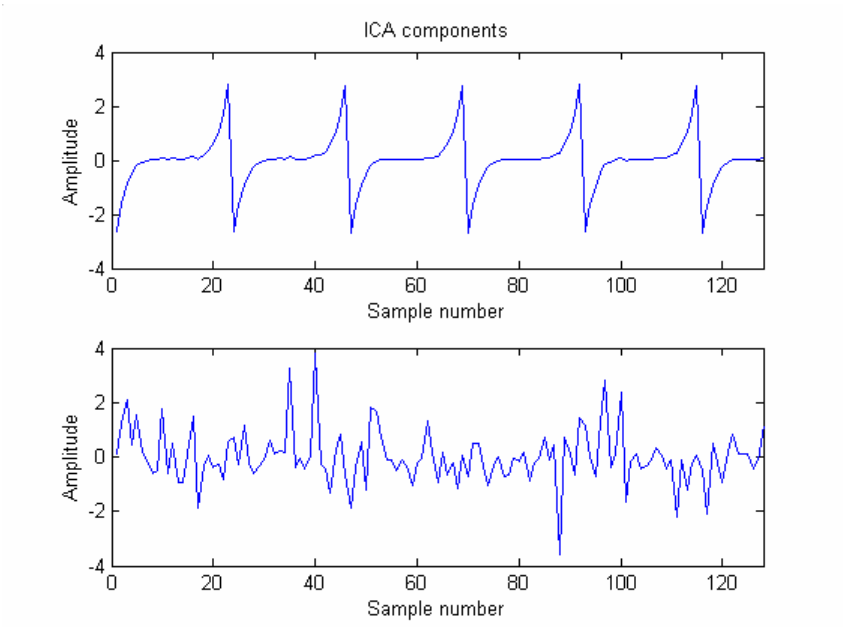


圖 4-13 利用 PC 的 ICA 分離結果

經過比較可以看出 ICA 分離的結果，和原先設定的 Component 相當接近。附錄 C 將會有其他訊號如 Sub-Gasssian 測試的結果。

3. ICA 權重向量

表 4-3 測試訊號的 ICA 權重向量

OMAP(ARM+DSP)	$\mathbf{w} = \begin{bmatrix} -13.015753 & 3.267332 \\ 9.357069 & -1.117348 \end{bmatrix}$
PC or ARM	$\mathbf{w} = \begin{bmatrix} -13.015749 & 3.267330 \\ 9.419549 & -1.117347 \end{bmatrix}$
PC(Matlab)	$\mathbf{w} = \begin{bmatrix} -13.015749 & 3.267330 \\ 9.419549 & -1.117347 \end{bmatrix}$

表 4-3是所得權重向量的結果比較經過比較，其差異度並不大，以這三組來看，比較不同的是 OMAP，其計算有些微的不同是由於 DSP 是定位點的架構，在運算上造成的小誤差，但是整體而言影響並不大，就計算結果來看依然是正確的。

4.5 系統效能測試實驗

經過前面將系統架構實現和實際訊號的測試完成之後，接下來將針對 OMAP 效能做一個測試。OMAP 和一般僅有 ARM 微處理器的嵌入式系統最大的不同在於其運算效能能大幅度的提升。另一個特點是在 DSP 處理資料時，不會影響到 ARM 的部分，此時 ARM 可以處理其他行程，和 DSP 做平行處理，本節將對系統做一個效能測試比較，讓具有 DSP 和不具有 DSP 的系統比較其中的效能差異度。

(1) 實驗說明及實驗方法

在腦波訊號分析系統中，由系統架構規劃，將 ICA 之演算法交由運算能力較佳的 DSP 微處理器來處理。實驗中，除了由 ARM 和 DSP 組成的 ICA 原始碼外，另外，準備了一個僅在 ARM 平台下處理的 ICA。並且比較兩者之間做 ICA 運算時，執行之效能比較。

實驗組：硬體—OMAP 平台，由 ARM 和 DSP 分工

軟體—FastICA 分成 ARM 和 DSP 兩個部分，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA

對照組：(1)硬體—OMAP 平台，僅由 ARM 的部分處理以及 LART 單板

軟體—FastICA 僅使用 ARM 處理，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA

(2)硬體—LART 單板(StrongARM)

軟體—FastICA 僅使用 ARM 處理，可做輸入為兩筆 128 點訊號輸出兩個成分的 FastICA

實驗中，主要是針對 ARM 和 DSP 處理 ICA 運算之部分能力，做比較探討，在測試中，執行所花費的時間，分別可以下面表示：

1. ARM and DSP：

   Total Execution time = ARM execute + Data word swap + Data transfer + FastICA(DSP execute)

2. ARM or LART(StrongARM)：

   Total Execution time = ARM execute(beside FastICA) + FastICA(DSP execute)

圖 4-14表示實驗組和對照組在執行同樣資料的 ICA 運算的時間關係圖

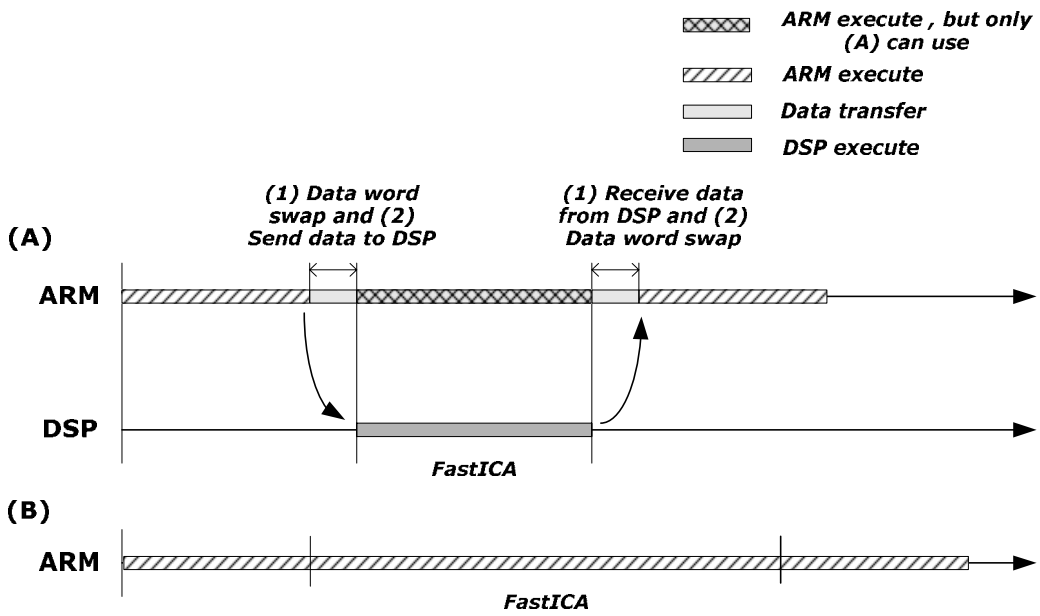


圖 4-14 效能測試中，實驗組和對照組的執行時間表示

(2) 實驗結果：

實驗測試一：前一個實驗的運算效能為測試的目標依圖 4-14所規劃的幾個重要部分進行時間的測試。測試結果，依 5 次計算加以平均的時間。

表 4-4 效能測試表

	Handler	***Excution time***
ARM and DSP	Data transfer	0.004885 sec
	ICA compute	2.297103 sec
	Total execution time	2.339638 sec
ARM	ICA compute	29.548805 sec
	Total execution time	29.575264 sec
LART(StrongARM)	ICA compute	21.917961 sec
	Total execution time	21.926881 sec

在計算上 ICA 計算時間，和計算資料量和一開始所取的隨機矩陣有相當大的關係，所以在測試中，輸入的隨機矩陣都是一樣的。表 4-4為測試結果，ARM 和 DSP 處理 ICA 演算法平均時間為 2.297103 秒，比 ARM 的 29.548805 秒和 StrongARM 的 21.917961 秒分

別快的有 9~11 倍不等的速度。圖 4-15可以很明顯的看出 OMAP 的執行時間遠少於其他兩種平台。

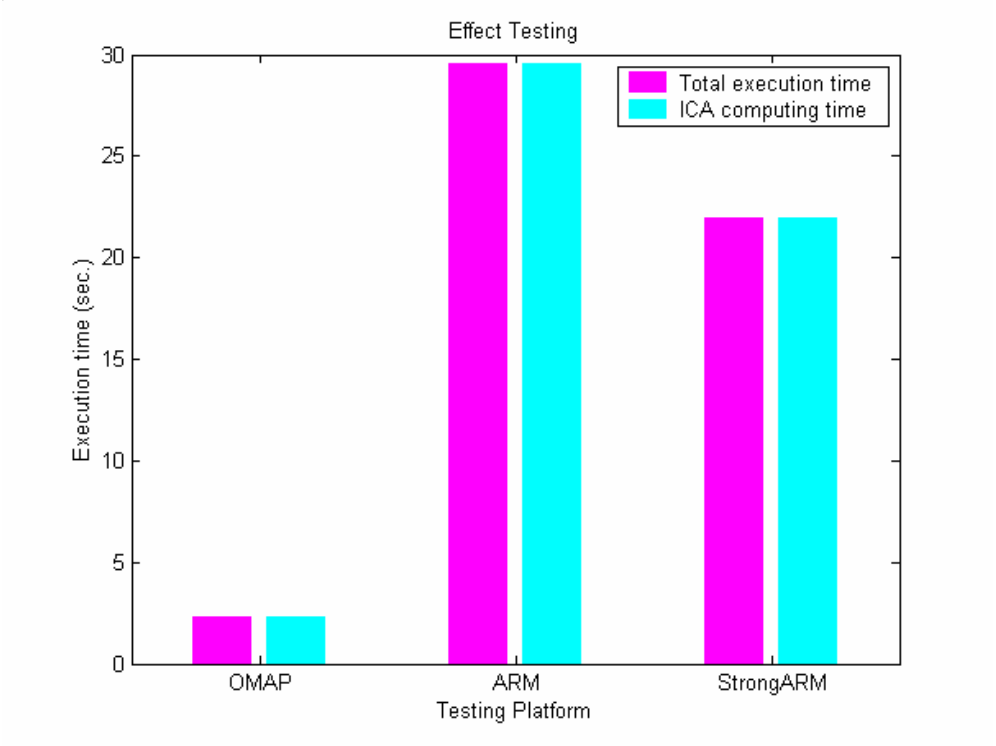


圖 4-15 ICA 計算時間比較圖

五、 研究成果與展望

本論文已分別於第二章介紹 OMAP 的特性及開發環境，第三章介紹 OMAP 內部 ARM 和 DSP 的溝通方式，並於第四章說明整個腦波訊號分析系統架構及設計方法，並且將系統實現及測試，對其效能做一個展示。本章將對整個系統所做過的測試做個回顧並且做延伸分析探討，最後實際應用於腦波訊號的狀況及分析結果。選擇一個的受測者受到外部刺激所產生的腦波反應訊號，並分析其腦波訊號之意義，將其訊號分離出獨立特徵成分，並且利用 FFT 得到其頻譜分析，之後將會提出腦波訊號處理系統、腦波量測的系統與虛擬實境之整合，論文最後將對未來的方向進行說明。以下分別就「研究成果」、「系統整合」、「發展討論」及「未來展望」進行說明。

5.1 研究成果

本論文目前為止已完成腦波處理系統，並於前一章節末端對整個系統正確性和系統效能做個測試，實際展現出 OMAP 比一般嵌入式微處理器優秀之處。以下會先對前述的成果做個整理，並且將對整個測試做延伸的探討，最後應用於將腦波量測到的訊號實際使用腦波處理系統去除，得到腦波成分，並且可以用 FFT 轉換成頻域觀察其頻譜分佈。

5.1.1 實驗測試回顧及延伸探討

在前一個章節中，本論文對腦波處理系統的特性做一連串的測試，其中包括系統正確性的測試和系統效能的測試，以下將對這兩個測試結果做個整理，並且舉出和上一章節略有不同的結果探討，讓整體測試情形更為完整。

1 系統正確性：

系統正確性測試架構由前一章節圖 4-9表示，在不同的平台上將測試結果做比較，是否一致。除了前面章節中以含 Super-Gaussian 的訊號為例，在這裡將再提供一些其他不同的訊號測試。

(1) 輸入一組含有 sin 波的雜訊，利用 ICA 將 sin 波分離出來

輸入 ICA 的訊號　　ICA 分離出訊號

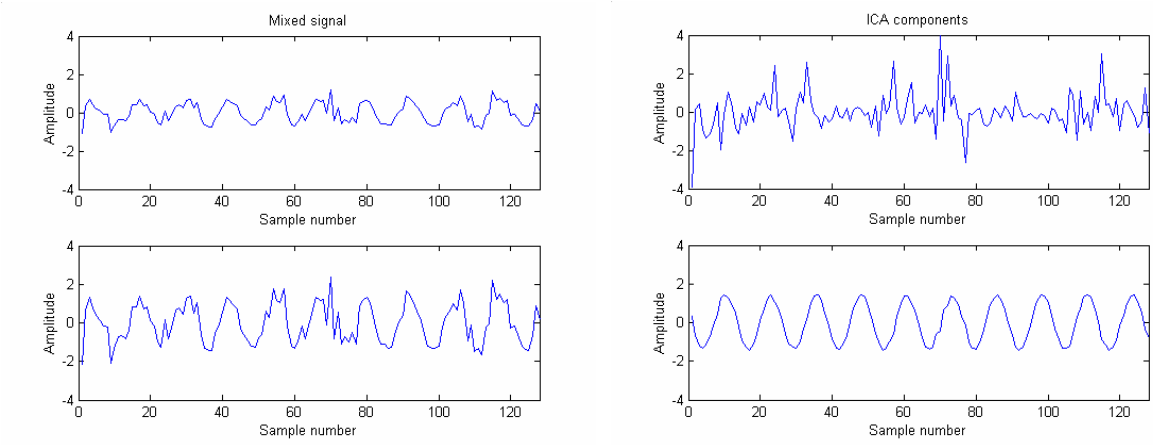


圖 5-1　ICA 雜訊去除測試一

表 5-1　權重向量比較表

OMAP(ARM+DSP)	**w** = [99.37367 -49.87024; 189.7796 -96.86836]
PC(Matlab)	**w** = [-191.125001 97.542218; -99.557755 49.708948]

所得到的權重向量來看，和前一章測試略有不同，就是權重向量內的數值位置和正負號位置有所不同。但事實上，這只是 ICA 演算法上寫法的問題，在 PC 的 Matlab 上有關於演算法的特徵值求解以及矩陣開方根等所用的運算在 Matlab 上都已經有相對應之工具，而在 OMAP 嵌入式單板上，並沒有這些特殊的數學函式庫，需要自己根據一些線性代數的原理去寫撰寫這些函式庫，故在數值求解的判斷上會有所差異，但是實值上是一樣的，可以由對角相乘和鄰近的關係看出。另外數值上大小的微小差異是由於 DSP 是採用定點數之故，所產生的小誤差。但在 ICA Testing 模式下，所得到的 ICA 成分圖形並無所異，均可成功的將 sin 的訊號分離出來。

以下再針對其他的輸入範例，做 ICA 分離的結果展示。

(2) 輸入一組含有鋸齒波形的雜訊，利用 ICA 將其分離出來

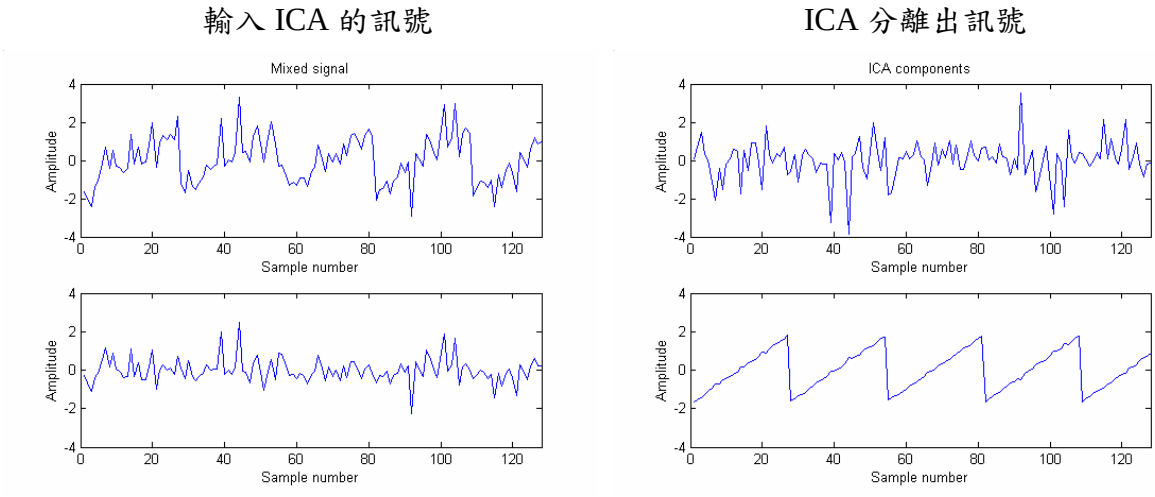


圖 5-2 ICA 雜訊去除測試二

(3) 輸入一組含有兩種規律波形的訊號，利用 ICA 將其分離出來

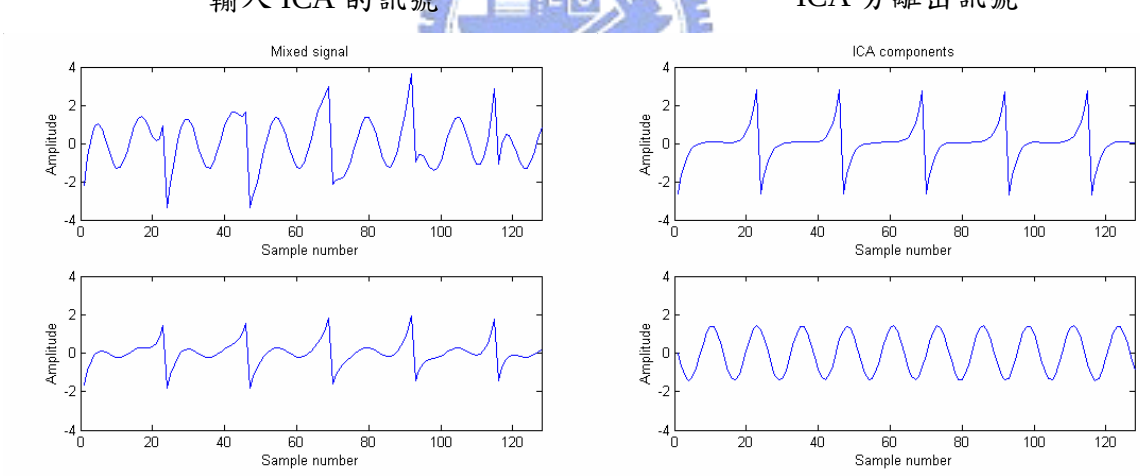


圖 5-3 ICA 雜訊去除測試三

再經過上面三個測試之後，除了分離的正確性之外，可以得到一個在 ICA 學習的共通點。這個共通點可以做為腦波用 ICA 分類選取的要點。這個共通點是在圖 5-1~圖 5-3中，兩個訊號其中之一個是類似做參考訊號的功用，而將另外一個分離出來。

2　系統效能：

系統工作執行如前一章節圖 4-14表示，OMAP 和一般微處理器最重要的一點是其效能較一般常見的嵌入式系統微處理器還要高。

表 5-2　效能測試整理

	Handler	***Testing1 Excution time(average)***	***Testing2 Excution time(average)***
OMAP	ICA compute	2.297103 sec	2.757421 sec
ARM	ICA compute	29.548805 sec	27.955059 sec
LART(StrongARM)	ICA compute	21.917961 sec	20.545351 sec

上面的數據是依照系統正確性測試所得到的實驗數據。

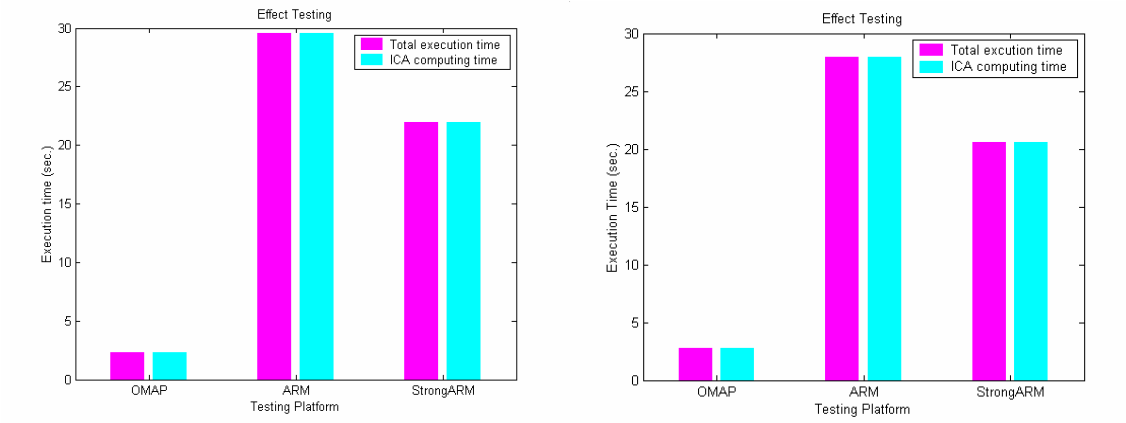


圖 5-4　ICA 計算時間和總執行時間比較圖

圖 5-4第一張圖為前面章節效能測試時間所整理出的比較圖其值如表 5-2測試一，第二章圖為另一個訊號所測試出來的結果其值如表 5-2測試二。證明了 OMAP 的效能比其他兩個平台快了十倍左右。另外，使用 OMAP 也大幅降低了 ARM 的負荷，利用 Linux 內部指令，當只使用 ARM 做 ICA 運算時其 CPU 負荷會達到 60%以上，而如果交由 DSP 運算，則情況正好相反，ARM 幾乎處於空載的狀態。如此一來，便可利用這一段時間來處理其他的行程。下面系統平行處理探討，將對此做個說明。

3　系統平行處理探討：

系統平行處理探討，主要是說明有關於當 ARM 將資料交由 DSP 運算時，如果在這段時間 ARM 處理其他的事情，是否會影響到整體運算時間。圖 5-5 中本實驗當 ARM 交資料給 DSP 運算 ICA 的時間，同時在 ARM 以一個 1 乘到 100 的運算和 sleep 函式來模擬 ARM 同時在執行，發現執行時間並不會受到影響，除非 ARM 在這段時間執行時間過久，而延後了接回 DSP 運算結果的時間。至少由此可以知道，利用 OMAP 不僅效能較一般的嵌入式處理器好，另一方面在執行中多了一段可以平行處理的時間。

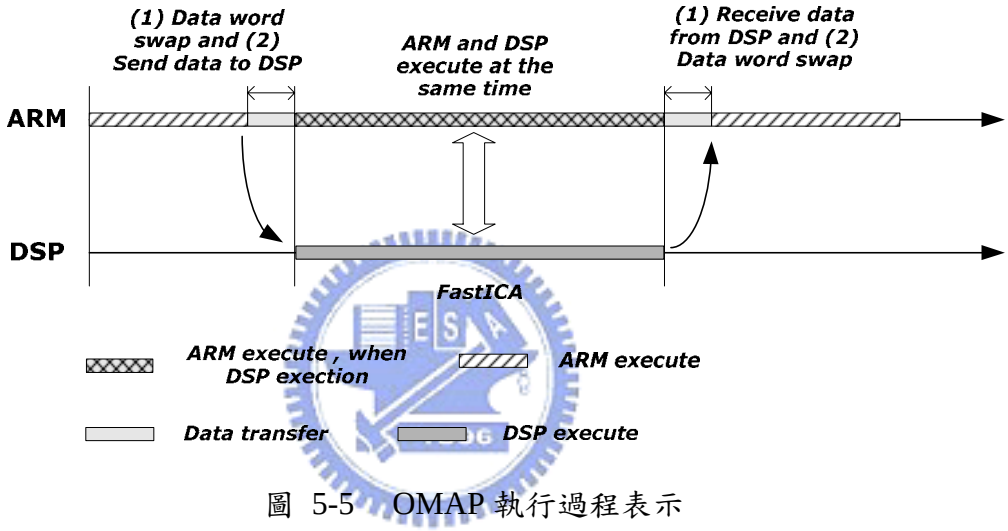


圖 5-5　OMAP 執行過程表示

5.1.2 ICA 腦波訊號分離

在上一小節中，ICA 正確性的測試中，最後曾經提到進行 ICA 分離時，訊號選取其中一個做為參考用，經過 ICA 分離出另一個訊號該有的成分，這是一個利用 ICA 做雜訊去除的基準。在這一小節中，將以此基準選取需分離雜訊的腦波資料，做分離測試。

就此系統，給定腦波量測到的訊號，用 ICA 將雜訊去除，得到腦波訊號的特徵。展示方式如下：

輸入：2 筆 128 點的腦波訊號。

輸出：2 個腦波訊號的成分。

腦波的資料區段的擷取是以人接受一個刺激點為基準，如特殊的燈光。往前取 200ms，往後取 1sec，共 1.2sec 長度的腦波訊號。其中腦波訊號給 ICA 運算前，先經過一些前處

理，把較大的雜訊經過一個低通濾波濾除。再將取樣頻率降低，得到 128 點的訊號資料。本試驗意在利用 ICA 將腦波中的雜訊去除，留下腦波訊號原本該有的資訊，例如，眼動訊號的雜訊去除、心跳訊號的雜訊去除。

實驗中選擇一眼動附近的腦波訊號，做為雜訊的參考依據，和一受眼動訊號影響的腦波訊號一起經過 ICA 運算。其輸入訊號和運算的結果如圖 5-6和圖 5-7。圖 5-6中，眼動雜訊參考為第一個訊號，而第二個訊號則是受到眼動影響的腦波訊號，經過 ICA 運算結果，圖 5-7中所得第一個訊號應該是去除眼動雜訊所留下腦波訊號的資訊，第二個訊號則是分離出的眼動雜訊。除了眼動之外，同樣的方式也可以用於去除心跳等雜訊，而得到正確的腦波訊號的資訊。

(1) 腦波訊號分離測試一

腦波輸入訊號

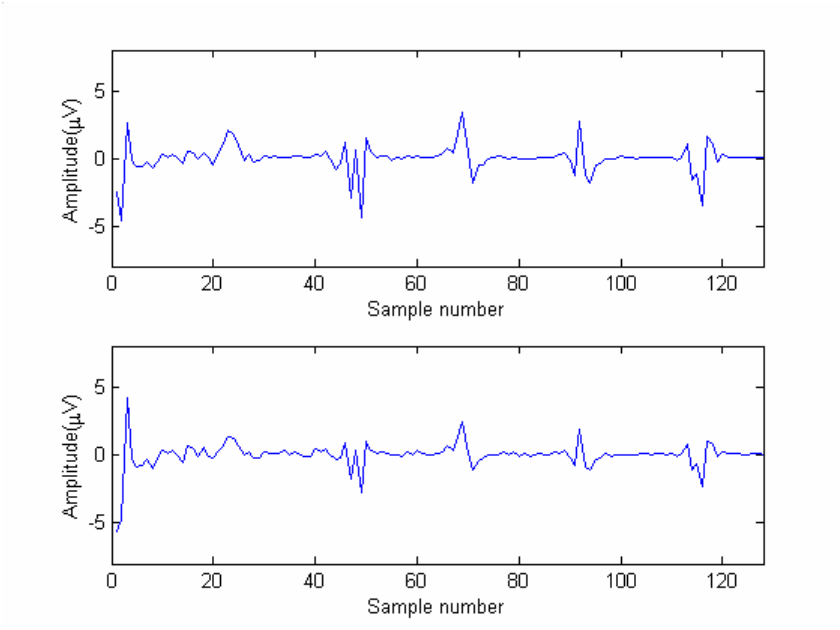


圖 5-6　輸入腦波測試訊號於 ICA(測試一)

圖 5-6為兩個訊號，其中訊號是在眼動附近電極所量測到的訊號，而訊號二是離眼動較訊號一較遠的電極所量到的訊號，推斷其應含有一些腦波資訊，故用 ICA 將其訊號分離，經過調降取樣頻率，取 128 點的訊號資料，把高雜訊先事先濾除，留下的較小的雜訊，再用 ICA 做細部的去除。

經 ICA 所得到的 ICA Components

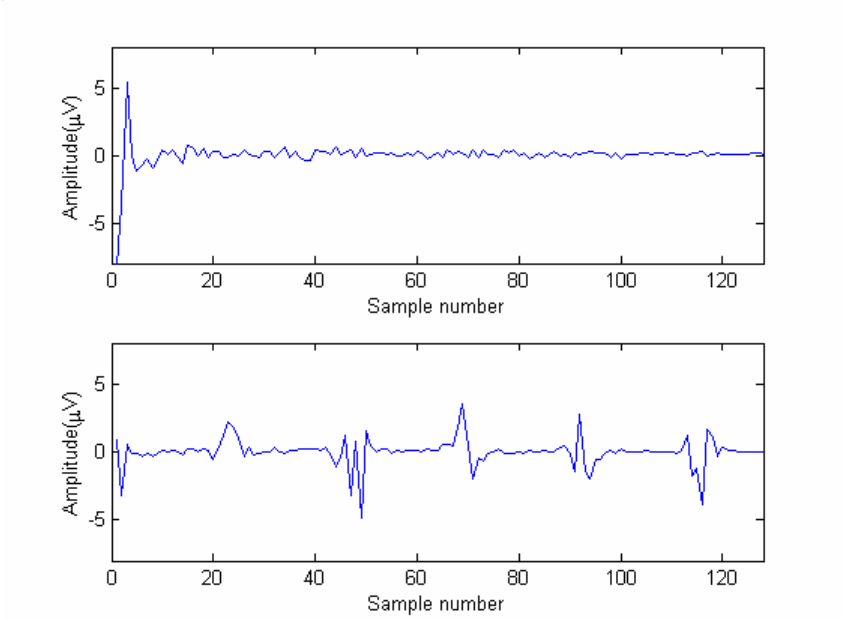


圖 5-7　由 ICA 得到腦波的成分資料(測試一)

圖 5-7為經過 ICA 所得到的結果，分離出的成分一，應該是屬於腦波該有的資訊，而成分二是屬於雜訊的部分。

(2)　腦波訊號分離測試二

腦波輸入訊號

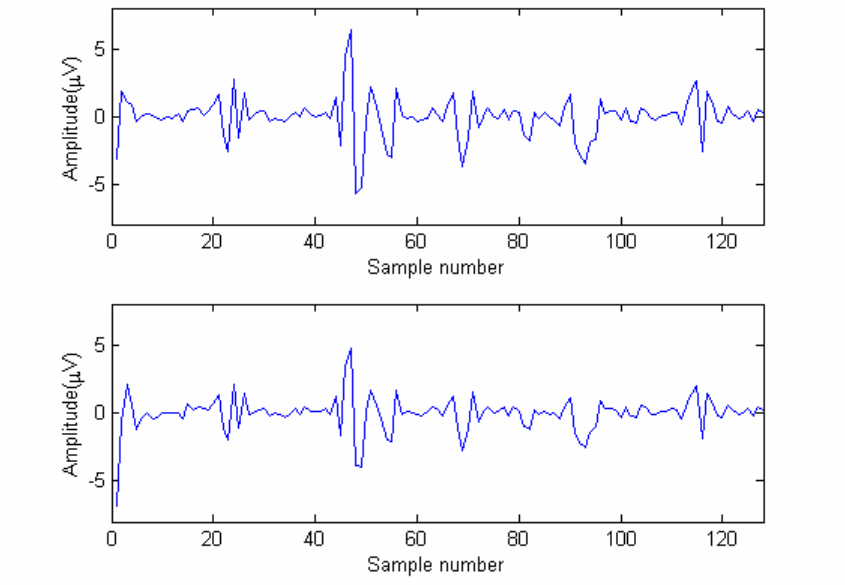


圖 5-8　輸入腦波測試訊號於 ICA(測試二)

經 ICA 所得到的 ICA Components

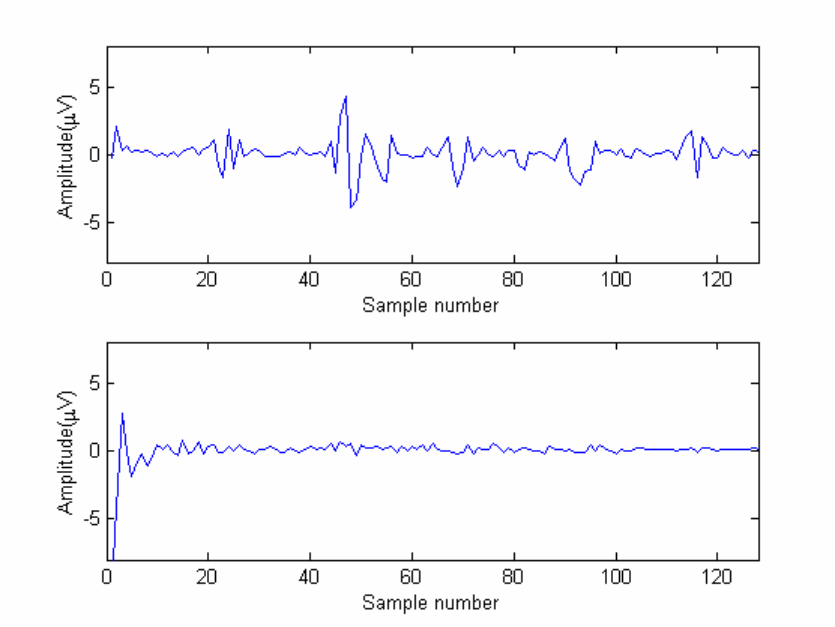


圖 5-9　由 ICA 得到腦波的成分資料(測試二)

為了說明測試一分離出的腦波應該是正確的在測試二和測試一相似，只是在振幅上略有差別，其中時間差也經前處理已調至正確的位置，由於測試二和測試一是由相同的點，做相同的刺激所做得到的訊號，經 ICA 分離之後，測試一和測試二的前 15 點有類似的訊號，推測這應該是腦波原有的訊號。

經前面分離後，接下來的工作便是分析腦波所代表可能的意義。在一些情況下，會有 ICA 分離不佳的情形，前一章有提到，這可能是因為 ICA 內所取用的非二次函數的關係，如針對一些例子改變其中非二次函數，可能會改善學習的結果。

5.2 系統整合

整個腦波分析系統結合虛擬實境的整合圖，如圖 5-10所示。系統包括：虛擬動態平台、虛擬場景、嵌入式腦波訊號處理系統及腦波量測系統部分。本論文主要是針對腦波訊號處理系統的部分發展一個小型嵌入式系統，在其環境上開發，以便未來技術上直接移植於一個量測和分析處理整合的小型嵌入式系統。整個整合圖的流程是，利用虛擬場景和史都華平台，產生一些模擬實際狀態，例如：駕車狀況。讓受測者於虛擬的狀態中感受到真實情況刺激，並且將腦波訊號記錄下來。將訊號經過一些前處理後，交由嵌入式訊號處理系統來做腦波訊號的處理，並且可以利用 ICA 將腦波資料做雜訊去除，另外，亦可由嵌入式單板 FFT 可得頻譜資料。

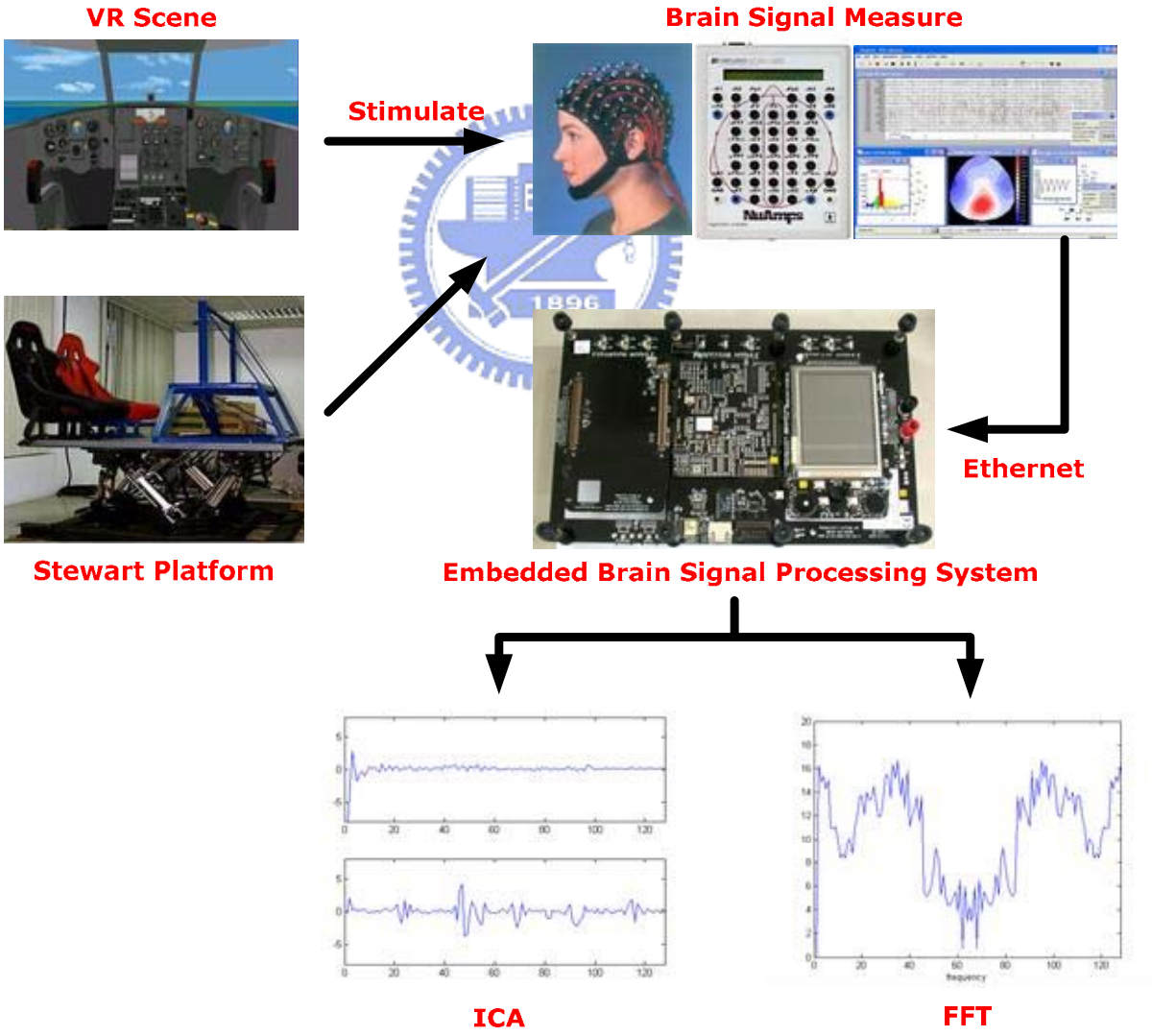


圖 5-10 系統整合架構圖

5.3 發展討論

本論文研究，主要是將 OMAP 應用於訊號分析處理上的實現，ARM 配合 DSP 系統效能展示，證實了 OMAP 效能在訊號處理上的確領先於一般 ARM 架構的微處理器。如此一來，來演算法上便可以不必因為牽就於處理器的能力，而降底其精確度的要求。雖然目前 OMAP 才剛有初步成果，但 TI 仍在積極發展新一代的 OMAP 處理器，依照不同的需求，TI 分別在不同型號的 OMAP 處理器上加上不同的功能，如繪圖硬體加速的功能等，近來 TI 也推出 OMAP 2 處理器。未來相信 OMAP 將會是嵌入式系統的主流之一，不管在 PDA、無線通訊上，其令人望其項背的運算能力，的確將是其一大優勢。目前國外許多著名廠商紛紛投入 OMAP 產品的開發，國內亦有許多廠商對此保有高度興趣。本論文選擇以 OMAP 平台，一方面是有鑑於舊系統能力對於訊號處理之能力嚴重不足，另一方面是嘗試開發測試新的研究平台，結果依所預期的，有長足的進步。在這之後，將會整合量測系統，如此一來，可以將整個腦波分析系統微小化，便於移植到任何的地點，做隨身的腦波訊號分析，利用腦波訊號的特徵加以分類腦波代表的意義，可應用於控制上的目標。雖然目前仍處於最初的階段，但已有一個雛形。

本研究中，較為困難的地方在於，如何從一個只有硬體的系統，在上面建構作業系統，並且建立程式開發環境，不同於以往在嵌入式系統建 Linux 的部分，在 OMAP 上需要做一些步驟上的修正。由於 Linux 龐大的社群自由軟體支援解答下，如 ARM 的 mailing list，找到了經過討論的解決方式。另外最重要的，在 ARM 和 DSP 溝通上的 DSP Gateway，使用了作業系統架構的知識，以及系統驅動程式觀念整合，設計上需要一步步的釐清其定義、可用的資料和傳送的流程，以便能撰寫出正確的程式。在撰寫過程中，由於目前相關於這方面的資料並不多，有很多地方都是經過推敲，慢慢找出規則，修正錯誤。例如在撰寫 DSP code 時，遇到了一些想不到的問題，雖然語法上沒有錯誤，但就硬體上的配置會有錯誤的發生，例如變數定義上的問題，全域和區域可給定的變數記憶體大小不同，如果將 ICA 資料存於區域變數，可能會出現錯誤，另外在 DSP 中呼叫 subroutine 有因為這個原因，而會有變數傳輸上的問題。在撰寫時要特別的注意。

5.4 未來展望

鑑於訊號分析需要複雜的運算，例如本論文中的腦波處理，在以往嵌入式系統的微處理器，並沒有如同 DSP 擁有優秀的運算能力，處理上受到能力的限制，不僅在時間上與功能上無法做到較為精確的運算結果，然而 DSP 卻有對於作業系統並不支援，造成在設計上操作不易，以及移植性的問題。本論文採用 OMAP 解決此一問題，大幅度的提升嵌入式系統的處理效能，以及保有對作業系統的支援度。但整個嵌入式腦波分析系統，仍有一些不足所在，有待改善，以下將列舉一些未來有待改進的方向：

1. 整合架構：

   本論文中整個腦波處理架構而言，並不包含腦波量測的工具，在資料的取得上仍然需要透過一些介面傳輸，例如：網路。然而為了實現將腦波訊號處理系統能整體的微小化，能攜於身上做隨時的腦波訊號分析，並且可以利用分析之腦波應用於控制電腦，做為腦機介面(Brain-Computer Interface)，勢必要將系統量測的部分也加入，這是未來可以參考發展的地方。

2. 深入效能高演算的方式：

   本論文中腦波訊號處理系統中，雖然運算效能上有大幅度的提升，但是對於運算上還是有資源浪費的情況，舉例來說，當 ARM 將工作交由 DSP 處理時，這時 ARM 仍然可以做其他的工作，且並不會影響到 DSP 的執行能力，而這段時間所做的工作時間，是可以調整的，將最大的工作份量加入，將會得到最佳的效能的提升。

3. 運算的修正：

   本論文中腦波分析系統所使用的微處理器是 OMAP1510，是結合 ARM9 和 C55x DSP 系列的處理器，但 C55x DSP 雖有強大的運算處理能力，但依舊是定點數的處理器，在處理如 ICA 的複雜運算時，會有些微的誤差。如果需要做更精密的運算，必需要將演算方式，針對定點的問題做一個修正。

4. DSP 資料配置：

   因為 DSP 在處理時可用資源有限，故對於一些記憶體空間的配置需要調整，減少不必要的空間浪費。一方面，在處理上會更加有效率，另一方面，可以利用剩下的空間給於其他 DSP task 配置使用。

參考文獻

[1] Toshihiro Kobayashi, Kiyotaka Takahashi, ***Linux DSP Gateway Specification*** Rev2.0, Nokia Corporation, November 13 2003.

[2] ***Innovator Development Kit for the OMAP Platform User's Guide***(SPRU667), Texas Instruments.

[3] ***Innovator Development Kit for the Texas Instruments OMAP™ Platform Deluxe Model User's Guide***, Texas Instruments.

[4] ***TMS320 DSP/BIOS User's Guide*** Rev B(SPRU423B), Texas Instruments.

[5] ***TMS320C55x Optimizing C/C++ Compiler User's Guide*** Rev E(SPRU281E) Texas Instruments.

[6] ***Code Composer Studio Getting Started Guide*** Rev C(SPRU509C), Texas Instruments.

[7] Alessandro Rubini, Jonathan Corbet, ***Linux Device Driver*** 2nd Edition, O'Reilly, June 2001.

[8] Karim Yaghmour(2003), ***Building Embedded Linux Systems***, O'Reilly, April 2003.

[9] W. Richard Stevens, ***Advanced Programming in the UNIX Environment***, Addison-Wesley, June 1992.

[10] Kay A. Robbins, Steven Robbins, ***UNIX Systems Programming-Communication, Cocurrency, and Threads***, Prentice Hall, 1996.

[11] Ashfaq A. Khan, ***Practical Linux Programming：Device Driver, Embedded Systems, and the Internet***, Charles River Media, 2002.

[12] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, ***Operation System Concept***, John Wiley & Sons, 2003.

[13] Aapo Hyvärinen, Erkki Oja, Independent Component Analysis : A Tutorial ***Neural Networks***, 13(4-5):411-430, 2000.

[14] Grady Booch, James Rumbaugh, Ivar Jacobson, ***The Unified Modeling Language User Guide***, 張裕益譯，UML 使用手冊，臺北縣：博碩文化，2001。

[15] Wayne Wolf, ***Computers as Components : Principles of Embedded Computing System Design***, Morgan Kaufmann, 2001.

[16] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, ***Discrete-Time Signal Processing*** Second Edition, Prentice Hall, 1999.

[17] TI OMAP™ Platform, http://focus.ti.com/omap/docs/omaphomepage.tsp

[18] rrload Bootloader, http://www.ridgerun.com/bsp/rrload.html

[19] Neuro Scan, http://www.neuro.com/

[20] 廖志昇，”虛擬實境動態模擬系統之即時嵌入式控制單板研製”，國立交通大學，碩士論文，民國九十一年。

[21] 陳建宏，”虛擬實境動態模擬器之嵌入式即時控制系統”，國立交通大學，碩士論文，民國九十二年。

附　　錄

附錄Ａ　硬體實驗環境

本論文所採用的硬體平台是 TI 的 Innovator Development Kit 發展板，以下針對此發展板做一個說明。

A.1　Innovator Development Kit

Innovator Development Kit 是 TI 為 OMAP 推出的發展平台，Innovator Development Kit 是很有彈性的發展與展示平台，不但支援所有主要無線標準，並能協助發展廠商在常用作業系統下開發各種應用，讓發展廠商使用所有主要的程式設計環境。

Innovator Development Kit 採用模組化硬體架構，將一套開放式軟體發展與展示環境提供給應用發展商。界面模組包含多項獨特界面功能，可以讓廠商對發展套件進行客製設定；另外還有可選用擴充模組，它們會支援 GSM/GPRS、802.11b 和藍芽，一張附加電路板（breakout board）則包含其它外部硬體和一組 10Mb 乙太網路連線。OMAP1510 處理器內含 TI 功能強大的 DSP 核心，應用軟體設計工程師只需透過 Innovator Development Kit 電路基板的協助，就能利用高階作業系統發展工具，獲得 DSP 所提供的媒體處理加強功能。

Innovator Delopment Kit 硬體包含三個部分，Processor Module、Interface Module 和 Expansion Module。主要的特性如下：

- TI OMAP1510 微處理器
- 32Mbytes SDRAM
- 32Mbytes User Flash
- 4MByte/256KBtyes Boot Flash/RAM
- 支援 2 Serial Ports，Ethernet，JTAG，USB，Audio、SD/MMC 等裝置介面
- 具有低功率，高效能的特性

其積體電路是採用模組化的設計方式，可以將硬體功能獨立，將沒有到的硬體抽換調整。另外 Expansion Module 提供了擴充的介面，參照 TI 所給的配置方式設計硬體，可以利用將此板的功能擴充。以下將分別介面 Processor Module 和 Interface Module 的部分。

A.2 處理器模組 Processor Module

處理器模組，顧名思意主要是包括 OMAP 微處理器的模組部分，除了處理器之外，一些系統記憶體。其中含 32Mbytes 的 SDRAM 供系統程式運作資料運算使用。32Mbytes 的 User Flash 可以區分為兩個 User Flash，分為別 User Flash0 和 User Flash1，而這個個 Flash 可以利用模組上的 switch 來切換儲存的目標，供系統資料的儲存，例如：DSPLinux 的核心，rrload bootloader。另外還有 JTAG 的介面，可以透過 JTAG 將 CCS 編出的程式碼載入到 Innovator Develoepment Kit 上。圖 A-1和圖 A-2分別是 Processor Module 的正反面。

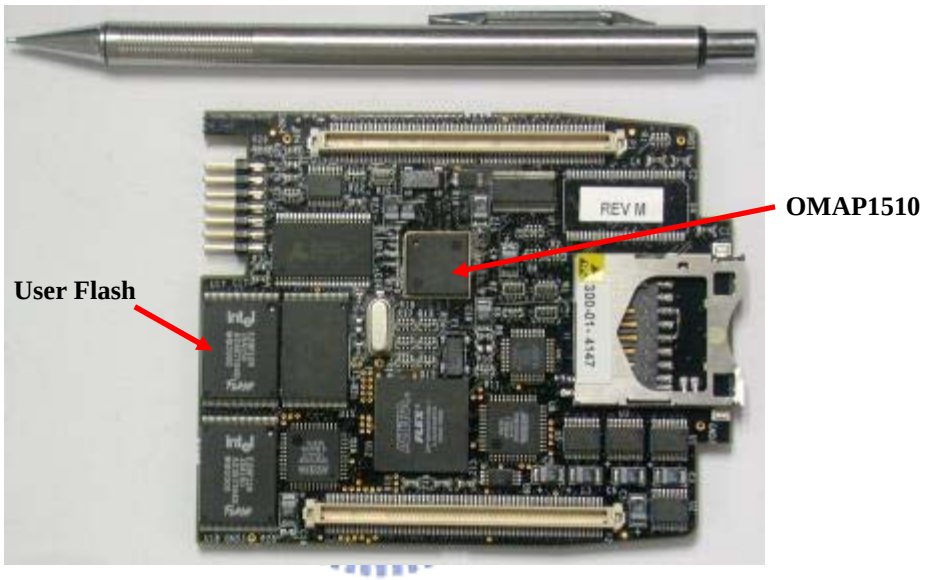


圖 A-1 Innovator Development Kit 處理器模組正面

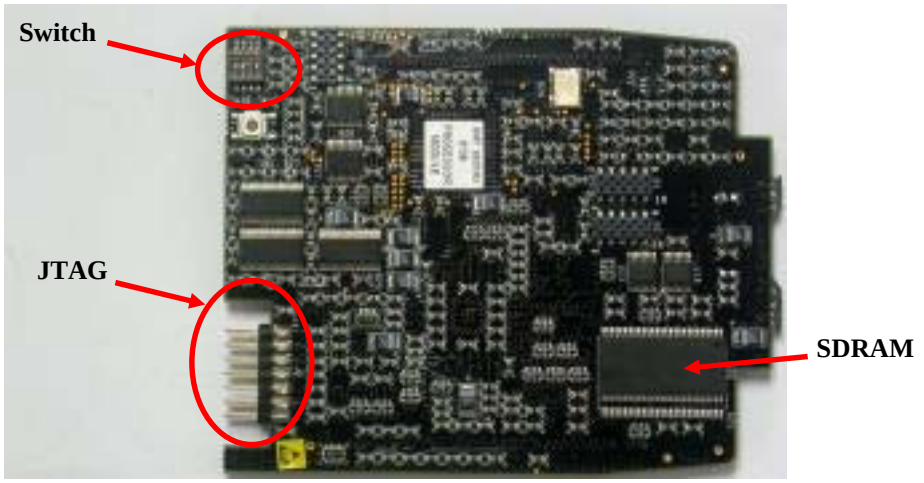


圖 A-2 Innovator Development Kit 處理器模組反面

A.3　週邊介面模組 Interface Module

Interface Module 是 Innovator Development Kit 的週邊模組如圖 A-3所示，一般溝通的介面 I/O 都在這個模組上，以下是介面模組上的裝置列表：

- 觸控式 LCD 螢幕，及四個控制按鈕
- Daul RS232 的埠
- 聲音連結器，立體聲放大器
- Digital Camara 介面
- USB Host 及 Client 的埠
- Power on/off

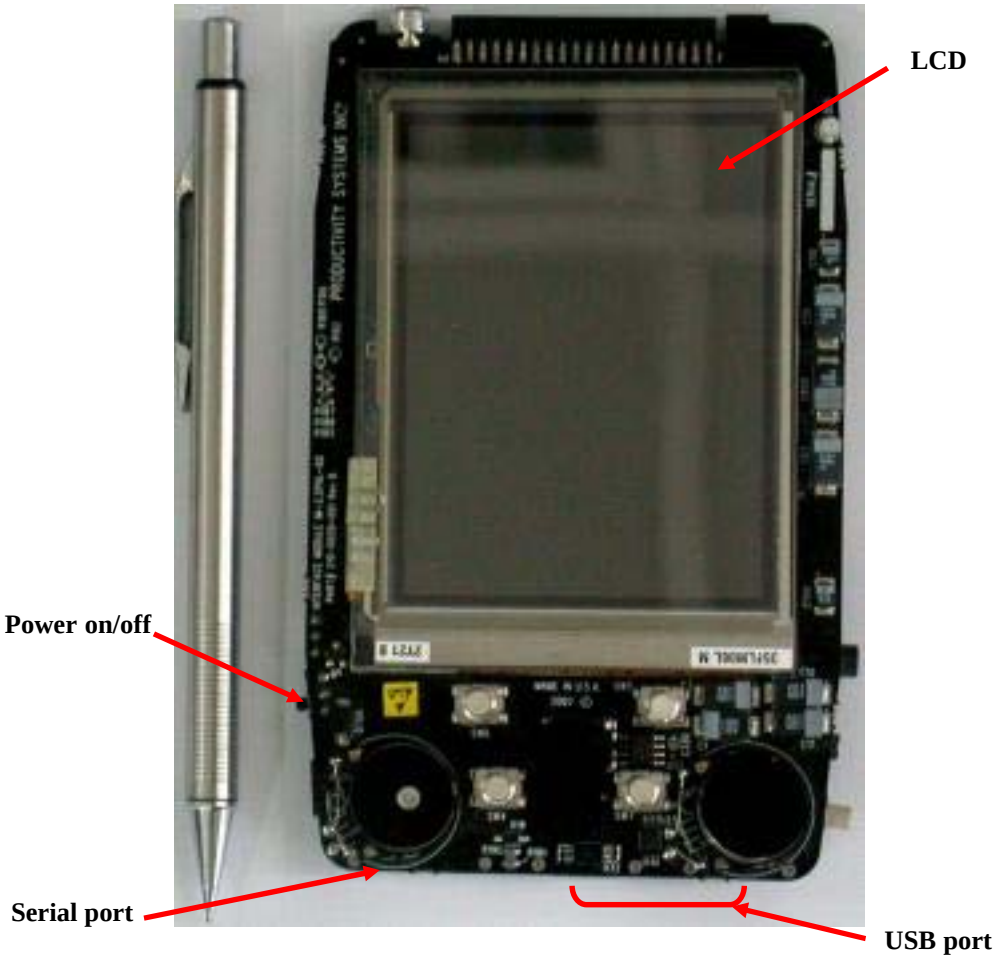


圖 A-3　Innovator Development Kit 介面模組

A.4 Break Out Board

在前面已經介紹 Innovator Development Kit 的模組，另外 TI 針對 Innovator Development Kit 在發展上的便利性，便開發了 Break Out Board(BOB)的套件，Break Out Board 是可以將 Innvator Development Kit 各個模組攤開平擺在 BOB 上使用，方便於開發時接線的方便性，另一方面，BOB 提供了原本 Innovator Development Kit 上所沒有的 LAN 的網路晶片模組。多了這個模擬將在發展上廣闊。如圖 A-4所示：

圖 A-4 Innovator Development Kit 的 Break Out Board 配件模組

附錄Ｂ　DSPLinux 環境建構方式

本論文的軟體實驗環境是在 OMAP 平台上運用 ARM 和 DSP 的作業系統，此附錄將其執行流程與建構方式分成「Innovator Development Kit 啟動程序」、「DSPLinux 作業系統核心之製作」以及「檔案系統製作流程」。

B.1　Innovator Development Kit 啟動程序

在整個嵌入式系統中，除了硬體平台外，重要的是還需要嵌入式系統內部的作業系統核心運作。一般而言，一個嵌入式 Linux 要正常運作，須要具備三種構成系統的單元：

(1) Bootloader—在 Innovator Development Kit 上採用 rrload bootloader[4]

(2) Linux 作業系統核心—在 Linux 作業系統中為 zImage[5]

(3) 檔案系統(root filesystem)—在 Linux 作業系統中的系統架構

rrload bootloader 是一個系統開始前的機制，主要的工作是：初始化單板硬體裝置（如：記憶體、系統執行速率、I/O 埠等）並且找尋作業系統進入點進行載入執行。一般可以視其為嵌入式系統的 BIOS。其次，Linux 作業系統核心 zImage 是一個作業系統的映象檔（Image），核心本身是一個塊的執行碼，在其內部沒有明顯劃分，但是可以依角色工作劃分，大致上可以分成：行程管理（Process management）、記憶體管理（Memeory management）、檔案系統（Filesystem）、裝置控制（Device control）以及網路作業（Networking），而可以在運行期才加入核心的程式碼，稱為模組（module）。核心可以視為整個運作單元中最為重要的部分。而第三個部分檔案系統(root filesystem)是 Linux 樹狀檔案系統架構，其內部包括了基本系統架構外，還包括發展之應用程式、系統指令集、靜態和動態函式庫（Library）以及未編入核心的裝置驅動模組。

[4] rrload bootloader：http://www.ridgerun.com/bsp/rrload.html

[5] 本論文中，採用的核心版本為 2.4.21

整個 Innovator Development Kit 系統開發環境規劃如圖 B-1所示，主機 PC 提供跨平台環境和根檔案系統(root filesystem)，而目標板 Innovator Development Kit 則包括 rroad bootloader 和 Linux 核心。

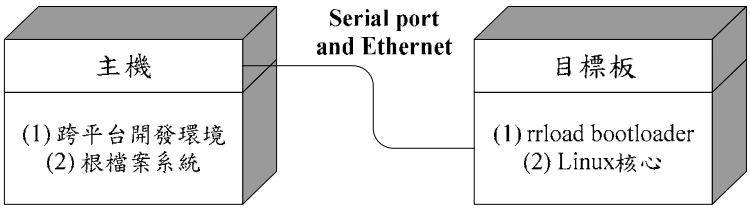


圖 B-1　主機、目標板採用連結式規劃

整個 Innovator Development Kit 的開機程式，如圖 B-2所示。首先，是由 rrload 進行硬體的初始化動作，並將 zImage 由靜儲存裝置 ROM 載入至 SDRAM 中，由 rrload 載入執行。在 kernel 載入系統當中，其中 kernel command line 會自動取得 IP address，並且到內定的位置載入根檔案系統，並且掛載至系統中，成為 DSPLinux 的檔案系統。接下來，作業系統便會執行系統第一個程式 init，並且完成系統其他運行程式，所有其他程式，皆是由 init 所 fork 出來的子程序。

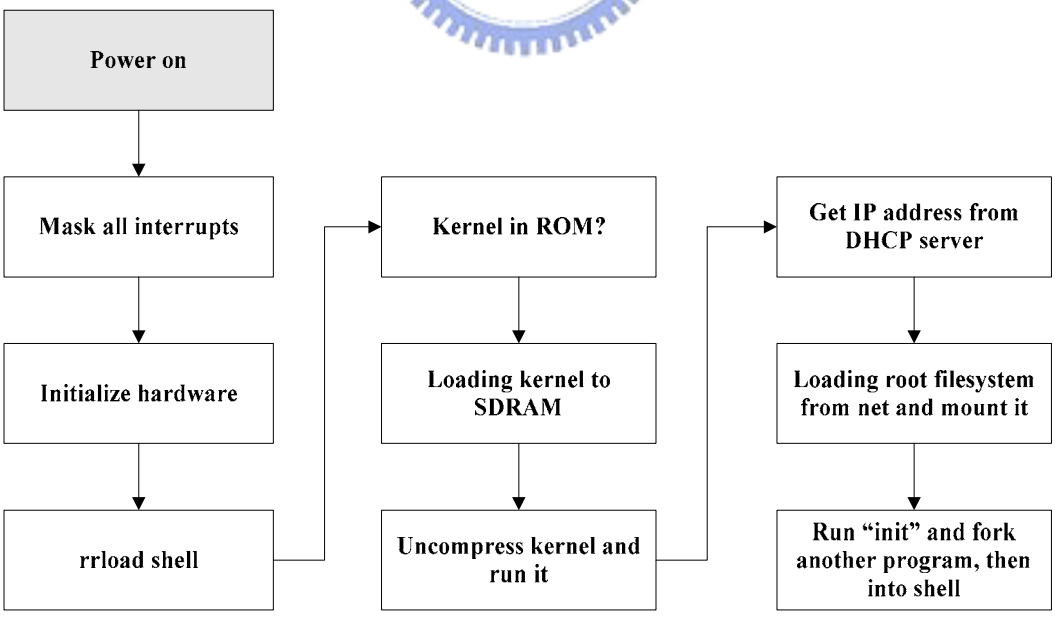


圖 B-2　Innovator Development Kit 啟動流程

B.2 rrload bootloader 的安置

一個系統在啟動時，需要一個引導的機制，如同 PC 的 BIOS，在 Innovator Development Kit 單板上，載入 rrload bootloader 做為啟動載入系統的機制。其安裝於 Innovator Development Kit 的流程，可以歸納如下：

Step1：下載 rrload bootloader 的原始檔

Step2：編輯 rrload 內部有關於跨平台環境的設定檔 bsp.def 和 innovator.def，指定為 arm-linux-的交叉編譯

Step3：執行 makefile，產生出 setup 和 rrload 兩個 COFF 的 binary 檔

Step4：利用 CCS 透過 JTAG Emulator 來載入 setup 來對環境做初始化的動作

Step5：於 CCS 中透過 JTAGE Emulator 再載入 rrload 於單板上

Step6：在 Linux 系統下，用 minicom[6]與單板環境做連結

Step7：儲存 rrload 於 ROM

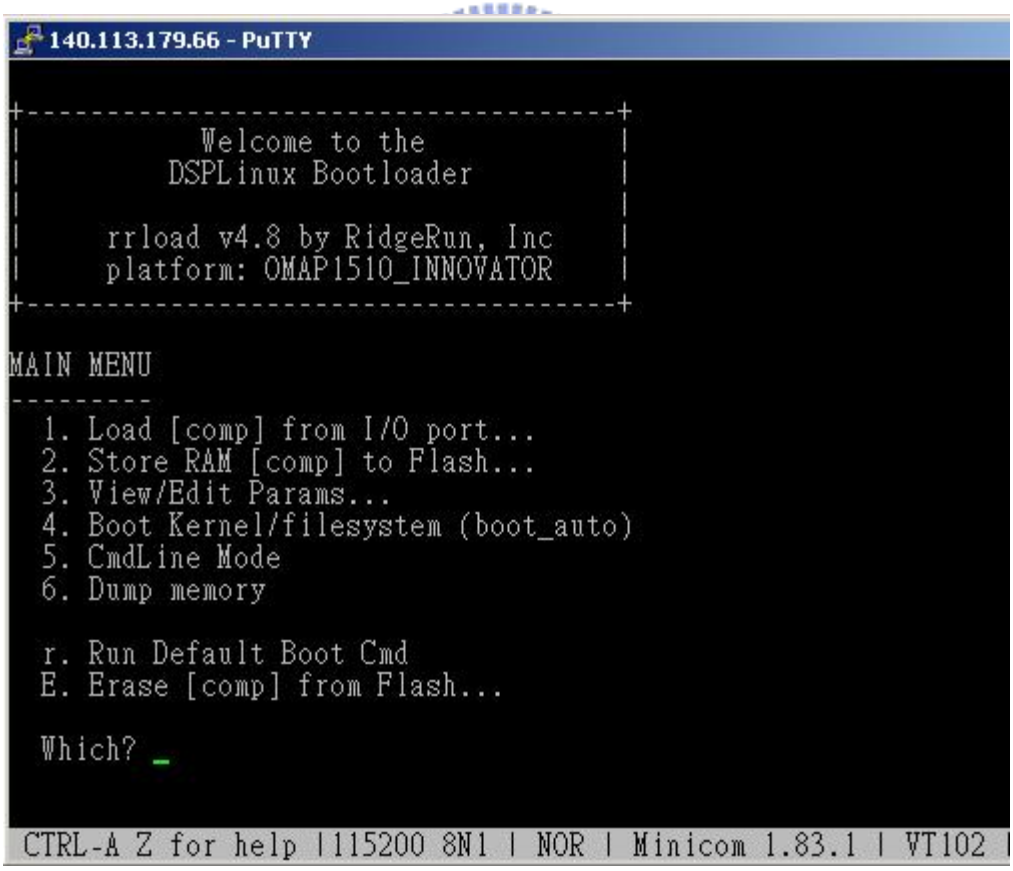


圖 B-3 rrload bootloader 操作介面

[6] minicom：Linux 下常用一種的終端機，目前已含入一般常用的 Linux distrubution

B.3 DSPLinux 作業系統核心之製作流程

Innovator Development Kit 是一個 OMAP 的發展平台，對於 OMAP 的硬體架構，必須將對 Linux 系統核心進行適當的修正調整。本研究所採用的是 DSPLinux，事實上 DSPLinux 是以 ARMLinux 為藍本，再加上 OMAP 整體架構和其中 DSP 核心的架構做修正，故需要 ARM、DSP 和 OMAP 三種修補檔（Patch）。系統經過修補過後，在 Linux 核心的編譯過程中，便會出現支援 OMAP，DSP 的相關選擇，由於需要 DSP 功能的支援，故必須要圈選。如果一來，DSPLinux 才會有 DSP 的功能支援。

在挑選核心過程中，把握的原則是，只選擇所需要的系統功能，不必要的儘量不要選擇，因為核心強調在於精簡，執行才會有效率，另一方面則是，嵌入式系統的記憶體儲存有限，核心的大小，將會直接影響到系統的儲存空間。在挑選過程中，除了 DSP 功能要選取之外，DHCP 和 Bootp 的支援亦需要選取。這是由於根檔案系統（root filesystem）是透過網路取得的。故在一開機時，必須讓 kernel 先自動從 DHCP server 取得 IP address。在挑選的最後，還需要選取 devfs 的支援，在前面第三章中曾經提到過，DSP 是透過 DSP task Device，而 DSP task Device 是由 devfs 機制自動產生的，故需要選取這一個機制。製作流程如圖 xxx 所示

挑選完畢之後，經過交叉編譯（Cross-compiler）系統核心來產生 zImage 檔案，再經過 rrload 中的 mkimage 的 binary 檔，轉換成 rrload 載入核心的編碼方式載入系統。

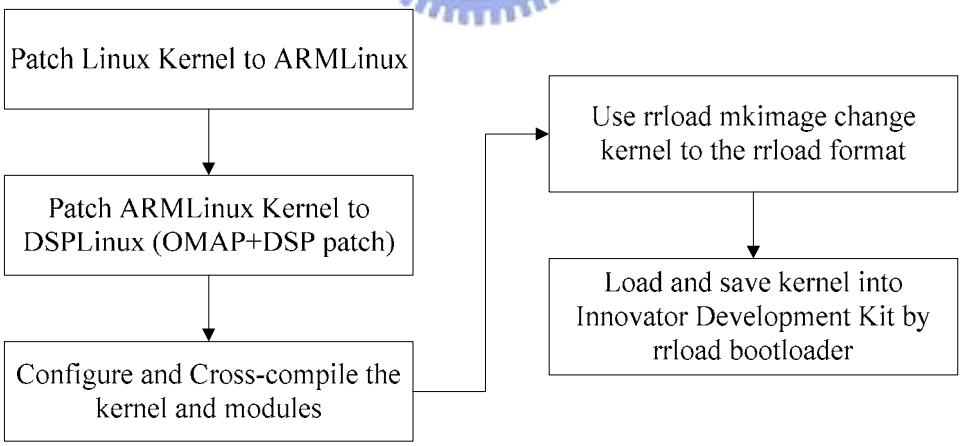


圖 B-4 DSPLinux 核心製作及載入單板流程

B.4 根檔案系統製作流程

一個系統的正常運作，除了系統核心之外，還需要一個檔案系統來來提供應用程式的執行環境。Linux 核心在系統啟動期間所進行的最後一個步驟就是掛載根檔案系統。Linux 根檔案系統正式規定是於 Filesystem Hierarchy Standard(FHS)中，在建立根檔案系統時，其目錄架構可以參照裡面的規定。一個基本的檔案系統是由以下幾個部分所構成的。

(1) 基本目錄結構：/、/bin、/sbin、/etc、/lib、/dev、/usr、/proc 等

(2) 系統指令工具：init、getty、ls、cd 等系統指令

(3) 系統設定檔：inittab、fstab 等設定檔

(4) 裝置檔案：mem、null、zero、tty0、ttyS0、console 等

(5) 系統函式庫：ld.so、libc.so、libm.so、libcrypt.so 等

(6) 系統模組：未編進系統核心的模組

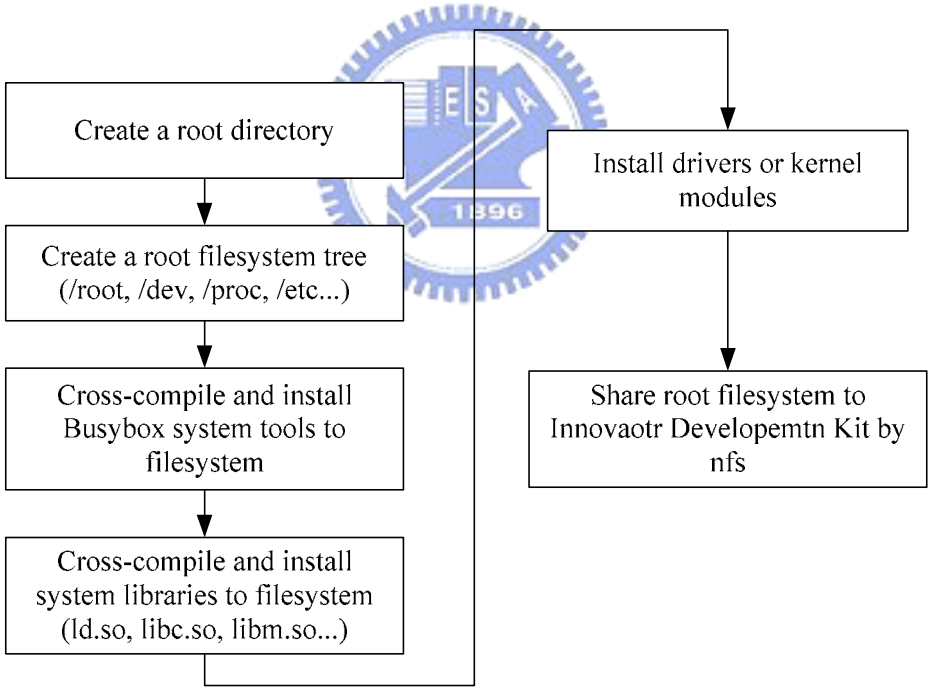


圖 B-5 Innovator Development Kit 根檔案系統建構流程及載入方式

圖 B-5是 Innovator Development Kit 的根檔案系統建構的流程，以及載入的方式。在流程中，一開始，主要是建立系統目錄的部分，接下來，採用 Busybox 來建立系統指令集，Busybox 是一個在嵌入式系統相當好的指令集解決方式，它除了具有一般常用指令外，重要的是它縮小這些指令大小並且整合起來，Busybox 亦提供系統開機時所需要的指令 init 供系統啟動。有了系統目錄和指令之後，還需要函式庫，除非將所有的應用程式都做靜態連結，否則一般應用程式需要一些重要的函式庫於執行時做動態連結。之後，用 ldconfig 指令建立動態連結的參照設定檔，再將一些重要設定檔如 inittab、fstab 等加入，最後將模組放入根檔案系統中。完成簡易的檔案系統。

在 Innovator Development Kit 載入根檔案系統，所採用的是利用 nfs 將檔案由網路載入，也就是將根檔案系統放置於主機，但 Innovator Development Kit 在開機時可以到主機去索取根檔案系統載入單板內使用。所以在建立根檔案系統的最後，要利用 nfs server 將根檔案系統分享給 Innovator Development Kit。