國立交通大學

電機與控制工程學系

碩士論文

CAN 網路同步應用協定之設計與實現

Realization of the Synchronized CAN Application Layer

研究生：鄧元銘

指導教授：徐保羅　博士

中華民國九十六年七月

CAN 網路同步應用協定之設計與實現

研究生：鄧元銘　　指導教授：徐保羅 博士

國立交通大學 電機與控制工程學系

摘要

CAN bus 為常見於各類應用場合的網路協定，為了分析其傳輸特性，本論文以四項實驗參數：(1)網路傳輸速率、(2)資料封包大小、(3)系統取樣時間以及(4)傳輸資料量進行傳輸實驗，根據實驗結果，歸納出影響網路資料遺失量的兩個主要因素：網路延遲與時脈漂移現象，並以此作為網路時脈同步機制的設計基礎。

在系統整合上，必須建立網路架構與控制程序之間的共同規範，即應用層協定的建立，在 CAN 應用層協定的設計上，本論文以 CANopen 的概念為基礎，規劃出較精簡的網路協定，並配合前述實驗結果，設計網路時脈同步機制，此同步機制可將時脈漂移現象加以修正，使得網路資料遺失量大幅降低。

最後，將所設計之 CAN 應用層通訊協定實現於四輪全向平台系統的控制網路，透過網路將控制系統加以整合，並在完成的全向平台網路系統下，驗證所設計之網路時脈同步機制，使得各軸速度響應的精密度提昇 16.31%。

Realization of the Synchronized CAN Application Layer

Student：Yuan-Ming Deng **Advisor：Dr. Pau-Lo Hsu**

Department of Electrical and Control Engineering
National Chiao-Tung University

ABSTRACT

To analyze the synchronization property of CAN (controller area network) bus among multiple network nodes, four factors: (1) bit rate, (2) data length, (3) sampling period, and (4) message amount are examined in this thesis. Experimental results indicate that the data dropout rate of a synchronized network control system (NCS) is mainly affected by the network delay and the clock drifting.

The application layer protocol, which is the mechanism between the communication network and the application process, is modified according to the CANopen structure to obtain a clock synchronization mechanism in this thesis to overcome the unavoidable clock drifting; thus, the data dropout rate in a synchronized network system is improved.

Finally, the proposed protocol and synchronization mechanism is applied to an omni-directional moving robot system with four independent wheels. Results indicate that the present synchronized mechanism improves 16.31% accuracy of velocity control in each axis.

誌　　謝

本論文的完成，首先要感謝指導教授徐保羅博士，在這段研究生活中的指導與照顧，讓我學習到研究的正確觀念與態度，去克服研究上的各個難題，並與他一同分享成果的喜悅，在此，獻上我最誠摯的敬意與感謝。同時，感謝口試委員王伯群博士、胡竹生博士和連豐力博士在論文上面的建議和指導，使本論文更加完整。另外，感謝系上王宜楷老師，在研究遭遇困難時，提供寶貴的意見，使問題得以解決。

感謝實驗室謝鎮洲學長、幸琮政學長、張昭琳學長、黃煒生學長、賴建良學長和鄭景文學長，在研究與生活上，給予我的意見和幫助，以及實驗室同學興漢、宗翰、孝麟、瑞原、學弟宗勝、雨坤、林億、也強、醇偉、建龍，在學業上的相互砥礪與指教，並且為生活帶來許多的歡樂。感謝 816 實驗室的所有夥伴，你們讓這段研究生活，變得有趣而豐富。

最後，感謝我最愛的父母與家人，有了你們的鼓勵與支持，我才能全心全意完成學業，才有力量去克服所遭遇的困難，一步步地向前邁進。

謹將本論文獻給我最愛的家人以及所有關心我的朋友們，謝謝你們。

目　　錄

表　目　錄

圖　目　錄

第一章　緒論

1-1　研究動機

現代工業與商業系統的趨勢是將運算、通訊和控制單元，加以整合成各種不同層級的處理程序，以方便系統的建立與管理，而透過網路匯流排(Network bus)概念的加入，可以增進整合後的系統，在工作效率與系統維護所帶來的改善。因此，各種網路協定便因應而生，常見於工業界的協定，例如：WorldFIP、Profibus、P Net、SERCOS、CAN [1]。然而，網路化系統首先面臨的問題，便是網路傳輸狀況所帶來的影響，網路引起的訊息傳遞延遲、訊息封包遺失與訊息排程，以及節點之間時脈的不同步問題。為了探討上述的種種因素，本文將以 CAN (control area network)為系統的通訊網路，進行各種傳輸條件下的網路分析實驗，探討各項網路的傳輸狀況。

然而，在網路系統的設計上，各個網路節點(node)之間通訊格式的問題，包括訊息識別碼(ID)、資料長度和內容規劃，甚至於訊息行經的節點與目的節點，都是系統建構上必須加以考慮的，針對此部份，本文將 CAN 的應用層通訊協定加以延伸，設計出符合應用場合的通訊協定，並根據網路分析實驗結果，針對網路節點之間的時脈同步，提出解決方式並加以驗證。

最後，本論文將所設計的應用通訊協定和時脈同步方法，實現於全向性移動平台的控制系統上，完成全向平台之分散式運動控制系統。

1-2　文獻回顧

半導體技術的發展，大幅縮短了各類元件的製造週期，帶動了電子電機相關研究領域，同樣也帶來了不同於以往的新概念，網路便是一個最明顯的例子，

從電子電路上的元件溝通到系統資料傳遞，乃至於乙太網路(Ether Net)、無線網路(wireless network)，網路概念的延伸與應用隨處可見，因此，網路控制系統(NCS, network control system)的概念也跟著成型。

網路控制系統相關的研究課題紛紛被提出[2]，對於整體系統而言，足以影響系統效能的網路因素，被歸類為以下幾點：網路傳遞所產生延遲(network-induced delay)[3-4]、節點間時脈不同步(jitter)[5-6]、資料遺失(data dropout)[7-8]、取樣週期(sampling period)[9-10]、網路排程(network scheduling)[11]等等，針對這些因素加以探討，便能對網路控制系統的效能進行分析。

然而，為了因應不同的使用場合，眾多的網路規格一一被開發，CAN bus是最常被使用應用於工業用控制的網路之一，因此，前段所述各項網路因素於CAN 網路的分析，也同樣成為相關的研究課題[12-14]。另一方面，如何在原有CAN 的架構之上，建立上層的通訊協定，以縮短系統開發與整合的時程，如CANopen[15]、Device Net[16]等，或是改良其本身的不足之處，如延長其傳輸距離[17]，除此之外，為了系統整合的目的，更延伸到與其他網路結合[18]，使得 CAN bus 網路的應用範疇更為廣泛。

本文將 CAN 應用到四輪獨立控制的全向位移動平台，機器人移動平台近來成為一個熱門的研究領域，其目的在於協助人類進行繁瑣工作，或是進入人類不易到達的場所，因此，如何提高移動平台的移動能力，使其更具機動性，便成為機器人領域的一項課題。於是，一種新的全向平台被提出[19]，它採用全方向性輪，如圖 1-1，這種輪子同時擁有橫向及縱向滾動的輪子，透過輪子間合力分力的關係，決定其移動方向，相較於差動輪移動平台，全向輪移動平台有更好的移動性，其優點為：(1)改變平台方向時，輪軸不需移動，(2)可在原地旋轉平台，不需旋轉半徑，(3)可同時自旋與曲線運動。此類的移動平台，已被廣泛的討論[20-21]。

目前常見的全向性移動平台，大多以三軸到五軸的平台為主，在不同的考

量下，這三類平台各有本身的優缺點[22]，三軸在以 60 度為倍數的移動角度上，可以達到較好的移動效率，四軸則是在 90 度為倍數的角度上，能獲得遠大於其他兩者的效果，至於五軸的效能，則是在各種角度效能都相當平均，此外，在負重能力上，軸數越多負重能力就相對的越好，本論文中是採用四軸全向平台，對於直線路徑，移動速率與節省消耗功率都有所幫助。

圖 1-1 全向輪

1-3 問題描述

為了建立全向平台的網路系統，從網路的傳輸架構到分散式全向平台控制系統的實現，本論文所需克服的問題如下：

1、 了解 CAN 網路傳輸特性

在網路控制系統中，除了在集中式系統的設計考量，網路傳輸的特性更是影響系統架構和效能的重要因素之一，包含傳輸的媒介、傳輸速率、封包傳遞方式等特性，而網路傳輸所衍生的各項課題，如：傳輸延遲、資料遺失現象、網路時脈的同步性等因素，同樣是必須納入考量的相關因素。此外，除了系統架構之外，系統的控制規格也與網路特性有關，如系統取樣時間的設計，便必須配合網路頻寬而設計。

在本論文中，所採用的網路通訊協定為 CAN bus，為了系統架構的設計，必須對 CAN 網路協定與系統規格加以分析，了解相關的網路與系統

特性，以利網路系統架構的建立。

2、 如何制定 CAN 應用層通訊協定

在多數的工業用通訊網路中，主要都以提供穩定可靠的傳輸協定為主，對於如何與應用程序互動，並未詳加規範，所以，網路系統設計時，開發者必須建立網路與處理程序之間的對應關係，而本論文所採用的 CAN 網路，雖有相關的應用層協定，對網路與應用端的互動加以規範，但卻必須付出額外的軟硬體成本，因此，本論文在全向平台網路系統的建立上，並不採用其應用層協定，而自行制定控制資料的傳遞方式以及網路的傳輸行為等等，以達到網路系統的建構。

3、 如何實現全向平台的分散式控制系統

本論文以全向移動平台作為網路系統的實現平台，除了控制架構的設計，如何將上述兩項網路課題與控制架構結合，以移動平台作為所設計之網路協定的驗證，也是必須解決的問題之一，因此，如何設計控制架構以及網路系統的整合，都是本論文將解決之問題。

1-4 研究方法與步驟

本論文研究方法與步驟如下：

1、 CAN 網路的傳輸分析實驗

為了獲得 CAN bus 的傳輸特性，本論文以硬體實驗的方式，進行網路傳輸實驗，首先，量測各項的網路時間參數，包含軟體的前(後)處理時間、各種傳輸速率和封包大小下的傳輸時間，接著，以不同的系統條件作為實驗參數，包括傳輸封包大小、傳輸速率、取樣週期、傳輸資料量四項，並統計傳輸的資料遺失量作為實驗結果的指標。在實驗結果的分析上，藉由網路傳輸時脈模型的概念，作為結果分析的基礎，針對各個實驗結果提出

解釋，並歸納出網路系統的傳輸性質。

2、 CAN 應用層通訊協定的設計

CAN 應用通訊協定，以 CANopen 和 Device Net 最為常見，兩者雖都是以 CAN 作為底層通訊協定，但在實際應用上，都需額外的軟硬體支援，並不適用於本論文的控制架構，但其設計概念仍可加以運用，因此，本論文將 CANopen 的概念作為基礎，以全向平台控制為主要考量，制定應用通訊協定，並根據分析實驗的結果，提出網路時脈同步方法，並透過實驗加以驗證，結合應用通訊協定和時脈同步方法，作為全向平台的控制網路架構。

3、 分散式全向平台控制系統的整合

全向平台控制系統的實現中，本論文將建立平台的硬體控制電路，以 DSP(digital signal processor)和微控器 8051 作為各個節點的運算核心，搭配相關的週邊電路，達到平台的閉迴路控制，並與前述的應用層通訊協定結合，將網路節點的管理與控制參數設定等系統管理行為，透過網路架構加以整合，完成分散式的平台控制系統。

1-5 論文架構

本論文共分為五章，第一章旨在說明研究動機與目的、相關文獻回顧、敘述所面臨的相關問題、本論文的研究方法與步驟、以及論文架構。第二章先對 CAN bus 加以說明，包括通訊協定、訊息封包格式以及錯誤處理機制三部份，並以 CAN 建構網路分析實驗環境，在不同的傳輸條件下進行實驗，以資料遺失量作為實驗結果的指標，並以網路時脈模型協助結果分析，歸納出影響網路系統的相關因素。第三章則先介紹 CANopen，包括其通訊協定與網路管理兩部份，接著，將其設計概念加以延伸，制定出以全向平台控制系統為考量的應用

通訊協定，此外，根據第二章的網路分析結果，提出網路時脈同步方法以及驗證結果，也於此進行說明。第四章則為分散式全向平台控制系統說明，包括全向平台硬體架構、運動模型與系統程式架構，系統整合的相關實驗結果，包含網路系統的實現結果、平台移動控制的實現，同樣於此章進行說明。第五章則為本論文的結論以及未來的發展方向。

第二章　CAN 網路實驗分析

本章先介紹 CAN 通訊協定，並且透過硬體傳輸實驗，對 CAN 網路的傳輸情形進行分析，以了解網路系統之特性，並歸納出影響網路遺失量的主要因素作為結論。

2-1 CAN (Controller Area Network) [23-24]

2-1-1 CAN 網路協定簡介

CAN 起源於 1980 年代後期，由德國 Robert Bosch 公司所開發，直到 1990 年代其通訊協定逐漸成型，相關的 IC 元件紛紛被推出，成為一種具有高安全性且適用於分散式架構的高速串列通訊協定，最高傳輸速率可達 1M bps(bit per second)。CAN 原先是應用在汽車電子系統，用來連接汽車內防鎖死煞車系統(ABS)或是引擎控制與感測單元等的電子元件，以簡化車內複雜的硬體配線；但由於 CAN 提供可靠快速的資料傳遞，適合用在即時系統(real-time system)且價格低廉，因此，發展至今也被廣泛運用在各類的控制系統上。目前 CAN 已成為國際標準規格(ISO11519, ISO11898)，而以 CAN 為基礎的高階通訊協定，已有 CiA (CAN in Automation)的 CANopen、Honeywell 的 SDS 及 Allen-Bradley 的 DeviceNet 等等。

根據其傳輸協定的規範，可歸類出以下特點：

- 使用差動訊號的方式傳遞
- 任一節點皆可主動發出訊息(multi-master)
- 資料訊息具有優先權(priority)
- 網路優先權的仲裁(arbitration)為非破壞性
- 採用廣播的方式(broadcast)，並藉由傳輸的訊息做時序同步的動作

- 未傳送成功的訊息會自動重新傳送
- 完整的錯誤檢查機制，並可排除錯誤過多的節點

接下來，2-1-2 節將介紹 CAN 的通訊協定，2-1-4 節介紹 CAN 的各種訊息封包，2-1-4 節則介紹其錯誤處理機制。

2-1-2 CAN 通訊協定

CAN 通訊協定主要可分為網路架構、傳輸方式、網路仲裁機制三個部份。

◆ CAN 網路架構：

以 CAN 本身的傳輸特性定義其網路架構，可分為 CAN 對象層(the CAN-object layer)、CAN 傳輸層(the CAN-transfer layer)、物理層(the physical layer)，對象層的功能包括尋找被發送的訊息、從傳輸層中挑選所需的訊息、提供應用層所需的硬體介面；而傳輸層主要是規範通訊協定相關定義，即控制訊息欄框的結構、訊息的仲裁、錯誤偵測、錯誤標定、故障界定，CAN bus 發送(接收)訊息的時機、位元時脈(bit timing)的定義也屬於傳輸層的一部分；實體層則是電子訊號的傳遞與硬體接線。若由 ISO/OSI 網路模型，將 CAN 的通訊協定加以歸類，其通訊協定僅包含 OSI 模型中的資料連結層(Data Link Layer)和實體層(Physical Layer)兩個部份，圖 2-1 為 CAN 在 OSI 定義下，各個部份的功能與定義。

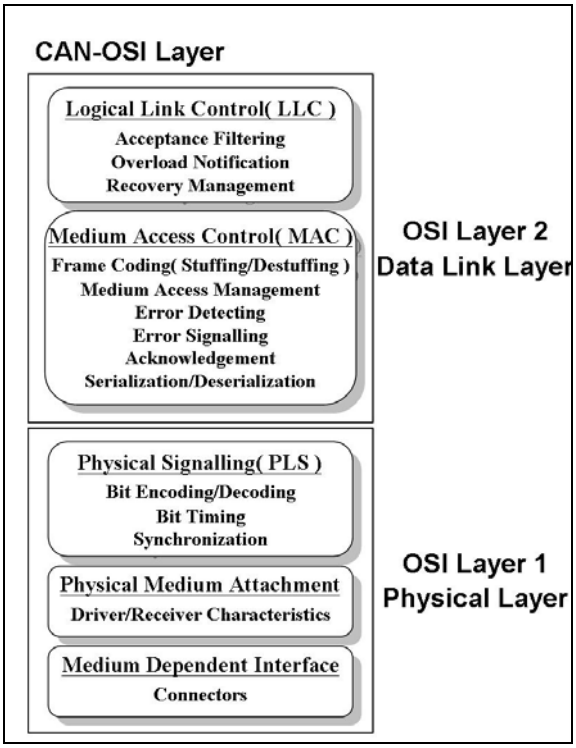


圖 2-1 CAN-OSI 網路模型

◆ CAN 傳輸方式：

這部份針對 CAN Bus 在網路傳輸方式進行說明：

- 傳輸速率(bit rate)

在 40m 以內的傳輸距離，最高傳輸速率可達 1Mbps，隨著距離增長，可達到的傳輸速率也跟著下降，例如距離拉長到 400m 時，只能用 100kbps 的最高速率傳輸。

- 傳送節點(transmitter)和接收節點(receiver)

CAN 允許 multi-master 的傳輸方式，當任一節點取得網路使用權，並進行資料傳輸時，此一節點稱為傳送節點，網路中其餘節點則稱為接收節點。

- 傳送節點對網路的偵測

傳送節點在進行資料傳輸的同時，也進行網路上出現的位元值是否與所送出的相同。

- 網路位元值(bus value)定義

CAN 所定義的網路位元為顯性(dominant)和隱性(recessive)兩種邏輯位元，其中 dominant 是數位訊號”0”，recessive 則是數位訊號的”1”，且 dominant 位元具有較高優先權。

- 網路存取方式

由於 CAN 採用多重節點隨機存取(random access)網路，訊息封包碰撞(collision)的情形在所難免，因此，CAN 採用的解決方法是 CSMA/CD +AMP (Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority)，每個節點進行傳輸之前，先進行網路監聽，確定網路閒置後才開始傳輸，但並無法完全避免碰撞的問題，一旦碰撞發生將進入優先權仲裁(Priority Arbitration)，決定網路使用權，優先權較低者喪失網路使用權後，在下次網路閒置時，自動重新訊息的傳遞。這樣的機制決定了網路媒介的存取行為和網路封包的傳輸次序，因此，接著對此仲裁機制進行詳細介紹。

- 網路使用權的仲裁機制(Arbitration)

當有兩個以上節點要傳送訊息時，網路上便發生封包碰撞的情形，此時便進入仲裁階段，仲裁的方式是將每個節點發送的訊息封包識別碼(identifier)，逐一進行位元比對，而當上述的兩種位元值同時被發送時，dominant 位元具有較高優先權，所以會被保留下來，recessive 位元將不會出現，因此，當所有發送節點對自己發送的位元值和網路線上的位元值比照時，如果線上的值和自己發送的相同，則繼續發送訊息，而如果發送訊息為 recessive 訊號，而線上為 dominant 訊號時，則此發送節點失去仲裁權，必須退出發送狀態，等下次網路閒置時再重新發送，以下圖 2-2 為例，節點 1 和節點 3 先後失去仲裁權，節點 2 在此仲裁取得優先權，所以可繼續訊息的傳輸，另外兩個節點則必須等到下次網路閒置才能繼續傳輸。此外，由圖可看出當識別碼越小，則擁有越高的優先權。

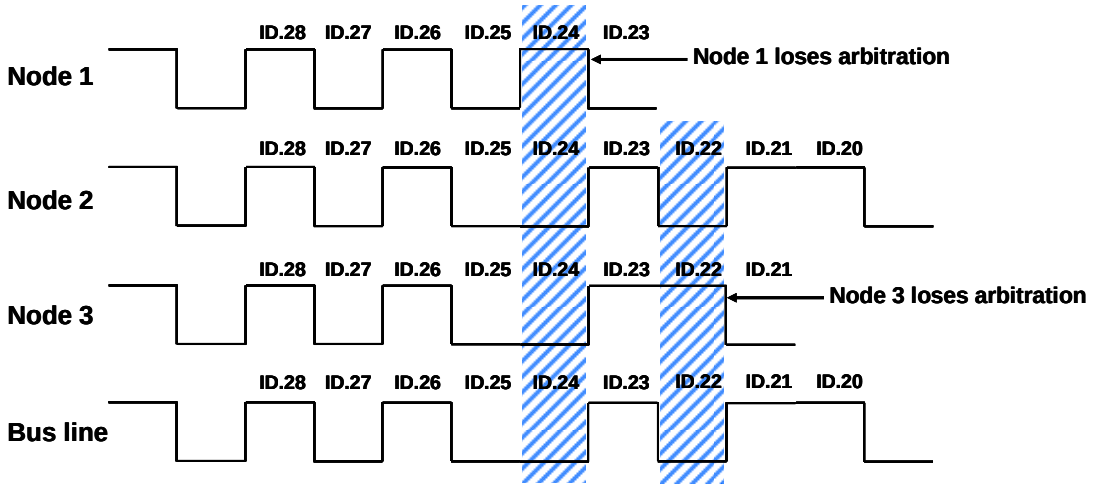


圖 2-2 節點優先權仲裁

2-1-3 CAN 訊息封包格式

關於 CAN 的封包格式，由 CAN 2.0A 和 CAN 2.0B 兩種規格所定義，兩者主要的差異在於訊息識別碼(Identifier)長度不同，前者定義的是 11 位元識別碼，為標準 CAN 格式；後者則是 29 位元識別碼，為擴展的 CAN 格式，此節所介紹 CAN 的各種欄框(Frame)格式，除了識別碼的定義之外，其餘的定義皆適用於兩種規格，而 CAN 的封包格式可分為資料欄框(data frame)、遙控欄框(remote frame)、錯誤欄框(Error Frame)和過載欄框(overload frame)。另外，兩個資料(或遙控)欄框或是資料欄框和遙控欄框之間都需欄框間隔(interframe spacing)加以區分。

在說明欄框前，先介紹 CAN 的填充/編碼(bit stuffing/coding)傳輸方式：當發送節點在傳送資料或遙控欄框的時候，若是傳送五個連續且相同位準的位元，則需在第五個位元之後加入一個相反位準的位元。接收節點在接收時會自動將這填充進來的位元給刪除，以得真正的訊息。錯誤欄框及過載欄框並不會經過位元填充的動作，而是以固定格式發送，因此，位元填充只適用於資料欄框與遙控欄框的部份區段：欄框起始、仲裁區、控制區、資料區和循環多餘碼區，各區段的意義將在接下來的部份說明，以下就分別說明各類欄框：

◆ 資料欄框(Data Frame)

當節點有資料要傳輸的時候，可透過資料欄框將資料發送到網路，所以，其訊息長度是所有欄框中最大的，也是格式最為複雜的，下圖 2-3 是標準 CAN 的資料欄框格式(standard data frame)，圖 2-4 是擴展 CAN 的資料欄框格式(extended data frame)，兩者的差異僅在識別碼的長度，因此，底下就以擴展格式為例，說明各段的意義：

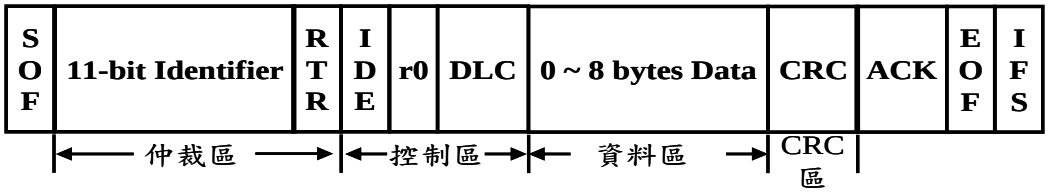


圖 2-3 標準 CAN 資料欄框格式

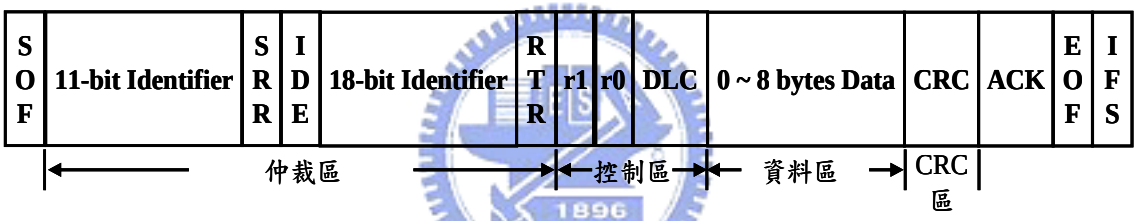


圖 2-4 擴展 CAN 資料欄框格式

- SOF(Start of Frame)

欄框起始，資料欄框及遙控欄框的起始，由 1 個 dominant bit 組成，當網路閒置的時候，藉由此位元的傳送，開始新的資料(遙控)欄框傳輸。

- Identifier

識別碼，用來決定訊息優先權並區分不同的資料(或遙控)欄框，識別碼越小的時候仲裁優先權越高，在標準 CAN 中識別碼為 11 bits，在擴展 CAN 中識別碼為 29 bits。

- RTR(Remote Transmission Request)

遙控傳輸請求，RTR 為 dominant bit 時，表示此欄框為資料欄框，當 RTR

為 recessive bit 的時，表示此欄框為遙控欄框。

- SRR(Substitute Remote Request)

替代遙控請求，在擴展 CAN 中用來取代標準 CAN 中的 RTR 位置，為一 recessive bit，當標準 CAN 和擴展 CAN 起衝突時，由於 SRR 為 recessive bit，根據 CAN 的仲裁方式，標準 CAN 的優先權大於擴展 CAN。

- IDE(Identifier Extension)

當此位元為 dominant 時，此訊息的識別碼為標準 CAN 的識別碼，當此位元為 recessive 時，此訊息的識別碼為擴展 CAN 的識別碼。

- r0, r1

保留位元，必須為 dominant bit。

- DLC(Data Length Code)

傳送資料的長度，由 4 個位元所組成。

- Data

欲傳送資料的存放地方，最少可傳送 0 個位元組，最多可傳送 8 個位元組。

- CRC(Cyclic Redundancy Check)

循環冗餘檢查，用來確認傳送資料是否正確，由 15 個位元組成。

- ACK(Acknowledgement Filed)

確認區，由 2 個位元組成，分別為 ACK Slot 和 ACK 確認邊界，一開始訊息送出去的時候都是 recessive bits，當匯流排上面有任何一個節點成功接受訊息，則將 ACK Slot 的 recessive bit 改成 dominant bit，如此一來，發送端可以藉由偵測 ACK Slot 來判斷訊息是否有正確被接收。

- EOF(End of Frame)

  欄框終止，由 7 個 recessive bit 所組成，表示一個資料欄框的結束。

- IFS(Inter-frame Space)

  欄框間隔，由 3 個位元組成，其用途主要是用來區隔資料欄框或是遙控欄框，以提供節點處理資料的時間。

◆ 遙控欄框(Remote Frame)

用來請求其他節點傳送所需要的資料，結構和資料欄框相似，但是有兩點不同的地方，第一點是遙控欄框 RTR 為 recessive bit，另外一點是遙控欄框是不帶任何資料的，其餘的格式皆與資料欄框相同，下圖 2-5 為一擴展遙控欄框，值得注意的是，雖然資料長度被忽略，但仍應設定欲接收的資料長度。

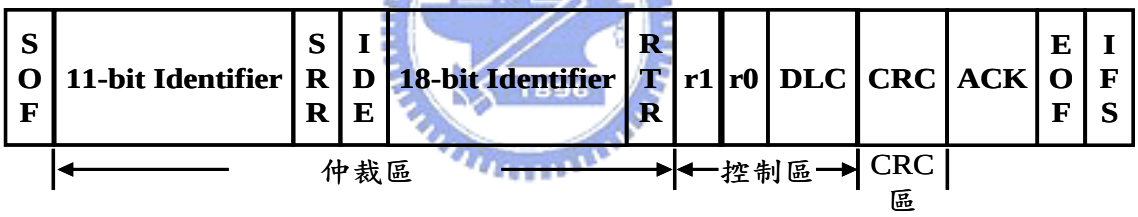


圖 2-5 擴展 CAN 遙控欄框

◆ 錯誤欄框(Error Frame)

當節點偵測到錯誤發生時，便以錯誤欄框通知各個節點，錯誤欄框由兩個部份所組成，第一個部份是錯誤旗標(error flag)，第二個部份為錯誤邊界(error delimiter)。錯誤旗標(error flag)有兩種形式，一種是由連續 6 個 dominant bit 所組成的主動錯誤旗標，另一種則是由 6 個 recessive bit 所組成的被動錯誤旗標，兩者的差異將在下節詳細說明。錯誤邊界(error delimiter)由 8 個 recessive bits 組成，當完成錯誤旗標的傳輸後，節點開始偵測網路線上的值，直到偵測到由 dominant

bit 轉為 recessive bit 的過渡(transition)為止，表示此時所有主動反應錯誤節點都完成錯誤旗標的傳送，而開始發送錯誤邊界。當所有節點偵測到 8 個 recessive bit 之後，錯誤欄框即傳送完成。若在錯誤欄框期間又發生新的錯誤，則各節點會重新發出新的錯誤欄框。

◆ 過載欄框(Overload Frame)

過載欄框由過載旗標(overload flag)和過載邊界(overload delimiter)所組成，其格式跟錯誤欄相似，過載旗標由 6 個 dominant bit 所組成，過載邊界由 8 個 recessive bit 所組成，主要是為了替接收節點爭取處理資料的時間，節點會在三個情況下發出過載欄框：

- 當接收端在下筆資料欄框或遙控欄框進來之前，需要一個延遲時間的時候。
- 當節點在欄框間隔期間，偵測到 dominant bit 的時候
- 當節點在錯誤邊界及過載邊界的第 8 個位元，偵測到 dominant bit 的時候。

2-1-4 CAN 錯誤處理機制

CAN 對於錯誤的處理方式，是其提供可靠穩定傳輸的重要因素，主要可分為兩大類：錯誤偵測和錯誤頻繁節點的排除，前者經由檢驗訊息封包的各項細節，判斷其正確性，一旦發現錯誤便利用錯誤欄框的發送，使錯誤的訊息封包重傳；對於錯誤發生頻率過高的節點，則經由後者減少其對於網路的使用權，甚至禁止對網路的存取。

◆ CAN 錯誤偵測機制

CAN 的錯誤偵測機制，根據訊息傳輸可能發生的問題，分成以下五種不同的錯誤類型：

- 位元錯誤(Bit Error)

  節點在發送訊息的同時也會偵測匯流排上面的狀況，當發送的值與偵測的值不一樣時，此時則產生一個位錯誤。但是在仲裁區時，某一節點所發出的 recessive bit 被另一節點發出的 dominant bit 所覆蓋的時候，或是在確認區(acknowledgement filed)時，發送節點送出的 recessive bit 被其他接收節點發出的 dominant bit 所覆蓋的時候，不會被認定為位元錯誤。

- 填充錯誤(Stuff Error)

  若在適用位元填充的區域中(從欄框起始到循環多餘碼區)，偵測到連續六個相同位準的位元時，則產生填充錯誤。

- 循環冗餘碼錯誤(CRC Error)

  CRC 序列中包含發送方節點 CRC 的計算結果，假如接收節點收到的 CRC 序列與計算的結果不同時，則產生循環冗餘碼錯誤。

- 格式錯誤(Form Error)

  當一個固定形式的區域：循環多餘碼區、欄框終止、欄框間隔、確認邊界，含有一個或多個非法位元的時候，則產生格式錯誤。

- 確認錯誤(ACK Error)

  當在 ACK SLOT 所監視的位元不為 dominant bit 的時候，表示沒有任何節點接收到資料，則產生確認錯誤。

以上是 CAN 所定義錯誤類型，若訊息的傳輸過程中，節點偵測到以上各類錯誤之一，便發送錯誤欄框通知其餘節點，發送這筆訊息的節點必須重新傳送訊

息，這樣的方式可保證接收訊息的正確性，但卻無法避免經常發生問題的節點對於網路所造成的負擔，當網路中有這樣的節點存在，造成反覆的重傳訊息與錯誤訊息的產生，都會提高網路負載且降低傳輸效能。因此，如何將錯誤過多的節點排除，成為提高網路傳輸效率的課題，CAN 對此採取的解決方式，是利用計算節點發生錯誤的次數，得知每個節點傳輸的狀態，一旦錯誤發生地太頻繁，便減少其存取節點的次數，底下就是相關的細節說明。

◆ CAN 錯誤排除方式

根據 CAN 的規範，將 CAN 網路中的所有節點，分成下列三種可能的工作狀態，單一節點只能處在其中一種狀態，而當錯誤發生時，隨著狀態的不同，所發送的錯誤欄框也有所不同。

- 主動反應錯誤(Error Active)

主動反應錯誤的節點可正常參與網路通訊，且在偵測到錯誤時，發送的是主動式錯誤旗標。

- 被動反應錯誤(Error Passive)

被動反應錯誤的節點可參與網路通訊，但在偵測到錯誤時，發送的是被動錯誤旗標。此外，此類節點在傳輸完一個欄框後，除了傳送欄框間隔，還必須額外送出 8 個 recessive bit，才能進行下一筆傳輸，此 8 個 recessive bit 稱為暫停傳輸(suspend transmission)，若在暫停傳輸期間，其他節點開始進行傳輸，則被動反應錯誤節點轉為接收節點。藉由暫停傳輸的方式，可有效減少此類節點對於網路的存取。

- 離線(Bus Off)

顧名思義，離線的節點不允許參與任何網路通訊，所有訊號輸出的部份全部停止，此類節點便不會對網路造成任何影響。

上列三種狀態的區分，在於節點錯誤次數的多寡。每個 CAN 節點都具有一個傳送錯誤計數器(transmission error counter)和一個接收錯誤計數器(receive error counter)，當節點偵測到錯誤，其對應的計數器便遞增，反之，節點完成一次傳輸或接收動作，對應的計數器便遞減。當任一錯誤計數器值超過 127，狀態則由主動反應錯誤進入到被動反應錯誤，若是傳送錯誤計數器值超過 255，則進入離線狀態。狀態在被動狀態的節點，可在持續的傳送接收成功後，減少計數值而回到主動狀態；狀態在離線的節點，若要回到主動狀態，必須經由使用者提出要求(normal mode request)，且偵測到 128 次的連續 11 個 recessive bit 之後，才能再次回到主動狀態。由以上敘述，我們可將 CAN 的狀態轉換整理成的狀態機圖。

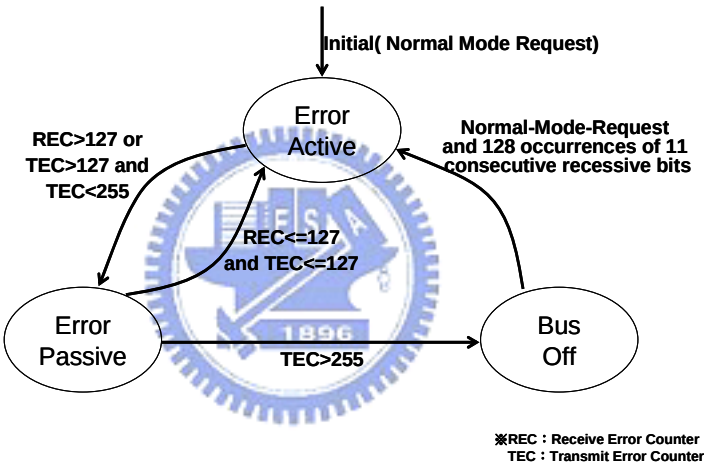


圖 2-6 CAN 錯誤狀態轉換

以上是 CAN 網路的介紹，了解其基本特性後，進一步經由實驗的方式，探討 CAN 在應用上，可能遭遇的相關問題。

2-2 CAN 網路分析實驗

網路系統所形成的分散式架構與集中式系統的不同，在於控制命令與迴授都必須經由網路傳輸，系統的效能參數不再只由控制器設計的好壞決定，各類網路架構下所衍生出的課題：網路傳輸延遲(network-induced delay)、取樣時間(sampling period)、資料遺失(data packet dropout)、時脈不同步(timer jitter)、網路的排程

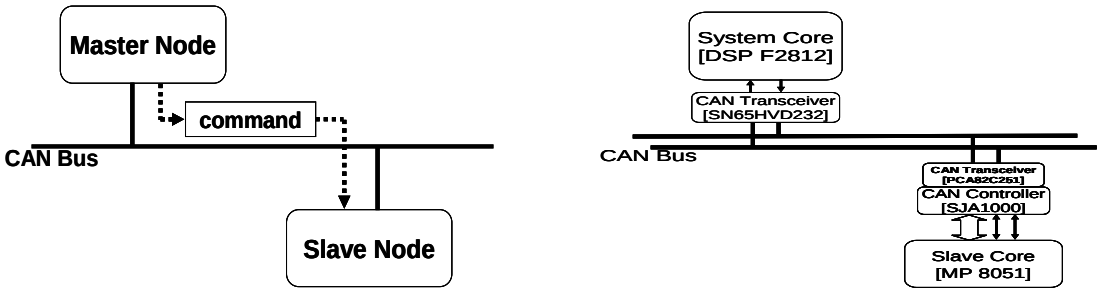
(network scheduling)，都將影響系統效能，甚至於對系統穩定性有極大的影響。

在本節的討論中，以網路傳輸延遲和節點間時脈同步為主，而網路傳輸條件和系統規格則間接影響系統表現，因此，吾人首先以不同的系統條件作為實驗參數，以資料遺失量為系統指標，藉由 CAN 網路進行傳輸實驗，另外透過網路傳輸時脈模型概念的引入，作為系統分析的基礎，歸納出 CAN 網路在不同傳輸條件下的結果分析。

2-2-1 CAN 分析實驗架構

CAN 的傳輸分析實驗，以兩個節點的傳輸架構為考量，其中一個節點為主控(master)節點，負責控制從屬節點的啟動停止，並產生實驗命令；另一個為從屬(slave)節點，負責接收主控節點的命令，並計算資料遺失量結果，下圖 2-7(a)為系統架構圖，至於硬體實做方面，master 節點以 TI 的 F2812 DSK 板(詳見第四章)搭配 CAN transceiver SN65HVD232 組成，slave 節點則是以微控器 8051 搭配 CAN controller SJA1000 和 CAN transceiver PCA82C251(詳見第四章)組成，下圖 2-7(b)為硬體架構。

在分析實驗的流程，主要是由 master 節點發送啟動指令，觸發 slave 節點的時脈中斷系統，在接下來的每個取樣週期中，master 節點不斷傳送命令，slave 節點負責接收並比對是否為該取樣時間的命令，如果不是，便將資料遺失計數器



(a) (b)

圖 2-7 (a) CAN 分析實驗架構，(b) CAN 分析實驗硬體架構

累加，以此方式循環，直到 master 節點傳送完所有命令，且發出停止指令，此時，slave 節點將關閉其時脈中斷，並回傳此一次實驗的資料遺失量作為結束，下圖 2-8 為 mater 的 DSP 程式流程圖，下圖 2-9 為 slave 的 8051 程式流程圖，依據這樣的架構，我們進行在不同傳輸條件下的實驗，便可以得到網路傳輸的特性。

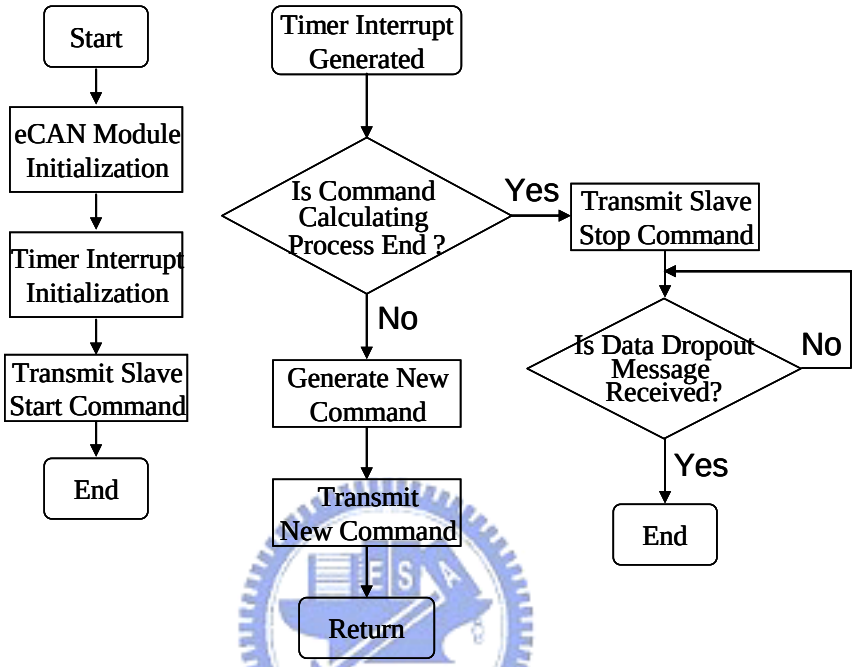


圖 2-8 CAN 分析實驗 DSP 端程式流程圖

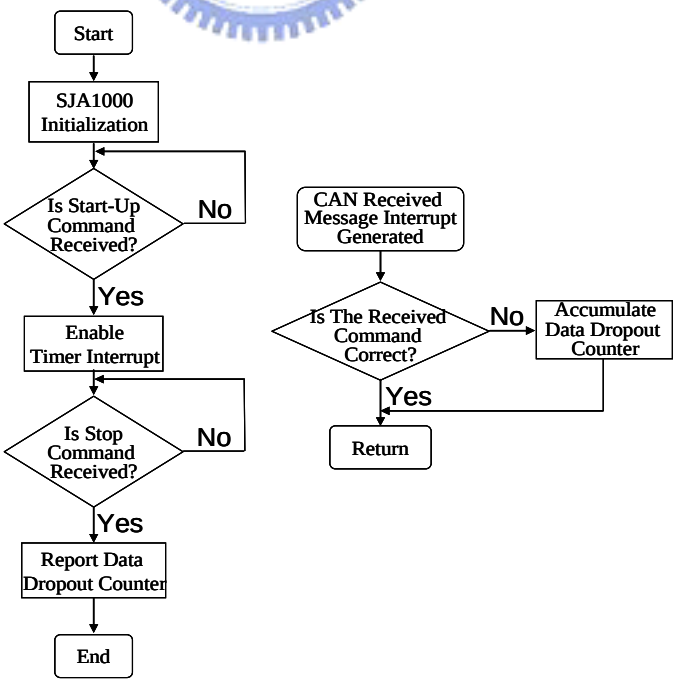


圖 2-9 CAN 分析實驗 DSP 端程式流程圖

2-2-2 網路傳輸時脈模型(Timing Diagram)[10]

在說明實驗結果之前，吾人希望藉由網路傳輸時脈模型的方式，作為結果分析的基礎，而網路傳輸時脈模型的概念，主要來自於訊息在節點之間耗費的傳遞時間，也就是傳遞上所造成的延遲，在這當中，可分成兩個基本類別：元件延遲(device delay)、網路延遲(network delay)，前者是指傳送(接收)節點處理訊息的時間，後者則是訊息在網路上傳遞的時間。在這樣的基本概念下，可加以細分成不同的區塊，進而得到訊息從傳輸(來源)節點到接收(目的)節點，總共所產生的延遲時間 T_{delay} 如下式(2-1)：

$$T_{delay} = T_{pre} + T_{wait} + T_{tx} + T_{post} \tag{2-1}$$

其中

T_{pre} 為傳輸節點的資料前處理時間(preprocessing time)

T_{wait} 為傳輸節點的等待傳輸時間(waiting time)

T_{tx} 為訊息在網路中的傳輸時間(transmission time)

T_{post} 為接收節點的資料後處理時間(postprocessing time)，

因此，進一步考慮兩個節點間的互傳時，便可得到如下圖 2-10 的結果，藉由此模型的建立，我們可對訊息傳遞有基本評估，進而規劃系統取樣時間、訊息排程等等參數，然而，吾人則透過這樣的模型，對接下來的實驗結果進行分析，以了解 CAN 網路實際的傳遞狀況。

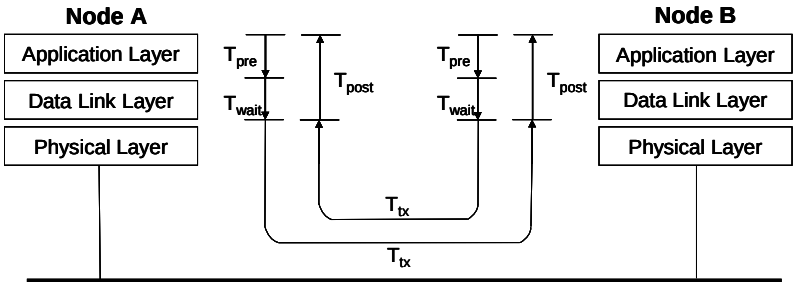


圖 2-10 網路傳輸時脈模型

2-2-3 CAN 分析實驗結果

實驗結果的討論上，本文主要以兩項因素：網路延遲和時脈漂移現象，作為探討的主題，首先，在網路延遲部份，以網路時脈模型在本實驗架構中的量測作為說明，本實驗架構中，僅有單一節點進行資料傳輸的動作，可將網路傳輸的等待時間(T_{wait})視為零，因此，T_{delay} 便由 T_{pre} 、T_{tx} 、T_{post} 三者決定，而三項參數的量測方法，是沿用 2-2-1 節的實驗架構加入一個 IO pin，透過程式改變此 pin 腳的準位並搭配示波器觀察所得，以 T_{pre} 的測量為例，master 端在主程式把 IO pin 設為低準位，發生時脈中斷的同時則設為高準位，直到完成命令運算後，重新設回低準位，如此便可得到如圖 2-11，T_{tx} 、T_{post} 也依照同樣的方式測量，前處理與後處理時間經多次實驗後，取其平均數作為近似值，傳輸時間則依照傳輸參數的不同而有所差異，但仍可獲得其改變的趨勢，實驗結果如下表 2-1、表 2-2，前者為 T_{pre} 、T_{post} 的實驗結果，由於前處理過程由 DSP 運算，執行時間相較於後處理的 8051 快了許多；而表 2-2 為不同條件下的傳輸時間，我們可進一步得到圖 2-12，由圖上可知，隨著傳輸速率降低與傳輸資料大小增加，傳輸時間便跟著上升。

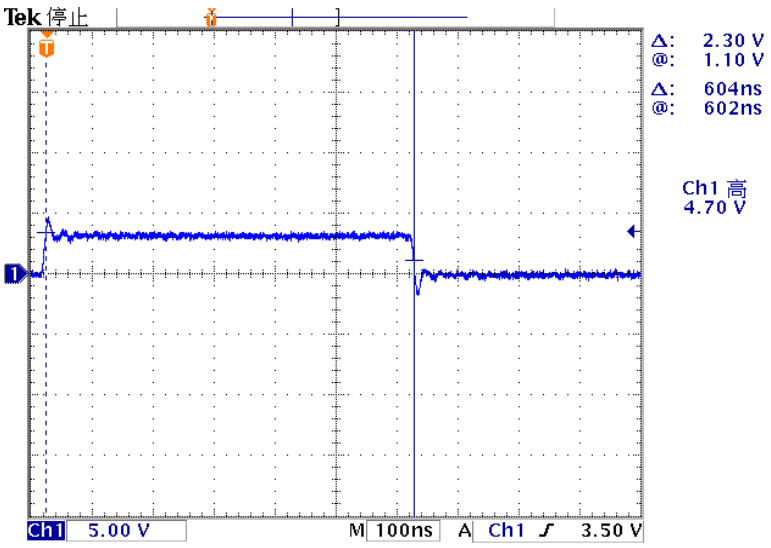


圖 2-11 前處理時間實驗波形

表 2-1 前處理與後處理時間實驗

實驗次數	1	2	3	4	5	平均
T_{pre}(ms)	0.000604	0.000604	0.000604	0.000604	0.000604	0.000604
T_{post}(ms)	0.0892	0.0888	0.0888	0.0888	0.0884	0.0888

表 2-2 網路傳輸時間(ms)實驗

資料大小(byte) / 速率(kbps)	0	1	2	3	4	5	6	7	8
1000	0.0828	0.0904	0.101	0.111	0.121	0.13	0.141	0.15	0.16
500	0.16	0.18	0.2	0.218	0.24	0.257	0.278	0.298	0.317
250	0.316	0.356	0.394	0.434	0.476	0.51	0.55	0.592	0.63
125	0.636	0.708	0.788	0.868	0.944	1.02	1.1	1.18	1.26
100	0.776	0.88	0.976	1.08	1.18	1.26	1.37	1.46	1.57

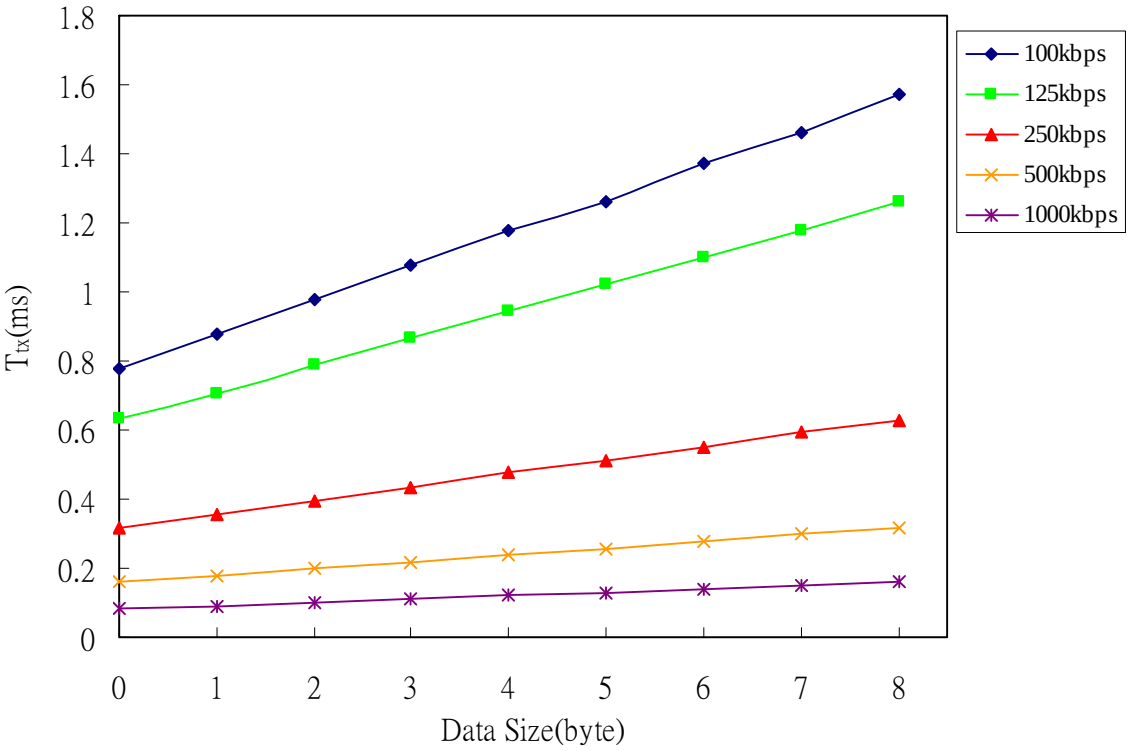


圖 2-12 網路傳輸時間實驗結果

在時脈漂移現象的部份，master 和 slave 間的時脈在起始狀態因傳輸延遲時間，造成兩者的時脈差異(t_{dif})，然而，隨著時間推移，此一時脈差距並非維持定值而是逐漸增大，所以，經過一段時間的運作之後，兩者時脈差距t'_{dif}必大於或等於原先的t_{dif}，圖 2-13 為其示意圖，虛線代表取樣點的位置，若以 master 的取樣週期為準，slave 的取樣點會呈現漂移的現象。這是由於各個節點間的時脈震盪，並不可能完全吻合，而這個微小的差異隨著時間增長，時脈同步性必定出現劣化的情形，導致時脈差距增大。因此，為了解此現象發生的狀況，沿用前述測量時脈模型參數的方法，在 master 和 slave 端各使用一 IO pin，在取樣中斷期間產生一高準位脈衝，經由示波器抓取兩者的輸出，可得到如圖 2-14 的一組脈衝序列，時間總長為 199ms，此處取樣週期為 5ms，總共進行了 40 筆訊息傳輸，其中 master 端為 DSP 端，使用的電壓準位較低(3.3V)，所以為振幅較小者，而圖 2-15 則為圖 2-14 中序列初始時的狀態，master 和 slave 端的取樣週期皆約為 5ms，由圖上可看出，此時兩者的時脈差距為 1.74ms；圖 2-16 則為序列末段時兩者的狀態，兩者的取樣週期都和剛開始的值相同，但進一步觀察時脈差距，此時差距已擴大為 2.46ms。因此，在所擷取的 199ms 中，時脈的差距約增大了 0.72ms，以此推估，相對於 master 的時脈，slave 1μs 約漂移 $3.62\times10^{-3}$$\mu$s 而漂移率約為 0.362%，數值看似影響不大，但經過時間累積，一旦 slave 落後 master 一個或者更多取樣週期，除了造成系統時脈錯亂，也將造成資料遺失。

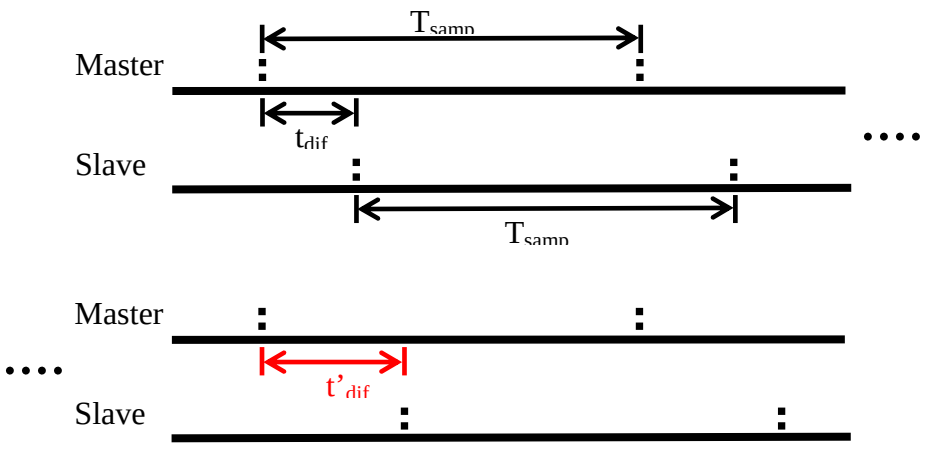


圖 2-13 時脈漂移示意圖

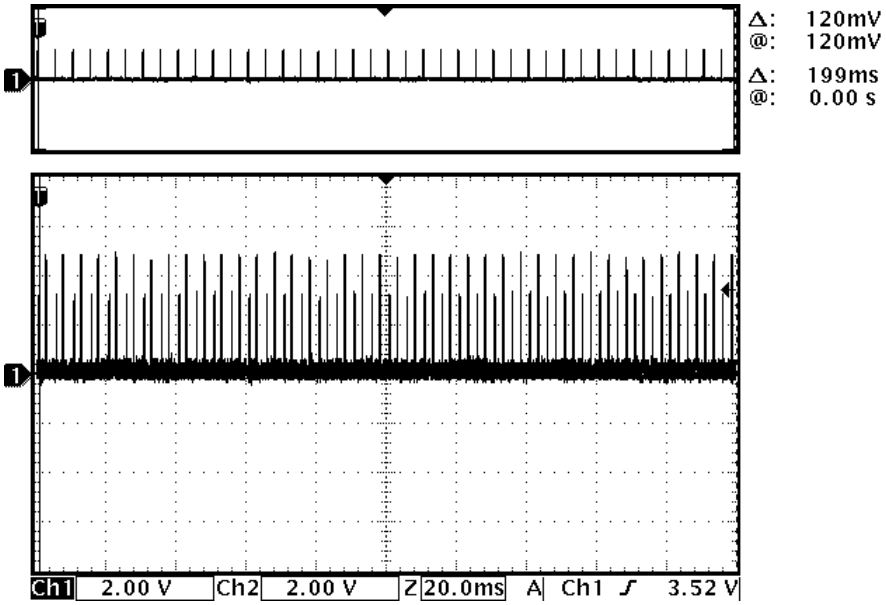


圖 2-14 系統取樣中斷脈衝序列

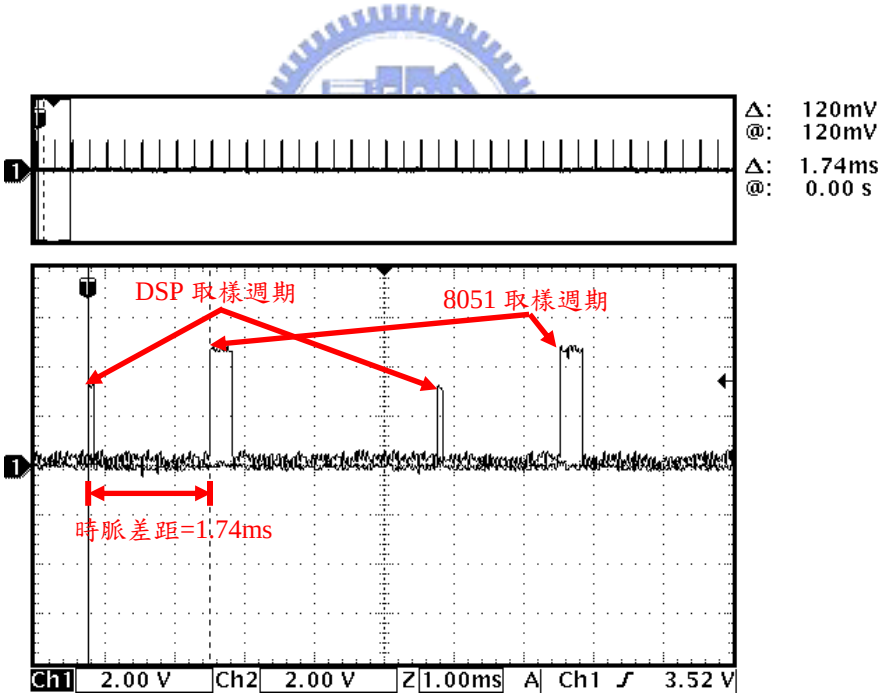


圖 2-15 序列初期時，master (DSP) 取樣週期=5ms 和 slave (8051)取樣週期=5ms，兩者時脈差距=1.74ms

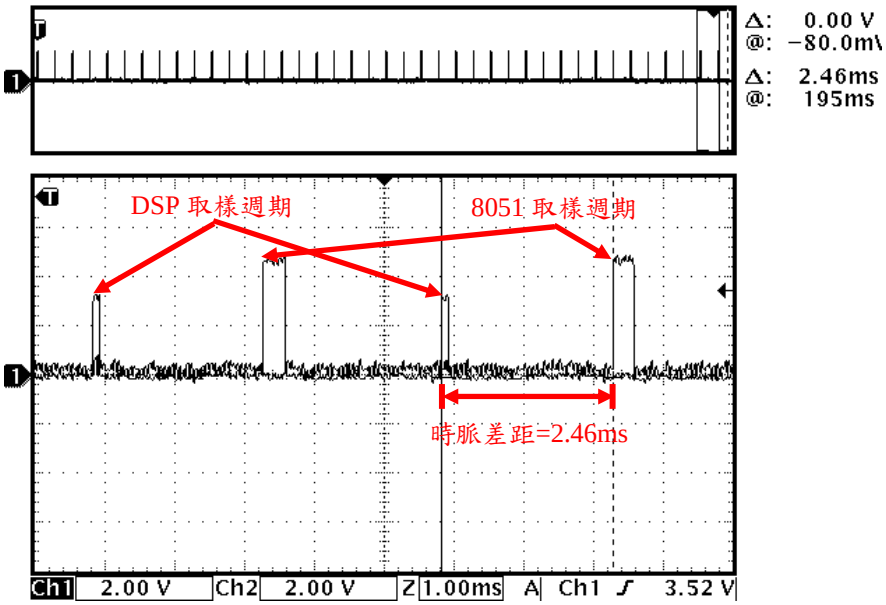


圖 2-16 序列末段時，master (DSP) 取樣週期=5ms 和 slave (8051)取樣週期=5ms，兩者時脈差距=2.46ms

以上為網路傳輸時間和時脈漂移現象的說明，接著，根據 2-2-1 節的實驗架構為基礎，吾人將傳輸資料大小、網路傳輸速率、系統取樣週期和傳輸資料量作為實驗參數，並計算網路的資料遺失率(p)作為實驗結果的主要評估，其定義如式 2-2，

$$p = \frac{N_{dropout}}{N} \times 100\% \qquad (2\text{-}2)$$

其中 N 為總傳輸資料量

$N_{dropout}$ 為接收端在第 i 個取樣點時，未接收到該取樣點之命令，此情形發生的次數

此外，本文對於資料遺失的定義為，只要非該取樣週期所需的命令，即視為資料遺失，因此，當 master 和 slave 的取樣週期受到時脈漂移影響，而相差一個取樣週期之後，slave 所接收的命令皆非該取樣點所需，其後的命令接收都將認定為資料遺失，所以，此處資料遺失率所涵蓋的意義，除了傳輸延遲的影響，也包含了 master 和 slave 時脈差異超過一個週期時的發生點，本文將此定義為 N_{drift}，如式 2-3，

$$N_{drift} = N - N_{dropout} \tag{2-3}$$

其中 N 和 $N_{dropout}$ 定義如 2-2 式，N_{drift} 代表時脈差異超過一個週期時，經過了幾次的取樣週期，換言之，就是時脈漂移現象對於系統造成影響所發生的時間點，如圖 2-17 所示，而 N_{drift} 和資料遺失率的關係則成反比，資料遺失率越大，時脈漂移的影響發生越早，N_{drift} 也就跟著越小。

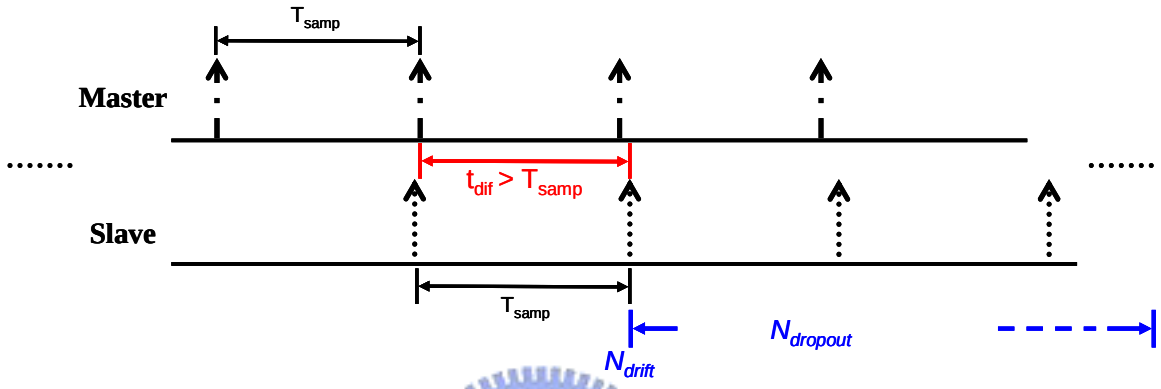


圖 2-17 N_{drift} 定義

接著，就以此兩項指標，對各個實驗結果加以說明：

(1) 傳輸資料大小：

本部份的實驗以改變傳輸資料的大小進行，資料長度為 2~8 位元組，圖 2-18 為傳輸速率 1000kbps、系統取樣時間 5ms 下，傳輸 2000 筆資料的實驗結果，由圖上可看出，在相同的傳輸狀況下，當傳輸資料長度增加，資料遺失量隨之增加，從 2-2-1 節的架構可知，slave 端的啟動是由 master 端發送訊息控制，兩者的時脈在起始狀態就因傳輸延遲，而存在一時脈差異(t_{dif})，當之後的資料傳輸延遲時間(t_{delay})超過t_{dif} ，也就是超過 slave 接收資料的時脈中斷時間，便造成了資料遺失的狀況，而根據前述的時脈模型，可知隨著傳輸資料長度增加，t_{delay} 的時間跟著增長，t_{delay} 大於 t_{dif} 的機率也隨之提高，資料遺失的情況就越發明顯。因此，我們可知在相同的傳輸速率與取樣週期下，隨著資料長度增加，資料遺失率的情況將隨之增加。

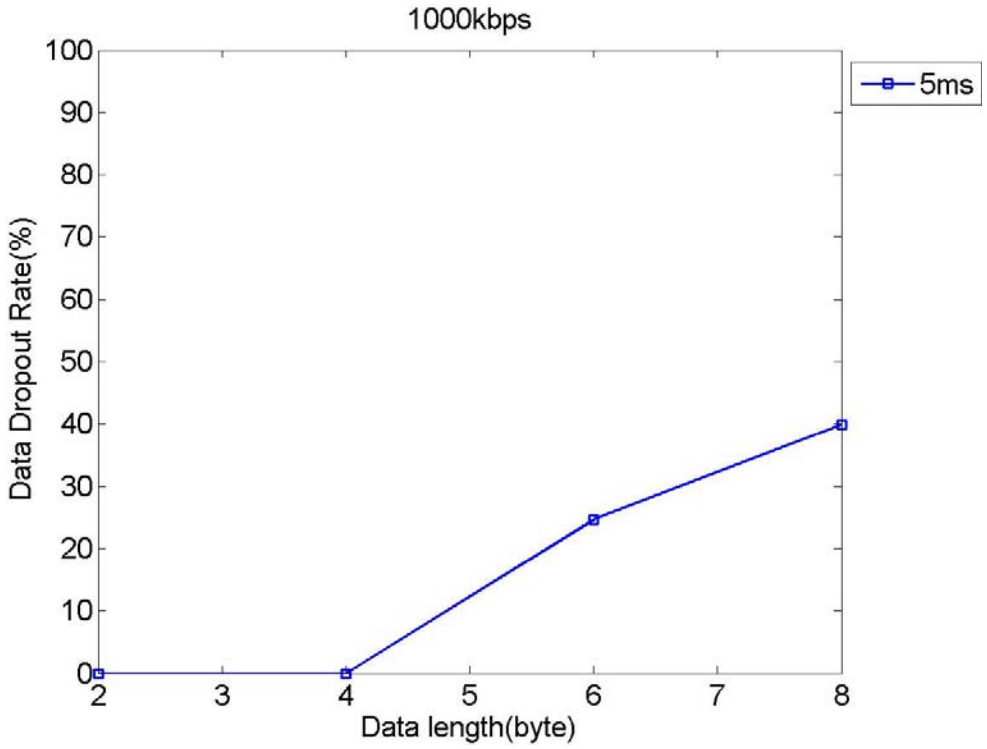


圖 2-18 不同傳輸資料大小的資料遺失率

(2) 傳輸速率：

本部份的實驗以改變 CAN 傳輸速率的方式進行，所使用的速度在 100kbps 到 1000kbps 之間，下圖 2-19 為系統取樣時間 5ms 下，傳輸 3000 筆資料的實驗結果，圖上可看出，在資料長度較小的情形，傳輸速率的影響並不明顯，其原因和取樣時間與傳輸資料量有關，稍後實驗結果會加以說明；在資料長度較大的情形，資料遺失率則隨著傳輸速率下降而顯著的上升，其原因和改變資料傳輸長度的情況類似，隨著速率下降而大幅增加了網路傳輸時間(t_{tx})，進而反應在t_{delay}之上，而由表 2-2 可知，在同樣資料長度的條件下，1000kbps 與 500kbps 的傳輸時間相近，實驗結果便相去不遠，同樣地，125kbps 與 100kbps 傳輸時間類似，其表現也呈現相同情況，至於 250kbps 則介在以上兩類之間。由此可知，在相同的取樣週期與資料長度下，隨著網路傳輸速度下降，資料遺失率將增加。

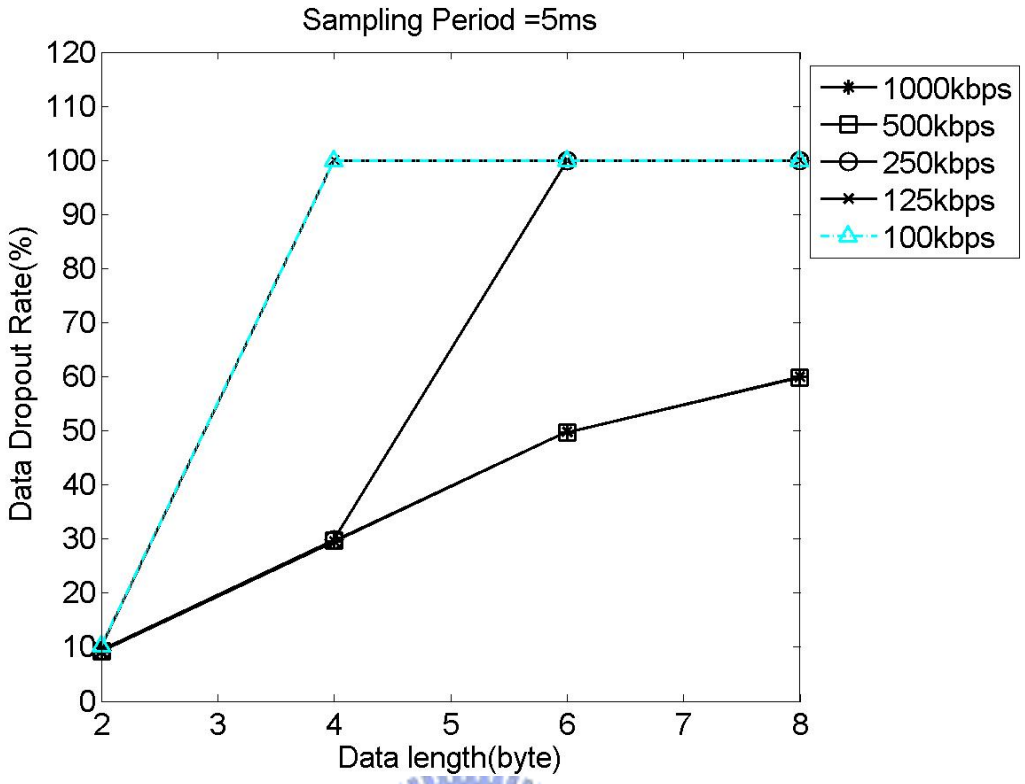


圖 2-19 不同傳輸速率的資料遺失率

(3) 取樣週期：

前述的傳輸速率實驗結果中，在傳輸資料量長度為 2bytes 時，所有傳輸速率的結果都相去不遠，主要是由於此時網路傳輸所造成的延遲時間，相較於時脈漂移所造成 slave 取樣點延後的時間，後者漂移的時間較長，所以，在取樣點向後移的情況下，傳輸速率的影響相對地減少，導致各個速率的實驗結果都很相近。

因此，本部份實驗經由改變系統取樣週期的方式進行，以觀察時脈同步性對於網路系統的影響。在 1000kbps 下的實驗結果，如圖 2-20 所示，由圖上可看出，三個不同取樣週期的結果完全被區分出來，而 10ms 下的實驗在資料長度短的情況下，遺失率幾乎為零，1ms 的實驗則從一開始就發生相當劇烈的資料遺失現象，而 5ms 則介在兩者之間，其主要原因是，若將 slave 時脈漂移率視為定值，而由圖 2-13 可知，取樣週期較大者，可容許的時脈漂移時間($t_{sample} - t'_{dif}$)相對地增大，因此，在 10ms 的取樣週期下，對於漂移現象的容忍度相當高，只在 8bytes 的狀

況下，才開始發生遺失現象；相對地，前面的漂移時間測量中，在 40 筆傳輸裡，slave 已漂移了 0.72ms，與 1ms 的取樣週期相當接近，所以，1ms 的取樣週期下，經過一段時間的傳輸後，slave 的時脈便落後 master 一個取樣週期，自此之後，slave 所收到的每一筆資料，都與該取樣週期所應接收的訊息不同，而被視為資料遺失，導致即便在 2bytes 的實驗下，資料遺失卻十分嚴重。

由 N_{drift} 的角度來看，如圖 2-21 所示，虛線部份為各個取樣時間下的 N_{drift} 估測值，主要由先前的網路延遲以及時脈漂移實驗加以估測，實線部份則是由圖 2-20 的實驗結果所得，由圖上可看出，隨著取樣週期縮短，N_{drift} 值跟著減少，這是由於取樣週期縮短後，時脈漂移越早使得節點的時脈差距，相差一個取樣週期以上。此外，在 N_{drift} 估測值與實測值可看出，估測值僅以時脈漂移的影響為主，各個封包大小的值都相同，但實際傳輸中，包含了網路延遲的影響，因此，N_{drift} 將因為網路延遲的影響，隨著封包大小增加而減少。

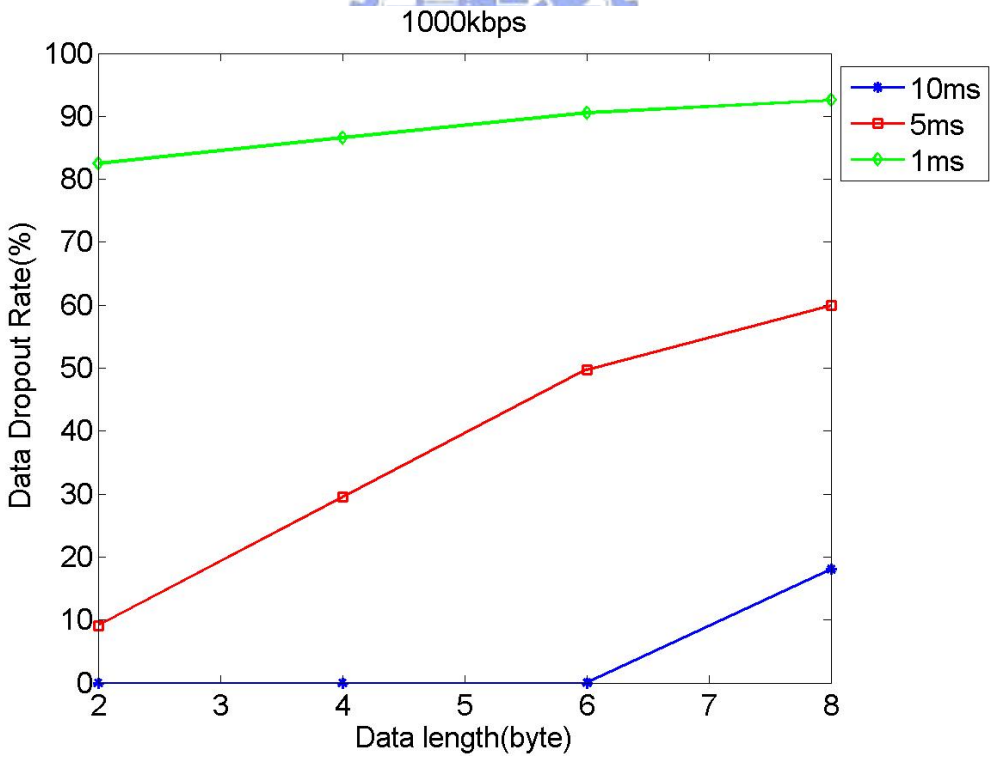


圖 2-20 傳輸速率 1000kbps 下，不同取樣週期的資料遺失率

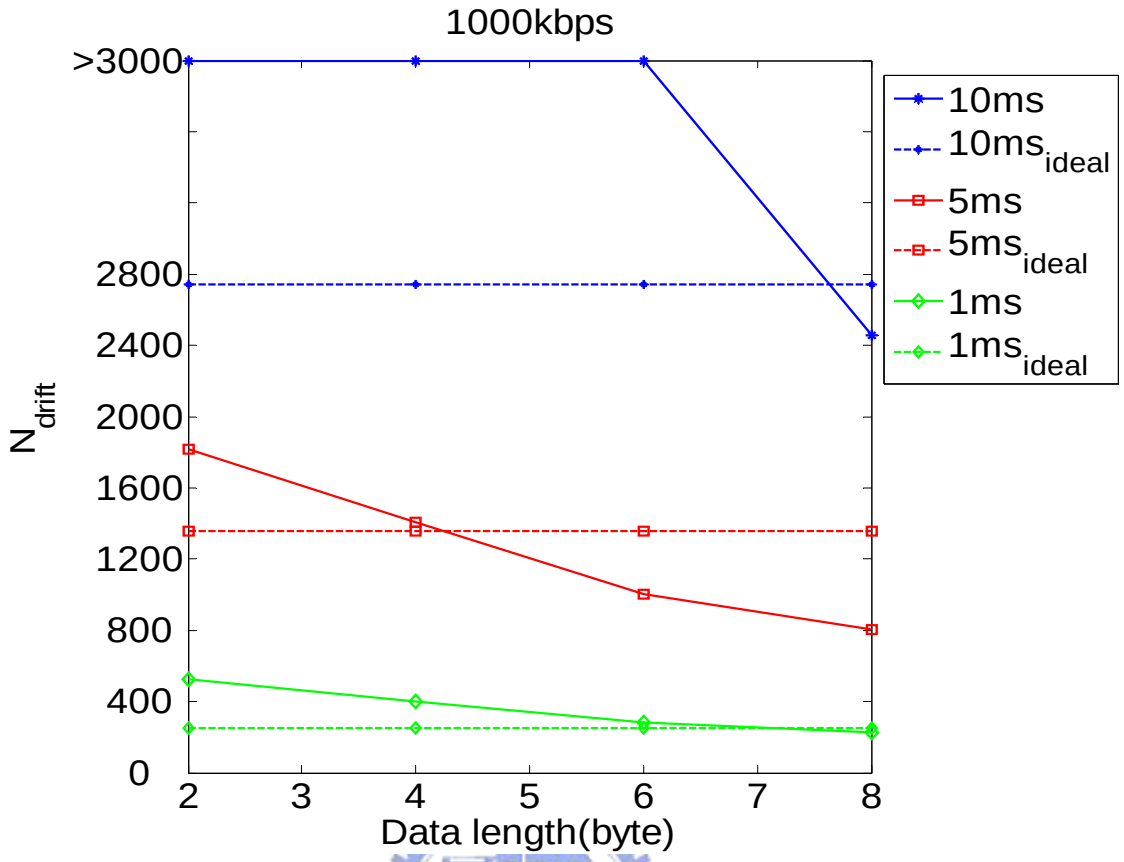


圖 2-21 不同取樣週期的 N_{drift} 值比較

(4) 傳輸資料量：

在改變取樣週期的結果中，說明了時脈漂移的影響，本部份的實驗則由另一個角度探討時脈同步性的問題，由於時脈漂移的現象隨著時間累積而擴大，而透過改變總傳輸訊息量的方式，可在相同的傳輸條件下，凸顯出其影響，在 1000kbps、1ms 取樣的實驗結果，如圖 2-22，由圖上看出三組實驗結果被區分開來，隨著傳輸量增加，資料遺失率也跟著上升。由 N_{drift} 值來看，如圖 2-23 所示，由圖上可知，由於傳輸條件相同，三種傳輸資料量的 N_{drift} 值十分相近，在 N_{drift} 值相近的情況下，進行較多筆傳輸的實驗，將造成越多的資料遺失，因此，隨著資料傳輸量增加，其資料遺失量將隨之增加。

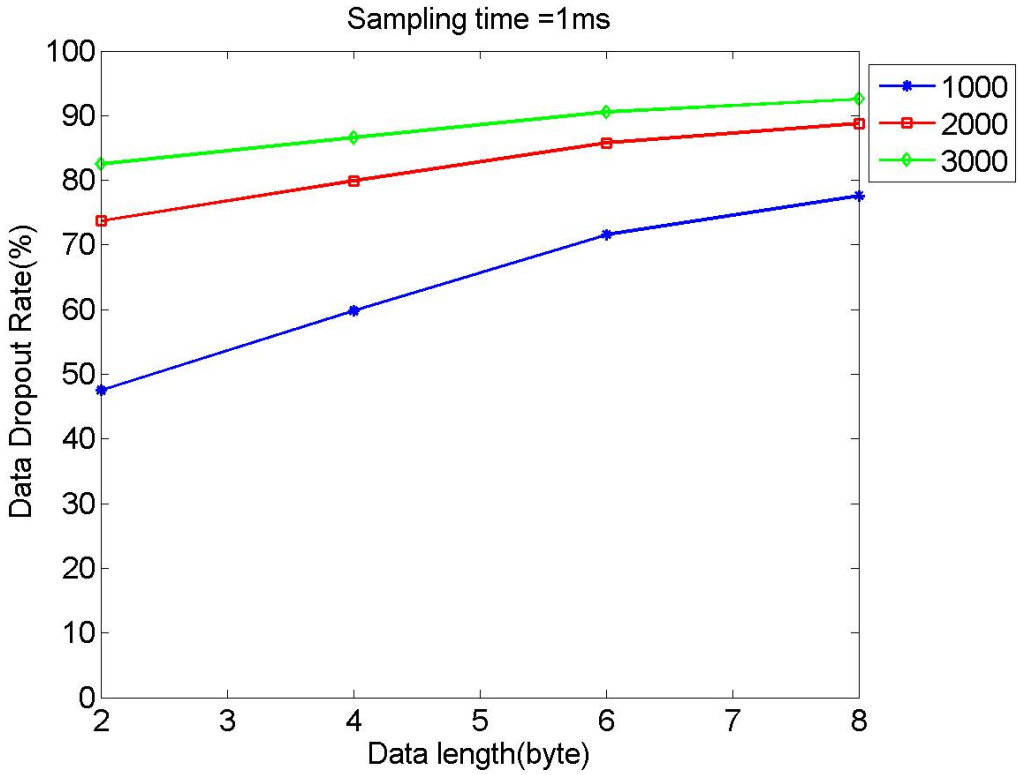


圖 2-22 傳輸速率 1000kbps,取樣週期 1ms,不同傳輸資料量的資料遺失率

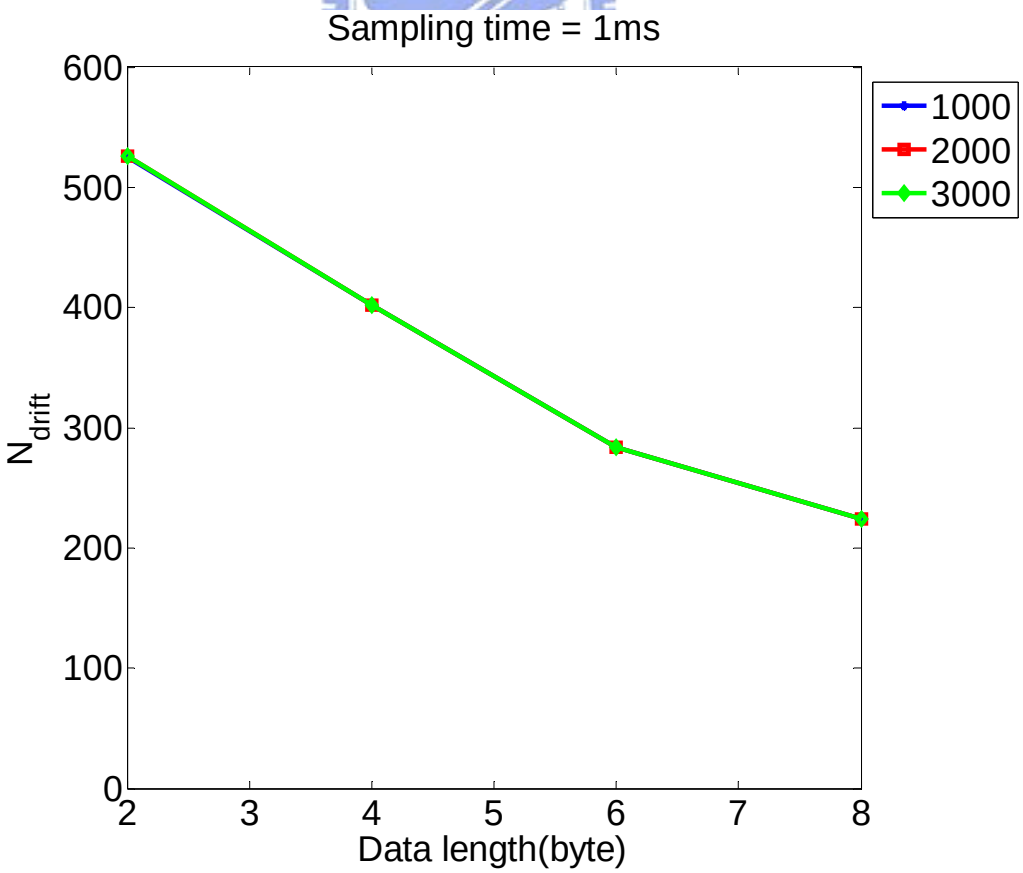


圖 2-23 不同傳輸資料量的 N_{drift} 值

2-2-4 小結

在本節中，經由 Master-Slave 網路架構的建立，完成了網路分析實驗的設計，而透過網路時脈模型概念的引入與相關參數的量測，對所設計之網路系統與 CAN 網路的各項特性，如：前(後)處理時間、傳輸時間等，得到了基本的了解，也作為實驗結果分析的基礎。

在實驗結果的分析上，藉由所測量的時脈模型參數，逐一對各個結果提出討論，而由傳輸資料大小和傳輸速率的實驗中，可得到兩者所造成之網路延遲時間，對於資料遺失率造成之影響；由取樣週期和資料傳輸量的實驗，則驗證了時脈漂移現象的影響。同時，由此歸納出影響網路系統資料遺失率的兩項主要因素：網路延遲時間、時脈漂移現象，此外，若以取樣週期長短作為區隔，當取樣週期較大時，可允許較差的時脈同步性，時脈漂移的影響力相對降低，資料遺失率主要受傳輸的延遲時間影響；而當取樣週期較小時，網路延遲時間的影響較少降低，主要影響則由時脈漂移現象所決定。

第三章　以 CANopen 為基礎之應用層同步協定

本章主要介紹 CAN 應用層通訊協定，並根據其架構與概念，設計出所需的協定，以利系統之整合，且延續第二章的實驗結果，提出網路時脈同步方法，以改善時脈漂移問題。

3-1 CAN 應用層通訊協定：CANopen [14]

3-1-1 CANopen 簡介

在 CAN 原本的通訊規範中，僅包含 OSI 網路模型中的資料連結層和實體層這兩個部份，其餘各個部份並未加以定義，因此，當使用者以 CAN 作為通訊媒介時，便必須自行定義所使用的資料型態、高階網路行為等等規格，以 OSI 模型的角度來看，即是定義應用層(application layer)的通訊協定。然而，這卻造成系統開發上的緩慢，且系統間缺乏共通性。因此，以 CAN bus 為基礎的應用通訊協定因應而生，其中最常見的為 CANopen 和 DeviceNet，透過這些應用層協定，不同系統間便建立了共通的通訊規範，如此一來，便可加速系統間的整合，而此處的介紹則以 CANopen 為主。

CANopen 最初同樣是由德國 Robert Bosch 所開發，在 1995 年轉由 CiA (CAN in Automation)處理發表，原先是為了機械控制所開發的網路協定，然而，近年來，第四版的 CANopen (CiA DS 301)已成為歐洲的標準規格(EN 50325-4)，使用範疇跟著被推展到各類應用上，其主要優點在於，相較其他以 CAN 為基礎的通訊協定除了對原先的 CAN 加以修改，且採用較複雜的網路架構，以 DeviceNet 為例，DeviceNet 修改了實體層規範，並加入了網路層(network layer)和傳輸層(transport layer)的概念，反觀 CANopen 的實現概念則較簡單而直接，並未對 CAN 原先的規格加以修改，而是直接規劃出應用層協定，由結合前述的 OSI 網路模型來看，可得到如下圖 3-1 所示，其中實體層與資料連結層如同前述的 CAN 網路架構，

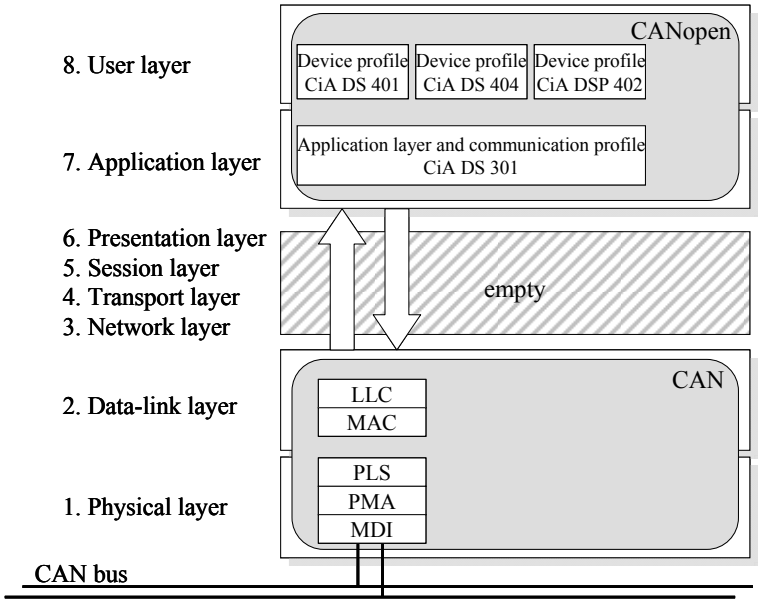


圖 3-1 CANopen OSI 網路模型

應用層(application layer)是上述的 CiA DS 301 為 CANopen 主要通訊協定，提供通訊用物件，使用者層(user layer)則是 CANopen 依據不同應用場合，加以制定的物件規範，而同一類型的物件集合則可稱為 device profile，如 CiA DS 401 提供一般 I/O 的功能，而 CiA DS 301 則提供 CANopen 通訊物件，兩者皆屬於 CANopen 協定的一部分。

根據 CANopen 的通訊規範，可歸納出以下特點：

- 提高系統整合性。因應各種應用場合，透過 CANopen 提供的應用物件，系統整合可經由網路達成。
- 系統擴充性佳。延續 CAN 網路本身的特性，節點可隨時加入或移除，配合應用物件的使用，提高系統建構上的彈性。
- 完善網路管理機制。CANopen 提供網路管理主要物件，包含了節點同步、節點操作控制和監測，都可透過 CANopen 達成。

以上為 CANopen 簡介，接著，在 3-1-2 節介紹 CANopen 的通訊物件與通訊模型，3-1-3 節則為 CANopen 的網路管理協定。

3-1-2 CANopen 通訊協定

在上節中，介紹了 CANopen 的 OSI 網路架構，是由網路特性加以區分，若從物件功能的角度來看，則可以將 CANopen 節點分為三個部份(如圖 3-2)：

(a) 通訊端(communication)

通訊端主要是透過底層網路架構提供網路通訊功能，達到網路傳輸與接收訊息的目的，圖上的通訊物件可視為網路線上傳輸的訊息封包。

(b) 物件表(O.D., object dictionary)

物件表(以下簡稱 O.D.)主要是所有對通訊端和應用端物件能造成影響的訊息封包集合，並將這些封包轉換到目的地，因此，O.D.就如同處在兩端的介面，提供通訊端和應用端物件間的轉換機制，而圖中的每個項目(entry)即是定義其對應關係。

(c) 應用端(application)

應用端則是負責與處理程序(process)互動，提供其所需的應用物件並將需要傳輸的應用端物件提供給 O.D.。

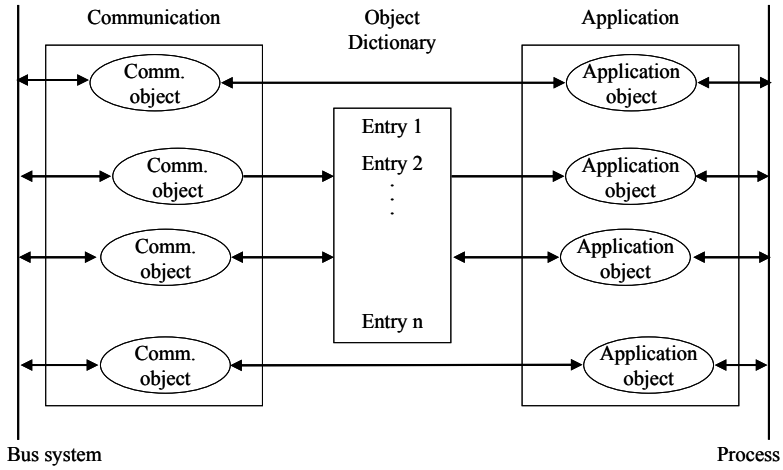


圖 3-2 CANopen 物件模型

由上述可知，CANopen 協定主要提供底層通訊物件和上層應用物件之間的轉換機制(O.D.)，且對於應用物件加以規劃定義，而因應不同的物件類型，也衍生

出了各類的網路行為，底下就針對 object dictionary、通訊模型和通訊物件[1]加以介紹。

◆ Object Dictionary (O.D.)

O.D.的功能在於提供物件的對應關係，實現方式類似一個預先規劃的表格，紀錄所有使用的物件資訊，包含資料類型、通訊參數與物件定義等等。至於存取O.D.內容的方式，則是透過 16 位元索引(index)和 8 位元的副索引(sub-index)，前者可支援最多 65536(2^16)個項目，後者則是為了存取複雜的資料型態(如：陣列)所設計，而根據資料型態和使用目的，可分為如表 3-1 中的各種類別。

0001h~001Fh 的靜態資料型態和 0020h~003F 的複合資料型態，為 CANopen 預設的資料類型，前者包含一般標準類型，如布林數、浮點數、字串等等，後者則是根據這些基本型態所組成的複合型態，如整數(字串)陣列；0040~005F 同樣提供標準資料類型，差異在於隨著製造商的設計而有所差異；0060~009F 的資料類型，則隨著不同的應用場合而額外規範。以上部份為資料型態的定義。

至於通訊參數和物件定義，則由 1000~BFFF 的各個 profile 加以規範，其中 1000~1FFF 為通訊用參數設定用，即由前述之 CiA DS 301 所規範；2000~5FFF 為製造商自行定義之參數設定；6000~9FFFF 則為應用層使用的所有資料項目，然而，不同 device profile 使用在不同的應用場合，一個 device profile 便對應到一群物件，而單一節點可能有多個應用目的，為此，CANopen 允許單一節點具有 8 個 device profile，每個 profile 擁有 2048 個資料項目，所以，6000~67FF 為第一個 profile 所使用，6800~6FFF 為第二個 profile 所使用，以此類推。

[1] 此處的通訊物件係指應用端的通訊物件

表 3-1 CANopen O.D.分類

Index(hex)	Object
0000	not used
0001~001F	Static Data Types
0020~003F	Complex Data Types
0040~005F	Manufacturer Specific Complex Data Types
0060~007F	Device Profile Specific Static Data Types
0080~009F	Device Profile Specific Complex Data Types
00A0~0FFF	Reserved for further use
1000~1FFF	Communication Profile Area
2000~5FFF	Manufacturer Specific Profile Area
6000~9FFF	Standardised Device Profile Area
A000~BFFF	Standardised Interface Profile Area
C000~FFFF	Reserved for further use

以上的介紹以 O.D.架構為主，至於其細部的各個項目，由於種類繁雜，便不詳加介紹。

◆ 通訊模型(Communication Model)

CANopen 建立了應用層物件定義和其對應關係，基於這些規範而發展出各類的通訊模型，本部份便以此為主，然而，網路與處理程序間的互動也與通訊模型相關，因此，底下先就這部份說明，接著介紹通訊模型，以了解 CANopen 節點的通訊形式。

根據 CANopen 所制定的規則中，可歸納出應用程序和網路應用層的溝通方式可分為：

- 要求(request)

  由應用程序發給應用層，以存取資料或要求網路服務。

- 指示(indication)

  由應用層發給應用程序，表示應用層的事件觸發產生或是應用層由其他處理程序接收到要求。

- 回覆(response)

  由應用程序發出，以回覆前一個接收到的指示。

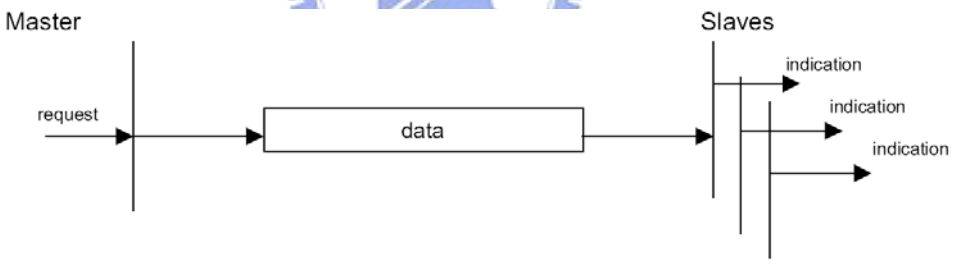
- 確認(confirmation)

  由應用層發給應用程序，以回報前一個提出的要求，所得到的結果。

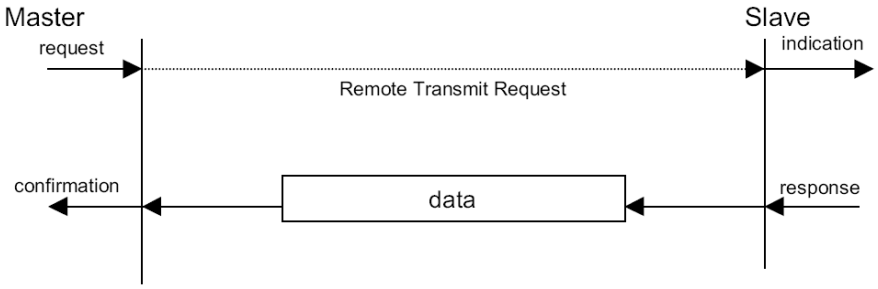
基於這四種方式，則可將 CANopen 的通訊模型，歸納成以下三個類別：Master/Slave、Client/Server 和 Producer/Consumer，底下就針對各個模型說明。

- Master/Slave

CANopen 架構下必定存在一 master，其餘節點則為 slave，後者的動作皆由前者所控制，其通訊模型如圖 3-3 所示，(a)是不需經確認的方式，master 傳送資料給各個 slave 之後，slave 發出指示給應用程序且引發對應的處理機制；(b)是必須經過確認的方式，master 傳送遙控傳輸要求(遙控欄框)，引發 slave 產生指示以及回覆，進而回傳資料到 master，並對 master 端的應用程序回報確認結果。



(a) Unconfirmed Master/Slave model



(b) Confirmed Master/Slave model

圖 3-3 Master/Slave 通訊模型

- Client/Server

Client/Server 的通訊方式類似需確認的 Master/Slave 通訊，差別為後者主

要用於讀取遠端節點的資料或狀態，而 Client/Server 方式主要用於點對點之間的大型資料寫入與讀取，因而較適用於非即時性(non-real-time)的資料傳輸，其通訊模型如圖 3-4 所示。

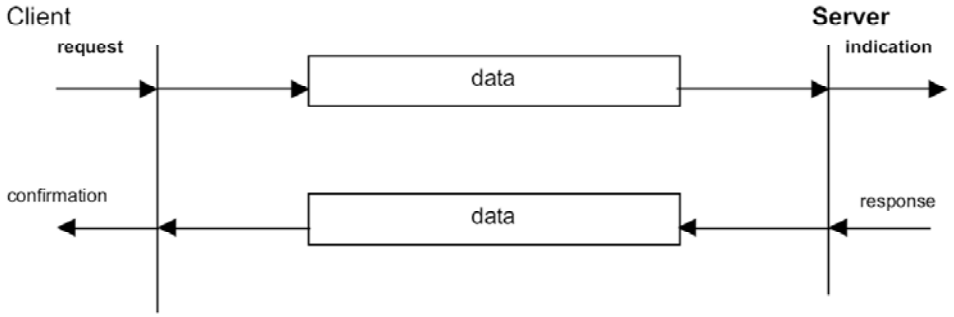
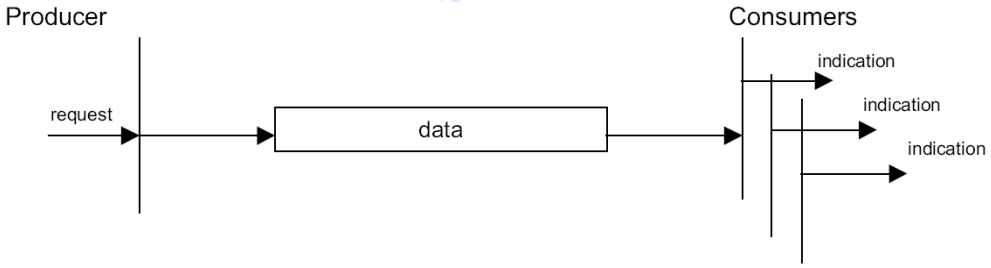


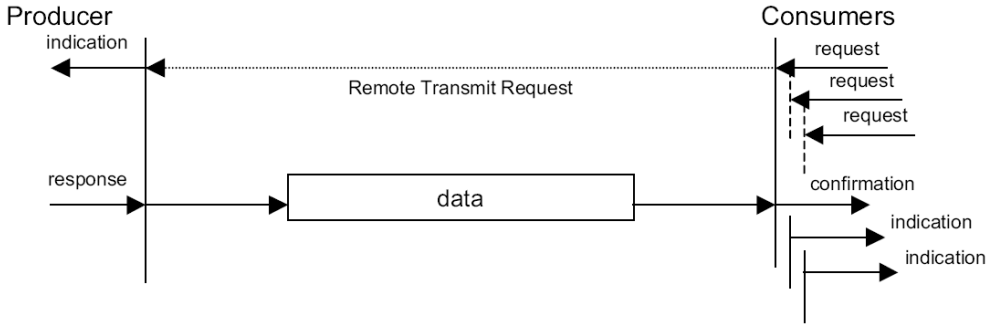
圖 3-4 Client/Server 通訊模型

- Producer/Consumer

Producer/Consumer 的通訊方式，就如製造者與消費者的關係，單一個 producer 可對應零個或多個 consumer，其通訊方式如下圖 3-5，(a)是由 producer 主動送出資料後，而引發各個 consumer 產生對應的機制，此方式稱之為 Push 模型；(b)則是由各個 consumer 透過遙控欄框，要求 producer 提供所需資料，此方式稱之為 Pull 模型。



(a) Push model



(b) Pull model

圖 3-5 Producer/Consumer 通訊模型

◆ 通訊物件(Communication Object)

在通訊模型的介紹中，說明了以通訊物件發展出的三種通訊形式，然而，通訊行為主要則是由通訊物件完成，CANopen 通訊物件是將 CAN 的訊息封包加以分類，而每個訊息封包即為一個通訊物件，其分類方式則是將標準 CAN 格式中的 11 位元識別碼，規劃為物件識別碼(COB-ID)，前 4 位元為功能類別(function code)，用以表示物件功能，後 7 位元為節點識別碼(Node-ID)，用以表示發送訊息的節點，Node-ID 象徵著每個節點都有專屬的編號，因此，CANopen 網路中最多可有 128(2^7)個節點，而根據其功能分類，可得到如表 3-2，其中可分為一般資料傳輸用以及系統管理用兩大類，PDO 和 SDO 主要用於資料傳輸和參數設定，其餘各類物件則用於系統管理，在此，以前者的介紹為主。

表 3-2 CANopen 通訊物件

Function code (binary)	COB-ID	Communication object
0000	0	NMT Service
0001	128(80h)	SYNC Message
	129(81h) ~ 511(FFh)	Emergency Messages
0010	256(100h)	Time-Stamp Message
0011	385(181h) ~ 511(1FFh)	Transmit PDO 1
0100	513(201h) ~ 639(27Fh)	Receive PDO 1
0101	641(281h) ~ 767(2FFh)	Transmit PDO 2
0110	769(301h) ~ 895(37Fh)	Receive PDO 2
0111	897(381h) ~ 1023(3FFh)	Transmit PDO 3
1000	1025(401h) ~ 1151(47Fh)	Receive PDO 3
1001	1153(481h) ~ 1279(4FFh)	Transmit PDO 4
1010	1281(501h) ~ 1407(57Fh)	Receive PDO 4
1011	1409(581h) ~ 1535(5FFh)	Transmit SDO
1100	1537(601h) ~ 1663(67Fh)	Receive SDO
1110	1793(701h) ~ 1919(77Fh)	NMT Error Control

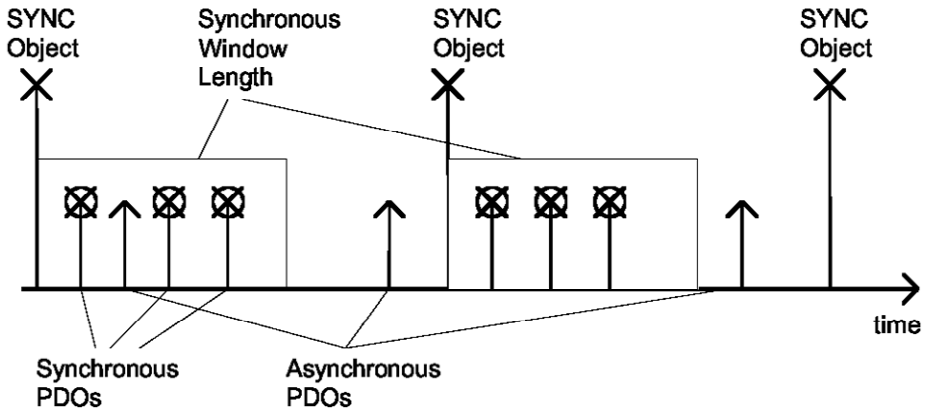


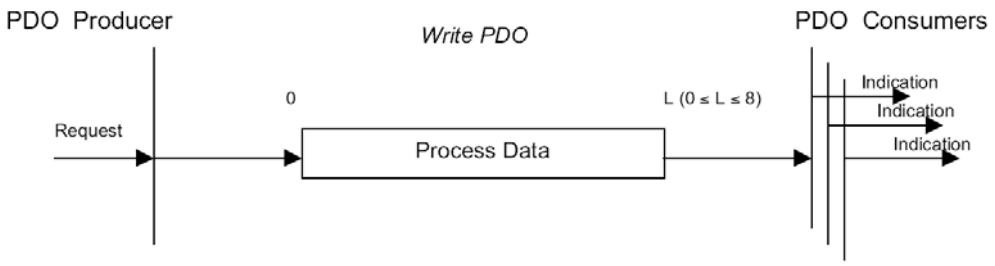
圖 3-6 同步與非同步傳輸方式

PDO (process data object)和 SDO (service data object)是為了資料傳輸之用，前者主要用於即時性資料傳輸，後者則用於非即時性的參數傳遞，因此，所使用的通訊模型也有所差異，以下則為個別說明。

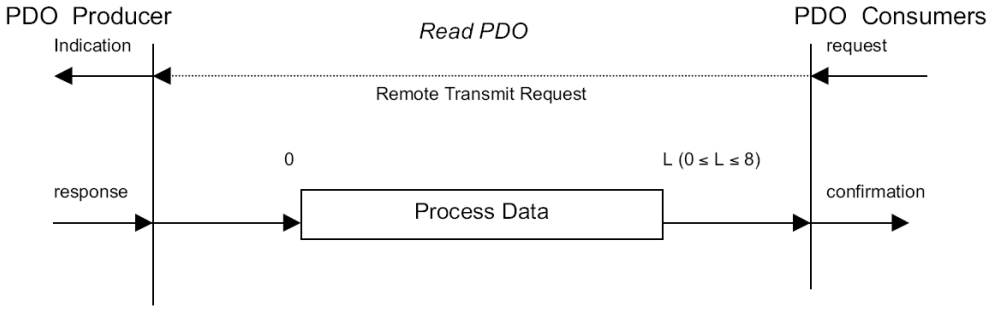
- PDO

PDO 主要用於即時性資料傳遞，而其傳輸方式分為同步(synchronization)傳輸與非同步(asynchronization)傳輸兩類，圖 3-6 為一範例，同步傳輸是基於同步物件(SYNC object)(詳見 3-1-3 節)的發送，一旦接收到同步物件後，觸發各個節點在預設的同步時間(synchronization window length)之內，發送同步 PDO，這樣的方式可達到節點同步的效果；而非同步性 PDO 則不需同步物件觸發即可發送。

PDO 的類型可分為傳輸(TPDO)和接收(RPDO)兩類，TPDO 為實際用來傳輸的物件，RPDO 則與欲接收的 TPDO 搭配，所以，對應的 TPDO 和 RPDO 具有相同的 COB-ID。而由於 PDO 採用 Producer/Consumer 的通訊形式，因此，傳輸 TPDO 的節點稱為 PDO producer，接收 TPDO 的節點則為 PDO consumer，其通訊協定如圖 3-7 所示，其中 PDO 的傳輸資料長度和 CAN 的資料長度相同(0~8bytes)，相對於 SDO，PDO 沒有額外的處理機制和網路負載，因此，較適用於即時性資料的傳輸。而寫入 PDO 的方式是採用 Producer/Consumer push 模型，讀取 PDO 則是採用 pull 模型。



(a) Write PDO 通訊協定



(b) Read PDO 通訊協定

圖 3-7 PDO 通訊協定

- SDO

為了達到即時性傳輸，PDO 將其傳輸方式限制在一次傳輸之中，且資料長度最多為 8bytes，然而，在 O.D.的介紹中，曾提及 CANopen 所支援的資料型態包含字串或陣列等複雜的資料型態，這些資料物件明顯無法透過 PDO 的傳輸方式完成，因此，SDO 的設計便是為了大型資料傳輸所設計，這樣的方式必須將資料分為多筆訊息分批傳送，接著由接收端組合為完整資料，但這樣一來，訊息之間可能因網路傳輸狀況不佳，使得接收端遲遲無法得到完整資料，所以，SDO 並不適用於即時性資料傳遞，而是用於大型參數資料傳遞。

SDO 的傳輸基本上為非同步方式，其通訊形式則使用 Client/Server 模型，依資料流向則分為上傳(upload)和下載(download)，至於傳遞資料的方式則可分為三種類型：快速(expedited)、分批(segmented)和集體(block)，主要差別是 expedited 最多僅能傳輸 4bytes 資料，segmented 和 block 都可支援到 128bytes 傳輸，但 segmented 必須在每筆傳輸結束時進行確認，block 則是在完整的傳輸結束之後，再加以進行確認。在此，以 block 傳輸為例，下圖 3-8 為 block 下載通訊模型，其中 c 代表是否繼續傳輸。0 表示尚有資料需傳輸，1 表示資

料傳輸結束。seqno 為資料編號，用以表示資料次序。每次傳輸都必須進行起始動作，而在資料傳輸的過程中，單次資料傳輸之間，不需對各筆資料進行確認，而是在資料傳輸完成後，server 端將回傳資料比對結果作為結束。透過這樣的方式，便可達到傳遞大型資料的目的。

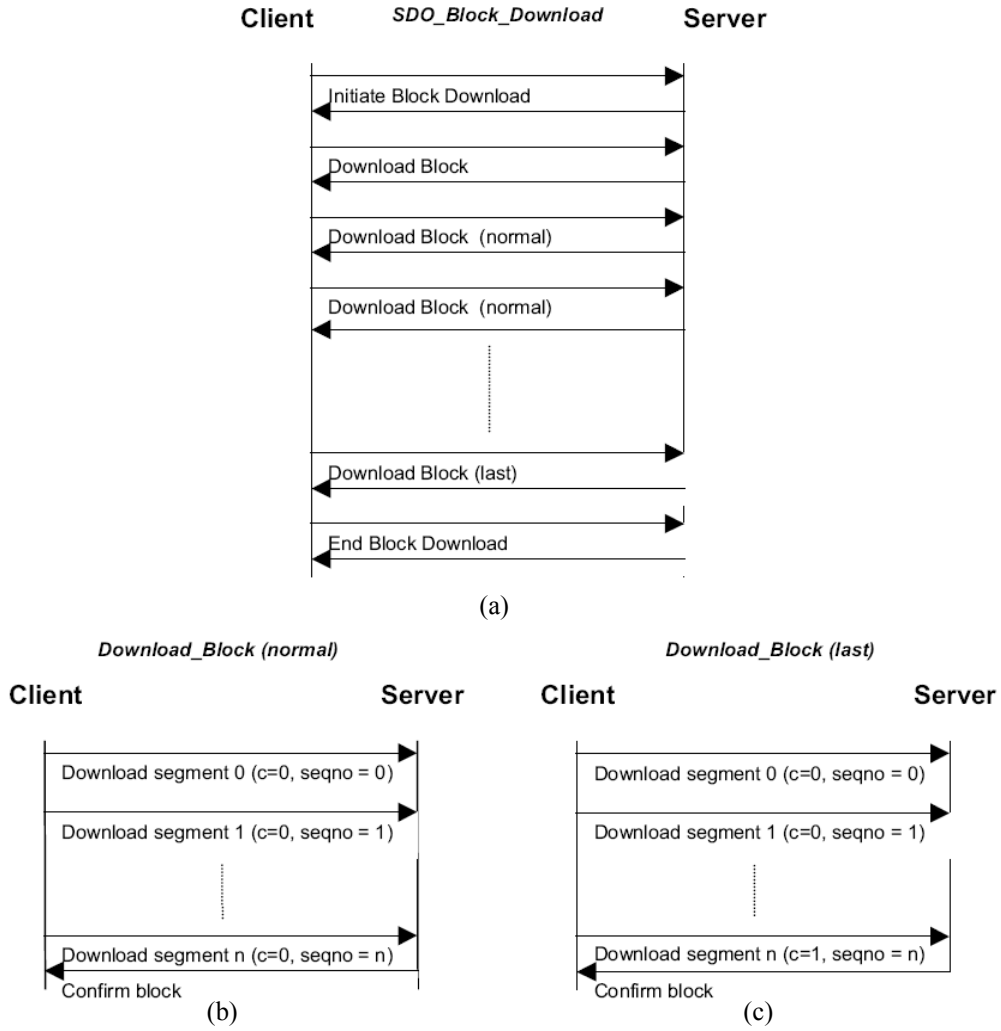


圖 3-8 SDO block 下載通訊模型

3-1-3 CANopen 網路管理

在網路管理的方面，CANopen 所提供的方式包含同步物件(SYNC object)和網路管理物件(NMT object, network management object)等等，其目的在於系統的控制操作皆可經由網路加以整合，底下就是相關的介紹。

◆ 同步物件(Synchronization Object)

時脈同步向來是網路系統整合上的主要考量之一，若節點間的時脈配合不當，便可能發生資料遺失甚至系統不穩等問題，關於這點，已由第二章的分析實驗獲得驗證。對於時脈同步的解決方式，CANopen 主要是藉由同步物件作為同步性資料傳送的觸發源，所有需發送同步性資料的節點，都在接收到同步物件之後，在預設的時間範圍內，將資料封包發送出去，如此一來，只要同步物件發出的週期固定，即可得到一個基本的網路時脈，各個節點以此為基準，就可達到同步的效果。同步物件的通訊協定如圖 3-9 所示，而為了減少網路傳輸造成的延遲，其訊息封包並不帶有任何資料，此外，由表 3-2 可知，同步物件的 COB-ID 相當小，因而具有極高的網路優先權，這則可降低等待傳輸時間的影響，進而達到更好的同步性。

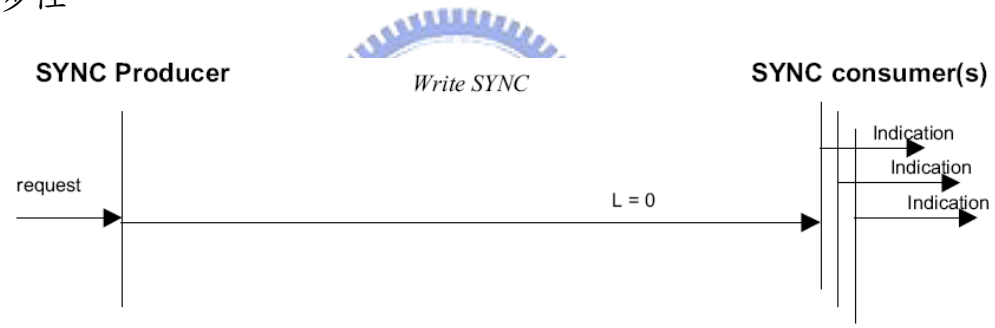


圖 3-9 SYNC 通訊方式

◆ 網路管理物件(Network Management Object)

網路管理物件主要是用於節點操作控制和節點狀態監控，透過 Master/Slave 的方式，由網路管理者(NMT master)控制其餘節點(NMT slave)的操作狀態，並進行各個節點的狀態監控，以便節點發生錯誤時，進行適當的處理機制。

CANopen 對於節點工作狀態的分類為停止(stopped)、前操作(pre-operational)、操作(operational)、初始(initializing)四種。停止狀態為預設狀態，僅能接受啟動(或停止)的網路管理命令；前操作則是參數設定狀態，上述的 SDO 主要是用於此處以進行大量參數設定；操作則是一般工作狀態，可進行任何的控

制命令傳送與回授；初始狀態則是節點進行重設的過渡時期。然而，各個節點工作狀態由 NMT master 所決定，透過管理物件的發送，更改 NMT slave 的狀態以便進行相對的設定或控制流程，下圖 3-10 為一起動通訊方式，其中 CS 為網路管理命令，1 表示啟動節點進入操作狀態，Node-ID 則是要啟動的節點識別碼。

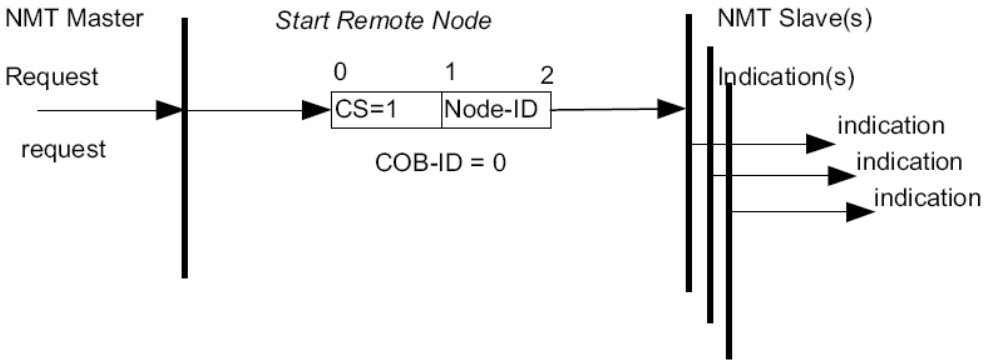


圖 3-10 節點啟動通訊方式

對於節點的網路監控，CANopen 採取 Node Guarding 和 Heartbeat 兩種方式，兩者都是藉由回傳狀態參數以進行評估，前者是由 NMT master 要求 slave 回傳狀態，後者則是由節點主動發佈目前狀態，一旦其餘節點發現錯誤，即回報給處理程序，接著，透過上述的節點操作控制將節點重新設定或提出錯誤警告，以減少降低節點錯誤對於系統的影響。下圖 3-11 為 Node Guarding 的通訊方式，NMT master 每隔固定週期(node guard time)即發出一個遙控欄框，要求各個節點回報目前的狀態，當有錯誤發生時，便會產生錯誤警示以供後續處理，而節點啟動到發生錯誤的時間則稱之為 Node Life time。下圖 3-12 則為 Heartbeat 通訊方式，不同於 Node Guard，各個節點是每隔固定週期(heartbeat producer time)便主動將狀態發佈給其他節點，而其他節點則設定收取此筆狀態訊息的週期(heartbeat consumer time)，一旦超過此週期未收到狀態訊息或發現狀態錯誤，便提報給應用程序加以處理，透過這種形式，各個節點可互相監控彼此的工作狀態，對於錯誤的偵測就更家有效率。

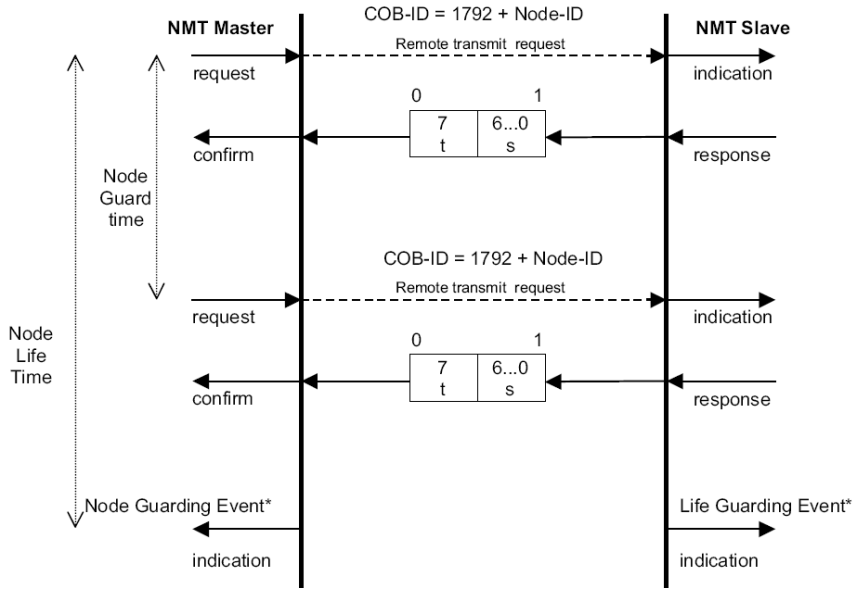


圖 3-11 Node-Guarding 通訊方式

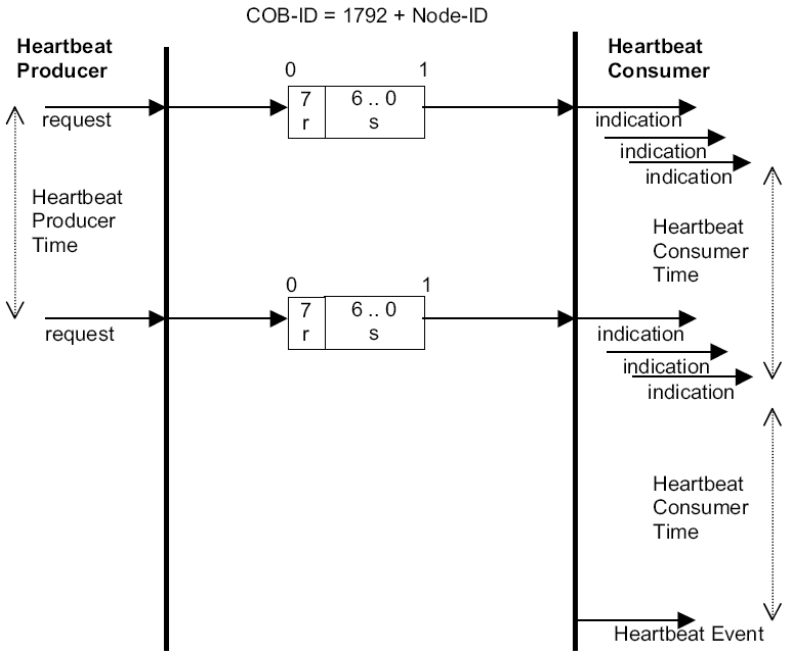


圖 3-12 Heartbeat 通訊方式

3-2 改良 CANopen 應用層通訊協定設計

由上節介紹可知，CANopen 的優點在於規劃了不同使用場合的整合環境，使得 CAN 網路系統的建構上，縮短了不同節點與應用介面的開發時程，並提供各類網路管理方式，不僅方便監測各個節點，同時系統的所有設定操作整合到網路

介面中，構築成一個完整的網路控制系統。

為了能符合各類的需求，CANopen 建立了極為完善的 O.D.，以提供各種應用物件，並且制定了許多網路通訊規範，然而，這卻造成 CANopen 的實現必須使用相當的記憶體容量，對於多數的微控器(如 8051)而言，並無法容納如此龐大的應用程式。由此可知，CANopen 並不適用於以微控器為運算核心的網路節點，因此，吾人以其通訊概念作為基礎，設計適用於分散式移動平台的應用層協定，此處主要以即時性資料傳遞、網路管理與節點時脈同步為主。接著，3-2-1 節介紹資料傳遞方式和網路管理，3-2-2 則為網路時脈同步方法。

3-2-1 全向平台應用通訊協定規劃

本部份的規劃主要以通訊物件的設計方式和對應的通訊形式為考量，而實現的系統為全向性移動平台(omni-directional moving robot)，因此，以下的說明皆以此為例。

在通訊物件的規劃上，沿用 CANopen 的方式，將原本的 CAN 訊息識別碼加以劃分成 function code 和 Node-ID，前三位元為 function code，後八位元為 Node-ID，如圖 3-13 所示，此物件識別碼稱之為 Omni-ID，其中 Node-ID 隨著 function code 的不同，所代表的意義也有所不同，在通訊行為的說明將詳細闡述。根據此規劃方式，通訊物件被分為 8 種類別，如表 3-3 所示，以功能可分為網路管理用、廣播用、資料與參數設定用，其說明如下：

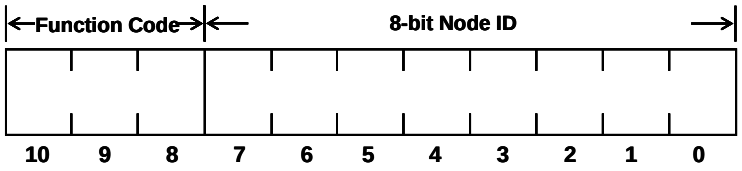


圖 3-13 Omni-ID 規劃

表 3-3 Omni 通訊物件

Function code (binary)	Omni-ID	Communication object
000	0	NMT Service(For master)
	1(001h) ~ 255(0FFh)	NMT Service(For slaves)
001	257(101h) ~ 511(1FFh)	Broadcast(Up to 8nodes)
010	513(201h) ~ 767(2FFh)	Process Data Transmission
011	769(301h) ~ 1023(3FFh)	Process Data Transmission
100	1025(401h) ~ 1279(4FFh)	Broadcast(Up to 4nodes)
101	1281(501h) ~ 1535(5FFh)	Process Data Transmission
110	1537(601h) ~ 1791(6FFh)	Process Data Transmission
111	1793(701h) ~ 2047(7FFh)	Parameter-Setup Transmission

◆ 網路管理用物件(NMT Service)

網路管理採取的架構為 Master/Slave 方式，每個物件的 Node-ID 代表各個節點本身的識別碼，NMT master 的 Node-ID 固定為 0，NMT slave 則在 1~255 之間。在通訊行為上，主要分為管理和監控兩種，在管理的方式上，將節點的狀態分為啟動與停止兩種，啟動狀態用於控制程序的執行，停止狀態則用於參數設定之用，節點狀態的變更皆由 master 所控制，其通訊方式如圖 3-14，其中 control state 則為指定的狀態，1 為啟動狀態，Node-ID 為欲控制的節點編號，若編號等於 0，則是對所有節點進行控制，此外，將 Node-ID 設計在高位元組的方式，可與 CAN 的訊息過濾器(acceptance filter)搭配，用以濾除不必要的訊息。至於節點的監控方式，透過遙控欄框要求每個節點回應，通訊方式如圖 3-15 所示，master 發出 Node-ID 為 0 的遙控欄框，各個 slave 同樣透過遙控欄框回應，以此方式測試各個節點是否仍可正常傳收訊息，作為網路節點通訊的監測機制。

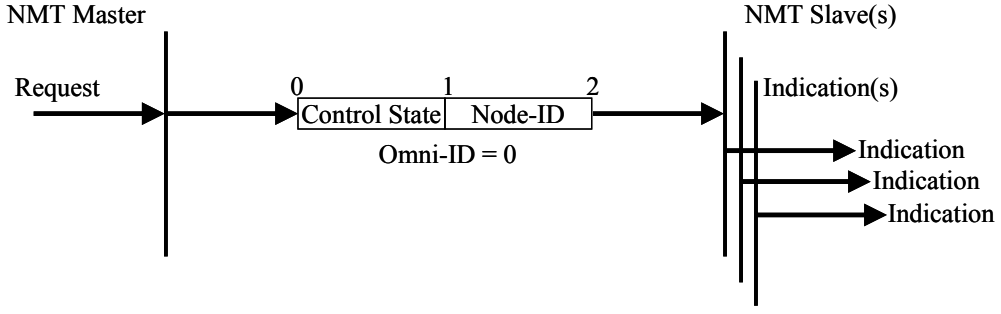


圖 3-14 Omni 節點通訊方式

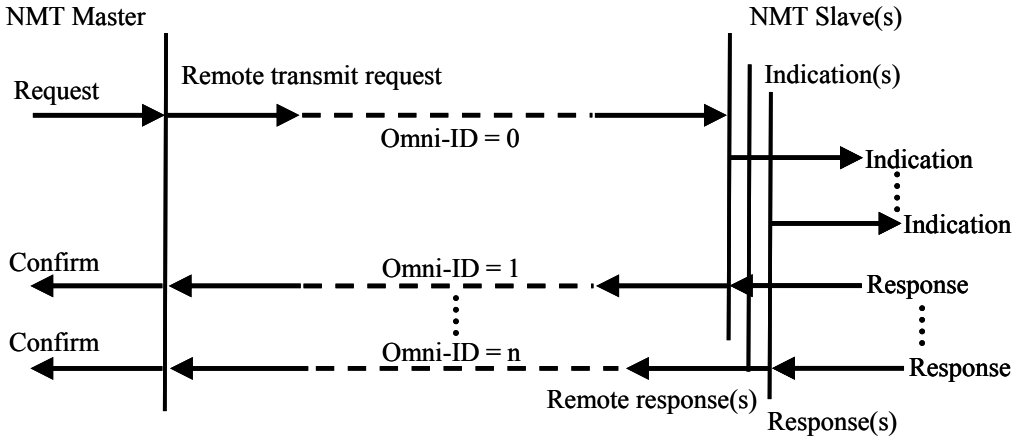


圖 3-15 Omni 節點監控通訊方式

◆ 廣播用物件(Broadcast)

在網路控制系統中，各個節點運作的協調性，對於整體系統的影響甚鉅，尤其多軸運動系統更是如此，然而，為了減少網路傳輸的影響，造成各個節點接收訊息的時間差異，進而引起節點運作的不一致，所以，藉由 CAN 網路本身傳輸的廣播特性，將多個節點的資料放在同一筆訊息之中傳送，而不是每個節點都發送一筆訊息，同時，此方式可減少訊息數量而提高網路使用率，此處則規劃兩個類別的廣播物件，Omni-ID 257 ~ 511 定為最多可傳送給八個節點，每個節點資料長度為 1byte，Omni-ID 1025 ~ 1279 則為四個節點，每個節點資料長度為 2bytes。廣播物件的通訊方式如圖 3-16，其中 Node-ID 為發送節點的識別碼，訊息封包內則涵蓋多筆資料，對應的接收節點完成整筆訊息的接收之後，再由當中取出所需的資料，如此便可同時傳送多個節點的資料。

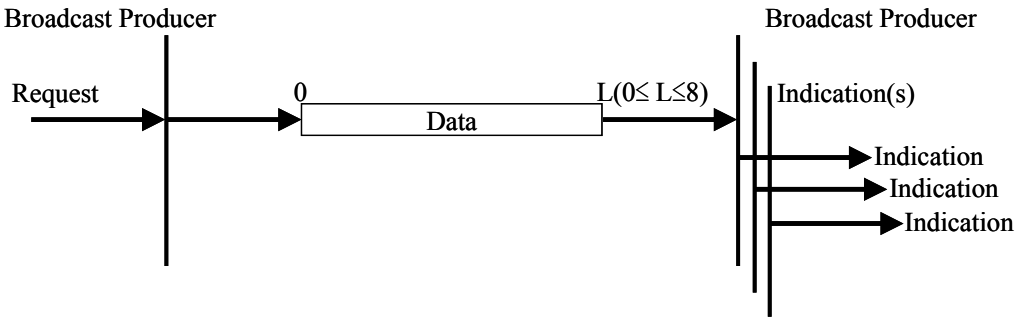


圖 3-16 Omni 廣播通訊方式

◆ 資料傳遞和參數設定物件(Process Data and Parameter Setup)

本部份的物件主要用於資料傳遞和參數設定，通訊方式類似上述的廣播物件，但採取的是點對點的傳輸，此外，不同於上兩類物件，其 Omni-ID 中的 Node-ID 為接收節點的識別碼，此方式同樣是為了和 CAN 的訊息過濾器搭配，隔離不需要的訊息以減少處理器負擔，通訊方式如圖 3-17 所示，其中最高位元組為發送節點的編號，接著為傳送資料的型態，最後為資料內容，資料內容依照 Omni-ID 的區分，可為即時性的控制命令或是節點的參數設定值，而由於參數設定是在停止狀態下使用，所以，將網路優先權最低的 Omni-ID 規劃為參數設定用，而即時性資料則賦予較高的優先權。如此一來，便可將系統的命令傳送與設定，皆透過此方式整合於網路架構上。

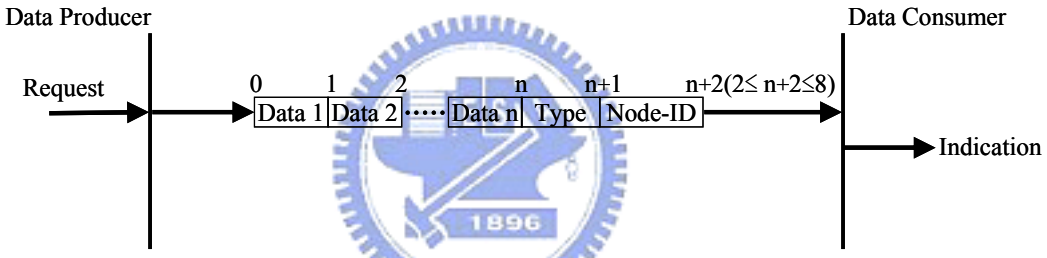


圖 3-17 Omni 資料傳輸方式

3-2-2 網路時脈同步方法

由第二章分析實驗的結果，針對時脈同步的影響加以討論，在系統取樣週期較小的架構下，時脈漂移造成節點間的時脈不同步，將導致大量的系統資料遺失量，由此可知，時脈同步對於網路系統是不可忽略的因素。

關於網路時脈同步的方式，可由系統架構、網路協定等層面解決，以系統架構來說，可利用額外的硬體接線作為同步線路，以此作為節點執行的基準；在網路協定層面，以 CANopen 為例，透過同步物件的傳送，提供整體網路的基準時脈。這兩類作法各有其優缺點，前者為獨立於原本網路架構之外的作法，可免除

網路傳輸的干擾，且經由硬體線路可達到極高的準確度，但正因為是獨立於網路之外，所以必須在系統建構時加以考量，若要應用於已建立的系統之上，就必須重新規劃系統架構；後者則是基於原先的網路協定所衍生的方式，主要優勢在於可經由軟體規範達成而不需改變系統架構，所以，實做上的可行性較高且提高系統維護的便利性，但由於是透過網路封包傳遞的方式，傳輸狀況的影響就無法忽略，傳輸速率、延遲時間和封包碰撞等等因素，都可能影響其效果而必須納入考量，以 CANopen 為例，由於封包碰撞可由 CAN 的重傳機制避免，而將主要考量放在傳輸延遲上，因此，賦予同步物件極高的網路優先權，降低傳輸的等待時間，便於同步物件順利傳送，以達到較好的同步效果。

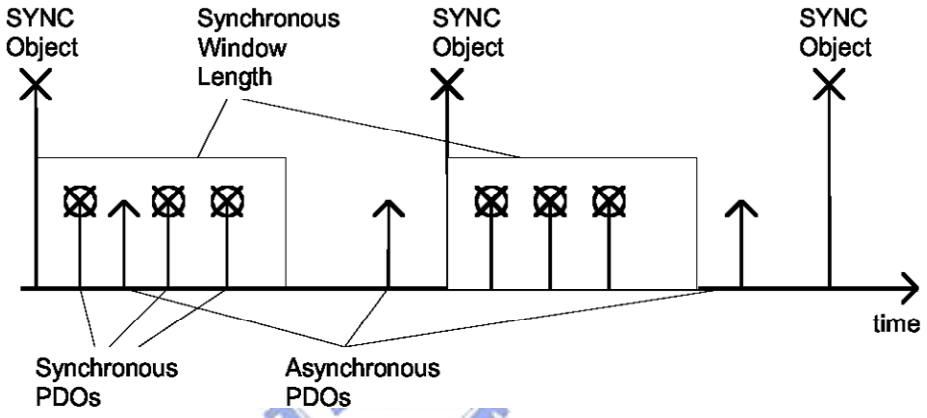
網路時脈同步方法主要可由系統架構和網路協定兩個方向去克服，以現行的網路系統而言，前者必須修改系統架構，可行性大為降低，而後者由通訊協定發展，易於實現在現行系統之上。顯然地，制定通訊方式的作法較為可行，然而，這類作法的關鍵在於，若透過額外的訊息封包進行同步，無疑會增加網路的負載，可能導致傳輸效率的下降。因此，若能由通訊協定的規劃出發，且不需透過額外的網路封包，而符合時脈同步的需求，便能既不修改系統架構，也不用付出額外的網路負載，卻能達到時脈同步的效果，這便是此部份同步方式的設計目標，底下為其說明和實驗結果。

◆ 網路時脈同步方法介紹

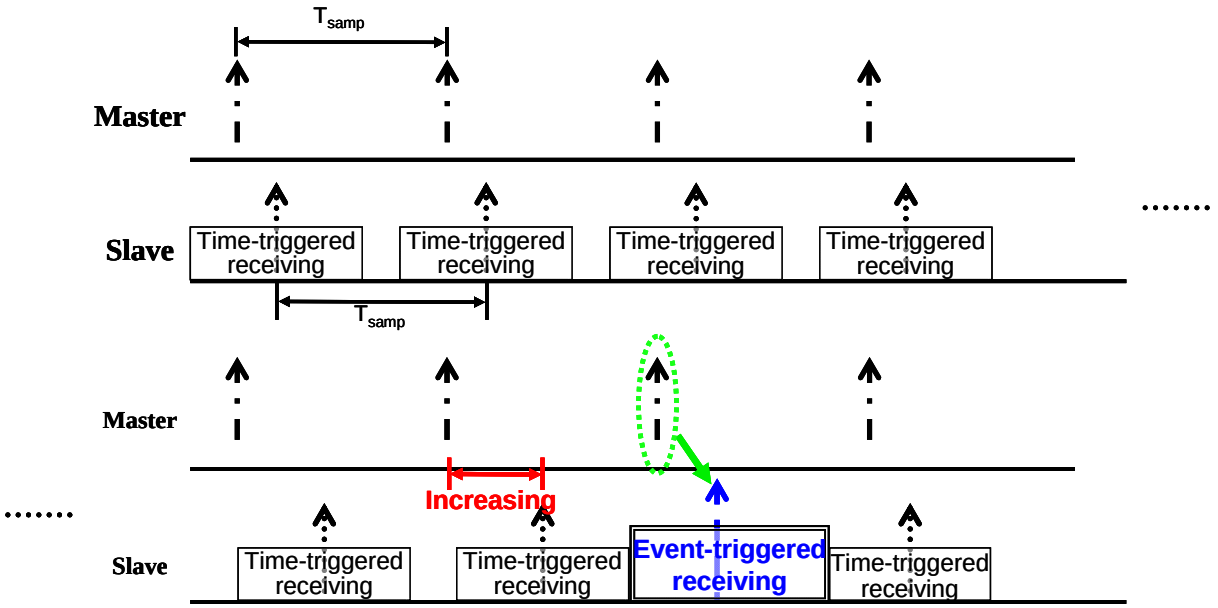
CANopen 以同步物件的傳送，作為各個節點傳送的基準，其方式類似事件觸發(event triggered)的架構，以同步物件作為傳送訊息的觸發源，達成各個節點動作的一致，然而，此方式主要是提供即時性資料傳輸的基準，至於如何達到各個節點間的時脈同步，CANopen 並未詳加規範，一旦考慮各個節點的時脈一致性，單以同步物件並無法達到同步的目標，並不能符合時間驅動(time triggered)的系統需求。

由於時脈驅動的系統中，必須考慮時脈的一致性，時脈漂移的影響就需加以

考慮，而由第二章的分析結果，可知兩個節點之間的時脈差異，會隨著時間增長而逐漸擴大，這是造成時脈不同步的主要原因。因此，若能即時校正時脈差異擴大的趨勢，使之維持在範圍之內，便能使各個節點的時脈保持固定差距而不至錯亂，而為了不修改架構且能達到時脈校正，所以，若能透過網路封包的傳輸實現，便能即時修正時脈漂移的現象，而 CANopen 雖以同步物件為基準，進行即時性資料的傳送，如圖 3-18(a)，此方式並未對時脈同步方法加以規範，但將此概念加以延伸，利用類似同步物件的訊息封包，強制各個節點進行校正動作，便可提高時脈同步性。



(a) CANopen 同步方法



(b) 本文所設計之時脈同方法

圖 3-18 時脈同步方式比較

然而，為了減少網路的負載，此處並不規劃額外的同步物件，而是直接利用原本傳輸的資料封包，搭配節點接收訊息方式的改變，引發強制時脈校正，其方式是將原先以時間驅動的接收方式，改變為事件驅動接收訊息，以控制器的角度說明，一般狀況下，採取計時中斷(timer interrupt)接收訊息，而進行時脈同步時，則改為外部中斷(external interrupt)的方式接收訊息，在接收訊息的同時，將時脈計數值加以歸零，其概念如圖 3-18(b)所示，因此，這個方式類似在時脈驅動的架構下，加入了事件驅動的概念，以進行時脈計數歸零，達到時脈同步的目的，然而，時脈同步的動作，必須在漂移現象對系統造成影響之前，也就是兩者時脈相差一個取樣週期以上之前，進行時脈校正，才能達到改善的效果。

◆ 時脈同步實驗結果與討論

本部份的時脈同步實驗，沿用第二章的分析實驗架構，採用 Master/Slave 的傳輸架構，實驗進行的流程，主要由 master 發送啟動指令，觸發 slave 的時脈中斷系統，接著在每個取樣週期中，master 持續傳送命令，slave 負責接收並計算資料遺失量，直到 master 完成所有命令的傳送並發出停止指令，關閉 slave 時脈中

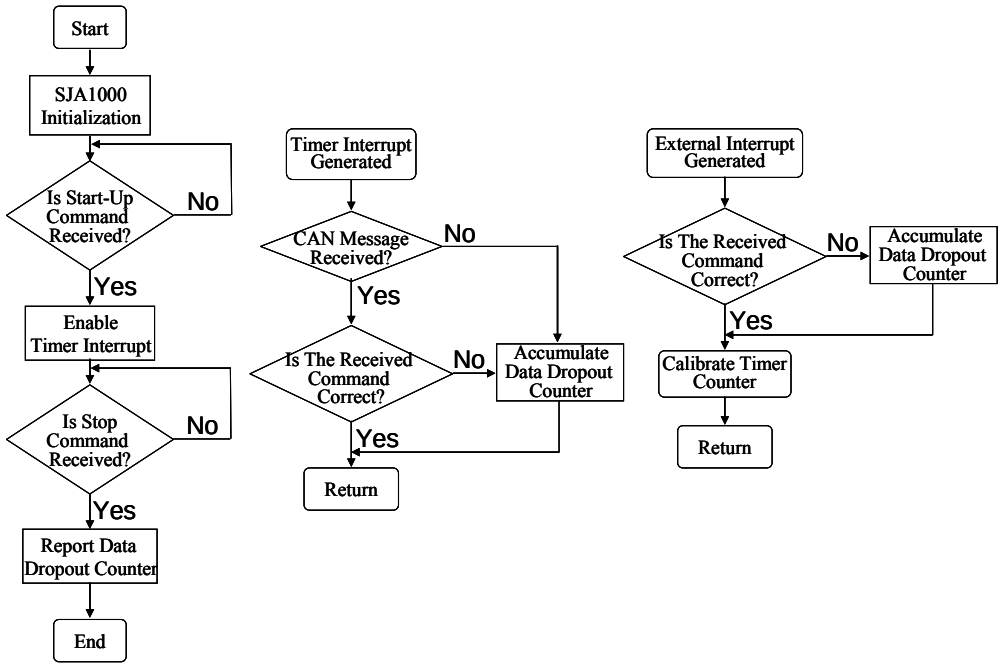


圖 3-19 時脈同步實驗 slave 端流程圖

斷，並由 slave 回傳資料遺失量，在實驗期間，slave 節點每隔固定次數的時脈中斷之後，便進行一次採用外部中斷的接收方式，並且將時脈計數歸零，重複此方式直到實驗完成，其中 master 端程式流程和第二章相同，slave 端程式流程則如圖 3-19 所示。

至於實驗結果的說明，由於採用時脈同步方法，目的主要是為了修正節點的時脈漂移，所以，先就此部份說明。在 master 和 slave 節點各使用一 IO pin，取樣中斷期間產生一高準位脈衝，透過示波器抓取兩者的輸出，便可得到一脈衝序列，如圖 3-20 所示，其中準位較低者為 master (DSP)端，圖 3-21 為序列初期，可看出 master 和 slave 的週期約為 5ms，此時兩者的時脈差距則約為 540 μs，；而在序列末期，master 和 slave 的週期依然維持在 5ms 左右，而兩者時脈差距則為 440 μs，如圖 3-22，並未持續擴大反而減少，這是由於時脈校正所得到的效果，將時脈漂移加以修正，擷取脈衝序列中的其中一段，如圖 3-23，由圖上可看出，圈起的位置為進行時脈校正的外部中斷，相較於其他脈衝，兩者時脈差距明顯減少，這是由於採用外部中斷接收所產生的效果，而此校正點前後的時脈差距也有所改變，修正後的差距相對降低，由此可知，經由此同步方式，時脈漂移現象可因而獲得控制。接下來，介紹同步方法在各種傳輸條件下的實驗結果，此處同樣以資料遺失量作為實驗結果的指標。

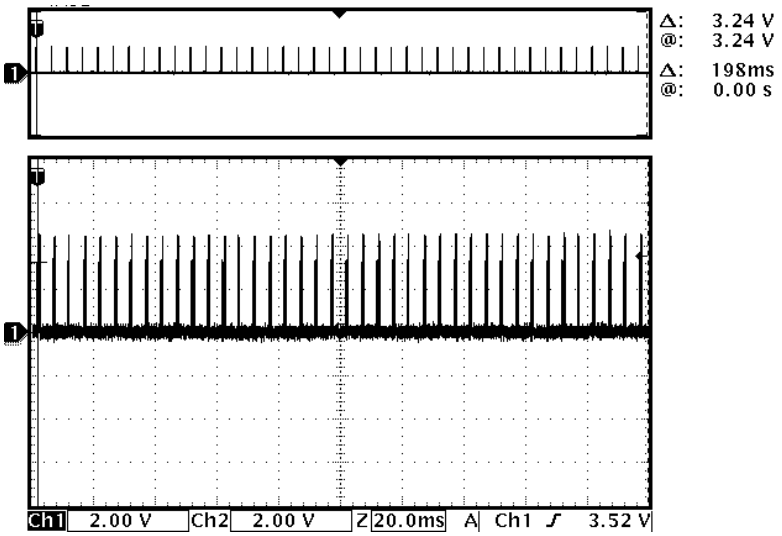


圖 3-20 取樣中斷脈衝序列

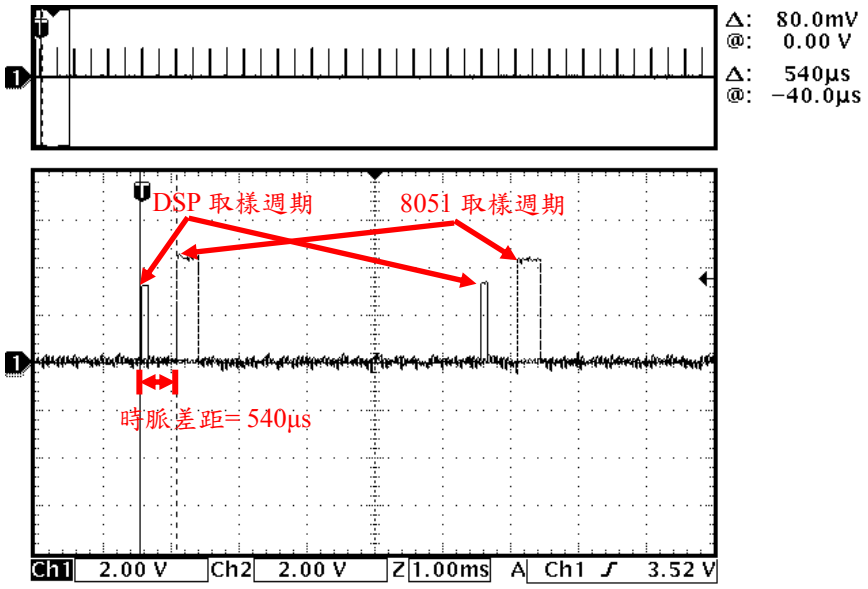


圖 3-21 序列初期，Master (DSP)和 Slave (8051)時脈差距 ＝540 μs

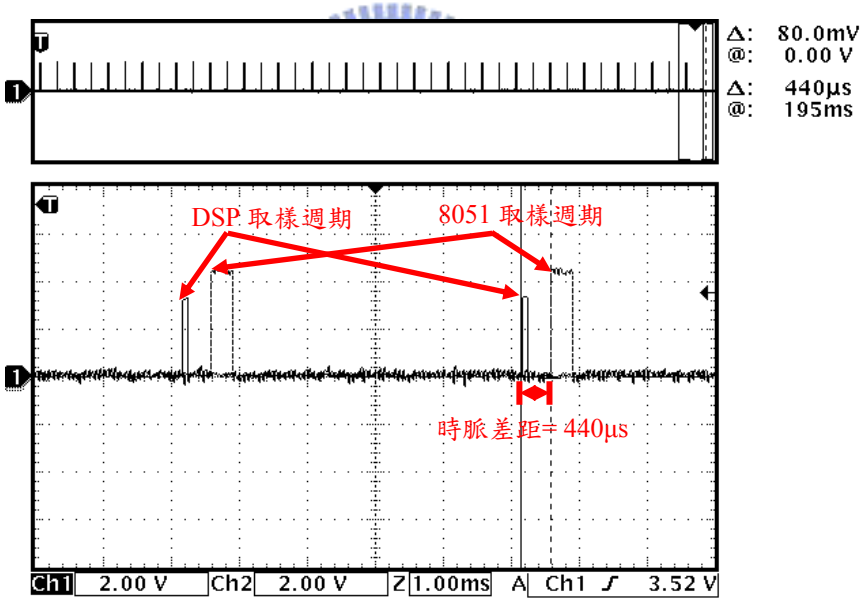


圖 3-22 序列末期，Master (DSP)和 Slave (8051)時脈差距 ＝440 μs

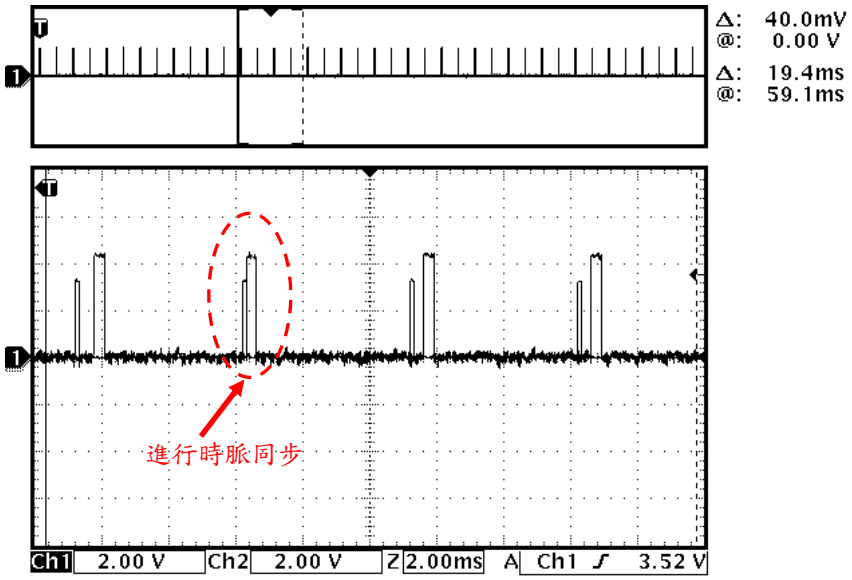


圖 3-23 進行時脈同步的結果

■ 傳輸資料大小

本部份實驗以改變資料封包大小進行，資料長度為 2 ~ 8 bytes，而在先前的分析實驗中，隨著資料長度增加，資料遺失量隨之增加，這是由於封包長度的改變，造成網路傳輸的延遲，導致訊息無法即時到達接收端。對於這樣的網路延遲情況下，加入時脈同步方法，所得的結果如圖 3-24，此實驗在傳輸速率 1000kbps、系統取樣時間 5ms 下，傳輸 3000 筆資料進行，由圖上可看出，加入同步方法後，幾乎可將資料遺失現象消除，這是由於經過同步之後，不僅可將時脈漂移修正，同時，原本的傳輸延遲會被涵蓋在重新修正後的時脈，使得接收端收取訊息的時間點往後移，訊息便可即時到達接收端。

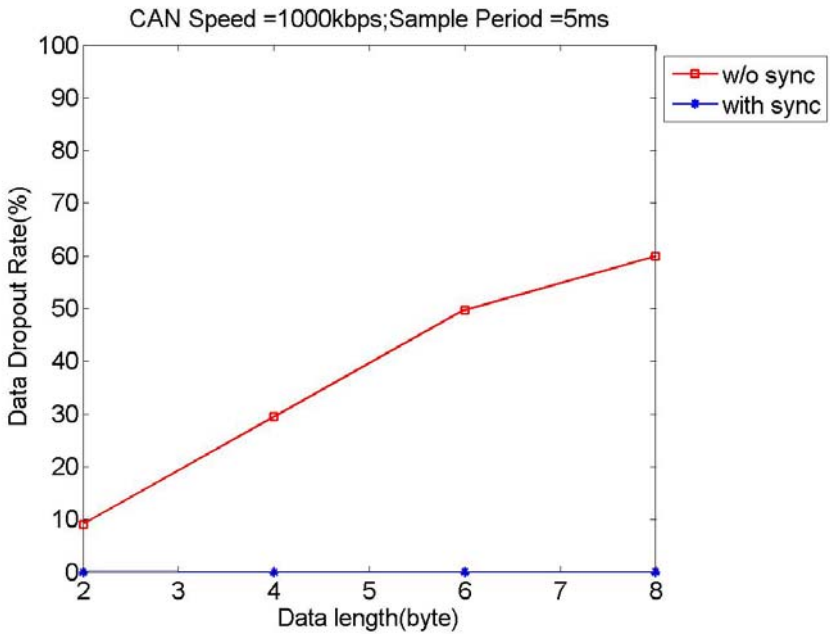


圖 3-24 加入時脈同步前後，不同傳輸資料大小的資料遺失率

■ 網路傳輸速率

在分析實驗中，隨著網路傳輸速率下降，傳輸延遲時間跟著增加，導致資料遺失量上升，圖 3-25 為系統取樣時間 5ms 下，傳輸 3000 筆資料的實驗結果，而加入同步時脈方法後，所得的結果如圖 3-26 所示，由圖上可看出，採用同步時脈後，資料遺失量大幅地降低，其原因和改變傳輸資料大小的實驗相同，原本的傳輸延遲被涵蓋於重新修正後的時脈，接收端收取訊息的時間點往後移，因此，接收節點可即時收到封包，而達到減少資料遺失量的效果。

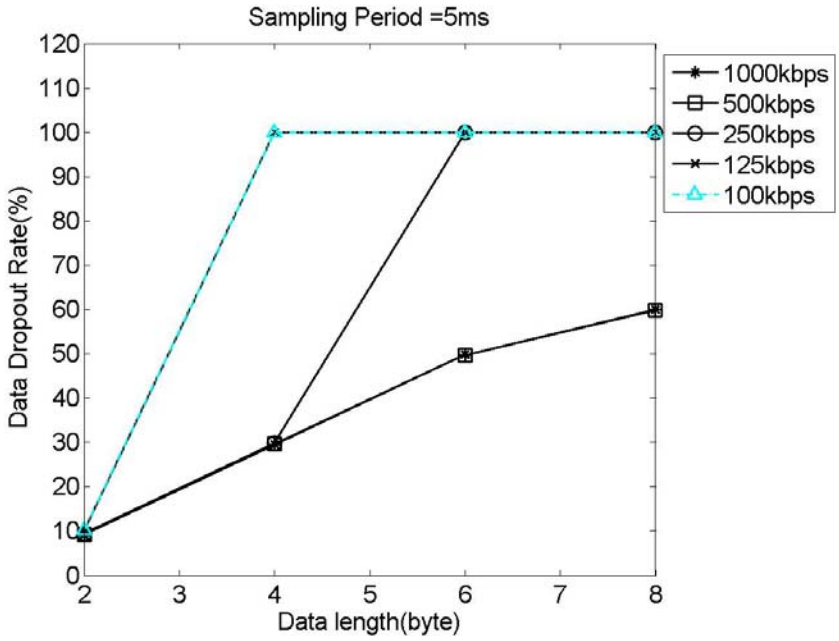


圖 3-25 5ms 取樣下，未加入時脈同步前，不同傳輸速率的資料遺失率

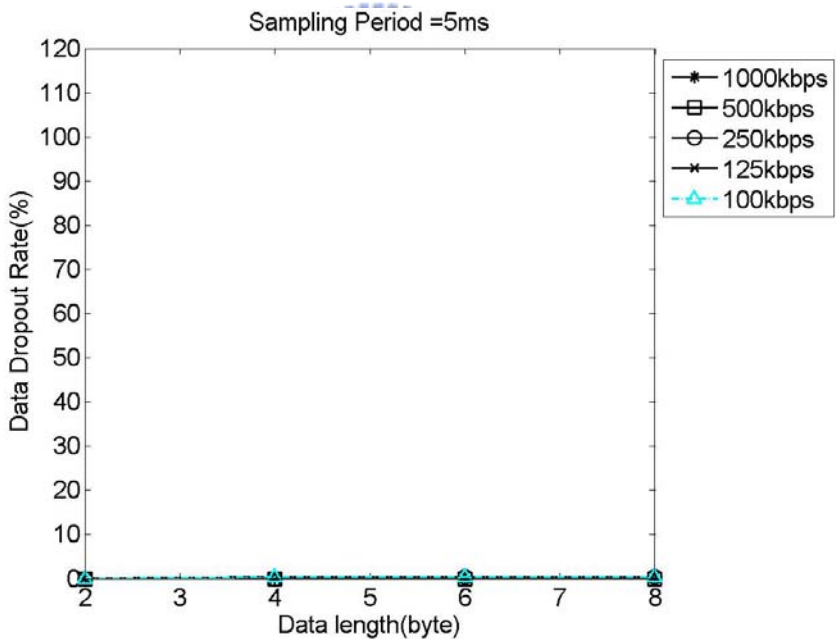


圖 3-26 5ms 取樣下，加入時脈同步後，不同傳輸速率的資料遺失率

■ 取樣週期

第二章的討論中，曾提及時脈漂移與取樣週期的關係，在取樣週期較小的系統中，時脈漂移的影響較為顯著，因此，先由取樣週期為 1ms 的實驗結果說明，由圖 3-27 和圖 3-28 的結果可知，在傳輸速率較快或資料封包較小

的情況下，資料遺失量可經由時脈同步機制，將漂移現象消除，但在 125kbps 和 100kbps 的傳輸速率下，若傳送較長的封包，即使採用同步機制，依然無法降低資料遺失量，這是由於此情況下的傳輸時間往往超過 1ms 以上，導致訊息不可能在取樣週期內將訊息傳送完畢，因而造成此現象，若以系統設計來說，便必須調整取樣週期或傳輸速率，以達到需求。因此，在可行的架構下，時脈同步方法仍可有效地改善時脈漂移現象。

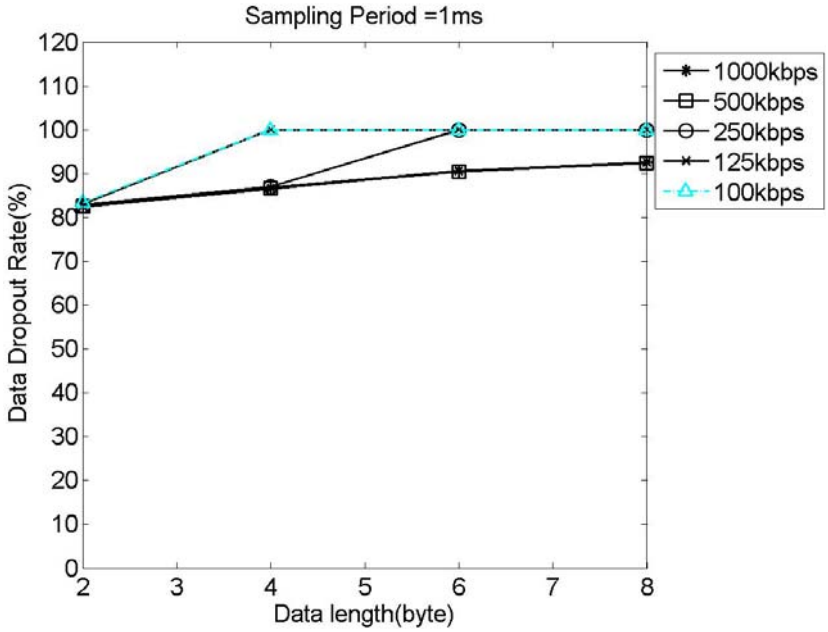


圖 3-27 1ms 取樣下，未加入時脈同步，不同傳輸速率的資料遺失率

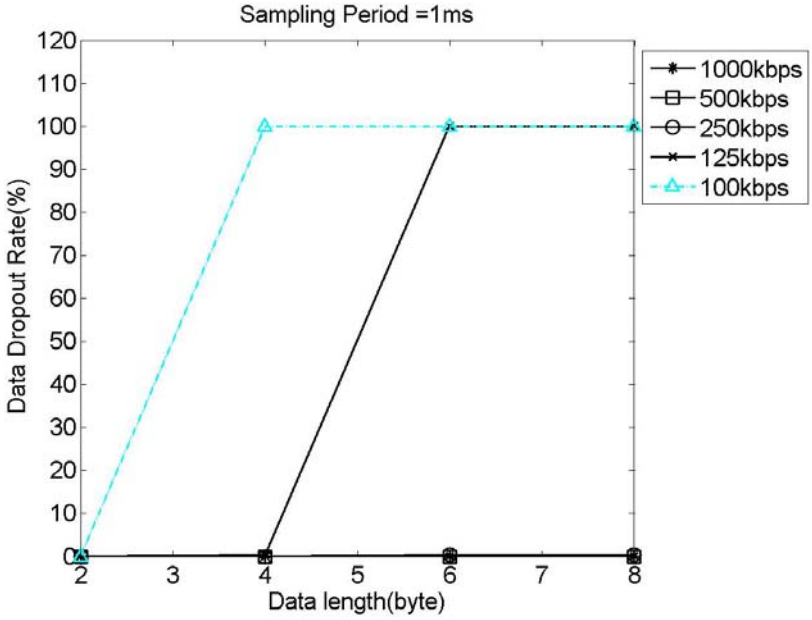


圖 3-28 1ms 取樣下，加入時脈同步後，不同傳輸速率的資料遺失率

3-2-3 小結

在本節中，基於 CANopen 的概念與架構和第二章的網路分析結果，延伸出對於 CAN 應用層協定的設計，由於 CANopen 所需的記憶體容量較大，並不適合實現於常見的微控器(如：8051)，此外，CANopen 的同步方式，並不適用於時脈驅動的系統，因此，本文將其概念延伸，設計較精簡的應用協定，主要以資料傳遞機制、網路管理以及網路時脈同步三個部份為考量，其中時脈同步的部份，不需用到額外的同步訊息，可減少網路傳輸的負載。

在資料傳遞方面，除了建立點對點資料傳輸與參數設定的通訊方式，並設計了廣播傳輸的傳輸方式，以同時傳輸多點資料，避免網路傳輸造成節點動作的不一致；在網路管理方面，則設計了 Master/Slave 架構，將各個節點的運作經由 Master 加以管理，並利用遙控欄框的方式監控節點，由以上兩點便可將系統各個環節透過網路整合；在時脈同步方面，將 CANopen 的同步概念延伸，透過原有的網路封包搭配外部中斷的方式，達到強制同步的效果，並經由實驗加以驗證完成。結合以上各點，建構出基本的應用層協定，表 3-4 為 CANopen 與所設計之同步通訊協定的比較。此通訊協定也將應用於全向性平台的網路控制系統之中。

表 3-4 CANopen 與本文之同步通訊協定比較

協定項目	CANopen	本文之同步通訊協定
Object Dictionary	定義網路訊息和應用協定之間的資料類型與內容(見 3-1-2 節)	未進行規範，由使用者規劃所使用的資料類型與內容(見 3-2-1 節)
PDO	用於即時性資料傳遞(資料長度在 0~8byte 間)(見 3-1-2 節)	僅規範訊息識別碼以及封包格式，對於傳輸長度與傳輸時機未進行規範。另外，規範多點資料傳輸封包，以同時傳輸多點資料(見 3-2-1 節)
SDO	用於非即時性或大型資料傳遞(資料長度可超過 8byte)(見 3-1-2 節)	
同步物件	以特定同步物件作為系統同步根據(見 3-1-3 節)	以原有的訊息封包作為同步基準，搭配中斷架構進行同步(見 3-2-2 節)
網路管理物件: 節點運作控制	以網路管理物件，決定各個節點的工作狀態(見 3-1-3 節)	以網路管理物件，決定各個節點的啟動與停止(見 3-2-1 節)
網路管理物件: 節點狀態偵測	以 node-guarding 和 heart-beat 兩種方式，確保節點工作狀態(見 3-1-3 節)	以遙控欄框詢問各個節點的連線狀況(見 3-2-1 節)

第四章　分散式全向性平台控制系統

本章主要介紹分散式全向平台的硬體實現架構、平台運動控制模型和系統程式架構，並將第三章所設計的 CAN 應用層通訊協定，運用於平台的網路架構之中，完成分散式平台控制系統，相關實驗結果也於此章加以說明。

4-1 分散式全向平台硬體架構

本論文所使用的控制平台為四軸全向性運動平台，以控制電路的架構來說，採用 Master/Slave 的方式，由一個 master 節點和四個 slave 節點所組成，master 節點主要以 DSP F2812 作為運算核心，slave 節點則以微控器 8051 作為核心，此設計方式是由於 master 端必須進行速度規劃和計算速度命令，並作為網路管理的主控者，所以，採用運算能力高的 DSP 晶片；而 slave 端則是負責執行 master 的速度命令，運算量相對降低，因而採用 8051 作為核心。此外，為了方便實驗的進行與傳輸狀態的觀測，在原本的控制架構下，加入了 PC 端節點，PC 端搭配 USB CAN，即可進行 CAN bus 的傳輸動作，但此節點並不影響平台的整體運作，

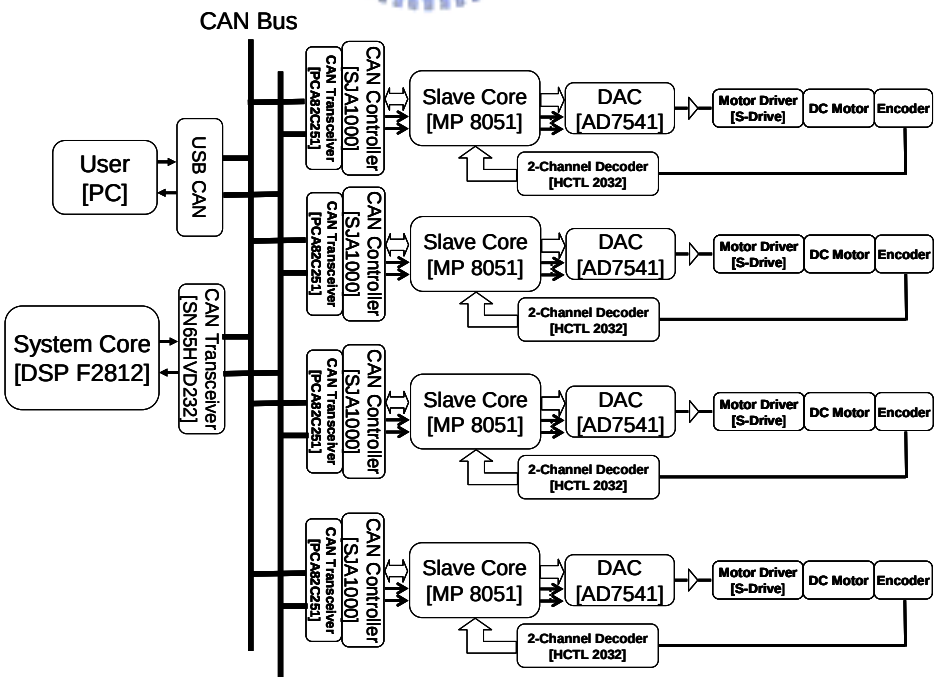


圖 4-1 分散式全向平台硬體架構

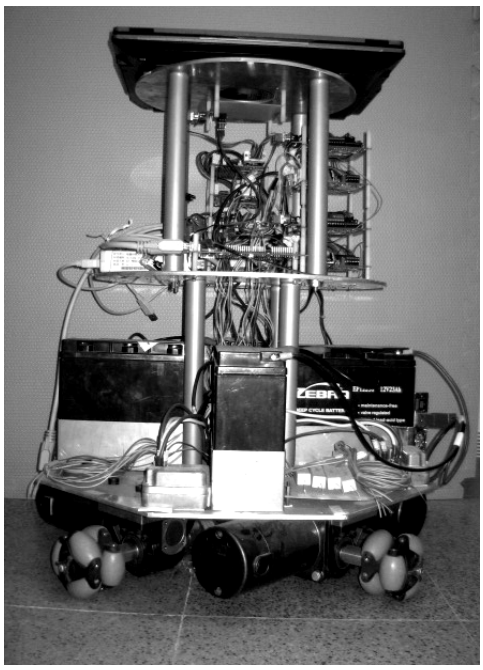


圖 4-2 平台實體圖

主要作為系統的觀測之用。因此，整體系統架構如圖 4-1 所示，平台實體則如圖 4-2 所示，底下為 USB CAN、master 端和 slave 端的硬體架構說明。

4-1-1 USB CAN 介紹

為了讓電腦能具備 CAN 的通訊介面，常見的方式是透過 PCI 和 USB 的 CAN 介面卡，將 CAN bus 上的訊息封包加以轉換，以便透過電腦對網路狀態進行分析與監控，本論文採用的是周立功單片機發展公司所開發的 USB CANII 智能 CAN 接口卡，USBCANII 提供兩個 CAN 介面，可同時連接兩個不同的網路，其外觀和硬體規格如下：

(a)CAN 介面端　　(b)USB 介面端

圖 4-3 USB CAN 外觀

硬體規格：

- USB 規格：USB1.1
- CAN controller：PHILIPS SJA1000
- CAN transceiver： PHILIPS PCA82C250
- CAN 網路傳輸速率：5Kbps~1Mbps
- CAN 介面：DB9，符合 Device NET 和 CANOpen 標準
- 支援 CAN 網路協定：支援 CAN 2.0B(兼容 CAN2.0A 協定)，符合 ISO/IS 11898
- 最高 Frame 流量：每個通道 5000 Frame/sec

4-1-2 Master 端硬體介紹

Master 端主要進行速度規劃和速度命令之產生，主要硬體架構以 DSP F2812 搭配 CAN transceiver，作為命令傳送和網路管理的介面。

◆ eZdsp^{TM} F2812 DSK 介紹

本論文 master 端所採用的運算核心為德州儀器公司(TI)所生產的 DSP ’C2000 系列產品，此系列是專為控制應用最佳化而設計的，其中本論文所使用的 F2812 晶片是目前此系列中運算速度最快的處理器，其特點在於擁有 150 MHz (6.67ns cycle time)的快速處理能力，比 TI 早期出產的 DSP ‘C240 快 7 倍。在核心部分，算數邏輯單元(ALU)、累積器(ACC)均採用 32 位元定點運算，為了提升數位訊號運算效能，以硬體方式實現乘法器、乘積位移器，可在一個指令週期(instruction cycle)內完成乘加運算。在整數計算方面，為減少數值計算所衍生如溢位(overflow)等問題，也採用硬體式的輸出倍率位移器來提高軟體執行的精確度。

記憶體容量方面，F2812 (on chip)主要擁有 128K*16 Flash EEPROM、兩組 4K*16 Single-Access RAM (SARAM)、一組 8K*16 SARAM，並採用哈佛匯流排(Harvard bus)架構。較舊型的 DSP 大部分將程式、資料、I/O 記憶體獨自分離(定址位址重複)，但 F2812 卻走向單獨的記憶體空間，包含了上述三種記憶體，使用上更有彈性，也可減少不同記憶體之間搬動資料的指令集。

Spectrum Digital 公司為 TI 的第三方供應商，eZdsp™ F2812 為此公司以 F2812 為主要核心，所開發之初學板(DSK, DSP started kit)，其週邊主要有 16 個通道的 12 位元類比數位轉換器(ADC)，串列傳輸支援了四種常用的型式：SPI (serial peripheral interface)、SCIs (two serial communications interface)、eCAN (enhanced controller area network)、McBSP (multi-channel buffered serial port)，最高可支援 56 個 GPIO(general-purpose I/O)。

由於 F2812 已內建 eCAN 的 CAN 控制器(CAN controller)，其作用為提供 CAN 通訊協定，其特色如下：

- 高輸入阻抗，可允許 120 個網路節點
- 支援 CAN 2.0A 和 CAN 2.0B 通訊協定
- 支援最快傳輸速率 1Mbps
- 提供 32 個 mailbox，可作為傳輸與接收之用
- 提供多種工作模式，包含等待(standby)、睡眠(sleep)和自我測試(self-test)等模式

其中每個 mailbox 可被設定為傳送或接收兩種類型，每個 mailbox 可獨立設定所使用的訊息識別碼，以傳送類型來說，所設定的識別碼即為所傳送之 CAN 訊息的識別碼；以接收類型來說，所設定的識別碼即為欲接收的訊息識別碼，此外，每個接收用 mailbox 可設定獨立的訊息過濾機制，因此，在應用上，可利用不同的 mailbox 將接收訊息加以分類。

eCAN 架構如所示，由於 eCAN 已提供 CAN 通訊協定，所以，只需 F2812 只需搭配 transceiver 即可進行通訊，本論文所用的 transceiver 為 SN65HVD232，以下為其說明。

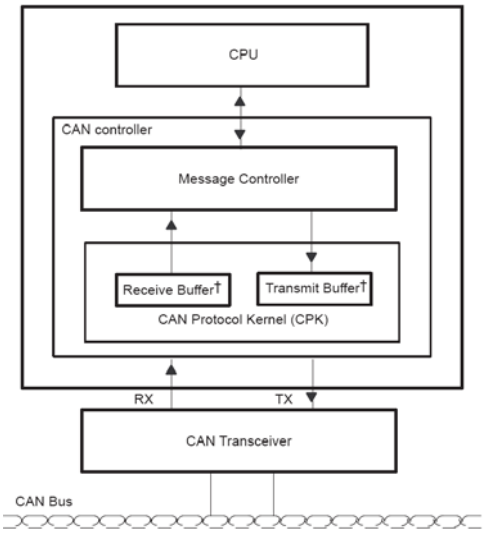


圖 4-4 F2812 eCAN 架構

◆ CAN Transceiver

F2812 內建的 CAN 控制器，主要是提供 CAN 的通訊協定，其輸出腳位為傳送(TX)和接收(RX)兩種，然而，CAN 的實體訊號傳遞方式為差動訊號，因此，必須透過 transceiver 將訊號轉換為差動訊號，此處所用之 transceiver 為同樣是 TI 所生產的 CAN transceiver SN65HVD232，其邏輯電路則如圖 4-5

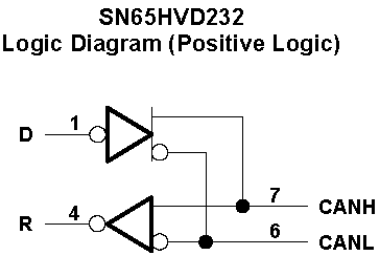


圖 4-5 SN65HVD232 邏輯電路

DRIVER (SN65HVD232)

INPUT D	OUTPUTS		BUS STATE
	CANH	CANL	
L	H	L	Dominant
H	Z	Z	Recessive
Open	Z	Z	Recessive

H = high level; L = low level; Z = high impedance

(a) D 輸入真值表

RECEIVER (SN65HVD232)

DIFFERENTIAL INPUTS	OUTPUT R
$V_{ID} \geq 0.9$ V	L
0.5 V < V_{ID} < 0.9 V	?
$V_{ID} \leq 0.5$ V	H
Open	H

H = high level; L = low level; X = irrelevant;
? = indeterminate

(b) R 輸出真值表

表 4-1 SN65HVD232 邏輯真值表

所示，其中 D (driver)是將輸入訊號轉為 CAN 網路線上的 CAN_H 和 CAN_L，其真值表如表 4-1(a)，一旦輸入訊號為低準位，將轉換為網路線上的 dominant 位元，否則轉為 recessive 位元，而由圖上也可看出 CAN_H 和 CAN_L 作為輸入而產生輸出 R (receiver)，因此，除了接收網路線上的訊息，當傳輸訊息時，可同時透過 R 輸出，比對傳出的訊息是否與線上的相同，進而進行 CAN 的仲裁機制，R 輸出的真值表如表 4-1 所示。

4-1-3 Slave 端硬體介紹

Slave 端接收來自 CAN 網路的速度命令，並進行速度迴路控制，主要架構以微控器 8051 為運算核心，並搭配 CAN 網路介面、DA 介面和 Decoder 介面所組成。

◆ CAN 網路介面

CAN 介面的硬體架構可分兩種方式，一種為內建於處理器之中的 CAN controller，如 4-1-1 節中，內建於 F2812 中的 CAN controller，處理器本身即可進

行通訊；另一種則為獨立(stand-alone)的 CAN controller，處理器透過 IO 接腳控制其運作，即可進行 CAN 的傳輸動作，至於細部的封包設定和傳輸行為，則由 CAN controller 負責處理。

在此，由於 slave 端所使用的 8051 並沒有內建 CAN controller，必須搭配獨立的 CAN controller，此處所使用的是 Philips 半導體所生產的 SJA1000，為目前最為常見的 CAN 通訊晶片，主要特色如下：

- 支援最高 1Mbps 傳輸速率
- 支援 CAN 2.0A 和 CAN 2.0B 兩種通訊規格
- 提供 64-byte FIFO 作為接收暫存器(receive buffer)
- 提供兩種接收訊息過濾器(acceptance filter)
- 提供各類的網路錯誤偵測

其架構圖如圖 4-6 所示，其中 64-byte receive FIFO，可同時儲存多筆訊息，使得程式設計上，將多筆訊息同時讀取並進行處理，增加處理器效能；訊息過濾器的設定，可將不需要的訊息先行濾除，其概念是藉由 CAN 封包的識別碼和資料欄

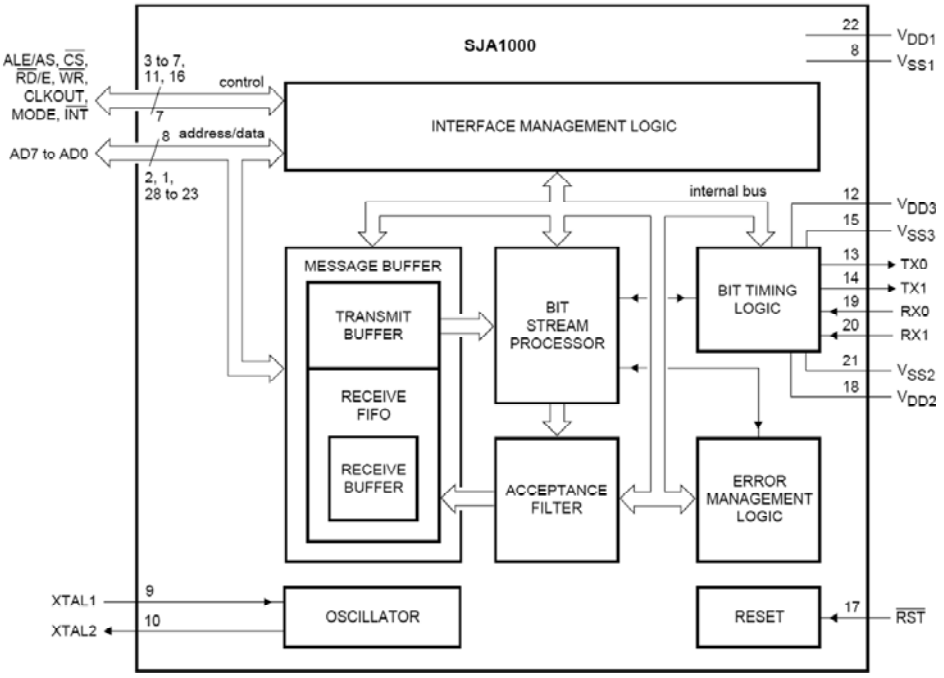


圖 4-6 SJA1000 內部架構

框的資料內容，作為過濾的比對基準，根據預定接收的封包識別碼，規劃過濾器設定值，下圖 4-7 為一範例，其中過濾器包含兩個部份：ACR (acceptance code register)和 AMR (acceptance mask register)，ACR 給定欲比對的識別碼以及資料內容；AMR 則與 ACR 格式相同，且與之相互對應，其主要作用為決定輸入訊息需和 ACR 之中的位元進行比對，若 AMR 中的位元設定為 0，表示該位元所對應的 ACR 位元必須進行比對，進入 SJA1000 的訊息識別碼必須與此 ACR 設定值相同，反之，若為 1 則不需比對，此範例中，訊息識別碼最低兩個位元(ID.19 和 ID.18)以及 RTR 位元，所對應的 AMR 位元皆為 1 而不需進行比對，其餘的每個位元則與必須和 ACR 中的相同，所以，在此設定之下，可接收的訊息是訊息識別碼 0x000 到 0x003 的資料或遙控欄框，其餘訊息皆被過濾而不進入 receive FIFO 中，如此一來，便可提昇處理器的效率。

此外，SJA1000 同樣必須搭配 transceiver 將訊號轉換為差動訊號，在此，所使用的 transceiver 為同樣是 Philips 半導體生產的 PCA82C251，其架構與上節所提到的 SN65HVD232 類似，主要差別在於允許的輸入準位和驅動能力不同，前者主要搭配 5 伏特的處理器，最多支援 110 個節點，後者則是用於 3.3 伏特的處理器中，最多可支援 120 節點，此處，由於 SJA1000 為 5 伏特輸出，所以，使用 PCA82C251 為 transceiver，所組成的 CAN 介面電路如圖 4-8。

n	0	1(upper 4bits)	2	3
ACRn(binary)	0000 0000	0101	1010 1010	0000 1111
AMRn(binary)	0000 0000	0111	1111 1111	1111 1111
Accepted messages (ID.28…ID.18, RTR)	0000 0000	0xxx		

圖 4-7 SJA1000 訊息過濾器範例

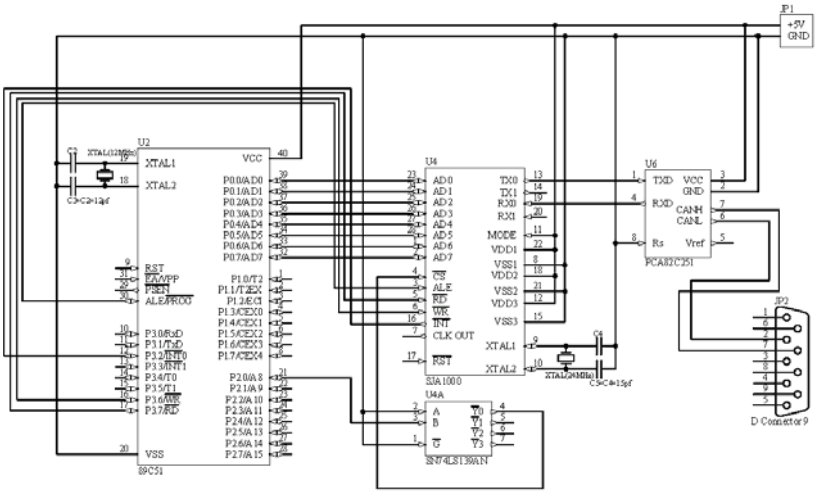


圖 4-8 Slave 端 CAN 網路介面

◆ DA 介面

本論文所使用的馬達驅動器是以類比電壓作為控制輸入，進而控制馬達轉速，所以，必須建立 DA 介面將數位命令轉為電壓輸入，此處所使用的 DA 元件為 Analog Devices 公司所生產的 AD7541，為 12 位元數位輸入的 DA 元件，其特色為隨著提供的參考電壓不同，可進行±25 伏特之間的電壓轉換。

在 DA 元件完成轉換後，輸入到驅動器之前，為了避免控制電路和馬達電路之間的干擾，必須透過緩衝(buffer)加以隔離，在此所使用的是 TI 所生產的 TLC2274，為 Rail-to-Rail 的運算放大器，以此設計緩衝電路作為 DA 電路輸出級，所設計之 DA 介面電路如圖 4-9 所示，由於 DA 輸入為 12 位元，因此，必須使用兩組 8051 輸出埠進行控制，所得之類比輸出電壓(V_{out})：

$$V_{out} = \frac{D}{4096}(Out1 - Out2) \tag{4-8}$$

其中 D 為數位命令值，即 DA 的 12 位元輸入；$Out1 - Out2$ 為類比參考電壓，此處為 5 伏特，因此，類比輸出值將在 0~5 伏特之間改變。

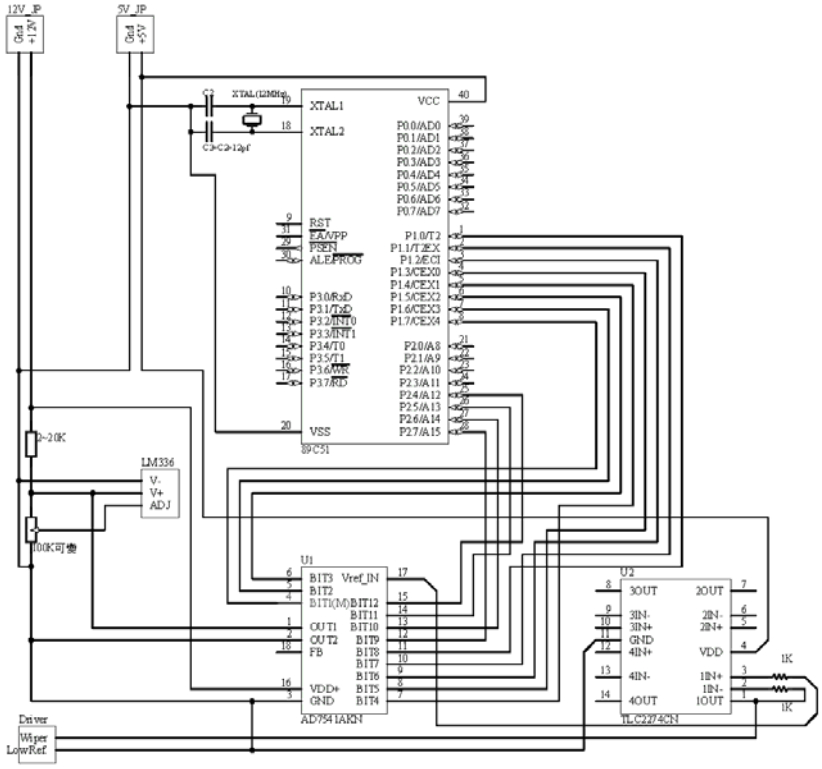


圖 4-9 DA 介面電路

◆ Decoder 介面

平台各軸接收速度命令後，進行閉迴路速度控制，為了獲得輪子的轉速，藉由輪軸上的增量型編碼器(encoder)，在輪子轉動時，進行編碼而產生對應的正交編碼器脈衝(QEP, quadrature encoder pulse)，QEP 訊號通常包含 A、B 相，兩者相位相差 90 度，造成不同的編碼組合，用以表示馬達的轉動方向，而 QEP 的脈衝數則與馬達轉動速度成正比。

因此，藉由將 QEP 訊號進行解碼，便可了解馬達的轉動方向與對應的轉速，此處所使用的解碼器(decoder)為 Agilent 公司所生產的 HCTL 2032，其主要特色如下：

- 最高 33 MHz 的時脈頻率
- 支援雙軸(Dual Axis)同時計數
- 支援 32 位元上下計數
- 高雜訊免疫性：施密特(Schmitt)觸發器、數位雜訊濾波器

- 8 位元並列式計數值輸出

所設計之 decoder 介面電路如圖 4-10 所示，而由於 8051 之 IO 埠有限，P0 埠同時作為 SJA1000 和 2032 的存取介面，因此，必須透過額外電路切換兩個元件的使用，在此，使用解多工器 SN74LS139 進行切換動作，以達到兩組電路的存取。

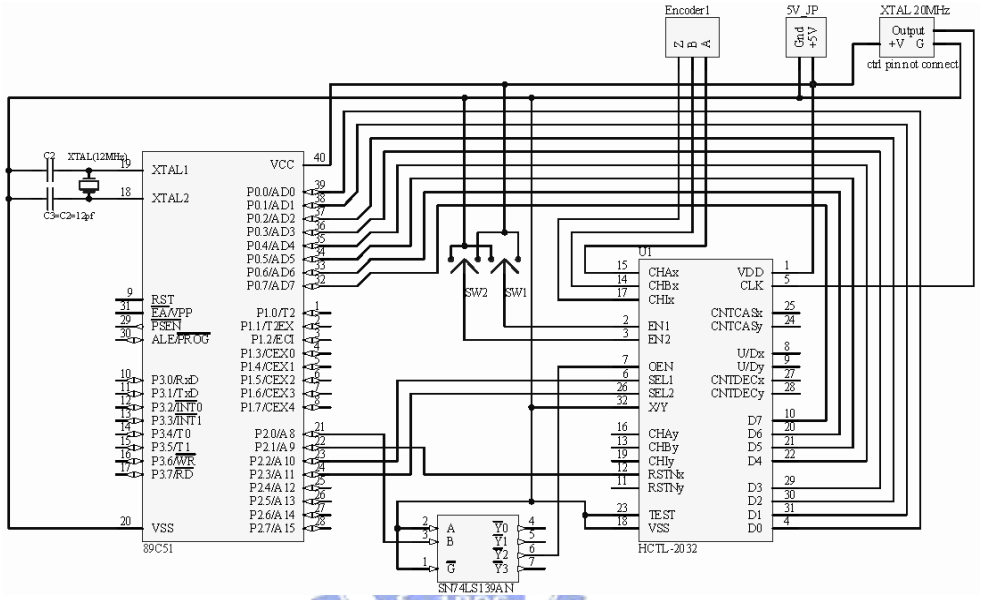


圖 4-10 Decoder 介面

4-2 平台運動模型[25]

為了進行平台之運動控制，本節定義移動平台的相關參數和座標系統轉換，接著以此推導其運動模型。

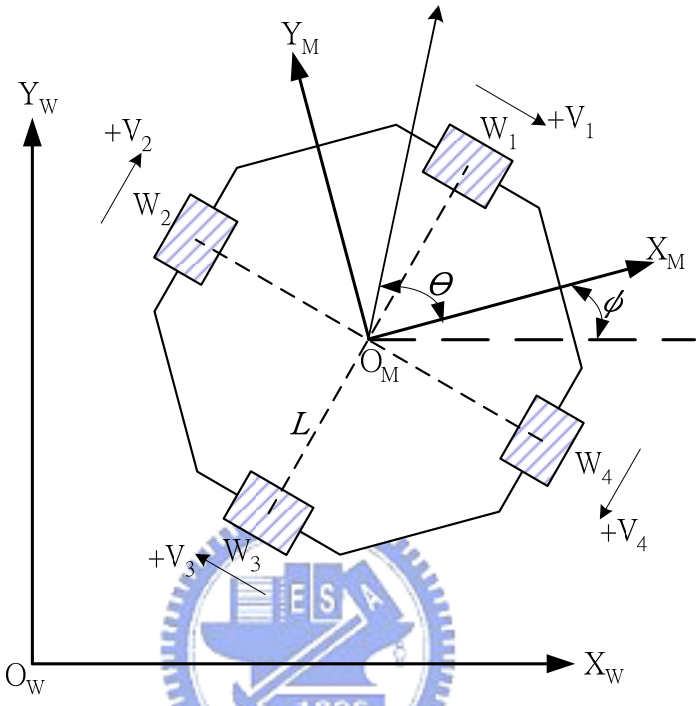


圖 4-11 全向性平台座標系統與符號定義

◆ 全向性平台座標系統與相關符號定義

圖 4-11 為全向性平台的座標系統和符號定義，各個符號所代表之意義如下：

X_W、Y_W、O_W：世界座標(world coordinate)X 軸、Y 軸與原點

X_M、Y_M、O_M：平台座標(mobile coordinate)X 軸、Y 軸與原點

W_1：平台座標第一象限的全向輪(wheel A)

W_2：平台座標第二象限的全向輪(wheel B)

W_3：平台座標第三象限的全向輪(wheel C)

W_4：平台座標第四象限的全向輪(wheel D)

$\alpha_1 \sim \alpha_4$：X_M 與 $W_1 \sim W_4$ 之間的角度

F：平台移動方向

θ：平台移動方向與平台 X 軸的夾角

ϕ：X_M 與 X_W 的夾角，平台逆時針旋轉為正

L：輪子與平台中心距離

r：全向輪半徑

$+V_1 \sim +V_4$：定義 $W_1 \sim W_4$ 之正轉方向

$\omega_1 \sim \omega_4$：全向輪 $W_1 \sim W_4$ 的旋轉角速度

◆ 全向性平台之運動模型

運動模型相關定義如上所示，根據其幾何關係，可將各軸轉速與平台運動速度整理成下式(4-1)，

$$r\varpi_i = [\sin\alpha_i - \cos\alpha_i]\begin{bmatrix}\dot{X}_M \\ \dot{Y}_M\end{bmatrix} - L\dot{\phi} \quad \text{i} = 1,2,3,4 \tag{4-1}$$

接著，建立世界座標與平台座標間的轉換矩陣(coordinate transformation matrix)C_T：

$$\begin{bmatrix}\dot{X}_M \\ \dot{Y}_M\end{bmatrix} = \mathrm{C_T}\begin{bmatrix}\dot{X}_W \\ \dot{Y}_W\end{bmatrix} \tag{4-2}$$

其中

$$\mathrm{C_T} = \begin{pmatrix}\cos\phi & \sin\phi \\ -\sin\phi & \cos\phi\end{pmatrix} \tag{4-3}$$

由(4-1 式~ 4-3 式)可得平台於世界座標移動形式與各軸轉速的對應關係，如下式(4-4)，

$$\begin{bmatrix}v_1 \\ v_2 \\ v_3 \\ v_4\end{bmatrix} \equiv \begin{bmatrix}r\omega_1 \\ r\omega_2 \\ r\omega_3 \\ r\omega_4\end{bmatrix} = D_e(\phi)\begin{bmatrix}\dot{X}_w \\ \dot{Y}_w \\ \dot{\phi}\end{bmatrix} \tag{4-4}$$

其中 $D_e(\phi)$為解耦合矩陣(decoupled matrix)：

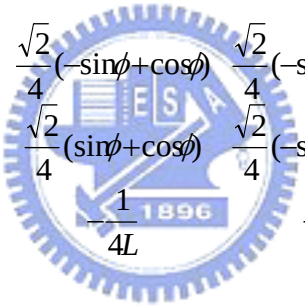
$$D_e(\phi)=\begin{bmatrix} \frac{1}{\sqrt{2}}(\sin\phi+\cos\phi) & \frac{1}{\sqrt{2}}(\sin\phi-\cos\phi) & -L \\ \frac{1}{\sqrt{2}}(-\sin\phi+\cos\phi) & \frac{1}{\sqrt{2}}(\sin\phi+\cos\phi) & -L \\ \frac{1}{\sqrt{2}}(-\sin\phi-\cos\phi) & \frac{1}{\sqrt{2}}(-\sin\phi+\cos\phi) & -L \\ \frac{1}{\sqrt{2}}(\sin\phi-\cos\phi) & \frac{1}{\sqrt{2}}(-\sin\phi-\cos\phi) & -L \end{bmatrix} \tag{4-5}$$

而為了從各軸回授反推平台的狀態，因此，建立以下對應關係：

$$\begin{bmatrix} \dot{X}_w \\ \dot{Y}_w \\ \dot{\phi} \end{bmatrix} = D_i(\phi)\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \tag{4-6}$$

其中 $D_i(\phi)$稱為 inverse kinematics 矩陣：

$$D_i(\phi)=\begin{bmatrix} \frac{\sqrt{2}}{4}(\sin\phi+\cos\phi) & \frac{\sqrt{2}}{4}(-\sin\phi+\cos\phi) & \frac{\sqrt{2}}{4}(-\sin\phi-\cos\phi) & \frac{\sqrt{2}}{4}(\sin\phi-\cos\phi) \\ \frac{\sqrt{2}}{4}(\sin\phi-\cos\phi) & \frac{\sqrt{2}}{4}(\sin\phi+\cos\phi) & \frac{\sqrt{2}}{4}(-\sin\phi+\cos\phi) & \frac{\sqrt{2}}{4}(-\sin\phi-\cos\phi) \\ -\frac{1}{4L} & -\frac{1}{4L} & -\frac{1}{4L} & -\frac{1}{4L} \end{bmatrix} \tag{4-7}$$



透過 4-4 式，可將平台運動於世界座標的移動行為，即 x 軸速度、y 軸速度和平台旋轉角速度，轉換成給予各軸的速度命令，以進行平台控制；另一方面，經由 4-6 式，可由各軸之速度回授，換算為平台於世界座標的移動速度，以進行平台的閉迴路控制。

4-3 系統程式架構

系統程式架構的部份，主要分為平台控制部份和網路通訊兩個部份，4-3-1 節首先說明平台回授運動控制架構部份，接著，4-3-2 節則為整體系統整合的程式部份。

4-3-1 全向平台運動控制架構

從整體的控制流程來看，可分成命令產生、平台控制迴路和各軸控制迴路三個部份，如圖 4-12 所示，其中命令運算(速度規劃)和平台控制迴路由 master 節點實現，所產生的平台運動命令經上節所述之解耦合矩陣，轉換成各軸運動命令並經由網路傳送到 slave 節點，slave 節點接收命令後執行，進行閉迴路速度控制並將速度迴授傳回 master 節點。底下為各部份的說明：

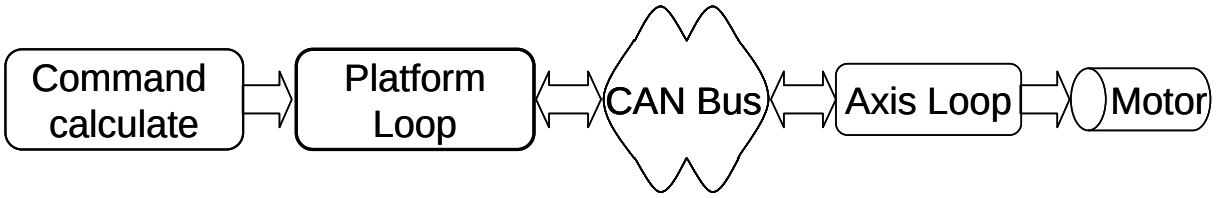


圖 4-12 全向平台運動控制架構

◆ 命令計算：

命令計算主要是將所設定的運動參數，如移動距離、移動時間、旋轉角度等參數，進行速度規劃並產生平台運動命令，提供給各軸控制控制器，因此，這部份的核心在於如何進行速度規劃(velocity profile)，使得運動軌跡更為平滑，以減少對於機構的磨損。

本論文所採用的速度規劃方式為梯形速度規劃，將平台在世界座標下的 x 軸移動速度、y 軸移動速度和旋轉速度，分配為加速、等速和減速三段過程，以逐漸加速到最高速，然後逐漸減速到靜止的方式，使平台移動方式更加平滑，其規劃方式如下：

已知平台總移動距離(D)與時間(T)，定義其加減速時間皆相同，可得其加速時間(Ta)，以此可計算出其平均速度(V_{max})和最大加速度(A_{max})

$$T_a = \frac{1}{3}T \tag{4-9}$$

$$V_{\max} = \frac{D}{T - T_a} \quad and \quad A_{\max} = \frac{V_{\max}}{T_a} \tag{4-10}$$

由以上可得梯形規劃的三段速度如下：

$$V(t) = \frac{t}{Ta} V_{\max} \mid_{t=0\sim Ta} \tag{4-11}$$

$$V(t) = V_{\max} \mid_{t=Ta\sim(T-Ta)} \tag{4-12}$$

$$V(t) = \frac{(T-t)}{T_a} V_{\max} \mid_{t=(T-Ta)\sim T} \tag{4-13}$$

因此，隨著時間 t 的增加，所得的速度命令 V(t)隨之呈現梯形形式，以此方式，便可逐步產生運動命令，避免一此給予過大的命令，造成機構的傷害。

◆ 平台控制迴路

平台控制迴路的部份，主要將速度規劃後所得的位置命令，以及經由逆轉換的平台位置進行差分，作為平台速度控制器的輸入，透過 PI 控制器產生平台速度命令，透過前述的解耦合運算將此命令分散成四軸命令，最後，經由 CAN bus 送到各軸；相對地，在回授運算上，同樣經由 CAN bus 接收各軸回授，經由逆轉換之後，得知平台的狀態，此部份架構如圖 4-13 所示。

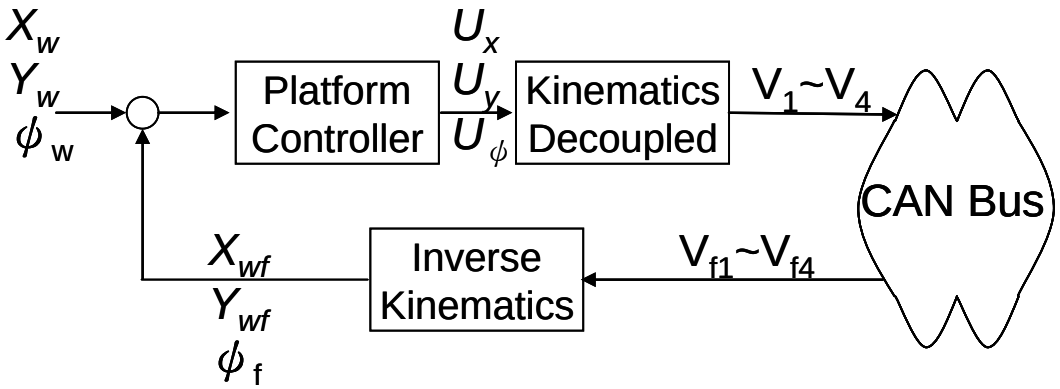


圖 4-13 DSP 端平台控制器架構

◆ 各軸控制迴路

速度規劃後的結果，為平台於世界座標下的運動速度，透過上節所述的解耦

合矩陣，將平台的運動速度轉換為各軸的運動命令，接著，透過 CAN bus 傳到各軸進行控制。因此，各軸的控制迴路部份主要是接收來自 CAN bus 的速度命令，以此進行速度 PI 控制，並透過 CAN bus 將速度迴授傳給 master 節點，此部份架構如圖 4-14 所示。

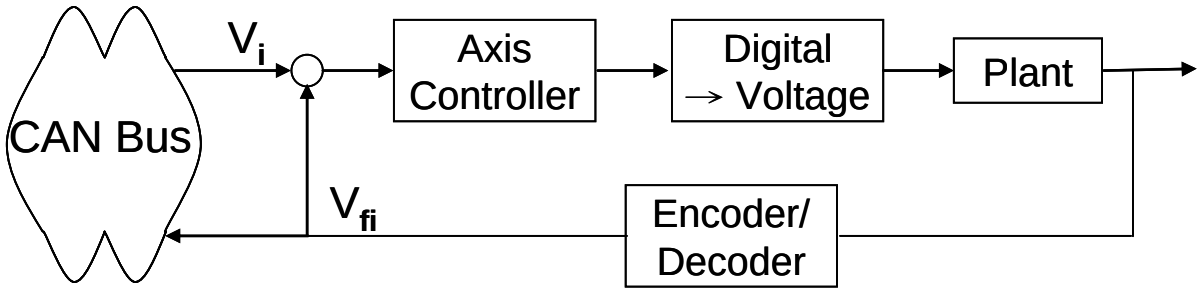


圖 4-14 8051 端單軸控制架構

4-3-2 全向平台整體程式架構

全向平台系統程式，主要是將運動控制架構和第三章之應用通訊協定結合，將 4-3-1 節的控制流程和網路管理行為整合，達到分散式的平台控制架構，基本架構如 4-1 節所述的 Master/Slave 架構，其中 master 端負責速度規劃與產生控制命令，並作為網路管理的主控者，slave 則接收來自 master 端的命令，加以執行回覆。接著，分別為 master 端與 slave 端的程式流程說明。

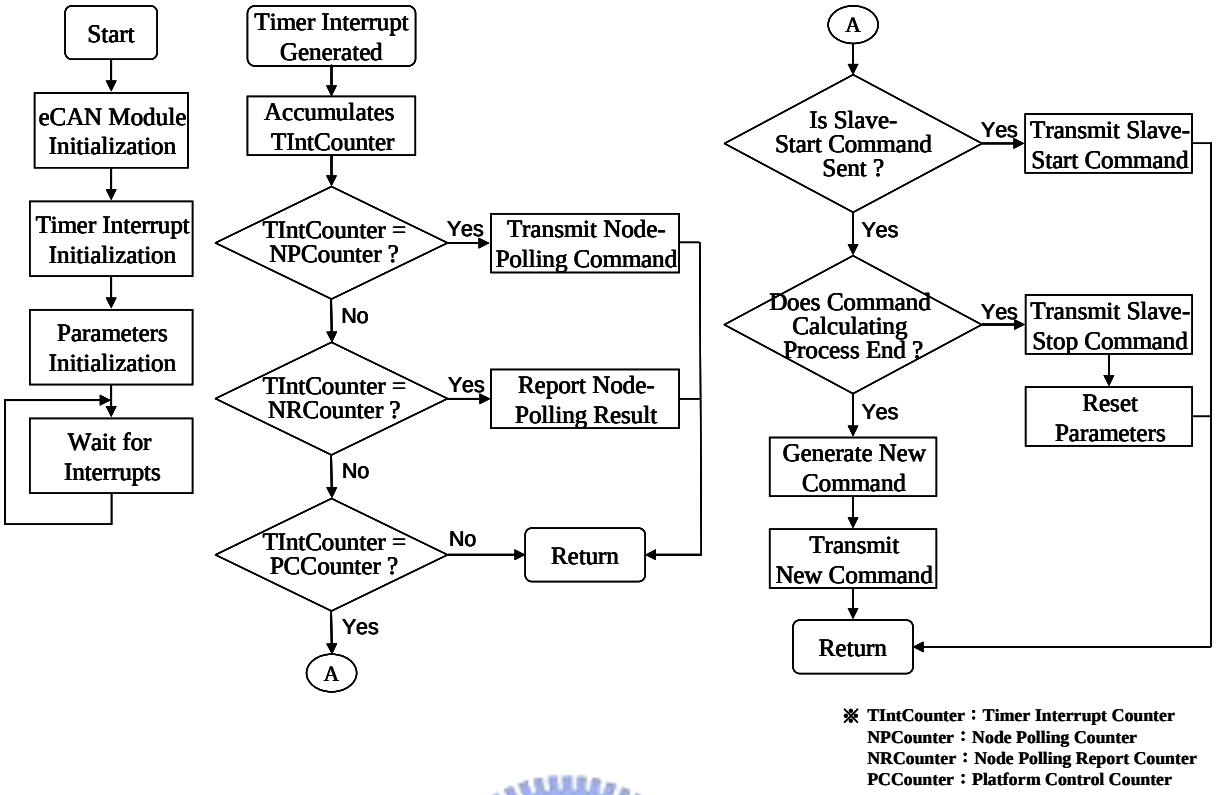
◆ Master 端程式流程

Master 端主要工作為速度命令的產生和網路管理，整體運作控制是透過使用者由 PC 端下達動作指令，啟動其時脈中斷系統，接著，在時脈中斷裡，進行速度規劃和各軸命令運算，並以第三章所設計之通訊方式，監控每個節點的狀態。至於 CAN 網路訊息封包的接收方式，則透過 eCAN 的事件中斷系統接收，包含 PC 端所下達之指令與參數值、網路管理回覆與各軸速度迴授。

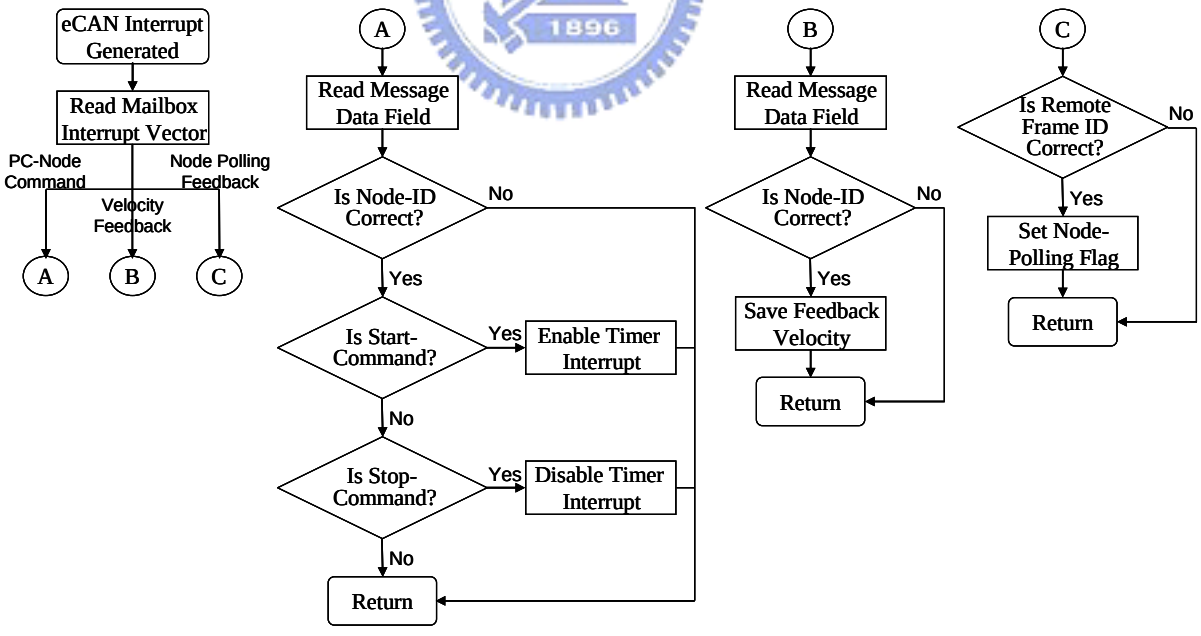
因此，master 端的程式流程如圖 4-15 (a)、(b)所示，其中在時脈中斷裡，主

要分為兩個部份：節點狀態偵測與速度命令的運算與傳送，當中斷發生時，以時脈中斷計數器的值為判斷基準，採取不同的動作，平台運動控制部份，根據系統取樣週期而執行，起始狀態由 master 端傳送起始命令給四軸 slave 節點，啟動各軸的運作，接著在每個取樣週期，master 端將依據所給的移動參數，進行速度規劃並且傳送速度命令到各軸，以此循環直到結束；至於節點狀態偵測與回報，則設計在兩個系統取樣週期之間執行，避免影響平台運動控制流程，master 首先傳送節點詢問訊息，要求每個節點回應，接著，經過一個取樣時間之後(同樣在兩個取樣週期間)，比對各個節點的回應狀況，一旦發現有節點未回應，便停止系統運作，等待使用者重新啟動或加以修復。

至於 CAN 訊息接收，包含 PC 端下達之指令與參數設定、節點偵測的回覆與各軸速度迴授，則透過 eCAN 事件中斷系統完成，依據上述三種訊息類型，設計對應的 mailbox 以接收訊息，每個 mailbox 中斷所對應到的中斷向量皆不同，當 eCAN 模組產生中斷時，根據中斷向量的值，便可得到哪一個 mailbox 接收到訊息，進而採取對應的措施，在圖 4-15 上，以單軸的訊息接收為例，進入 eCAN 事件中斷後，讀取 mailbox 的中斷向量，並執行對應的處理流程，以 PC 端的指令和各軸速度迴授來說，必須先行讀取資料區的節點識別碼，確定傳送節點正確，才執行指令或儲存迴授資料；而節點偵測的回覆，採用遙控欄框的方式，所以，只需比對訊息識別碼的正確性，若確認無誤便設定旗標，表示節點已回應。



(a) Master 端主程式與時脈中斷程式流程圖



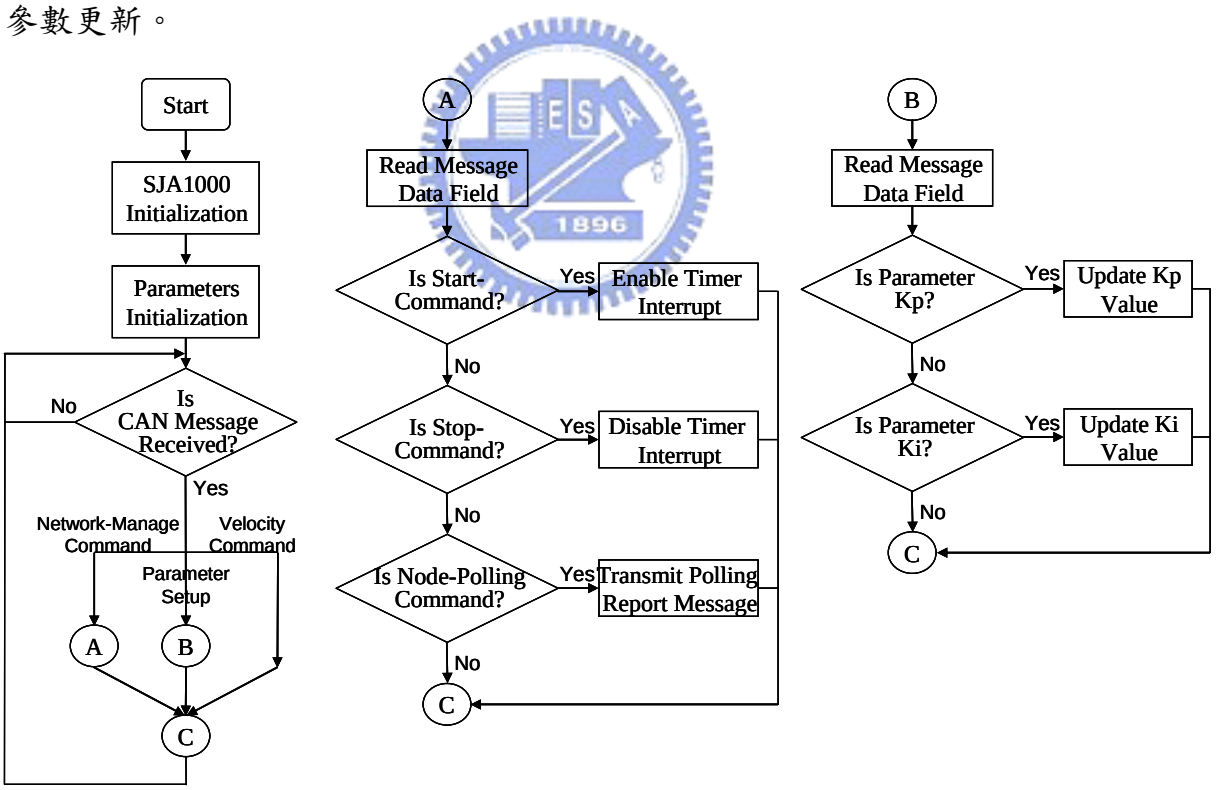
(b) Master 端 eCAN 程式流程圖

圖 4-15 Master 端程式流程圖

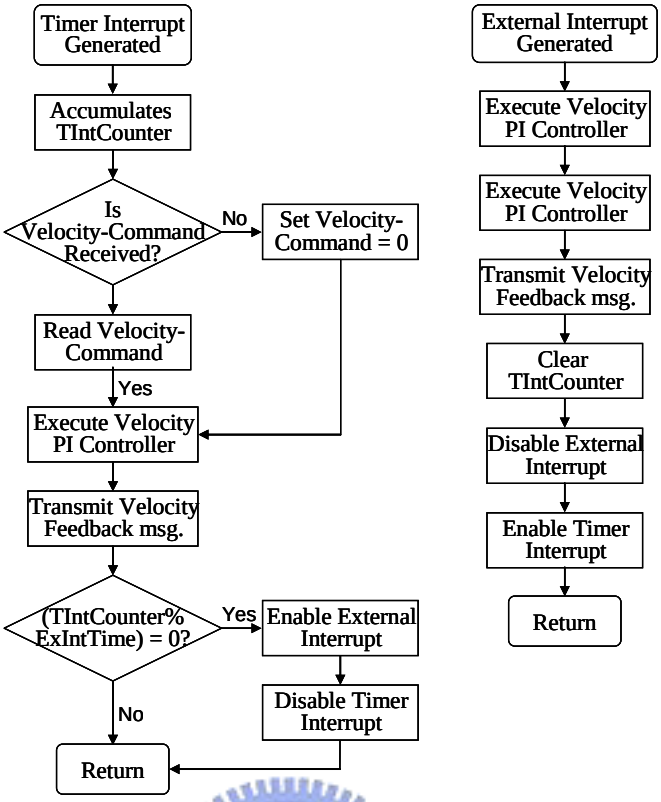
◆ Slave 端程式流程

Slave 端主要接收來自 master 的控制命令和網路管理訊息，並將速度迴授結果回傳到 master，其程式流程如圖 4-16 所示，在主程式部份，主要為 CAN 訊息封包接收和處理，除了速度命令的接收，網路管理訊息和參數設定皆在主程式完成，而各軸速度迴路則主要由時脈中斷系統，至於外部中斷系統則用於時脈同步方法。

以網路管理部份來說，包含節點操作指令(啟動和停止)與節點偵測命令，一旦接收到啟動指令，便開啟時脈中斷系統，開始平台移動控制流程，反之，接收到停止指令便關閉時脈中斷，結束平台各軸控制流程。另一方面，若皆收到節點偵測指令，便以遙控欄框加以回應。若接收到參數設定訊息，則將節點的控制器參數更新。



(a) Slave 端主程式流程圖



(b) Slave 端中斷程式流程圖

圖 4-16 Slave 端程式流程圖

在平台移動控制中，以各軸的時脈中斷系統為基準，在各個取樣週期裡，檢查是否接收到速度命令，若皆收到新的速度命令，便以之進行速度控制，同時將速度回授值傳給 master 端，完成控制流程；至於時脈同步方法，在完成平台控制流程後，以時脈中斷計數器值為準，判斷是否進行時脈同步，若需執行同步，將時脈中斷關閉而改以外部中斷的方式接收訊息，完成時脈同步，並在外部中斷程式裡，執行平台控制流程。

4-4 分散式平台控制系統實現

本節主要介紹分散式平台控制系統的實現結果，首先，介紹第三章的應用通訊協定於平台系統的實驗結果，接著，說明平台運動控制的結果與相關的探討。

4-4-1 應用層通訊協定實現

◆ 全向平台網路事件規劃

全向平台的分散式架構中，包含 PC 端、一個 master 節點和 4 個 slave 節點，根據第三章所設計之應用通訊協定，各個節點都應具有各自的識別碼(Node-ID)，其中 PC 端主要作為網路監控之用，並不參與系統控制流程，其 Node-ID 為 0x01，至於平台控制架構的五個節點，同樣給予獨立的 Node-ID，其中 master 的 Node-ID 為 0x02，4 個 slave 則分別為 0x04、0x08、0x10 和 0x20，依據這樣的定義，配合所設計之應用通訊協定，規劃出全向平台控制系統的網路訊息事件，如表 4-2 所示，其中 source node 表示傳輸訊息節點，destination node 表示接收節點，function code 為訊息功能類別，Omni-ID 為訊息的 CAN 識別碼表示接收節點，function

表 4-2 全向平台控制系統之網路訊息事件表

Source Node	Destination Node	Function Code(binary)	Omni-ID	Frame Type Data Size	Function
PC	Master or slaves	000	0 (0x000)	Data 2-byte	節點運作指令，控制節點的啟動與停止
PC	Master or slaves	000	0 (0x000)	Remote	節點偵測指令，測試每個節點的連線狀況
PC	Master	111	1794 (0x702)	Data 6-byte	節點參數設定，設定移動控制相關參數值
Master	Slaves	000	0 (0x000)	Data 2-byte	節點運作指令，控制節點的啟動與停止
Master	Slaves	000	0 (0x000)	Remote	節點偵測指令，測試每個節點的連線狀況
Master	PC	000	2 (0x002)	Remote	回應節點偵測指令
Master	Slaves	001	258 (0x102)	Data 4-byte	四軸速度命令傳輸，同時傳輸四軸速度命令
Master	Slaves	111	1796(0x704) 1800(0x708) 1808(0x710) 1824(0x720)	Data 4-byte	各軸控制參數傳輸，設定各軸控制參數
Slaves	PC or master	000	4(0x004) 8(0x008) 16(0x010) 32(0x020)	Remote	各軸回應節點偵測指令
Slaves	Master	010 011 101 110	514(0x202) 770(0x302) 1282(0x502) 1538(0x602)	Data 6-byte	各軸分別將速度迴授回傳到 master 端

code 為訊息功能類別，Omni-ID 為訊息的 CAN 識別碼(identifier)，Frame Type 和 Data Size 分別為訊息欄框的格式與大小，最後為訊息欄框的功能說明。

資料欄框的資料區規劃上，主要根據應用協定的設計方式，用於節點運作控制的資料欄框，最高位元組為接收節點 Node-ID，另一個位元組為運作指令；用於控制命令和參數傳遞的資料欄框，最高位元組為傳送節點的 Node-ID，接著為資料型態，之後的位元組則為資料內容；至於四軸速度命令傳輸用的資料欄框，則分別由四個位元組構成，最高位元組為第四軸速度命令，接著為第三軸命令，以此類推，將四軸的命令放在同筆訊息中加以傳輸。

◆ 全向平台應用通訊協定實現結果

本部份的說明，以上述的網路訊息事件實現結果為主，而網路時脈同步方法的實現結果，將與 4-4-2 節的平台控制實驗結果一起說明。

在應用協定結果的說明中，以 PC 端所接收的訊息，作為實驗結果的呈現，在此，藉由鄭景文學長以 USB CAN 為基礎，所開發的網路監控程式[26]，作為網路系統的監測之用，其介面如圖 4-17 所示，其中(a)部份為 USB CAN 的基本設定，如傳輸速率、訊息過濾器設定等等；(b)部份為傳輸訊息用的相關設定，包含訊息欄框的類型、傳輸的資料內容等，皆由此處設定；(c)部份為網路訊息的顯示，傳輸與接收的訊息皆可顯示於此處，包含訊息的各項特性也將於此處顯示，如封包傳遞方向、訊息識別碼、資料長度與內容等特性，都可透過此處加以監控。

至於實驗結果的說明，以三個部份為主，包含節點偵測機制、節點參數設定、節點運作與控制流程。在節點偵測機制上，實驗結果如圖 4-18 所示，第一筆訊息為節點偵測指令，由 PC 端以遙控欄框發送給系統的各個節點，接著為各個節點的回應，由 Frame ID 可看出分別由哪些節點所回傳的訊息。節點參數設定的實

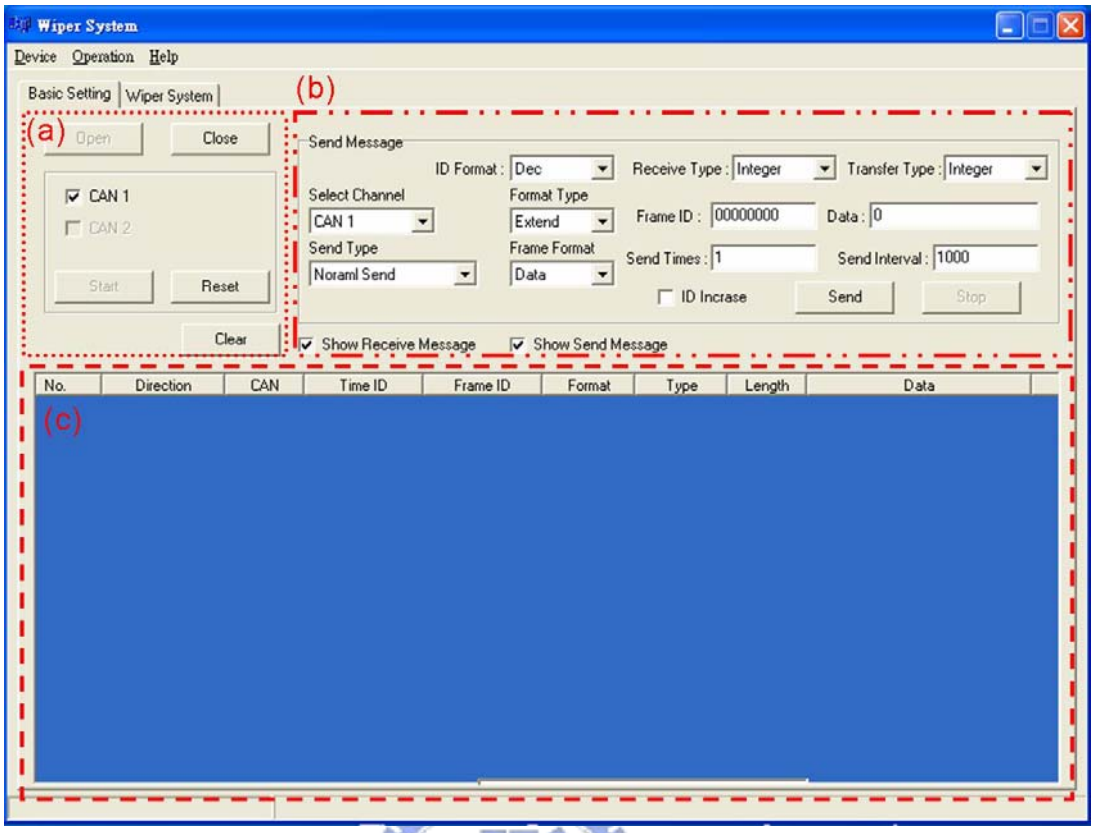


圖 4-17 網路監控程式介面[26]

驗結果如圖 4-19 所示，此處為 PC 端透過 master 進行各軸 slave 參數設定之用，由資料區的最高位元組可看出傳送節點為 0x01(也就是 PC 端)，接著為參數類型為 0x0C，表示為 slave 的控制器設定值，後面 4 個位元組則為參數值，master 接收到此筆訊息之後，便此設定分別傳給 4 個 slave 進行設定，此方式可同時設定多軸的參數，當然，若需對特定軸進行設定，同樣可由 PC 端加以設定。至於節點運作與控制流程的實驗結果，如圖 4-20 所示，這是平台控制系統啟動的流程，首先，PC 端發出命令啟動 master 的運作，再由 master 對 4 軸節點發出啟動命令，完成所有節點的開啟流程，接著，在下一個取樣週期，master 開始傳送各軸的速度命令(Frame ID = 258 者)，4 軸接收命令執行後，將其速度回授分別回傳到 master，系統便以此方式反覆運作，直到 master 送出停止命令並結束所有的控制流程，如此一來，便可達到全向平台分散式運動控制的目的。

No.	Direction	CAN	Time ID	Frame ID	Format	Type	Length	Data
1	Send	1	-	0	Remote	Standard	-	-
2	Receive	1	1503:58.881.8	2	Remote	Standard	-	-
3	Receive	1	1503:58.882.0	4	Remote	Standard	-	-
4	Receive	1	1503:58.882.1	8	Remote	Standard	-	-
5	Receive	1	1503:58.882.1	16	Remote	Standard	-	-
6	Receive	1	1503:58.882.2	32	Remote	Standard	-	-

圖 4-18 節點偵測機制實驗結果

No.	Direction	CAN	Time ID	Frame ID	Format	Type	Length	Data
1	Send	1	-	1794	Data	Standard	6	01 0C 00 00 00 64
2	Receive	1	1557:28.820.5	1796	Data	Standard	4	02 01 00 64
3	Receive	1	1557:28.820.6	1800	Data	Standard	4	02 01 00 64
4	Receive	1	1557:28.820.7	1808	Data	Standard	4	02 01 00 64
5	Receive	1	1557:28.820.8	1824	Data	Standard	4	02 01 00 64

圖 4-19 節點參數設定實驗結果

No.	Direction	CAN	Time ID	Frame ID	Format	Type	Length	Data
1	Send	1	-	0	Data	Standard	2	02 01
2	Receive	1	1507:01.348.1	0	Data	Standard	2	00 01
3	Receive	1	1507:01.358.6	258	Data	Standard	5	02 00 00 00 00
4	Receive	1	1507:01.361.7	514	Data	Standard	6	04 00 00 00 00 00
5	Receive	1	1507:01.361.8	770	Data	Standard	6	08 00 00 00 00 00
6	Receive	1	1507:01.362.0	1282	Data	Standard	6	10 00 00 00 00 00
7	Receive	1	1507:01.362.1	1538	Data	Standard	6	20 00 00 00 00 00
8	Receive	1	1507:01.368.9	258	Data	Standard	5	02 00 00 00 00

圖 4-20 節點運作與控制流程實驗結果

4-4-2 分散式全向平台控制實驗

本部份實驗目的是為了說明全向平台控制實驗結果，並驗證在全向平台網路系統下，時脈同步方法對於系統控制效能所帶來之改善。在實驗過程中，首先，在沒有加入時脈同步方法的情況下，進行平台控制，接著，在各軸節點中，加入時脈同步方法，藉此驗證時脈同步方法的改善效果。

本部份實驗以空載測試為主，假設起始位置的平台和世界座標相同，移動路徑則以直線行走的方式，沿著世界座標的 x 軸方向移動，移動距離為 200 公分，移動時間為 20 秒，根據所定義的平台參數，第一、二軸朝著正方向運動，第三、四軸則朝著負方向運動。

至於實驗結果的評估，以 IAE (integral absolute error)和標準差作為性能指標，前者為系統追跡能力的指標，其計算方法如下式 4-14；後者則為穩定性的指標，其計算方法如下式 4-15。

$$IAE = \sum_{i=1}^{N} | e_i | \tag{4-14}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_i)^2} \tag{4-15}$$

首先，說明在未加入網路時脈同步方法前的結果，圖 4-22 和圖 4-24 為 1000 kbps 下的實驗，圖 4-23 和圖 4-24 為 125kbps 下的實驗，其中位置響應結果可看出，平台在 y 軸上持續有震盪現象的發生，在各軸速度回授中，同樣存在程度不等的震盪，這主要是由於命令遺失和各個節點不同步所造成。

為了比較各個傳輸速率的結果，針對各個傳輸速率的 IAE 與標準差加以統計，進而得到表 4-4~表 4-6，由平台位置來看，除了 125kbps 效能較差，各個速率差異並不大；若以各軸的 IAE 和標準差平均值來說，各軸在此架構下的特性，在正轉的情況下，第二軸的控制效能較好，在反轉的情況下，第四軸的效能較好；由各個傳輸速率下的結果可知，傳輸速率所造成的網路延遲對於控制結果的影響，其中 1000kbps 和 500kbps 的延遲時間較低，造成的資料遺失現象較少，各軸速度命令較能即時送達，可得到較佳的追跡能力與穩定度，而 250kbps 和 125kbps 的傳輸延遲較大，導致速度命令無法即時送達，使得兩項指標都相對的下降，造成較差的控制效能，此外，不論在何種速率之下，時脈漂移的現象也將造成資料遺失現象，影響各軸的控制結果。

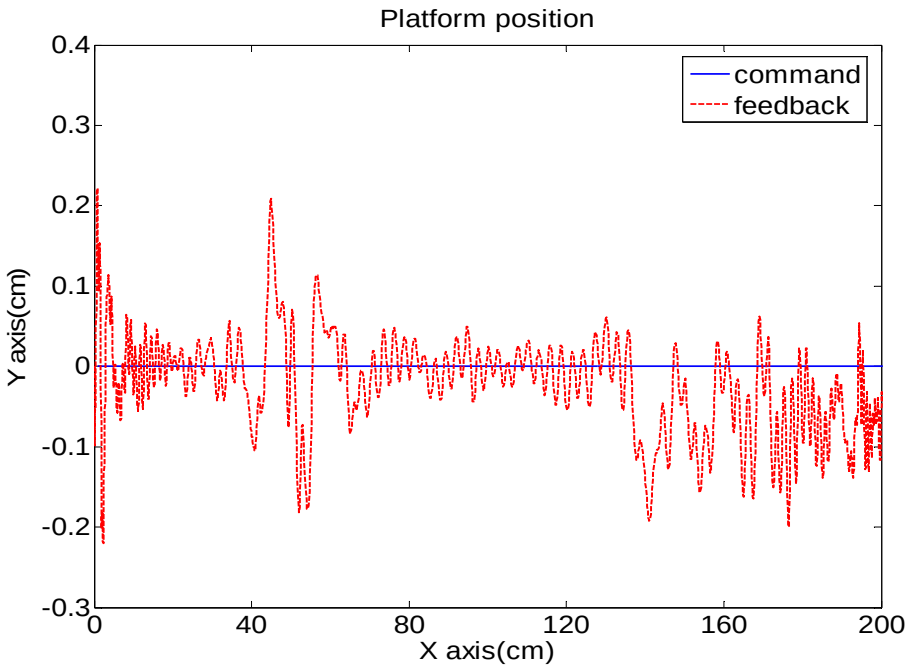


圖 4-21 1000kbps 未加入時脈同步下，平台位置實驗結果

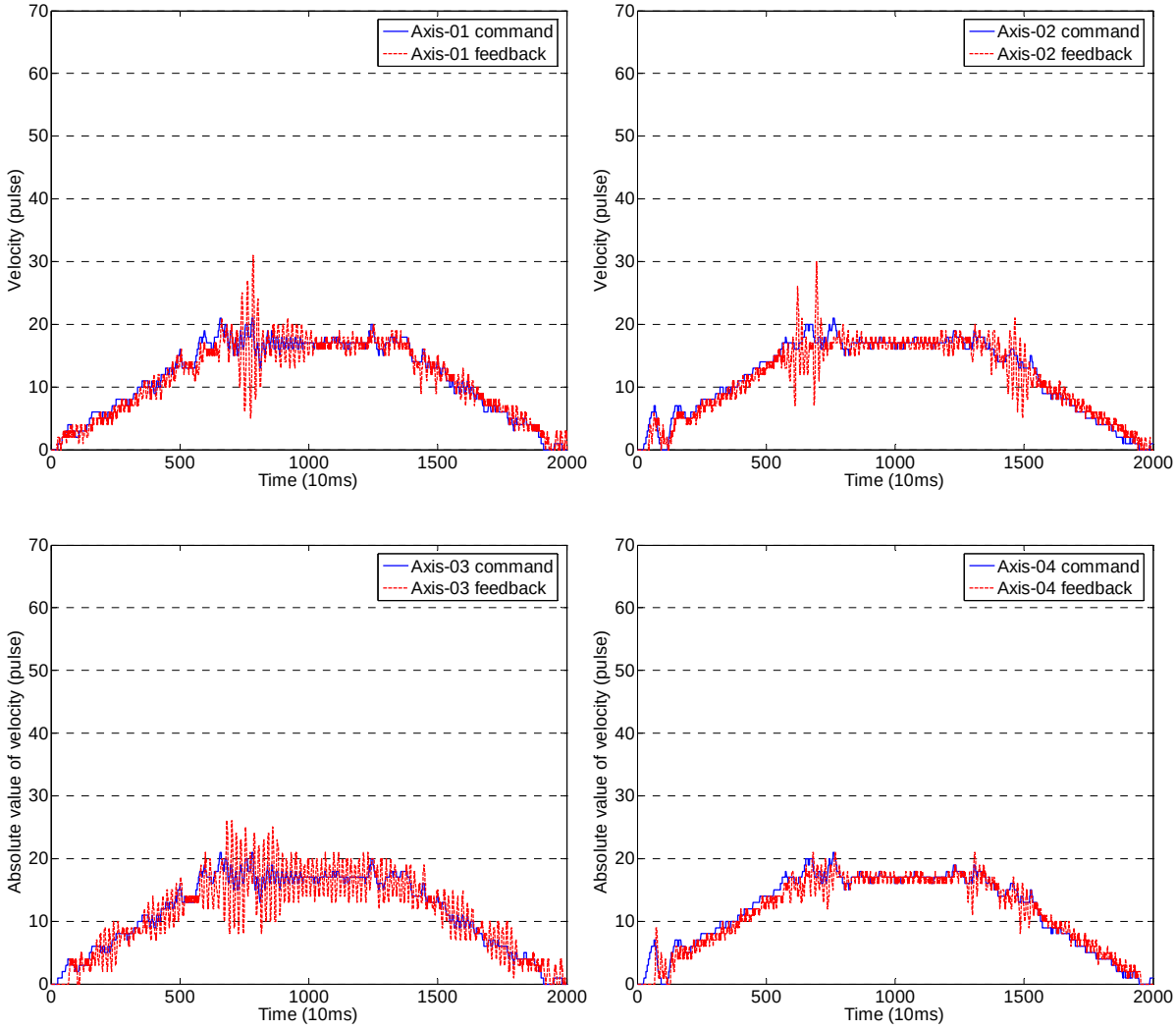


圖 4-22 1000kbps 未加入時脈同步下，各軸的速度響應

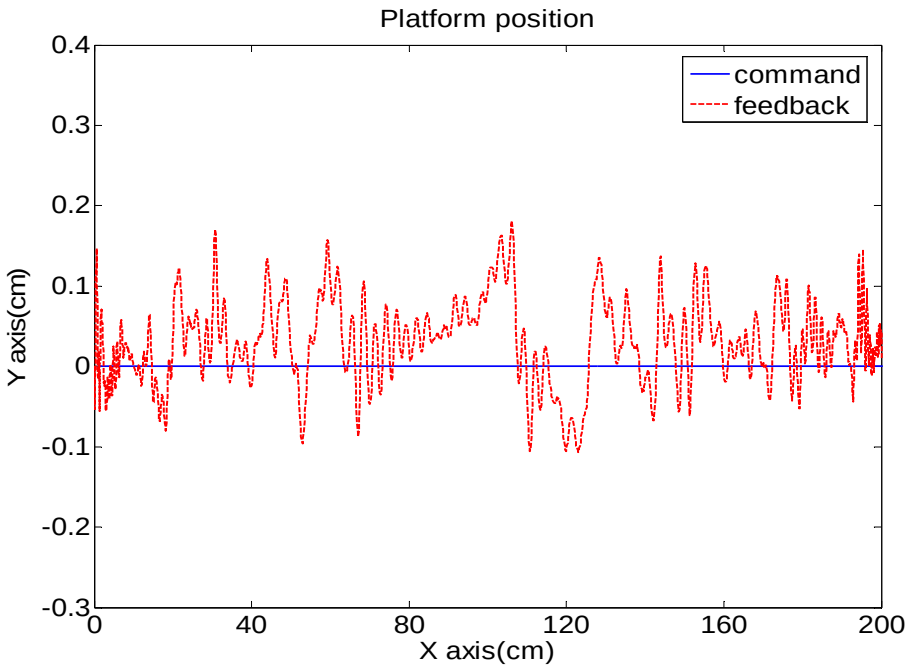


圖 4-23 125kbps 未加入時脈同步下，平台位置實驗結果

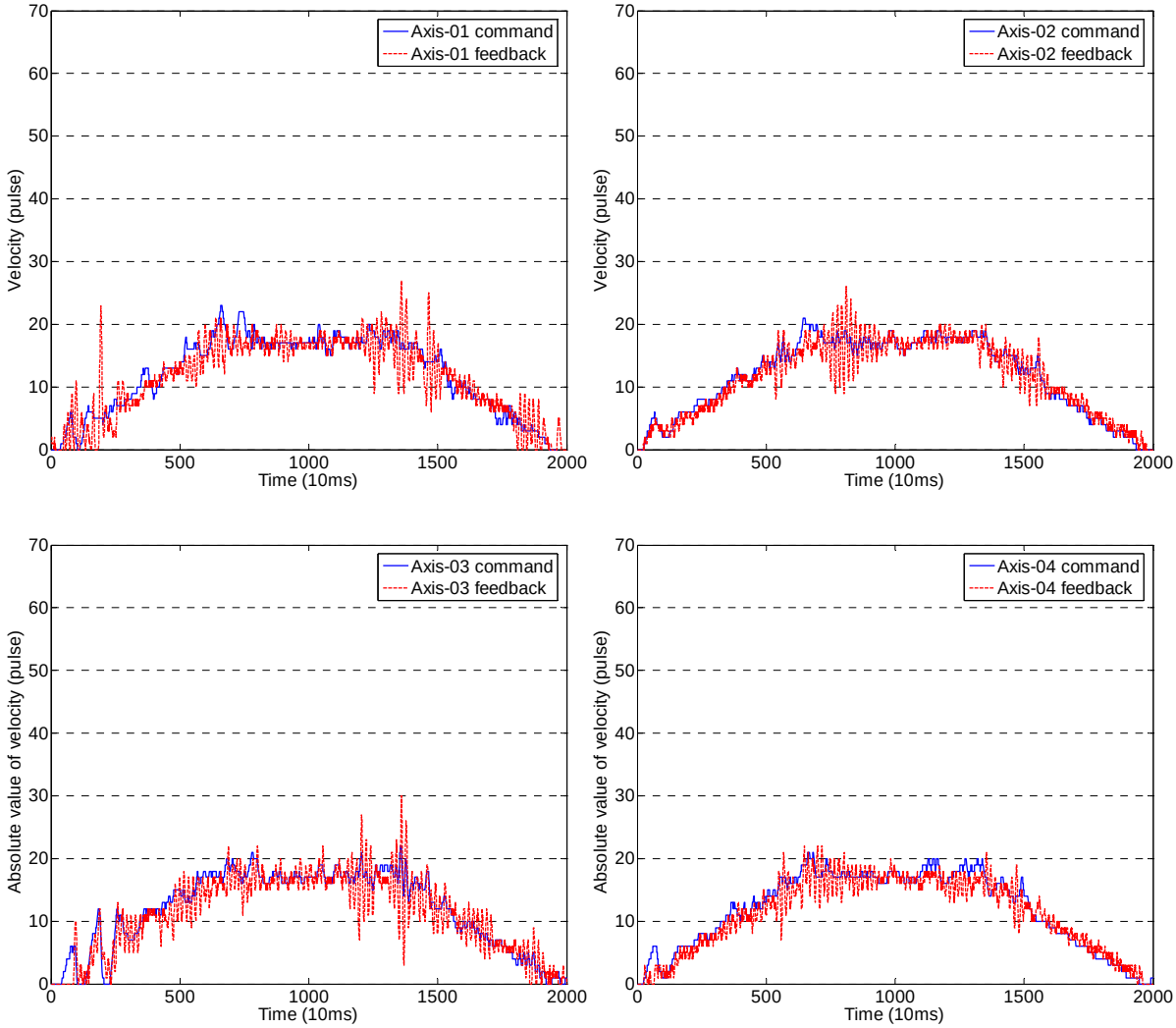


圖 4-24 125kbps 未加入時脈同步下，各軸的速度響應

表 4-3 未加入時脈同步下，平台位置的 IAE 值(cm)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	13612	11207	11456	11043	11829.5
Y axis	2091	2673	2514	2139.4	2354.35
Contour	13702	11549	11758	11263	12068

表 4-4 未加入時脈同步下，各個傳輸速率的 IAE 值(pulse)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	3174.5	3338	2659	2789.5	2990.25
Axis-02	2453	2406	2186	2292.5	2334.375
Axis-03	3655.5	4046	3343	3629	3668.375
Axis-04	2604.5	2487	2318	2240	2412.375
Ave. of each speed	2971.9	3069.3	2626.5	2737.8	

表 4-5 未加入時脈同步下，平台位置的標準差(cm)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	7.633	6.21	6.3	6.07	6.553
Y axis	1.092	1.432	1.349	1.151	1.256
Contour	7.652	6.370	6.451	6.177	6.663

表 4-6 未加入時脈同步下，各個傳輸速率的標準差(pulse)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	2.288	2.367	1.826	2.057	2.134
Axis-02	1.712	1.800	1.545	1.710	1.692
Axis-03	2.450	2.676	2.155	2.382	2.416
Axis-04	1.819	1.771	1.709	1.630	1.732
Ave. of each speed	2.067	2.154	1.809	1.945	

在上述的實驗結果中，系統受到網路延遲和時脈漂移的影響，導致效能的下降，然而，根據第三章的探討，本論文所提出之時脈同步方法，可將兩項因素的影響降低，因此，在原先的各軸節點中，加入時脈同步方法，將結果與上述的控制結果相比，圖 4-25 和圖 4-26 為 1000kbps 的實驗結果，在位置響應部份，可看出震盪現象大幅減少，各軸速度響應部份，也獲得較好的控制效能，至於圖上固定週期所出現的脈衝現象，是因為強制時脈同步，使得各軸的取樣時間縮短，所得到的迴授量跟著減少，導致如此的現象，然而，整體控制效能仍獲得了改善；圖 4-27 和圖 4-28 則為 125kbps 實驗結果，由圖上可看出，各軸速度響應的震盪相對的降低，但平台位置響應仍然存在相當程度的抖動。

若將各個速率下的 IAE 和標準差加以統計，如表 4-7~表 4-10 所示，其改善率則如表 4-11~表 4-14 所示，在平台位置部份，除了 125kbps 之外，各個速率的位置控制皆得到大幅改善；在各軸速度響應部份，由平均值可知，各個傳輸速率的控制效能皆獲得提昇，其中 250kbps、500kbps 和 1000kbps，由於網路因素的影響被消除，因此，效能和傳輸速率並無絕對關係，至於 125kbps，本身傳輸延遲大，加上節點競爭網路使用的情況，使得對於時脈漂移的容忍度下降，導致同步方法的改善效果下降，以單軸而言，雖獲得相當改善，但因為四軸匹配度下降，導致平台位置控制效能降低。

因此，以各軸速度控制而言，時脈同步方法可有效減少因網路所造成之震盪現象，整體而言，可提昇其精密度達 16%左右；在平台位置控制上，由於各軸同步性增加，同樣可達到大幅的改善，但此處必須考量各軸匹配度，若匹配性降低，將造成控制效能下降，平均而言，在平台位置響應上，時脈同步方法可提昇其效能約 22%左右。

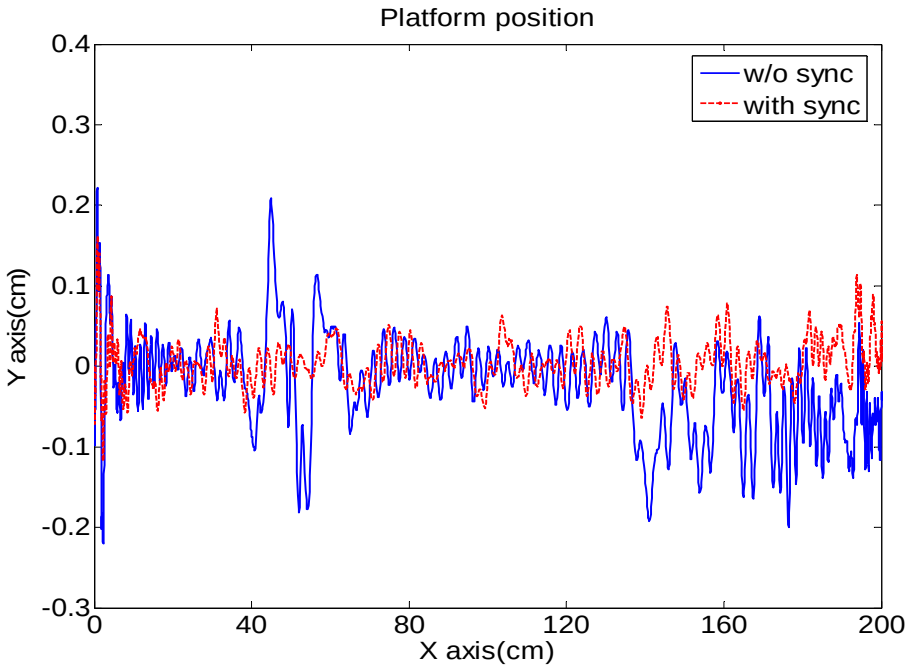


圖 4-25 1000kbps 加入時脈同步前後，平台位置響應比較

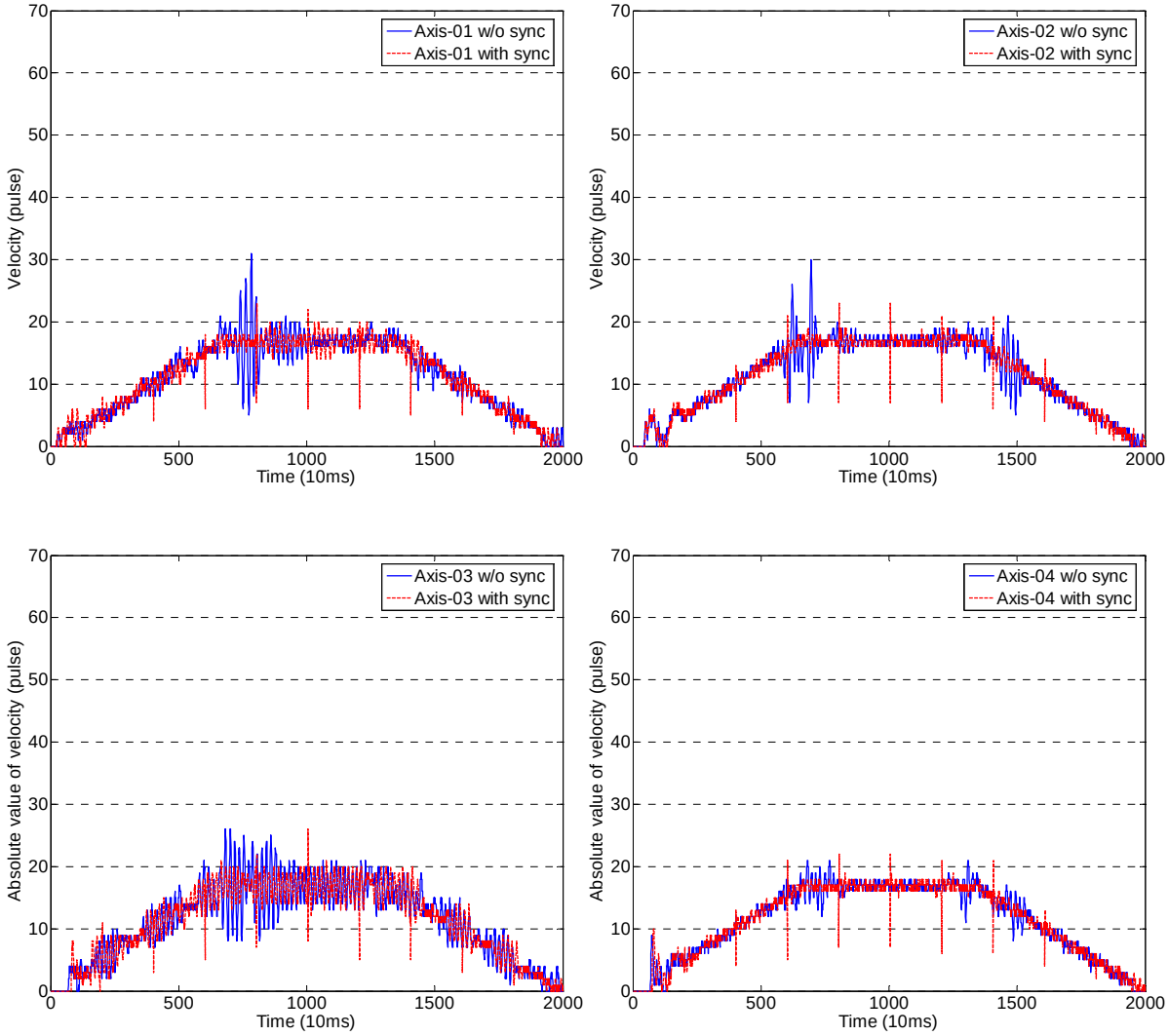


圖 4-26 1000kbps 加入時脈同步前後，各軸速度響應的比較

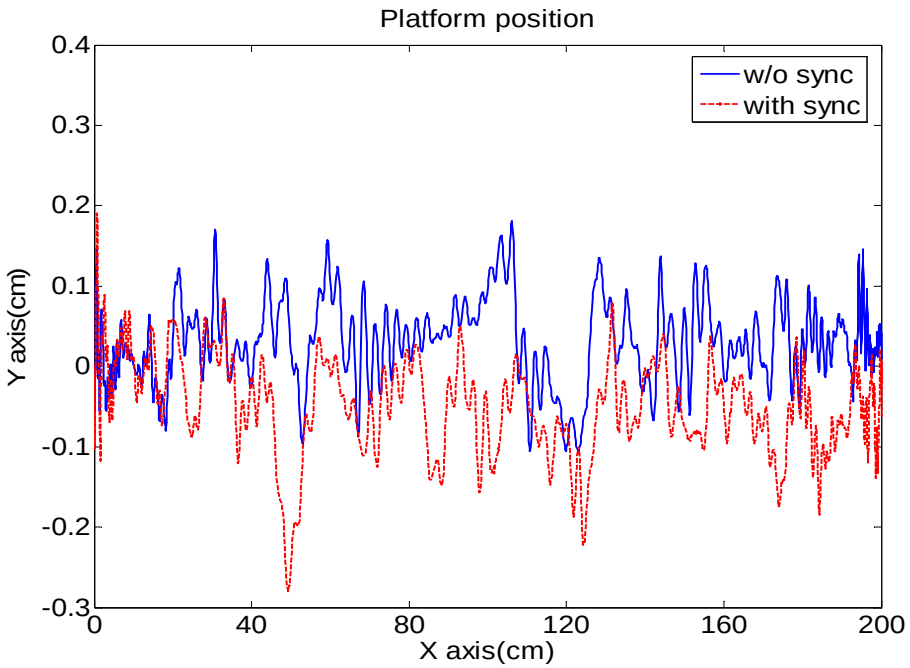


圖 4-27 125kbps 加入時脈同步前後，平台位置響應比較

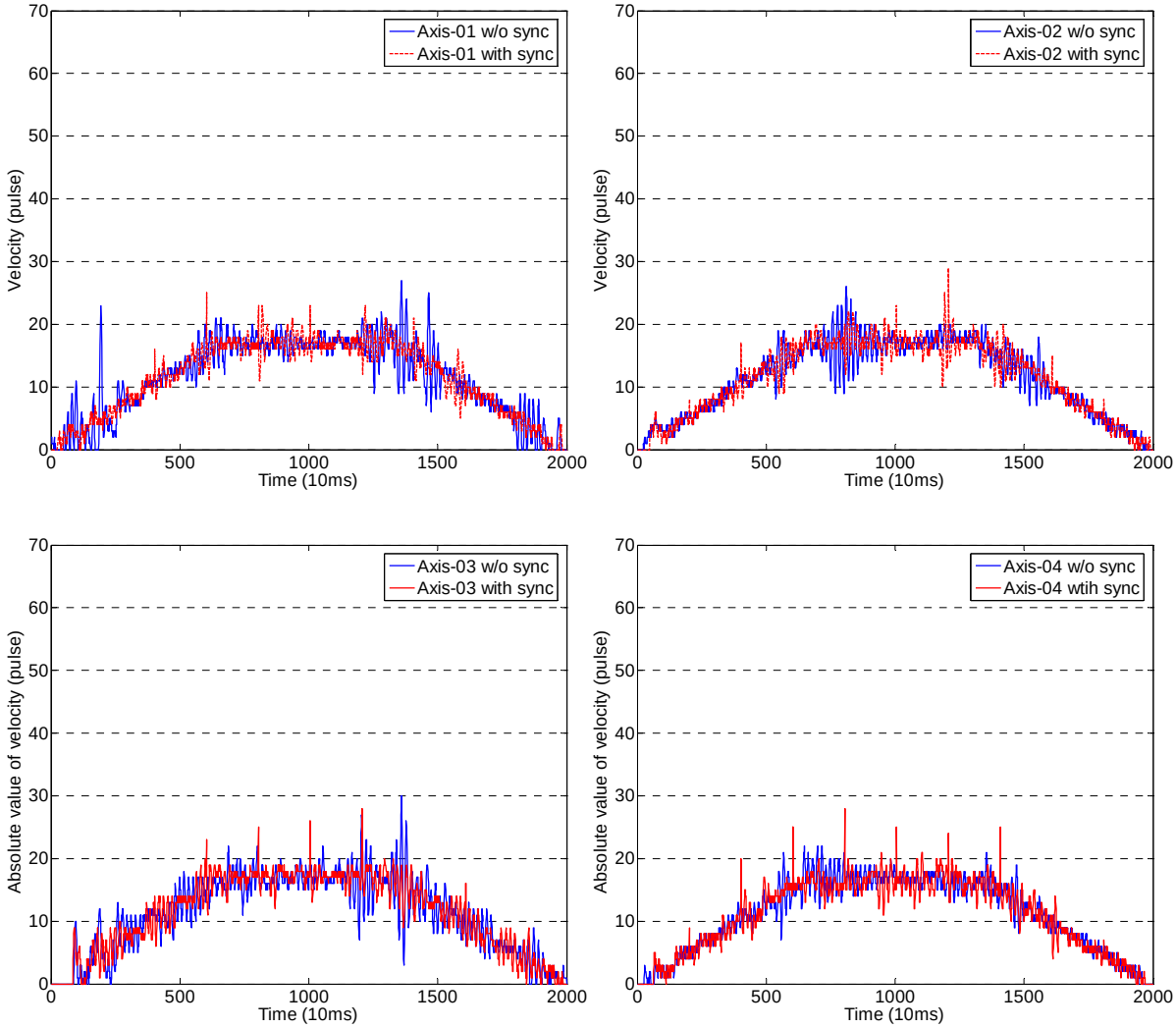


圖 4-28 125kbps 加入時脈同步前後，各軸速度響應的比較

表 4-7 加入時脈同步後，平台位置的 IAE 值(cm)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	13919	8448.7	7850.5	6741.1	9239.825
Y axis	2041.9	1752.4	1617.3	1307.9	1679.875
Contour	14419	8636.5	8035.2	6879.6	9492.575

表 4-8 加入時脈同步後，各個傳輸速率的 IAE 值(pulse)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	2278	2107	2111.3	2240	2184.075
Axis-02	2354	2084	2063	1891	2098
Axis-03	3021.7	2609.7	2600	3058	2822.35
Axis-04	2791.5	2071	2180.3	2043.7	2271.625
Ave. of each speed	2611.3	2217.9	2238.7	2308.2	

表 4-9 加入時脈同步後，平台位置的標準差(cm)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	7.755	4.585	4.384	3.726	5.113
Y axis	1.244	0.933	0.864	0.718	0.94
Contour	8.044	4.679	4.470	3.798	5.248

表 4-10 加入時脈同步後，各個傳輸速率的標準差(pulse)

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	1.634	1.460	1.462	1.501	1.514
Axis-02	1.663	1.458	1.420	1.380	1.48
Axis-03	2.004	1.794	1.788	1.954	1.885
Axis-04	1.927	1.488	1.563	1.566	1.636
Ave. of each speed	1.807	1.550	1.558	1.600	

表 4-11 各個傳輸速率下，平台位置 IAE 改善率

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	-2.3%	24.6%	31.5%	39.0%	23.196%
Y axis	2.3%	34.4%	35.7%	38.9%	27.831%
Contour	-5.2%	25.2%	31.7%	38.9%	**22.642%**

表 4-12 各個傳輸速率下，各軸速度 IAE 改善率

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	28.2%	36.9%	20.6%	19.7%	**26.354%**
Axis-02	4.0%	13.4%	5.6%	17.5%	10.140%
Axis-03	17.3%	35.5%	22.2%	15.7%	**22.699%**
Axis-04	-7.2%	16.7%	5.9%	8.8%	6.063%
Ave. of each speed	10.6%	**25.6%**	13.6%	15.4%	

IAE 整體平均改善率=16.31%

表 4-13 各個傳輸速率下，平台位置標準差改善率

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Average
X axis	-1.6%	26.2%	30.4%	38.6%	23.40%
Y axis	-13.9%	34.8%	36.0%	37.6%	23.62%
Contour	-5.1%	26.5%	30.7%	38.5%	**22.66%**

表 4-14 各個傳輸速率下，各軸速度標準差改善率

Bus speed / Axis	125 kbps	250 kbps	500 kbps	1000 kbps	Ave. of each axis
Axis-01	28.6%	38.3%	20.0%	27.1%	**28.474%**
Axis-02	2.9%	19.0%	8.1%	19.3%	12.335%
Axis-03	18.2%	32.9%	17.0%	18.0%	**21.533%**
Axis-04	-6.0%	16.0%	8.6%	3.9%	5.609%
Ave. of each speed	10.92%	**26.57%**	13.41%	17.06%	

標準差整體平均改善率=16.99%

4-4-3 小結

本節中，將所設計之 CAN 應用協定，運用在分散式平台系統架構之中，透過網路架構將平台系統的監控加以整合，包含各軸速度命令、速度迴授以及相關的控制參數等等，皆由所設計的應用協定達成，並搭配 PC 端介面，作為系統的監控介面。

在節點時脈同步上，在未加入時脈同步前，各軸速度迴授易由於資料遺失現象，產生震盪的情形，加入同步方法後，將各軸的時脈進行校正，減少資料遺失率，大部分的震盪情形獲得了改善，但由於時脈校正的動作，迫使各軸的取樣週期縮短，導致迴授量減少，使得迴授出現短暫的脈衝現象，但整體而言，時脈同步方法仍可有效提昇各軸的精密度和穩定度，整體可得到約 16%的改善效果，且由於時脈同步方法的加入，各軸同步性增加，使得平台整體位置控制效能約提昇 22%。

第五章　結論與未來發展

5-1 結論

本論文以 CAN 網路系統為探討的主題，透過 CAN 網路的分析實驗，了解其傳輸特性與系統架構之間的關係，接著，沿用 CANopen 的 CAN 應用層設計概念，規劃 CAN 應用層協定，以及網路時脈同步機制的設計。最後，於四輪全向平台系統，實現所設計之應用協定與同步方法，並進行相關的驗證。在此，歸納出以下結論：

1. 在 CAN 網路特性分析上，本論文以四項實驗參數：(1)網路傳輸速率、(2)資料封包大小、(3)系統取樣時間，以及(4)傳輸資料量進行傳輸實驗，根據實驗結果，歸納出影響網路資料遺失率的兩個主要因素：網路延遲與時脈漂移現象。以系統取樣週期來說，可分為兩種情形：

   (a) 取樣週期較大者，對於節點間的時脈同步性，具有較大容忍度，時脈漂移現象不易被凸顯，因此，資料遺失率主要受到網路傳輸延遲所影響。

   (b) 取樣週期較小者，相對於傳輸延遲，時脈漂移將造成節點間的時脈差異，超過一個取樣週期以上，導致節點持續接收到非該取樣時間的資料，導致極高的資料遺失情形。

   此外，以 N_{drift} 定義時脈漂移現象對於系統產生影響的時間點，N_{drift} 越小者，表示時脈漂移現象影響越早發生。

2. 在 CAN 應用層協定的設計上，本論文以 CANopen 的概念為基礎，主要設計的考量為資料傳遞機制、網路管理以及網路時脈同步三個部份。

   (a) 在資料傳遞方面，除了建立點對點資料傳輸與參數設定的通訊方式，並設計了廣播傳輸的傳輸方式，以同時傳輸多點資料，避免網路傳輸造成

節點動作的不一致。

(b) 在網路管理方面，設計了 Master/Slave 架構，將各個節點的運作經由 Master 加以管理，並利用遙控欄框的方式監控節點，由以上兩點便可將系統各個環節透過網路整合。

(c) 在時脈同步方面，根據先前的網路分析實驗結果，透過原有的網路封包搭配外部中斷的方式，達到強制同步的效果，並經由實驗加以驗證完成。

綜合以上三點，規劃適用於多點資料傳輸網路協定，以及網路時脈同步機制，其中所設計之協定針對主要的傳輸架構，可用較精簡的方式實現於微控器上，而同步機制則可將時脈漂移現象加以修正，使得網路資料遺失量大幅降低。

3. 全向平台網路系統的實現上，透過 DSP F2812 和 8051 作為各個網路節點的運算核心，完成移動平台控制架構，並將所設計之 CAN 應用層通訊協定實現於平台系統的控制網路，透過網路將控制系統加以整合，並在完成的全向平台網路系統下，驗證所設計之網路時脈同步機制，使得各軸速度響應的精密度提昇 16.31%，並增加各軸同步性，使平台位置控制效能約提昇 22%。

5-2 未來發展

1. 本文的討論主要以網路架構的方式，對於網路系統的效能加以改善，未來朝向控制器或補償器設計的方式，針對網路效能進行改善。
2. 目前架構主要以單一網路的探討為主，未來可朝向多種網路的整合發展，加入無線網路的應用，探討多種網路效應對於系統之影響，並達到遠端平台監控的目的，

參考文獻

[1] J. D. Decotignie, “Ethernet-Based Real-Time and Industrial Communications,” *Proceedings of the IEEE*, Volume 93, Issue 6, pp.1102–1117, June 2005.

[2] J. Y. Yu, S. M. Yu, and H. Q. Wang, “Survey on the Performance Analysis of Networked Control Systems,” *2004 IEEE International Conference on Systems, Man and Cybernetics*, Volume 6, pp.5068–5073, 10-13 Oct. 2004.

[3] D. S. Kim, “A Scheduling Method for Networked Control Systems in Fieldbus Environments,” Ph.D Dissertation, Seoul National University, Feburary 2003

[4] J. Nilson, B. Bermhardsson, and Wittermark, “Schtochastic Analysis of Control of Real Time Systems with Random Time Delays,” *Automatica*, Volume 34, Issue 1, pp.57-64, B.1998.

[5] P. Marti, G. Fohler, K. Ramamritham, and J. M. Fuertes, “Jitter Compensation for Real-Time Control Systems,” *22nd IEEE Real-Time Sysfems Symposium*, London, UK, December, 2001

[6] 廖建龍,“以CAN Bus為基礎的分散式即時伺服馬達控制器之設計與實做,” 國立交通大學,碩士論文,中華民國 89 年

[7] W. Zhang, M. S. Branicky, and S. M. Philips, “Stability of Networked Control Systems,” *IEEE Control Systems Magazine*, Volume 21, Issue 1, pp.84-99, February 2001.

[8] W. Zhang, “Stability Analysis of Networked Control Systems,” PhD Thesis, Case Westem Reserve University, 2001.

[9] F. L. Lian,” Analysis, Design, Modeling, and Control of Networked Control Systems.” Ph.D thesis, University of Michigan, May 2001.

[10] F. L. Lian, J. R. Moyne, and D. M. Tilbury, “Network Design Consideration for Distributed Control Systems,” *IEEE Transaction on Control System Technology*, Volume 10, Issue 2, pp.297-307. March 2002

[11] S. H. Hong, “Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems,” *IEEE Transaction on Control System Technology*, Volume 3, Issue 2, pp.225-231, Jun 1995.

[12] X. D. Ren, S. B. Li, Z. Wang, M. Z. Yuan, and Y. X. Sun, “A QoS Management Scheme for Paralleled Networked Control Systems with CAN Bus,” *The 29th Annual Conference of the IEEE Industrial Electronics Society*, Volume 1, pp.842-847, 2-6 Nov. 2003.

[13] F. L. Lian, J. R. Moyne, and D. M. Tilbury, “Performance Evaluation of Control Networks: Ethernet, ControlNet, and DeviceNet,” *IEEE Control Systems Magazine*, Volume 21, Issue 1, pp.66–83, Feb. 2001.

[14] J. A. Fonseca and L. M. Almeida, “Using a Planning Scheduler in the CAN Network,” *Emerging Technologies and Factory Automation*, Volume 2, pp. 815-821, Oct. 1999.

[15] *CANopen Application Layer and Communication Profile*, CiA DS301, Version 4.02, Feb. 13, 2002.

[16] Open Device Net Vendors Association, http://www.odva.org/default.aspx

[17] H. Ekiz, A. Kutlu, and E. T. Powner, “Design and Implementation of a CAN/CAN bridge,” *Second International Symposium on Parallel Architectures, Algorithms, and Networks*, pp.507–513, 12-14 June 1996.

[18] F. Moraes, A. Amory, N. Calazans, E. Bezerra, and J. Petrini, “Using the CAN

Protocol and Reconfigurable Computing Technology for Web-Based Smart House Automation," *14th Symposium on Integrated Circuits and Systems Design*, pp.38–43, 10-15 Sept. 2001.

[19] F. G. Pin and S. M. Killough, "A New Family of Omnidirectional and Holonomic Wheeled Platforms for Mobile Robots," *IEEE Transactions on Robotics and Automation*, Volume 10, No. 4, pp.480-489, 1994.

[20] H. Asama, M. Sato, L. Bogoni, H. Kaetsu, A. Matsumoto, and I. Endo, "Development of an Omni-Directional Mobile Robot with 3 DOF Decoupling Drive Mechanism," *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Aichi, Japan, pp.1925-1930, 1995.

[21] K. Watanabe, "Control of an Omnidirectional Mobile Robot," *Proceedings of the 1998 IEEE Second International Conference on Knowledge-Based Intelligent Electronic Systems*, Adelaide, Australia, pp.51-60, 1998.

[22] M. Ashmore and N. Barnes, "Omni-Drive Robot Motion on Curved Paths：The Fastest Path Between Two Points is not A Straight-Line," *Australian Joint Conference on Artificial Intelligence*, Australia, pp.225-236, 2002.

[23] "CAN Specification Version 2.0", *BOSCH*, 1991.

[24] W. Lawrenz, *CAN System Engineering: From Theory to Practical Applications*, New York: Spring-Verlag, 1997.

[25] 莊孝麟, "全向性移動平台之精密運動控制設計," 國立交通大學,碩士論文, 中華民國 95 年

[26] 鄭景文, "動態網路控制系統之時間延遲分析," 國立交通大學,碩士論文,中華民國 95 年

[27] CAN in Automation, http://www.can-cia.org/

[28] Embedded Systems Academy, http://www.esacademy.com/index.htm