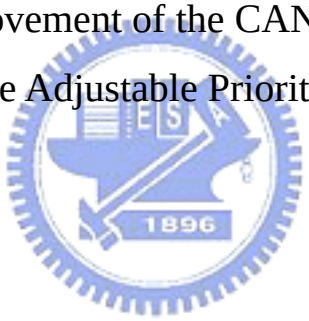
國立交通大學

電機與控制工程研究所

碩士論文

以硬體實現主動可調式CAN網路排程系統

Realization and Improvement of the CAN Bus Transmission via the Active Adjustable Priority Scheme



研　究　生：薛伊婷

指導教授：徐保羅　博士

中華民國九十四年六月

以硬體實現主動可調式 CAN 網路排程系統

研究生：薛伊婷　　指導教授：徐保羅 博士

國立交通大學電機與控制工程研究所

摘要

在本論文中分別使用 Borland C++ Builder 建立 CAN（Controller Area Network）網路系統的模擬軟體，並建立 8051 與 SJA1000 Standard CAN Controller 組成多組 CAN 節點的硬體實驗系統，實驗依據三項傳輸指標：平均傳輸時間、未即時傳遞訊息比例以及訊息遺失比例，在不同的負載量下，分析不同排程理論的優劣。

因為 Rate Monotonic scheme 與 Deadline Monotonic scheme 無法在訊息傳輸的過程中更改訊息的優先權，使得優先權低的訊息在高負載量時無法即時傳送，所以本研究希望在訊息傳送的過程中，可以即時更改訊息的優先權以提升傳輸效率，文中使用動態調整訊息的理論有 Earliest Deadline First scheme 以及本論文研發出的 Active Adjustable Priority scheme。EDF 是訊息快要錯過 deadline 的時候，即時調整訊息的優先權，使得訊息可以快速送出。而 AAP 是累積訊息一段時間的傳輸狀況後逐漸調整優先權。EDF 因為硬體 SJA1000 的設計限制，無法在硬體上實現，而 AAP 可以在硬體上實現。

對於在硬體實驗上顯示對於 missing deadline percentage 來說，DM＋AAP 有很顯著的效果。而 lost message percentage 則以 RM＋AAP 的效果

最好。對於網路控制系統而言，如果訊息是作為回饋控制，未能及時傳遞時，會造成系統不穩定，在這種系統架構下選用 DM＋AAP 最能達成系統穩定。而訊息若是作為控制命令，控制命令如果遺失，系統無法做出回應動作造成系統有錯誤發生，因此選用 RM＋AAP 可以使系統發生錯誤的機會最少。

Realization and Improvement of the CAN Bus Transmission via the Active Adjustable Priority Scheme

Student: Yi-Ting Hsueh Advisor: Prof. Pau-Lo Hsu

Department of Electrical and Control Engineering
National Chiao-Tung University

ABSTRACT

By using the Borland C++ Builder software, a controller area network (CAN) bus simulation system is developed in this thesis. Furthermore, results of a hardware system with different bus loads and nine CAN nodes, which combine 8051 with SJA1000 standard-alone CAN controller, are compared with simulation results in three indices: (1) the average transmitting time, (2) the missing deadline percentage, and (3) the lost message percentage.

Simulation and experimental results indicate that the rate monotonic (RM) scheme and the deadline monotonic (DM) scheme, which are formed as the fixed message priority during the period of transmission, successfully transmit the higher priority messages only in time. Therefore, to develop real-time changing priority of CAN messages in this thesis leads to two adjustable schemes: (1) earliest deadline first scheme (EDF) and (2) the proposed active adjustable priority (AAP) scheme. Theoretically, as messages are going to miss their deadlines, the EDF changes their priorities immediately while the proposed AAP gradually changes its priority according to performance of message transmission. Physically, the EDF cannot be implemented on real hardware system and the AAP is adopted in the present experiments.

Experimental results also indicate that the DM+AAP presents the best performance in the missing deadline percentage and the RM+AAP presents the best performance in the lost message percentage. For network control systems, the deadline for the sensors to provide the measurement is critical for performance and stability in feedback control. Therefore, the DM+AAP scheme is more suitable as bus message load increases. On the other hand, as messages of commands for each CAN node is more important, like in the synchronization for multi-axis systems, the lost message is become more critical in contributing the errors. Thus, adopting the RM+AAP scheme is recommended.

誌謝

碩士論文的完成，首先要感謝保羅老師的耐心指導與包容，特別是在研究的過程中，對於研究方向的選擇，給予相當的耐心以及支持。

感謝安平學長對於我的研究提供了很多的方向，以及鎮洲學長幫助我解決了很多研究上的問題。還有研究室俊賢學長，琮政學長以及我的同學政宏和政衍，學弟妹尚玲、景文、興漢、瑞原、宗翰、元銘、孝麟的幫助使我能夠完成碩士學業。而我的室友慈育、億如、依璇在生活上給予我很多的關懷。

最後將此論文獻給我的父母薛政雄先生以及郭淑麥女士以及我的姊妹博勻、蓓芳、詩蓉、舒珮，因為有他們的支持與關懷，我才有勇氣與信心繼續前進。

目錄

圖目錄

表目錄

第一章 緒論

1.1 前言

CAN 是在 1990 年由德國 Robert Bosch 公司所訂定的一個網路協定 [2-4]。訂此協定的目的是在以一種新的數位網路技術，取代原先在汽車內複雜且昂貴的硬體接線。CAN 原先是應用於連接汽車內防鎖死剎車系統 (anti-lock brake system, ABS)或是引擎控制單元等的，但由於它提供了快速且可靠的連線反應，適用在即時(real-time)系統上，並且費用低廉，故也被用於許多其他的控制場合。CAN 並已成為國際標準規格(ISO11898, ISO11519) [1]。應用 CAN 於場域匯流排的商品已有 Honeywell 公司的 SDS，Allen-Bradley 公司的 DeviceNet [3]，與 CANopen [23]。

由於車用系統越來越複雜，所需要傳遞的訊息也越來越多，而 CAN 取代了複雜的硬體接線，所有的訊息都在 CAN 上傳遞。因此如何妥善安排訊息，使訊息的傳送能夠達到系統的需求，達到系統安全穩定為本文的重點。因此本文並不考慮 CAN 的其他機制，如傳輸錯誤的處理機制，只針對如何透過排程理論來安排訊息的傳遞，來討論如何提升訊息傳輸的效率以及即時性要求，。排程理論有很久的歷史，常用於工廠流程與管理方面、網路或是嵌入式作業系統。在本文中由於 CAN 的有限的頻寬，使用排程理論希望使訊息的傳送能達到最好的傳輸狀況。

1.2 文獻回顧

排程理論用於工廠流程或電腦系統排程有很多的理論，也有很長的歷史。1990 年德國 Robert Bosch 公司所訂定 CAN 網路協定，因為 CAN 網路有 1Mbps 頻寬的限制，所以之後有許多文獻提出改善 CAN 網路效能的方式。

文獻中提及的排程理論分成兩種類型，第一種是 Almeida, et al.提出 planning scheduler [17-19] 。planning scheduler 是將所要傳送的訊息安排成一個訊息表來傳送，以 on-line 的 scheduler 來調整訊息表，文中提出三種調整訊息表的方式: static scheduler、planning scheduler 以及 dynamic scheduler。

static scheduler是先將訊息表安排好，在整個系統過程中不做任何改變。此種方法雖然彈性很差，但能夠有效減少因為改變訊息表所造成的負載加重，如果系統很穩定，不需要變動，是一個很有效的方法。dynamic scheduler是在傳送過程中，可任意改變訊息表，這種方法很有彈性，但是為了改變訊息表會增加BUS的overhead。planning scheduler是一個折衷的方法，它不是隨時在改變table，但如果系統效能不符合設定，可透過改變訊息表來改善。此三種方法的特性如表1.1所示。

表 1.1 static、dynamic 和 planning schedulers 的比較 [17]

scheduler	static	dynamic	planning
operational flexibility	low	high	medium*
run-time overhead	low	high	medium*
schedule size	potentially huge	negligible	bounded*
schedulability	guaranteed	not guaranteed	guaranteed each plan

* Depends on the plan duration

第二類的排程理論:訊息的傳遞是以訊息為單位，提出此方法的是 Zuberl, and Shin(1995)提出非強迫式的排程方法 DM，EDF 以及 MTS(DM 與 EDF 的結合) [12-14]，文中以軟體模擬 computer-integrated manufacturing (CIM)系統的訊號傳輸比較 DM、EDF 以及 MTS 的傳輸效能。其他研究 EDF 的文獻如[15-16]。

1.3 問題陳述

由於 CAN 網路將車用線路簡化，改善車用線路的複雜度，但車用訊號只透過 CAN 線路來傳送，雖然線路簡化，但車上大量的訊息資料並沒有減少，而 CAN 的頻寬有限(1Mbits/s)，如何在有限的頻寬來傳送車上大量的訊息，是本論文的重點。參考文獻中提到多種排程方法，本論文中除了建立軟體模擬 CAN 的運作，模擬排程方法，並將排程方法實際實現在硬體中。

在本論文中首先使用的方法為 rate monotonic (RM) scheme 來改善傳輸效能，但使用 RM scheme 是依據訊息的週期長短來安排優先權，一旦訊息的數量超過頻寬負載，優先權低的訊息會傳送不出去。因為在傳送的過程中沒有彈性，所有訊息的優先權都是固定的。RM scheme 希望即使超過頻寬的負載時，能夠彈性的調整訊息的優先權，以利於訊息的送出。

為了能夠在訊息傳送時，彈性的調整訊息的優先權，因此提出 earliest deadline first (EDF) scheme 在傳輸進行的過程中，如果訊息快要錯過 deadline，則調整訊息的優先權，使訊息能順利傳輸。EDF 在硬體的實現上有困難，因為在硬體上訊息開始傳送之後，就無法更改優先權，因此本文提出 active adjustable priority (AAP) scheme。AAP 更改優先權的方式和 EDF 不同，AAP 是觀察訊息傳輸狀況一段時間後，依訊息的傳輸表現，更改訊息的優先權。因為 AAP 不在線上即時更改訊息的優先權，而是在線上逐漸更改其優先權，因此 AAP 在硬體上是可實現的。

1.4 研究方法

為了研究排程理論對於改善網路傳輸的效果，建立評估 CAN 網路效能的三項指標:平均傳輸時間、未即時傳遞訊息比例、遺失訊息比例，利用此三項指標配合硬體以及軟體實驗，來比較排程理論的優劣。此三項傳輸指標與文獻中所運用的指標有所不同，在文獻〔12-14〕運用指標 workload，比較各式排程理論。此方式是觀察各種排程理論“可處理多少訊息量”，使系統還能正常運作。

而本論文利用三項指標來評估傳輸效能的原因，是希望能夠觀察在實際應用的狀況下訊息的傳輸情形。平均傳輸時間觀察訊息的平均傳輸時間，瞭解訊息從傳輸端開始傳送到接收端實際接收到訊息的時間，可幫助系統規劃設計。制訂未即時傳遞訊息比例此項指標，是因為在控制架構下，有些命令必須在 deadline 之前送到，不然系統會有錯誤的情形發生。而遺失訊息比例是為了瞭解瞭解各種排程方法處理時，訊息未送出的比例。

由於三項指標在實際應用系統的重要性，在本文中不論是在模擬或硬體實驗都以三項指標為分析的依據。本文以 RM，DM 為主要的排程理論，但由於 RM 在高負載量時，低優先權的訊息會有一直傳送不出去的現象，所以提出能夠與其互補設計概念的 AAP。AAP 與 EDF 同樣都能在系統傳輸過程中動態的調整訊息的優先權，但由於 EDF 在硬體上有實現的困難，只能在模擬實驗中實現，所以在本文中提出 AAP 的概念與 RM 搭配為論文的主軸。

因為上述的原因，在本論文中分成兩個部份:軟體模擬以及硬體實驗。在軟體模擬中，比較 RM，DM，EDF，RM+EDF，DM+EDF，AAP，RM+AAP 以及 DM+AAP 在各項傳輸指標的表現。由於 EDF 無法在硬體中實現，所

以在硬體實驗中，比較 RM，DM，AAP，RM+AAP 以及 DM+AAP 在各項傳輸指標的表現。

1.5 論文架構

本論文分成六章。第一章是緒論在說明論文的研究動機與目的，問題陳述以及研究方法。第二章介紹本文所採用的網路協定-CAN，並討論 TTCAN 與 CAN 的不同。第三章是將所利用的排程理論做一個詳細的介紹。第四章介紹模擬軟體的架構與模擬結果分析。第五章硬體實驗結果與分析。第六章是結論針對硬體與軟體實驗的結果做比較。

第二章 Controller Area Network（CAN）

2.1 CAN 網路協定介紹

CAN 是在 1990 年由德國 Robert Bosch 公司所訂定的一個網路協定。當初訂此協定的目的是在以一種新的數位網路技術，取代原先在汽車內複雜且昂貴的硬體接線。CAN 原先是應用於連接汽車內防鎖死剎車系統(anti-lock brake system, ABS)或是引擎控制單元等的，但由於它提供了快速且可靠的連線反應，適用在即時(real-time)系統上，並且費用低廉，故也被用於許多其他的控制場合。CAN 並已成為國際標準規格(ISO11898, ISO11519) [1]。應用 CAN 於場域匯流排的商品已有 Honeywell 公司的 SDS，Allen-Bradley 公司的 DeviceNet [3]，與 CANopen。

原始的 CAN 規格中只有針對相對於 OSI 架構中的資料連結層做規範，至於實際硬體如何完成連線，接收到的資料要作何處理等，都不在其規格範圍內。CAN 具有以下的特點：

- 資料訊息具有優先權(priority)
- 優先權的仲裁(arbitration)為非破壞性的(non-destructive)
- 可設定優先權與資料傳輸長度(1-8byte)
- 採用廣播的方式(multicast)，並藉所傳輸的訊息作時序同步的動作
- 任一節點皆可主動發送訊息(multimaster)
- 未傳送成功的訊息會自動重新傳送
- 錯誤檢查與錯誤頻繁的排除功能

以下將詳細地介紹 CAN 之架構與特性。2.2 節至 2.4 節介紹 Standard CAN。2.5 節至 2.7 節介紹 TTCAN。2.8 節介紹 CAN 與 TTCAN 的差別。

2.2 CAN 資料傳輸方式

在介紹 CAN 的傳輸格式前，有一些基本的設定必須要先了解：

- 傳輸速率

  協定中設定傳輸速率上限是 1Mbps，而實際的傳輸速率是依系統的要求而設定。

- 匯流排之值

  CAN 為數位傳輸的協定，數位傳輸的位準只有兩種，在 CAN 中規定為：主宰性的(dominant)與退讓性的(recessive)。在同一時刻，匯流排上只能有兩種位準之一存在，其規則是：當匯流排上同時有兩個節點分別送出主宰性與退讓性的位元時，只有主宰性的位元訊號會留下而成為匯流排上被測得的位準，如圖 2.1 所示。而在一般的實作中，是以電路上的 0 (low)代表主宰性的位準，而以 1 (high)代表退讓性的位準。

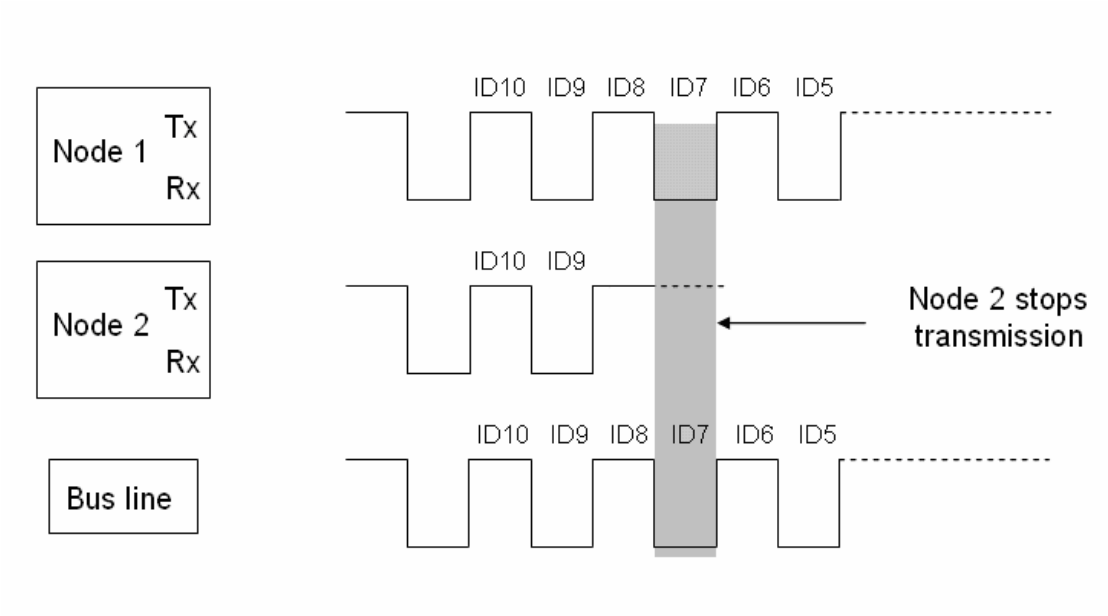


圖 2.1 位準仲裁圖

- 發送節點(transmitter)與接收節點(receiver)

  當有一節點在傳送資料時，此節點即稱之為發送節點，而其餘的全部節點都是接收節點。

- 發送節點對匯流排的偵測

  在網路中，發送節點在發送資料的同時，也需偵測匯流排上的值是否與其送出的相同。在 CAN 的網路傳輸中，共有四種不同的欄框，分別為資料欄框、遙控欄框、錯誤欄框與過載欄框。其中任兩個資料(遙控)欄框之間，或是資料欄框與遙控欄框之間，都需要以欄框間隔作分隔。以下分別詳述之。

2.3 資料欄框(Data Frame)

當節點有資料欲傳送出去時，需將資料封包在資料欄框內傳送至網路上。資料欄框為四種欄框中最長的一種，其格式如圖 2.2 所示。在說明欄框內各區段前，需先介紹 CAN 的填充/編碼(bit stuffing/coding)傳輸方式：當發送節點在傳送資料(遙控)欄框時，若是傳送出 5 個連續且相同位準的位元，則需在緊接著第 5 個位元之後加入一個相反位準的位元於要傳送的序列中。接收節點在接收時會自動地將這填充進來的位元刪除，以得到真正的資料。錯誤欄框或是過載欄框並不會經過位元填充的手續，而是以固定的格式發送。位元填充只適用於資料(遙控)欄框的下列區域：

- 欄框起始
- 仲裁區
- 控制區
- 資料區
- 循環多餘碼區
- 確認區
- 欄框終止

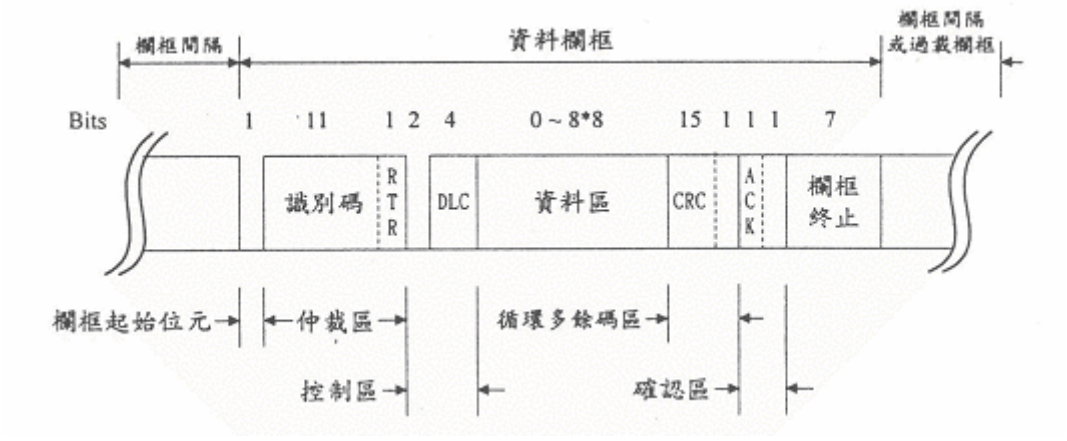


圖 2.2 CAN bus 之 Data frame

以下將分別介紹資料欄框中的七個區段：

- 欄框起始(Start of Frame)

  欄框起始為一單一的主宰性位元，標示一個新的資料框之開頭。只有在匯流排閒置時方可開始一個新的資料(遙控)欄框。

- 仲裁區(Arbitration Field)

  此區域可再分為識別碼(identifier)與要求遙控傳送位元(remote transmission request, RTR bit)兩部份。識別碼是用來識別各筆不同的資料，代表著與該筆資料相關的意義與在網路上競爭的優先順序，協定中並規定其前 7 位元不可皆為退讓性的位元。要求遙控傳送位元的目的是用來區別資料欄框與遙控欄框的，資料欄框設定在資料欄框中必須被設為主宰性的位元，而在遙控欄框中則要被設為退讓性的位元無資料。整個仲裁區域是用來判斷各節點存取匯流排的優先次序，其所採用的是非破壞性的二進位倒數(binary countdown)的方法：當有兩個以上的資料框同時要被送上匯流排時(當然此匯流排必須為閒置的狀態)，其整個欄框從欄框起始位元開始逐位元地傳送仲裁區內的位元逐次地送出，當送出退讓性位元的節點偵測到匯流排上之值為主宰性位元的時候，此節點便知道有其他優先權較高的節點也正在傳送，則此優先權較低的節點必須在下一位元送出之前便終止進行傳送，等待下一資的匯流排閒置。

  舉例來說，以 0 代表主宰性的位準，而 1 代表退讓性的位準的假設仲裁區只有 4 個位元長度，若有三個節點在同一時刻分別送出 0010、0011、1010 的仲裁區，在第 1 個位元的時刻，仲裁區為 1010 的節點便喪失了匯流排的存取權，再來第 2、第 3 位元的位準都相同，到了第 4 位元的時刻，0011 的節點也在仲裁中輸了，於是最

後只剩優先權最高的-仲裁區為 0010 的節點可進行傳送。

- 控制區(Control Field)

  此區有 6 個位元，前 2 位元為保留位元，必須為主宰性位元。後面 4 位元是資料長度碼(data length code)，用來標示在此資料欄框中包含了多少個位元組(byte)的資料。一個資料欄框的有效負載(payload)最多為 8 個位元組，故資料長度碼容許的範圍為 0000-1000。

- 資料區(Data Field)

  發送節點欲傳送的資料需放在此處，資料區的長度以位元組為單位，最少可傳送 0 個位元組，最多可為 8 個位元組。若是節點要傳送的資料超過 8 個位元組，則需發送另一個資料欄框。

- 循環多餘碼區(CRC Field)

  此區包含了循環多餘碼序列(cyclic redundancy check, CRC sequence)與循環多餘碼邊界(CRC delimiter)。循環多餘碼也稱為多項式碼(polynomial code)，而 CAN 所採用的產生多項式(generator polynomial)為

  $$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$$

  每一個 CAN 節點都有此一相同的產生多項式。當要傳送資料(遙控)欄框時，發送節點將從欄框起始到資料區結束之間的二進位位元序列(bit stream)視為一個係數只有 0 或 1 的多項式，並在其後附上長度為 15 位元的循環多餘碼序列，使得整個多項式(從欄框起始到循環多餘碼結束為止)能被產生多項式以 2 的餘數法(modulo-2)整除。也就是說，發送節點必須附上能使整個多項式被整除的循環多餘碼序列。循環多餘碼邊界為 1 個退讓性位元。

- 確認區(Acknowledgment Field)

確認區包含了 2 位元，分別為確認槽(ACK slot)與確認邊界(ACK delimiter)。這兩個位元在發送節點送出時都是送出退讓性的位元，匯流排上的接收節點在收到此欄框，並計算其送出的循環多餘碼正確無誤後，便會在確認槽之處發送出 1 主宰性位元，如此發送端可藉由偵測到確認槽的位準為主宰或的而得知網路上至少有一節點成功地接收到其所送出的欄框，不必準備重複傳送剛才傳送的欄框。

- 欄框終止(End of Frame)

  欄框終止由 7 個退讓性位元所組成，標示一個資料欄框(遙控欄框)的結束。

2.4 欄框間隔 (Interframe Spacing)

欄框間隔包括中斷欄框(intermission)與匯流排閒置(bus idle)，其用途是用以區隔資料欄框或是遙控欄框，以及提供節點時間處理資料。中斷欄框是由 3 個退讓性位元組成，在中斷欄框的過程中，全部的節點皆不可以嘗試傳送資料欄框和遙控欄框，而匯流排閒置是指匯流排上沒有任何節點正在進行傳送的動作。在中斷欄框結束之後(第 3 個退讓性位元結束時)，任一個有資料要傳送的節點都可以立即開始傳送，若是有兩個以上的節點都想要傳送，則進入仲裁的程序以決定可以傳輸資料的節點。若是在中斷欄框之後沒有節點要傳送，便進入匯流排閒置的狀態，在此狀態下，任一想傳送的節點都可隨時進行傳送。

2.5 時間觸發 CAN (Time Triggered CAN ; TTCAN)

一般 CAN 是使用事件觸發(event-triggered)，只有當有事件發生時才會傳送與時間不無相關性。時間觸發 CAN (time-triggered CAN； TTCAN)具有執行時間分配(time-slots)功能，因此當一次性傳輸模式準備就緒時，無論發生任何仲裁損失或錯誤的封包結構，資訊都必須被一次性傳輸。另外，封包起始點(SOF)輸出功能亦可幫助系統設計人員執行時間分配協定，或在 CAN 匯流排進行診斷以儘早判斷出匯流排資料流量以降低錯誤發生的情形。因為 TTCAN 的傳送與時間有相當密切的關係，所以 TTCAN 的架構與 CAN 有些許的不同，其不同的地方在於 TTCAN 能使所有的節點具有同步的機制。另外 TTCAN 的傳送與時間點有關係，也是因為此同步機制才能使每個節點上的訊息能夠在設定的時間點上送出而不會有訊息無法送出的狀況。以下的章節會詳細論述 TTCAN 的架構與傳輸方式以及 TTCAN 與 CAN 的比較。

2.6 TTCAN 的架構

TTCAN 的架構如圖 2.3 所示，TTCAN 比標準的 CAN 多了兩個功能區塊，觸發記憶體(Trigger Memory)和同步處理架構(Frame Synchronization Entity ; FSE)。Trigger Memory 是用來處理時間觸發的訊息傳送，它會從 message RAM 中取出所需的訊息，配合 FSE 在正確的時間點將訊息送出。

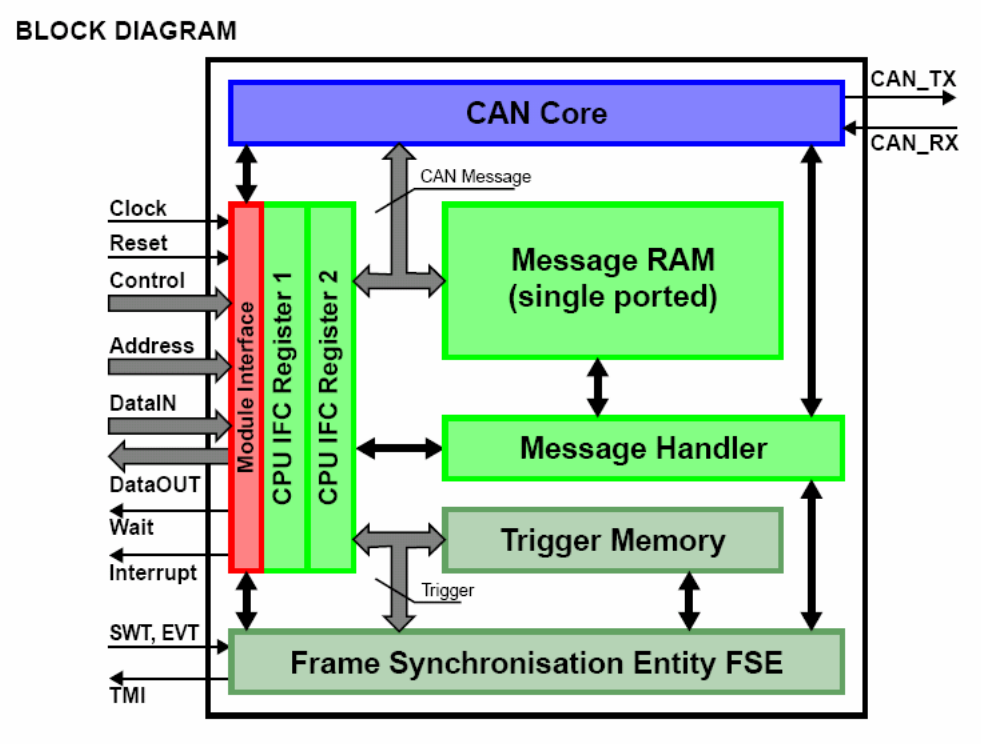


圖 2.3 TTCAN 的架構圖 [9]

FSE 是處理 TTCAN 的同步架構，它是一個狀態控制器(state machine)，控制 Time Triggered 訊息的溝通，FSE 的架構如圖 2.4 所示。FSE 可細分成以下六個功能區塊:

- TBB : Time Base Builder
- CTC : Cycle Time Controller
- TSO : Time Schedule Organizer
- MSA : Master State Administrator
- AOM : Application Operation Monitor
- GTU : Global Time Unit

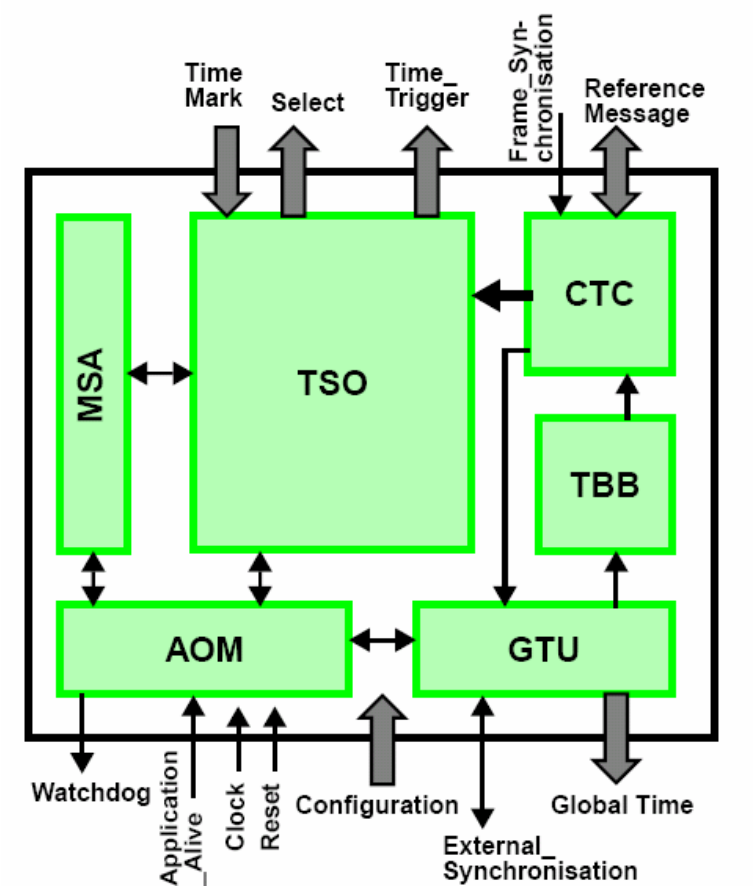


圖 2.4　FSE 的架構圖 [7]

2.7 TTCAN 的傳輸方式

在介紹 TTCAN 的傳輸之前，先介紹 TTCAN 的基本設定。

- 參考訊息(Reference Message)

  TTCAN 在開始傳輸之前，會由 master node 須先傳送一個 Reference Message，表示要開始傳送訊息了，接下來會連續傳輸數個訊息如圖 2.5 所示。

- 時間視窗(Time Windows)

  在兩個 Reference Message 中，區分成很多個 Time Windows，Time Windows 裡面放置等待傳送的訊息。Time Windows 分成 3 種: Exclusive Time Window 裡面放置週期性的訊息，Arbitrating Time Window 放置自發性的訊息(spontaneous messages) 和 Free Time Window 可傳送上面兩種訊息以及保留給其他的網路應用。

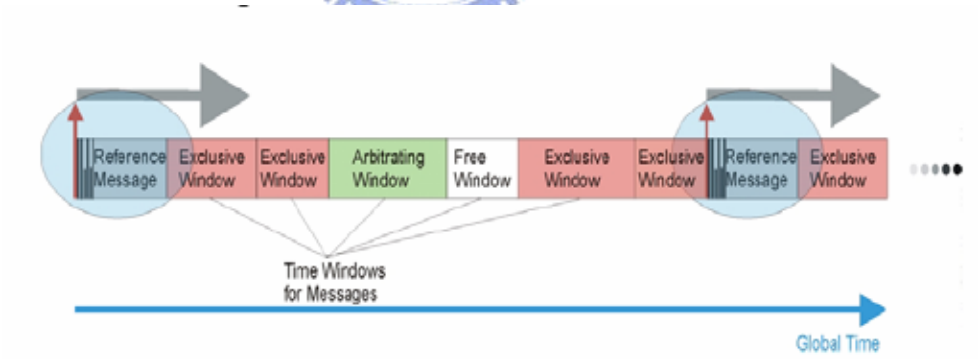


圖 2.5 TTCAN 的 Basic Cycle 傳輸架構 [7]

- 基礎循環(Basic Cycle)

  在兩個 Reference Message 的週期時間稱為 Basic Cycle。Basic Cycle 是 TTCAN 基礎傳輸單位，裡面包含很多個 Time Window 來傳輸資料。

- 系統矩陣(System Matrix)

  如果 Basic Cycle 不能滿足使用者的需求，TTCAN 允許將好幾個 Basic Cycle 組合成一個 System Matrix 來傳送訊息，如圖 2.6 所示。但 System Matrix 裡面的每個 Basic Cycle 仍要以 Reference Message 開頭來區隔每個 Basic Cycle。

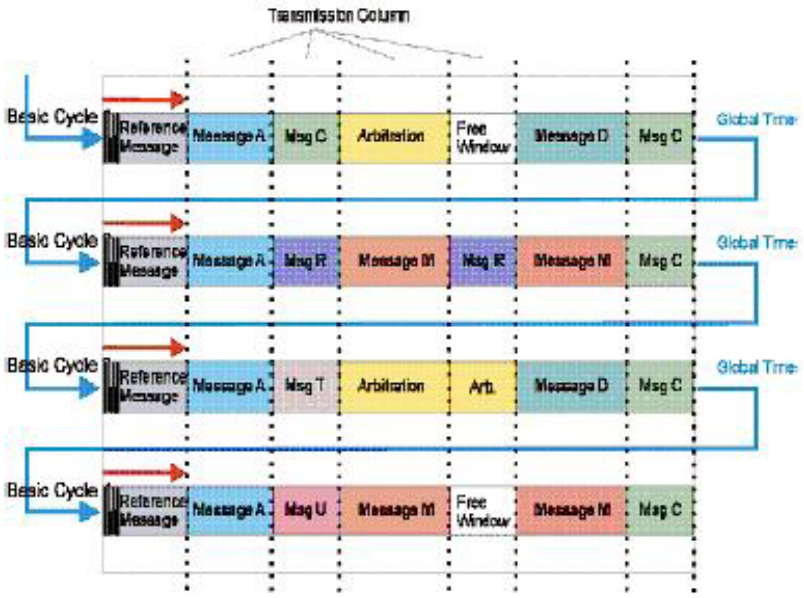


圖 2.6　TTCAN 的 System Matrix 傳輸架構 [7]

- 時間標記(Time Mark)

  Time Mark 是用來標記，每個 Time Window 在 Reference Message 後多少時間送出，如圖 2.7 所示。

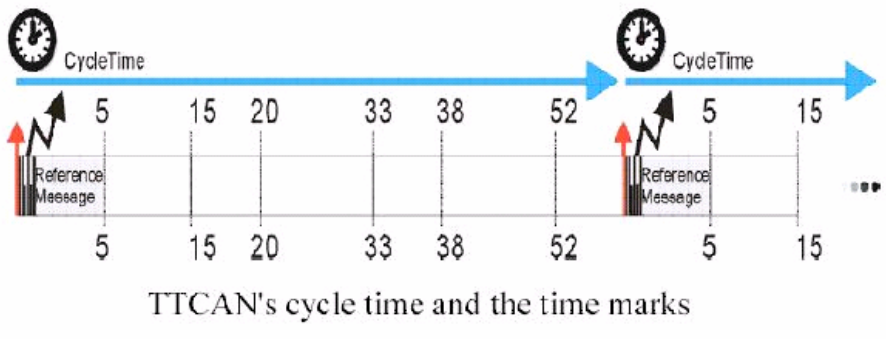


圖 2.7　TTCAN 的時間標記 [7]

TTCAN 的傳輸是依據 Cycle Time 或是 System Matrix 的順序來傳送訊息。TTCAN 上每個的節點不需要知道所有在 BUS 上的訊息，節點只需依據 Reference Message 以及 Time Mark 來傳送訊息。節點的傳送就不需要比較每個訊息的 Identify。因此 TTCAN 不會有失掉仲裁權(lost arbitration)的狀況發生。

2.8 CAN 與 TTCAN 的比較

TTCAN 之所以會出現是因為 event-triggered CAN 沒有同步的時脈，網路上沒有一個 master node 來主導，所有訊息的傳送都必須透過比較資料欄位的仲裁欄，當節點的仲裁欄的優先權低的時候，常常會有訊息一直送不出去的現象發生。所以在傳送上充滿不確定性，不知道何時訊息才會傳送出去。但 TTCAN 滿足了此部分的要求，TTCAN 所有的傳送與接收都透過 time slot，節點只需要在特定的時間去傳送或接收。TTCAN 也因為如此其傳送比較不具彈性，因為所有的訊息都要透過規劃來產生，它的設計較為嚴謹且繁複，不似標準的 CAN 傳送較為簡單有彈性。

由於 TTCAN 的特性藉由時間觸發來傳送訊息，所以在需要時間觀念的排程方法中有很大的用處。有很多的排程方法有假設系統有統一的系統時間觀念，利用訊息的緊急程度不同，改變訊息的仲裁欄決定優先權先後。因此在一些論文中，模擬排程方法假設在 TTCAN 的狀況下。

雖然 TTCAN 是因為有其需要才出現，現在還是 event-triggered CAN 比較普及，TTCAN 現在雖然有產品，但車上所使用的 CAN 仍然還是 event-triggered CAN。

第三章 排程理論

3.1 CAN的排程理論分類

有關CAN的排程理論，大致上可以區分成兩種:第一種是使用訊息表為單位，編排訊息表來改善網路效能;第二種以單個訊息為單位，使訊息依一定規則送出，改善網路效能。其分類如下:

- 使用訊息表:
  - static scheduler
  - dynamic scheduler
  - planning scheduler
- 利用訊息來排程:
  - rate monotonic scheme (RM)
  - deadline monotonic scheme (DM)
  - earliest deadline first scheme (EDF)

將所要傳送的訊息先排程一個表，所有訊息的傳遞都依照排程表的順序傳送，如果要改變一個訊息的順序，就必須整個變動排程表。此種方法如果用於TTCAN的話，因為TTCAN本身就是將訊息放在System Matrix來傳送，且matrix在傳送之前必須先排好message再傳送，所以較為適合。若用於普通的CAN如果要造一個訊息表，可能需要使用很多的訊息先做溝

通，並不適合。

在此論文中，主要以event-triggered CAN來架構軟體模擬及硬體，所以適合直接使用訊息為單位來排程。依據各種不同的排程理論，改變訊息的仲裁欄位以達成提高網路效能的目的。下面會仔細描述此種類別的排程理論。

3.2 排程理論介紹

3.2.1 rate monotonic scheme (RM)

RM scheme是一種使用很久的排程方法[11]，所使用的方式是週期越短的訊息，得到越高的優先權，使週期較短的訊息比其它週期較長的訊息先傳送出去，藉此使訊息的傳輸更為順暢。在CAN的設定上將週期反映在資料欄位的仲裁欄，由於在CAN傳輸上仲裁欄位值越小者，其優先權越高，所以取週期反應在仲裁欄上。但CAN的仲裁欄標準只有11個位元，如果週期無法充分反映在11個位元的仲裁欄，CAN仲裁欄可以擴充成29個位元，但此方法會增加系統的overhead，因為需多傳輸18個位元。

為了能夠將週期更完整的表現在仲裁欄上，將週期範圍分段，為了避免訊息週期在同一段，在仲裁欄留下幾個位元區分這些訊息，使每個訊息的仲裁欄都會是唯一的， 以利訊息的辨識。仲裁欄的分布方式如圖3.1所示。

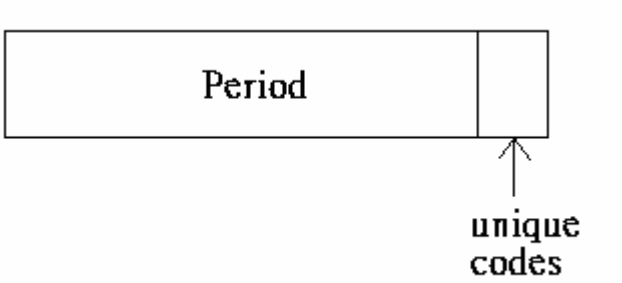


圖 3.1　RM scheme 表示在 CAN 仲裁欄的方式

3.2.2 deadline monotonic scheme (DM)

DM scheme與RM scheme很相似，不過一個是以週期的長短為依據，另一個是以deadline的長短為依據[12-14]。deadline是系統能容忍訊息最長的傳輸時間，如果訊息錯過deadline怎可能造成系統不穩定

DM scheme的想法是「造成系統出現問題是因為傳輸時間超過deadline所致」，因此優先權的順序應該以deadline越近的訊息優先權越高。由於DM於RM非常相似，其實現的手法也與RM相同，是將反應在ID上的週期換成deadline。為了避免造成ID的位元不夠長，所以所做的處理與RM相同，分出幾個位元使得每個訊息都是唯一。其仲裁欄的分布方式如圖3.2所示。

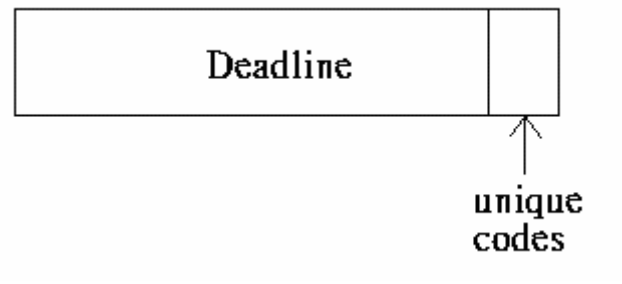


圖 3.2　DM scheme 表示在 CAN 仲裁欄的方式

3.2.3 earliest deadline first scheme (EDF)

EDF是一種動態的排程方法[12-16]，它與RM及DM不同，其原理是將最接近deadline的訊息，動態的更改其優先權使其擁有最高的優先權。但此方法需要比較所有待傳輸的訊息距離deadline時間的長短，在參考文獻中為了實現此方法都假設系統裡面要有一個同步的時脈，以方便將訊息距離deadline的時間差(圖3.3中的D)，反應在資料欄位仲裁欄中，如圖3.3所示。

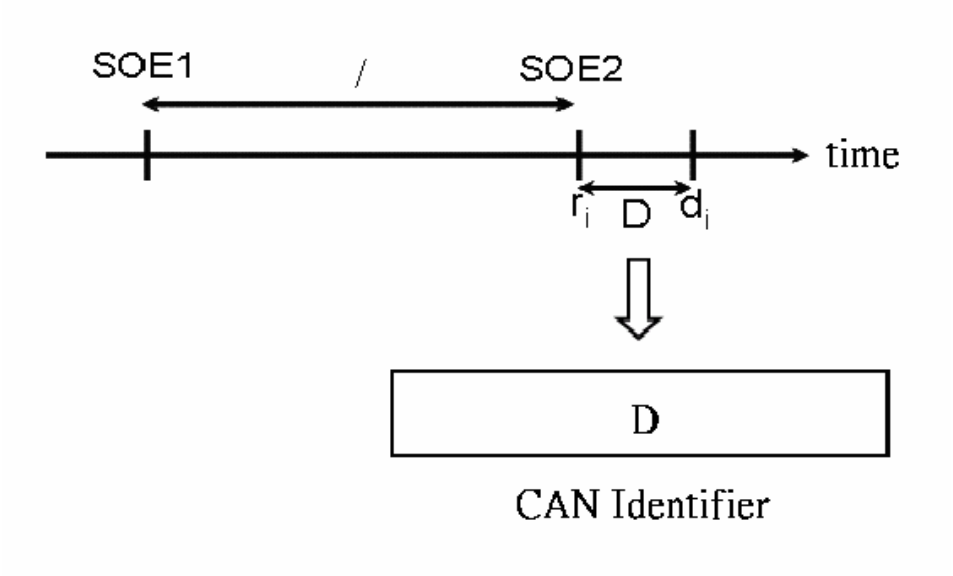


圖 3.3　EDF 的實現方式

此外在ID中deadline的欄位只有在SOE的時候更新，且必須維持到下一次SOE為止。在圖3.3中r_i是訊息i能最靠近deadline所能更改ID的時間點，di是訊息i的deadline。SOE1及SOE2是每次更改ID的時間點，兩次SOE所相差的時間差l。在實現EDF的時候，只須在最後最近deadline的SOE更改ID即可，所以D的計算如下:

$$d_i - SOE_1 = l + D$$

$$d_i - SOE_1 = d_i - \left\lfloor \frac{t}{l} \right\rfloor l$$

$$D = d_i - \left\lfloor \frac{t}{l} \right\rfloor l - l$$

$$= d_i - (\left\lfloor \frac{t}{l} \right\rfloor - 1)l$$

t: current time

l: depends on what fraction of CPU time that allow for ID updates

但在普通event-triggered CAN是並沒有同步時脈的，也沒有參考的SOE點，因為每個訊息的傳送都是獨立的，無法計算出上面的D值。而能夠擁有一個同步的時脈，每個訊息要在同一個時間點更新，以及知道整個bus的時間和自己本身的時間，只有TTCAN有此功能。TTCAN的傳輸是以cycle time以及system matrix的方式，一次傳遞好幾個訊息，且TTCAN在硬體設計上有同步的機制，能夠更改傳送內容只有在下次reference message的時候。

在CAN2.0b上此方法並不可行，因為CAN2.0b無法得知每個訊息的狀況，來安排訊息的仲裁欄。為了能夠將排程理論應用在在CAN 2.0b上，雖然CAN 2.0b沒有同步的時脈，無法知道所以無法比較每個時間點上，各個訊息deadline的長短。但EDF其精神在於將快接近deadline的訊息傳送出去，解決最緊急最接近deadline的訊息，為了能夠將EDF的精神應用在普通的CAN上，所以提出在仲裁欄中保留最前面的一個位元，一般設計為1，在時間快接近deadline之前，將最前面的一個位元從1改成0，使其擁有最高的優先權，使訊息能夠即時在deadline之前送出，其表現方式如圖3.4所示。

圖 3.4　本文模擬之 EDF 在仲裁欄的表現方式

3.2.4 RM，DM與EDF的結合

靜態排程理論RM scheme與DM scheme，是將排程理論反應在訊息的仲裁欄上。而動態排程理論EDF是將最緊急的訊息送出，此兩種想法並不衝突。其結合方式如圖3.5所示。仲裁欄的前面一個位元，第一個位元保留給EDF，當訊息接近deadline時，將此位元由1改成0，最後面則留幾個位元則是為了讓訊息的ID是唯一的。

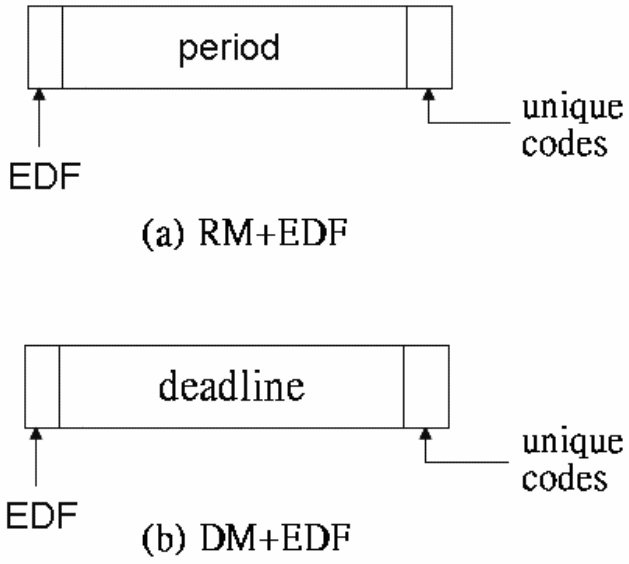


圖 3.5　RM,DM 與 EDF 的結合

3.2.5 active adjustable priority scheme (AAP)

由於EDF在硬體上無法實現，因此本論文提出AAP的概念，本質上與EDF有相同的概念，但EDF是當訊息傳送不出去的時候，直接更改訊息的優先權，使訊息能夠立即送出。AAP是經過一段時間的觀察後，觀察訊息的傳輸狀況，調整訊息的優先權。雖然不能達到立即調整優先權的效果，但因為調整是經過一段時間的觀察，因此能適當的調整訊息的優先權。

AAP的概念如下:

(1)將訊號的緊急程度分成M級，觀察N次傳輸的狀況，N次傳輸中傳輸成功的次數為t。需要調降緊急程度的次數為x，以及調升緊急程度的次數為y。

(2)如果在N次傳輸過程中，若t>x則訊息緊急程度(m)降一級(m-1)。

(3)如果在N次傳輸過程中，若t<y則訊息緊急程度(m)升一級(m+1)。

(4)如果在N次傳輸過程中，若y<t<x則訊息緊急程度(m)不變。

例如當M=3，N=10，x=10，y=5時，訊息的運作如下:

(1)AAP將訊號的緊急程度分成三級(index1、index2、index3)，訊息仲裁欄如圖3.6所示，實驗一開始所有訊息都設為index2。調整訊息仲裁欄的時間以節點十次傳輸時間為一個單位。

(2)如果十次都順利傳輸出去，則訊息降一級(index1→index1、index2→index1、index2→index3)。

(3)如果訊息有一至五次傳送不出去，則訊息等級不變(index1→index1、index2→index2、index3→index3)。

(4)若訊息有超過五次以上傳送不出去，則訊息調高一級(index1→index2、index2→index3、index3→index3)。

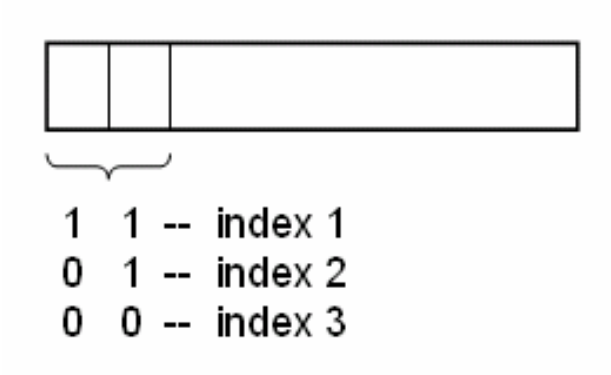


圖3.6 AAP在訊息仲裁欄的表現方式

3.3.6 RM、DM與AAP的結合

RM、DM與AAP的結合和與EDF結合相似，因為動態調整的排程方法與靜態的RM或DM結合並沒有衝突的地方。其結合方式如圖3.7所示。將前面兩個位元保留給AAP調整訊息的優先權。其餘的位元按照RM或DM的表現法。

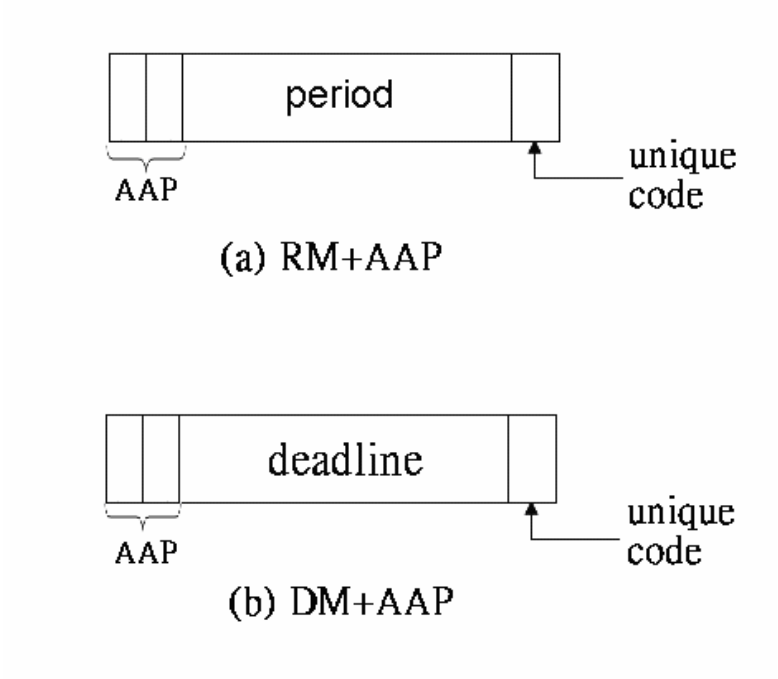


圖3.7 (a)RM+AAP在訊息仲裁欄的表現方式

(b)DM+AAP在訊息仲裁欄的表現方式

第四章　模擬軟體架構與模擬實驗結果

在本章中，將進行RM與其他排程方法的比較。首先說明如何架構CAN的模擬軟體，其次說明評估的三項指標： 平均傳輸時間(average transmitting time)、未即時傳遞訊息比例(missing deadline percentage)、訊息遺失比例(lost message percentage)。最後以三項指標作為評估的依據，並分析實驗結果進行討論。

4.1 模擬軟體架構

4.1.1 軟體架構

為了能夠比較各種排程方式的優缺點，建立一套能夠模擬分析CAN的軟體，以期能夠在電腦上先做分析模擬，直接比較其優缺點。所以在此論文中利用Borland C++ Builder 5 建立一個CAN的模擬軟體，模擬CAN訊息的傳輸方式，並直接在電腦上進行分析比較。此套系統並不執行細部的模擬CAN的各項軟體設定或硬體架構，而只是在邏輯上模擬CAN的傳輸，由於CAN的傳輸是以比較資料欄位仲裁欄的大小，仲裁欄的值越小者其優先權越高，以此方式來決定傳輸的先後順序。

建立此系統的主要目的是為了方便分析傳輸結果，因此在模擬的過程中，必須將所有傳輸的過程與結果記錄下來以方便分析。在過程中所要記

錄的有:訊息開始等待傳輸時間點，傳輸完畢的時間點以及訊息在傳輸過程中錯失幾次。為了達成這樣的目標，此系統藉助資料庫的幫助。在架構此程式時，使用兩個資料庫系統來記錄這些訊息。一個資料庫用來記錄等待BUS傳輸的訊息(BUS queue database)，另一個資料庫用來記錄BMS傳輸完的訊息(transmitted message database)。BUS queue與transmitted message兩個資料庫的欄位如表4.1所示，由表4.1所示所記載的欄位資料可以記錄訊息的傳輸狀況，幫助分析傳輸狀況。

表 4.1　資料庫的欄位

BUS queue database	transmitted message database
identifier	identifier
length	length
period	period
deadline	deadline
enter time	enter time
lost	lost
	leave time

程式的架構如圖4.1所示，節點將所要傳輸的訊息送到BUS queue的資料庫，並紀錄進入的時間點於資料庫的『enter time』的欄位。所有的訊息都送入BUS queue中等待傳輸。如果在下次同樣的訊息送入BUS queue時，發現原本訊息尚未送出，必須紀錄此訊息錯失，將其記錄在『lost』的欄位。當訊息依照所設定的傳輸速度，從BUS queue送入transmitted message的資料庫時，以代表訊息已經傳送，除了將此訊息的所有資料外還要再記錄離

開BUS queue資料庫的時間點，將其記錄在transmitted message 資料庫的『leave time』欄位，以方便後續的分析。

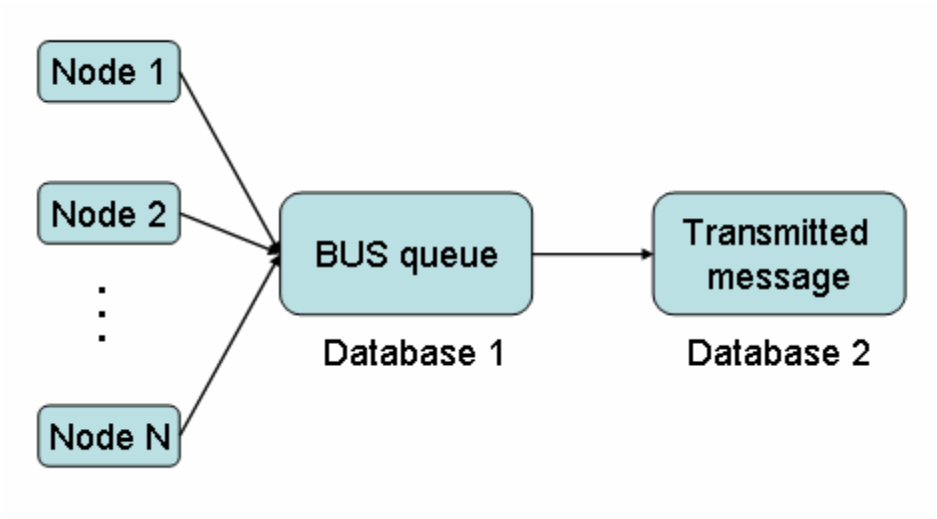


圖 4.1　CAN BUS 模擬軟體資料庫的操作

4.1.2 程式流程

整個程式流程可分成三個部份:第一部分是程式主流程，介紹整個程式的使用流程；第二部分是Message Thread的程式流程；第三部份是BUS Thread的程式流程。

◆ 程式主流程

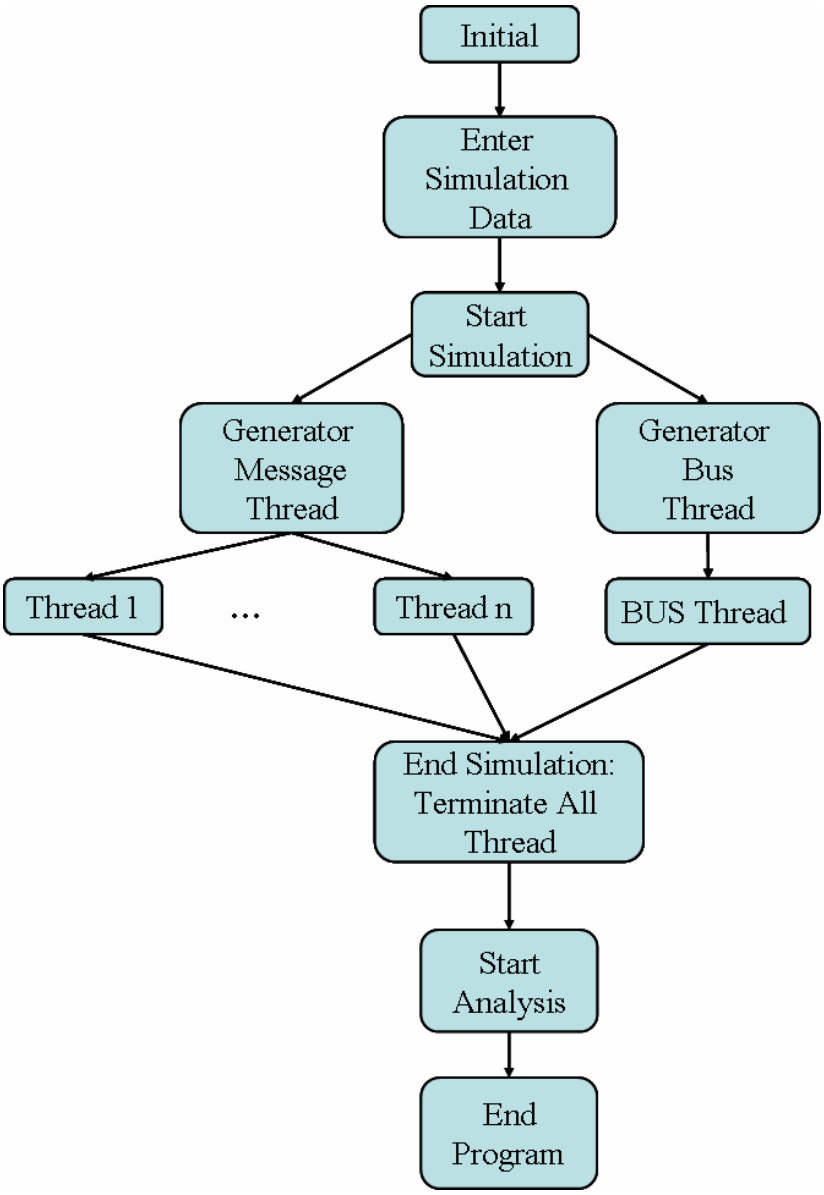


圖 4.2　CAN BUS 模擬程式主流程圖

◆ Message Thread的流程

整個程式是以多工處理來解決節點獨立運作的特性，在程式中每個節點都以一個執行緒(Thread)來表示，執行緒會依照設定週期性的將訊息送入BUS queue的資料庫中，若資料庫中已有此Thread的訊息，則表示此訊息在一個週期時間內還未送出，將『Lost』的欄位加1，表示其錯失了一個訊息。若資料庫中沒有這個訊息，則將訊息依其仲裁欄的大小插入資料庫中。其處理方式如圖4.3所示。

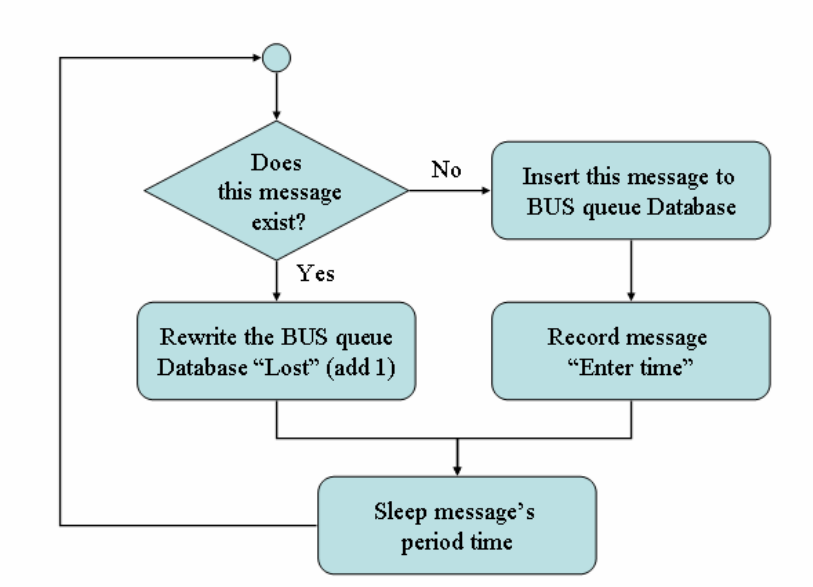


圖 4.3　Message Thread 流程圖

◆ BUS Thread的流程圖

BUS Thread的存在是為了模擬CAN的傳輸，此執行緒會依照所設定的傳輸速度，週期性的從BUS queue Database拿走優先權最高的訊息，並將此訊息的所有資料與此訊息離開BUS queue Database的時間點寫入Transmitted Database的『Leave time』欄位，也方便後面的分析。其處理方式如圖4.4所示。

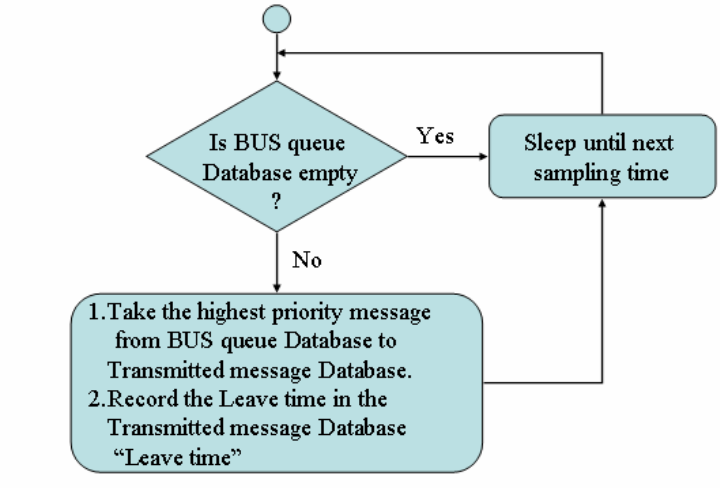


圖 4.4　BUS Thread 流程圖

4.1.3 網路效能指標

- 平均傳輸時間（average transmitting time）

  對於控制系統而言評估整體訊息的平均傳輸時間，以了解整個系統中訊息的傳輸狀況。命令的傳輸時間直接影響到下達命令的時間安排，如果訊息的平均傳輸時間越短，系統中的節點在做個別處理時，比較不需考慮傳輸時間的因素。相反的若平均傳輸時間很長，則節點必需遷就CAN BUS的傳輸狀況做處理。

  $$\text{平均傳輸時間} = \frac{\text{傳輸成功的訊息總傳輸時間}}{\text{傳輸成功的訊息數量}}$$

- 未即時傳遞訊息比例（missing deadline percentage）

  在即時多工的系統中 ，為了每個訊息都能發揮其效果，一般會有deadline的設計，訊息如果錯過deadline則可能造成系統運作的錯誤。如在馬達控制系統中，有一定的sampling time來計算馬達的轉動速率，回朔之後再下達控制命令。但如果訊息錯過這個時間送達，則會造成命令下達不符合實際狀況。而造成系統不穩定因為deadline是一個很重要的問題。

  $$\text{未即時傳遞訊息比例} = \frac{\text{傳輸成功的訊息中沒有即時傳遞的訊息數量}}{\text{傳輸成功的訊息數量}} \times 100\%$$

- 訊息遺失比例（lost message percentage）

  訊息如果沒有傳送出去對系統所造成的影響最大，如果漏掉的訊息只是一般普通的資料的話影響不大，但若漏掉的訊息是緊急的命令，那可能會造成系統嚴重錯誤，導致系統無法運作。如以車用系統為例，如果漏掉的訊息是緊急煞車的訊息，那會造成使用上的危險。所以在

使用上以訊息沒有傳送出去，造成的影響最大也最危險。

$$\text{訊息遺失比例} = \frac{\text{沒有傳輸成功的訊息數量}}{\text{所有傳輸的訊息數量}} \times 100\%$$

4.1.4 程式介面

程式介面分成三個個頁面:Messages，Simulation以及Analysis。第一個頁面【Messages】如圖4.5所示，此頁面的用途是編輯所要模擬的訊息。在『key in message』的欄位中可直接輸入訊息，輸入訊息的順序依次為訊息的仲裁欄位，傳送訊息的長度，訊息的週期以及訊息的deadline。輸入一組訊息後，按下[Add message]的按鍵後，訊息會出現在『Message information』中，接著可繼續鍵入下一組訊息。訊息也可以從檔案一次輸入，從『Message file』欄位中的[Open]按鍵，可以開啟所需的訊息檔，訊息同樣也會出現在『Message information』中。所有出現在『Message information』都可以直接修改，修改完後必須將訊息存檔(『Message file』欄位中的[Save]按鍵)。要進入模擬前，按下[Enter message to simulation]的按鍵，結束編輯模擬訊息。

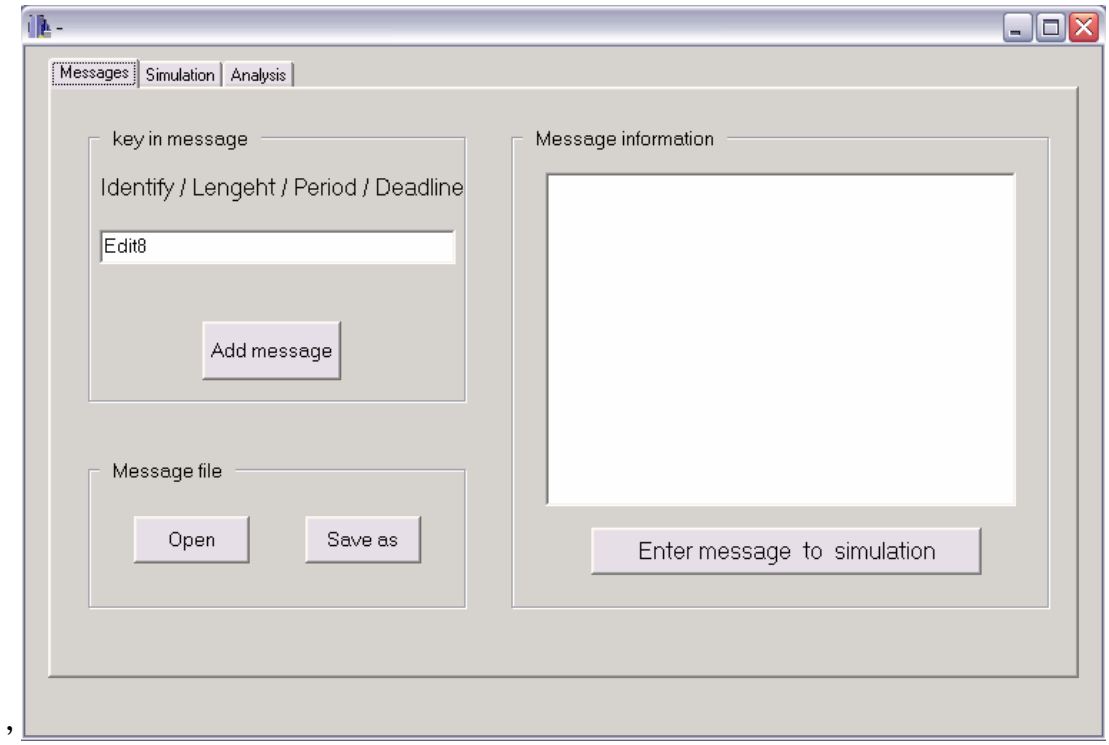


圖 4.5　訊息輸入介面

第二個頁面【Simulation】此介面設定CAN的模擬設定，並可操作觀察CAN的傳輸，整個頁面設計如圖4.6所示。在『Simulation option』的欄位設定傳輸速度，傳輸方法以及模擬的傳輸時間。在『Simulation』及『BUS status』兩個欄位，設定啟動模擬[Start simulation]，終止模擬[End simulation]，暫停模擬[suspend]以繼續模擬[resume]。在右邊的兩個資料欄顯示BUS上的狀況(BUS Queue)以及傳輸過的訊息狀況(Transmitted message)。

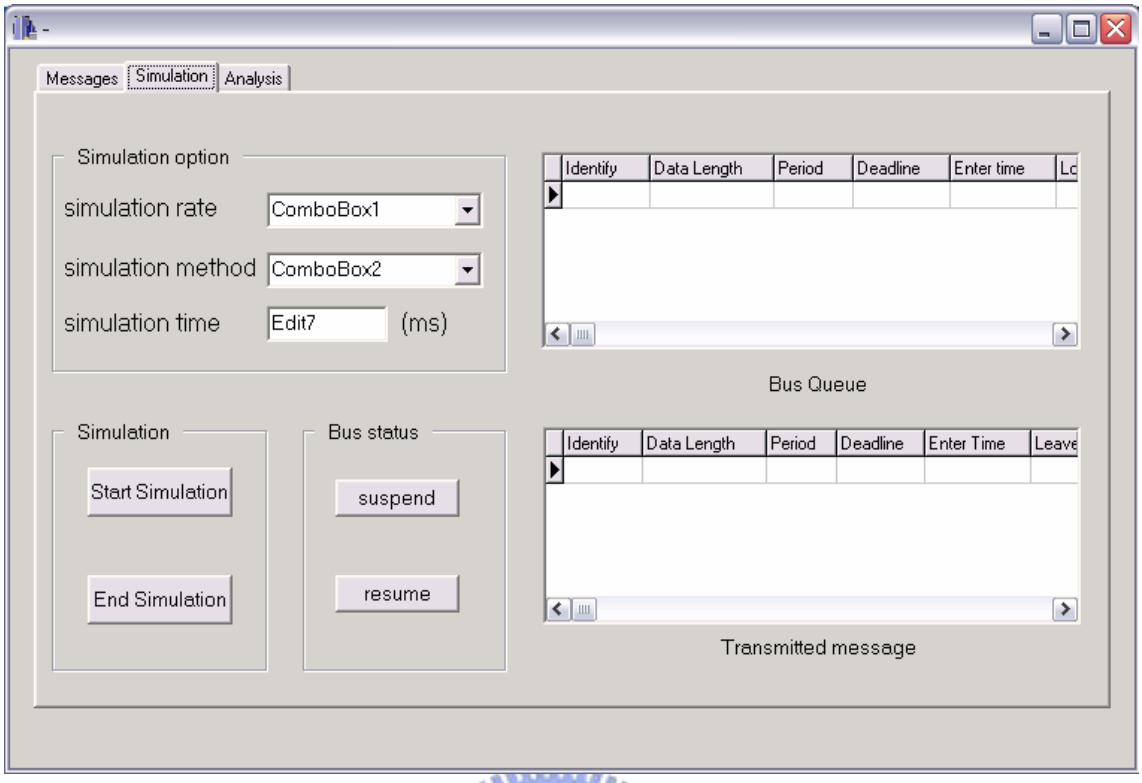


圖 4.6　Simulation 介面

第三個頁面【Analysis】是作為分析的頁面，其介面如圖4.7所示，按下[Start Analysis]的按鍵後，會將分析的結果average transmitted time，missdeadline，lost message以及BUS的流量顯示在頁面上。

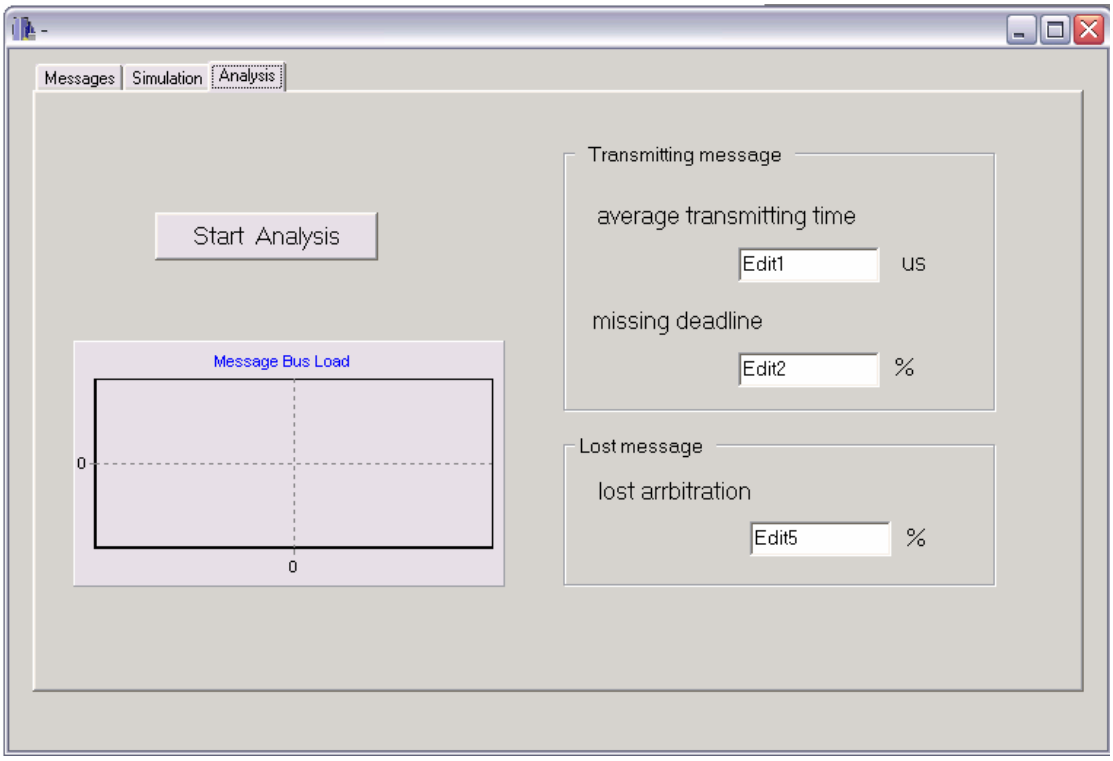


圖 4.7　Analysis 介面

4.2 模擬實驗數據

軟體實驗是模擬在傳輸速度250Kbits/s時，模擬CAN實際傳輸4秒鐘的傳輸狀況，由於是用模擬軟體模擬CAN的運作，實際的程式執行時間遠大於4秒鐘。以不同的頻寬負載量以量測average transmitting time，missing deadline percentage，lost message percentage藉此比較其效能。實驗中比較的排程法有:RM、DM、EDF、RM+EDF、DM+EDF、AAP、RM+AAP與DM+AAP。

負載公式計算方法如下:

BUS Transmitting Rate : K $bits/s$

Message : 1,2,⋯,i,⋯,n

Period : $T_1, T_2, \cdots, T_i, \cdots, T_n$ (ms)

Message Data Length : $L_1, L_2, \cdots, L_i, \cdots, L_n$ $(bits)$

Message overhead =44 $(bits)$

bit time = $\frac{1}{K}$ (sec)

Message Length (L_i) = D_i + overhead $(bits)$

one message transmitting time = $L_i \times \frac{1}{K} = \frac{L_i}{K}$ (sec)

1 sec can transmit maximum messages M = $\frac{K}{L_i}$

$$\text{Bus Load (\%)} = \frac{\sum_{i=1}^{i=n} \frac{1000ms}{T_i(ms)}}{M} \times 100\%$$

下面的例子是由表4.2 (a)的數據，計算其負載量：

BUS Transmitting Rate : 250K $bits/s$

Message : 1, 2, 3, 4, 5

Period : 4, 8, 10, 6.9, 7.8 (ms)

Message Data Length : 64, 64, 64, 64, 64 ($bits$)

Message overhead =44 ($bits$)

bit time = $\frac{1}{250\times10^{3}} = 4\times10^{-6}$ (sec)

Message Length (L_i) = $64+44=108$ ($bits$)

one message transmitting time = $108\times4\times10^{-6}$ =0.432 (ms)

1 sec can transmit maximum messages M = $\frac{1000ms}{0.432ms} = 2314.8$

Bus Load (%) = $\frac{1000/4+1000/8+1000/10+1000/6.9+1000/7.8}{2314.8}\times100\%$

$\doteq 30\%$

表4.2至表4.7分別為頻寬負載量在30%，50%，70%，90%，110%，120%進行實驗的實驗數據。(a)表是所有Node的傳輸資料，總共有五個Node在傳送資料，傳輸資料包括:訊息仲裁欄(Identifier)，訊息的資料長度(data length)，訊息的傳送週期(period)，訊息的截止期限(deadline)。(b)表是模結果依據average transmitting time，missing deadline percentage，lost message percentage三項指標分析的數據。

表4.2 (a) 傳輸速率250 *Kbits/s*時，30%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	4	15
2	65	64	8	25
3	12	64	10	30
4	10	64	6.9	33
5	28	64	7.8	28

表4.2 (b) 傳輸速率250 *Kbits/s*時，30%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	16.013	15.621	16.209	15.392	15.196
Missing Deadline Percetntage(%)	0	0	0	0	0
Lost Message Percentage(%)	0	0	0	0	0

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(10μs)	15.359	17.275	15.359	16.209
Missing Deadline Percetntage(%)	0	0	0	0
Lost Message Percentage(%)	0	0	0	0

表4.3 (a) 傳輸速率250 *Kbits/s*時，50%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	2.4	10
2	65	64	5.5	25
3	12	64	7.2	22
4	10	64	3.3	13
5	28	64	4.8	18

表4.3 (b) 傳輸速率250 *Kbits/s*時，50%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	21.395	20.807	20.059	19.764	19.823
Missing Deadline Percetntage(%)	0	0	0	0	0
Lost Message Percentage(%)	0	0	0	0	0

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(10μs)	21.297	20.923	19.665	20.177
Missing Deadline Percetntage(%)	0	0	0	0
Lost Message Percentage(%)	0	0	0	0

表4.4(a) 傳輸速率250 *Kbits/s*時，70%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	1.8	8
2	65	64	3.8	14
3	12	64	5.1	21
4	10	64	3.0	10
5	28	64	2.5	15

表4.4 (b) 傳輸速率250 *Kbits/s*時，70%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	17.879	17.963	17.851	17.668	17.809
Missing Deadline Percetntage(%)	0.4213	0.2809	0	0.2809	0.2809
Lost Message Percentage(%)	0	0	0	0	0

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	18.005	18.202	17.809	17.879
Missing Deadline Percetntage(%)	0.2808	0.4213	0.4213	0.1404
Lost Message Percentage(%)	0	0	0	0

表4.5(a) 傳輸速率250 *Kbits/s*時，90%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	1.5	8
2	65	64	2.9	14
3	12	64	3.5	15
4	10	64	2.2	12
5	28	64	2.0	10

表4.5 (b) 傳輸速率250 *Kbits/s*時，90%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	27.934	28.175	28.317	28.428	28.295
Missing Deadline Percetntage(%)	2.5137	0.7650	0.6557	0.5459	0.6557
Lost Message Percentage(%)	0	0	0	0	0

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	28.185	28.776	28.689	28.525
Missing Deadline Percetntage(%)	0.7650	2.7322	0.8743	0.7650
Lost Message Percentage(%)	0	0	0	0

表4.6 (a) 傳輸速率250 *Kbits/s*時，110%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	1.2	5
2	65	64	5.8	28
3	12	64	2.5	15
4	10	64	4.4	14
5	28	64	0.9	4

表4.6(b) 傳輸速率250 *Kbits/s*時，110%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	39.699	48.680	48.342	34.503	42.733
Missing Deadline Percetntage(%)	19.0763	15.9722	15.9888	12.9317	15.4154
Lost Message Percentage(%)	10.5924	9.5153	9.6050	11.4901	10.5640

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	46.096	51.781	79.063	77.473
Missing Deadline Percetntage(%)	15.5599	27.3642	22.6321	26.7732
Lost Message Percentage(%)	9.5878	11.0116	10.0448	10.1436

表4.7 (a) 傳輸速率250 *Kbits/s*時，120%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	25	64	1.2	6
2	65	64	2	10
3	12	64	2.4	14
4	10	64	1.7	7
5	28	64	1.5	8

表4.7 (b) 傳輸速率250 *Kbits/s*時，120%傳輸訊息量的軟體模擬結果

Scheme	Original	RM	DM	EDF	RM+EDF
Average Transmitting Time(μs)	46.673	44.612	45.621	98.787	99.394
Missing Deadline Percetntage(%)	22.4429	14.9105	14.2289	56.2624	56.9444
Lost Message Percentage(%)	17.5941	17.6085	17.6229	17.5410	17.4447

Scheme	DM+EDF	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	98.532	79.520	86.474	85.986
Missing Deadline Percetntage(%)	59.9444	22.8771	27.6892	26.6932
Lost Message Percentage(%)	17.5123	17.9508	17.8396	17.8396

4.3 模擬結果分析

4.3.1 RM 與 DM 的比較

由 4.2 節的實驗數據，依據 average transmitting time，missing deadline percentage 和 lost message percentage 三項指標整理成圖 4.8。由圖 4.8(b)(c)可知 RM 能有效改善 missing deadline percentage 與 lost message percentage。但對改善 average transmitting time 的效果差。由圖 4.8 也可知 RM 與 DM 的效果差不多。

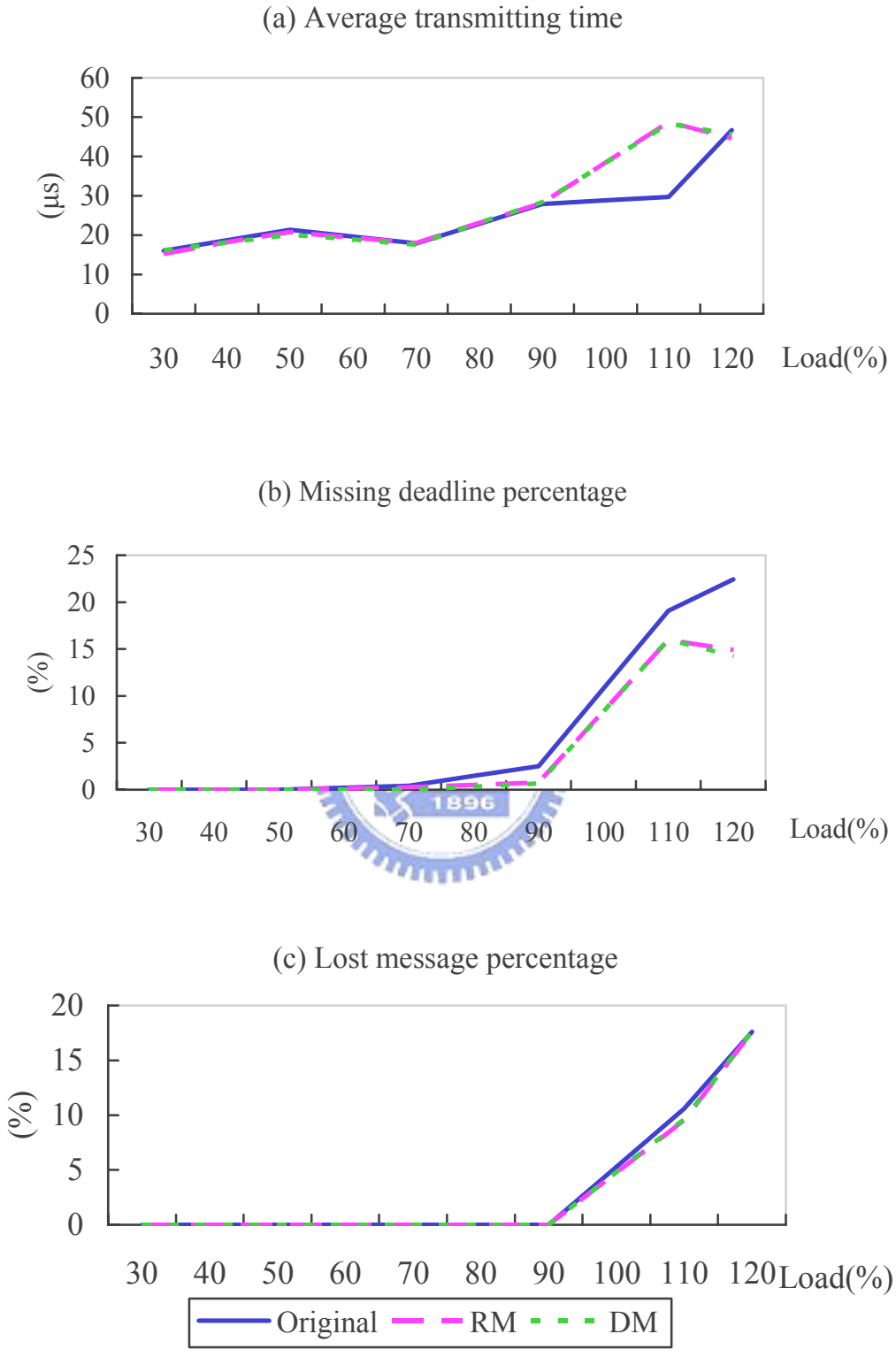


圖 4.8 Original，RM 與 DM 傳輸效能比較

4.3.2 EDF

圖 4.9 是由 4.2 節的數據將 EDF 與 Original 依據整理而成。由圖 4.9(b)可知 EDF 在負載量不太大時能有效克制 missing deadline 的狀況發生，但是若負載增加到 120%後則效果很差，missing deadline 的情形會突然急遽增加。而 EDF 在改善 Lost message 的效果如圖 4.9(c)所示，其效果並不好。

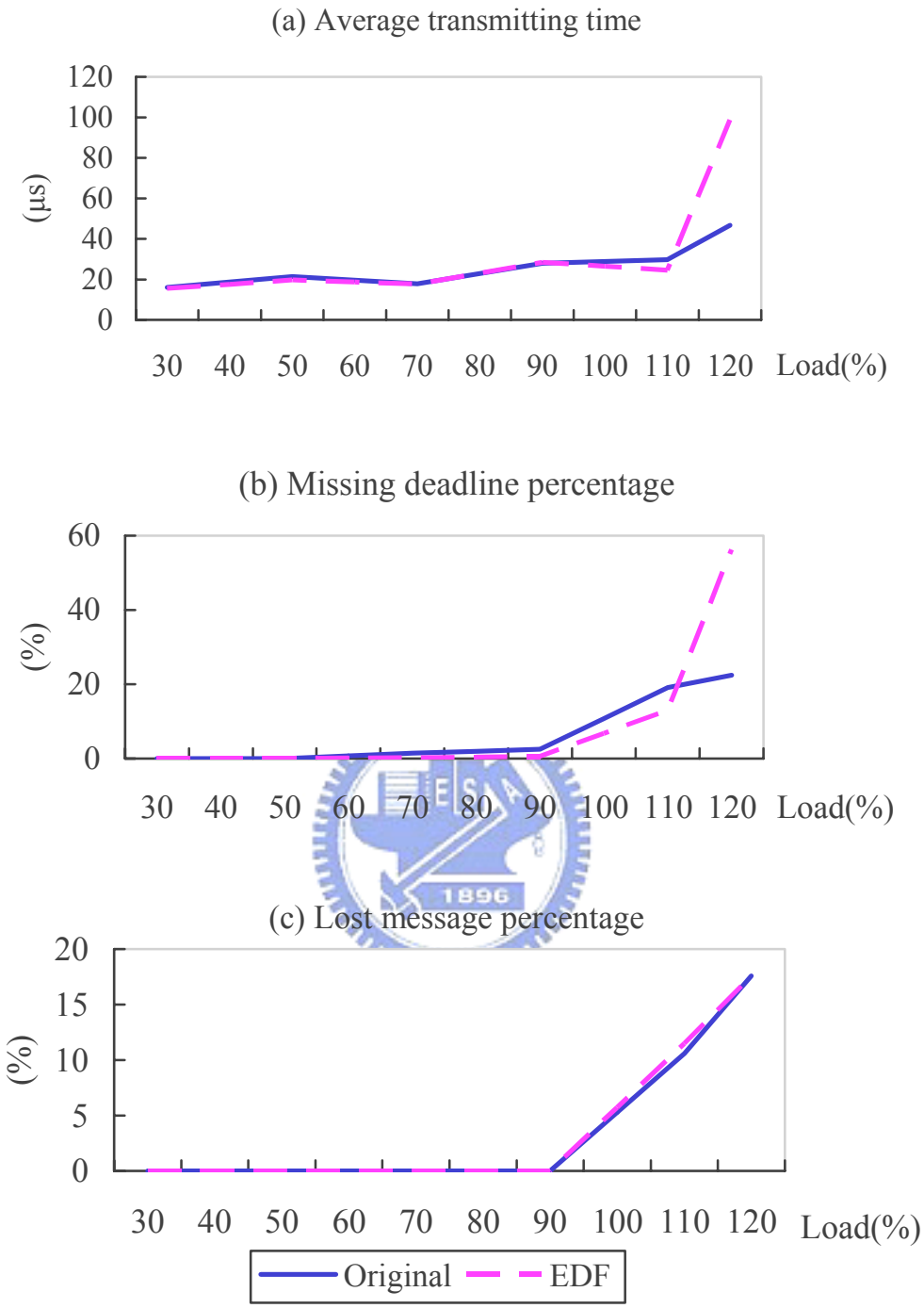


圖 4.9 Original 與 EDF 的傳輸效能比較

4.3.3 RM+EDF

使用 RM 設計訊息的仲裁欄，在訊息傳送時利用 EDF 調整仲裁欄，其結果如圖 4.10 所示。在高負載時 RM+EDF 如圖 EDF 一樣，average transmitting time 與 missing deadline percentage 會劇增。RM+EDF 的在改善

average transmitting time、missing deadline percentage 與 lost message percentage 的效果比單純的 EDF 佳但比 RM 來得差。

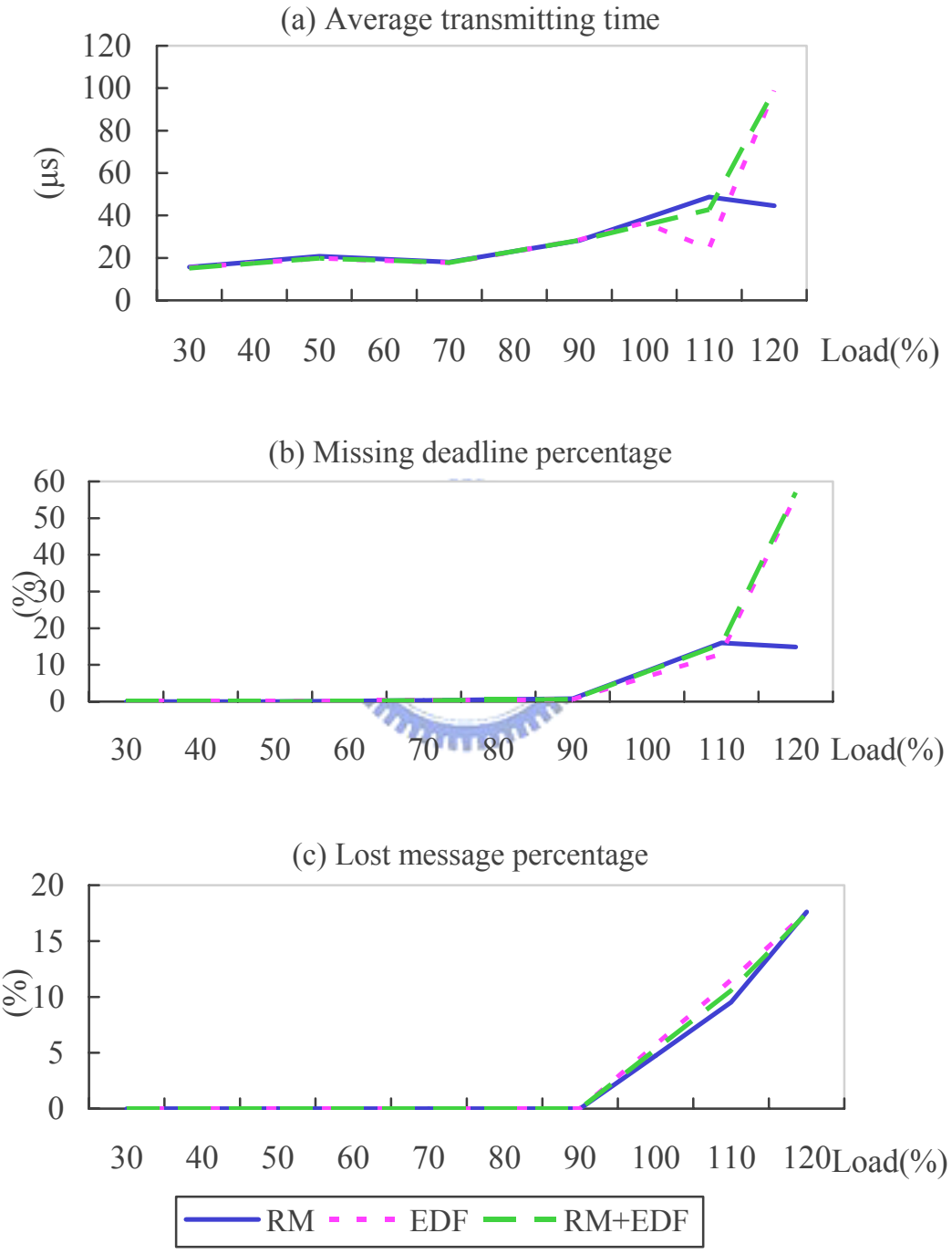


圖 4.10 RM，EDF 與 RM+EDF 的傳輸效能比較

4.3.4 DM+EDF

使用 DM 設計訊息的仲裁欄，在訊息傳送時利用 EDF 調整仲裁欄，其結果如圖 4.11 所示。DM+EDF 在改善 lost message percentage 的效果和 DM 差不多，且比單純 EDF 好。DM+EDF 與 EDF 在 missing deadline percentage 一樣劇增，DM 則沒有這個問題。

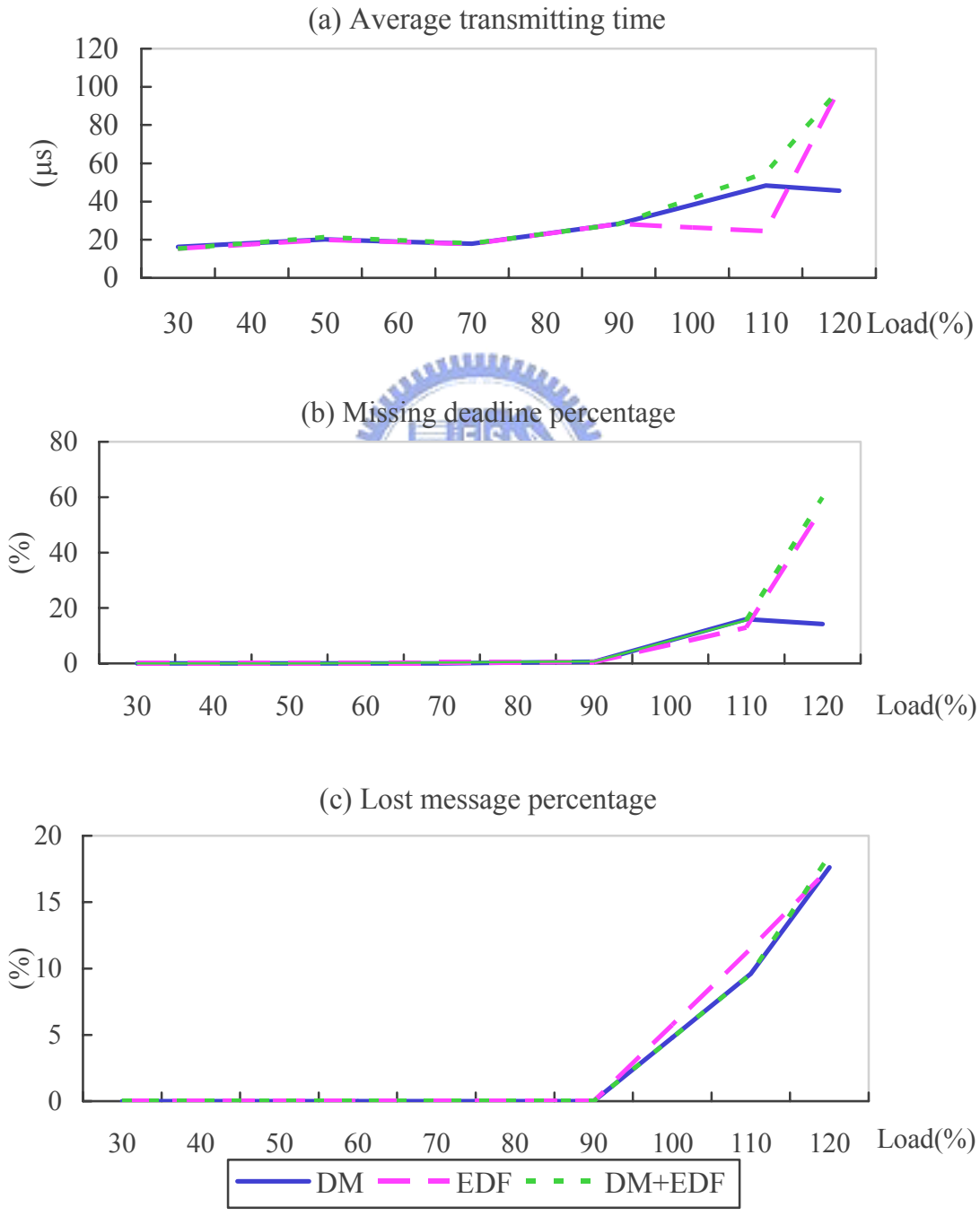


圖 4.11 DM，EDF 與 DM+EDF 的比較

4.3.5 Active adjustable priority (AAP)

Active adjustable priority(AAP)是一種動態調整訊息 Identifier 的方式，與 EDF 不同的地方是 AAP 是經過一段時間後再調整訊息仲裁欄改變訊息的優先權。將 4.2 節 Original 與 AAP 的實驗結果依據 BUS load 與三項指標的關係整理成圖 4.12。由圖 4.12(b)所示 AAP 只有在中低負載時對 missing deadline percentage 有效果。其他如圖 4.12(a)、4.12(c)所示，AAP 的效果比沒有加排程法的時候來得差。

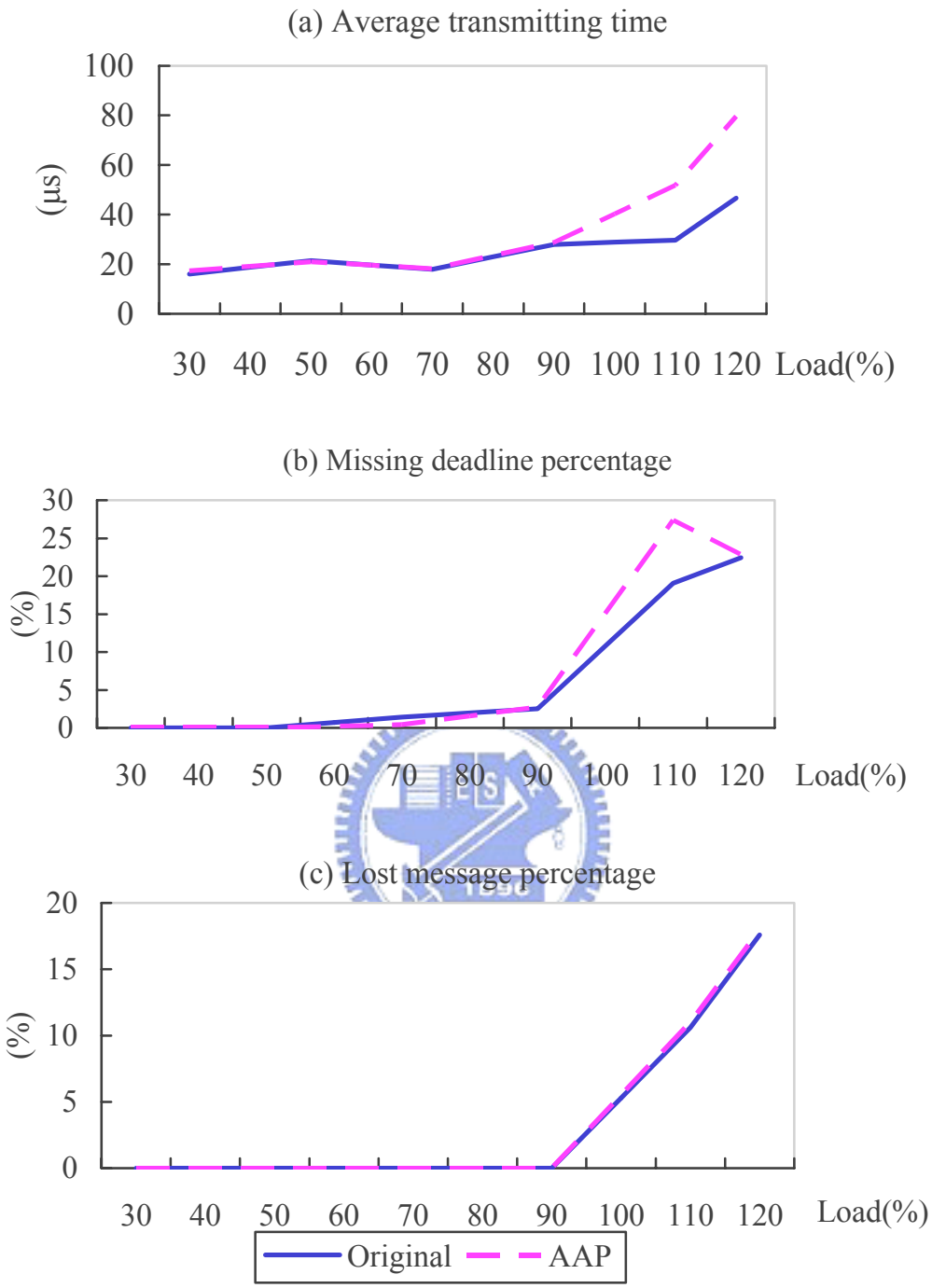


圖 4.12 Original 與 AAP 的傳輸效能比較

4.3.6 RM+AAP

將 4.2 節 RM、AAP 與 RM+AAP 的實驗結果依據 BUS load 與三項指標的關係整理成圖 4.13。RM+AAP 對與傳輸的改善在 missing deadline

percentage 與 lost message percentage 介於 RM 與 AAP 之間。對於 average transmitting time 此項指標 RM+AAP 比 RM 以及 AAP 差。

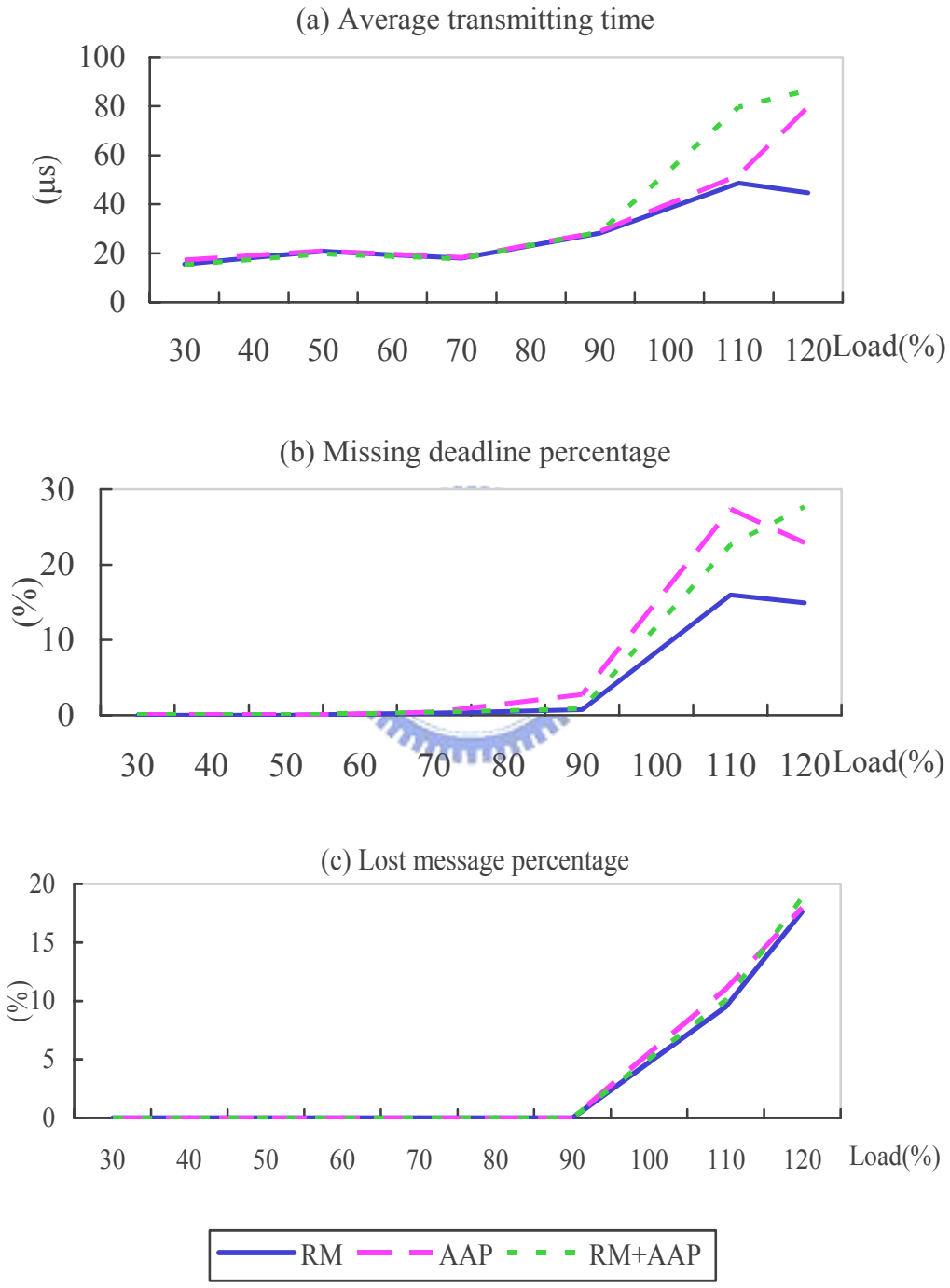


圖 4.13 RM，AAP 與 RM+AAP 的比較

4.3.7 DM+AAP

將 4.2 節 DM、AAP 與 DM+AAP 的實驗結果依據 BUS load 與三項指標的關係整理成圖 4.14。DM+AAP 對與傳輸的改善在 missing deadline percentage 與 lost message percentage 介於 DM 與 AAP 之間。對於 average transmitting time 此項指標 DM+AAP 比 DM 以及 AAP 差。

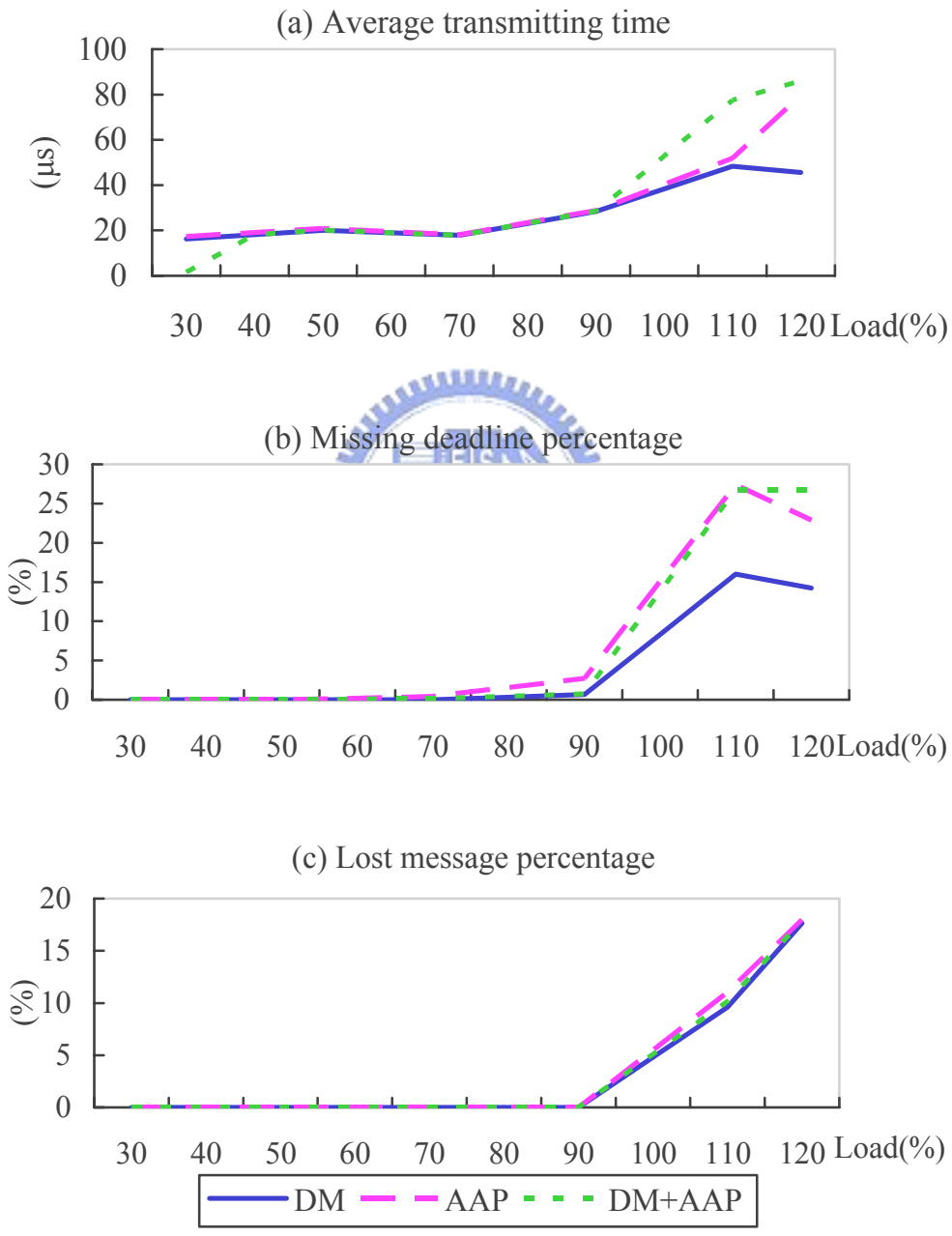


圖 4.14 DM，AAP 與 DM+AAP 的比較

4.4 討論

由 4.3 節的分析結果，依據 average transmitting time，missing deadline percentage，lost message percentage 的表現整理出表 4.8 與表 4.9。

表 4.8 軟體模擬實驗綜合整理（一）

Scheme	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time	***	***	***	*	*
Missing Deadline Percentage	*****	*****	**	****	***
Lost Message Percentage	*****	*****	***	****	****

[註] *worst ***fair *****best

表 4.9 軟體模擬實驗綜合整理（二）

Scheme	RM	DM	EDF	RM+EDF	DM+EDF
Average Transmitting Time	***	***	*****	****	**
Missing Deadline Percentage	*****	*****	****	****	****
Lost Message Percentage	*****	*****	**	****	*****

[註] *worst ***fair *****best

由表 4.8 與表 4.9 可做出幾點討論：

1. 利用 RM 來安排訊息的仲裁欄，確實能有效的降低 missing deadline percentage，lost message percentage。但對於 average transmitting time 並沒有改善效果。DM 的效果和 RM 的效果也差不多。
2. EDF 雖然在高負載時表現不好，但 EDF 在中低負載時，對降低 average transmitting time 比其他的排程法好。
3. RM+EDF 是利用 RM 來安排訊息的仲裁欄，在加上傳輸時使用 EDF 動態調整訊息的仲裁欄。RM+EDF 效果比不上單純使用 RM 來安排訊息欄，但比 EDF 來得好。
4. DM+EDF 在改善 lost message percentage 的效果和 DM 差不多，且比單純 EDF 好。DM+EDF 與 EDF 在 missing deadline percentage 一樣劇增，DM 則沒有這個問題。
5. 由三項傳輸指標來看，AAP 沒有任何改善效果。而 RM+AAP 效果雖然比不上單純使用 RM，但比 AAP 來得好。DM+AAP 也有同樣的情形。

第五章 硬體實驗結果

在這一章中，將進行排程理論的硬體實驗。首先介紹實驗平臺的硬體架構以及相關資訊；之後再說明硬體實驗設計流程，排程理論的實現方式；最後依據 average transmitting time、missing deadline percentage、lost message percentage 作為實驗分析的依據，並分析實驗結果，交叉比對進行討論。

5.1 實驗系統之硬體架構

5.1.1 硬體架構

用來作為實驗的平台是 Winbond 公司生產的 W78E516B 8051 微控制器，搭配 Philip 公司的 Stand-alone CAN controller SJA1000 以及高速光耦合器 PCA82C251。實驗的主要架構是由九個 CAN 節點（Node）所構成，其中 CAN 節點是由 8051 連接控制 SJA1000，再透過 PCA52C251 傳送接收訊息。實驗流程是由 PC 透過 RS232 將程式下載至 8051，由 8051 控制 SJA1000 傳送接收訊息，最後的實驗數據由 8051 透過 RS232 回傳到 PC 加以分析。硬體架構如圖 5.1 所示。圖 5.2 是節點的硬體圖。圖 5.3 是硬體的實驗系統圖，由九個節點所構成的系統，最後將資料傳回 PC 端。

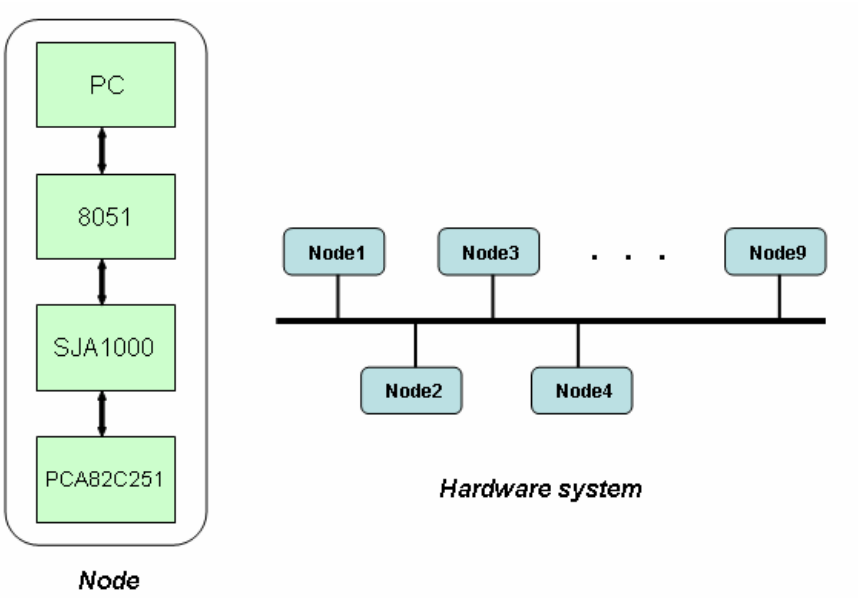


圖 5.1　實驗系統的硬體架構

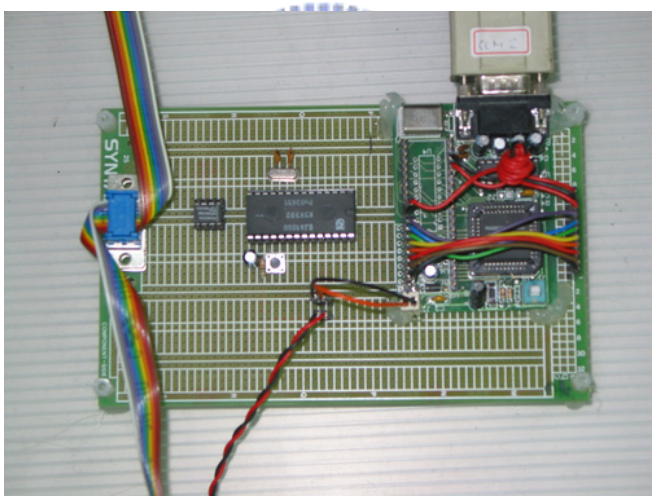


圖 5.2 單一節點的硬體圖

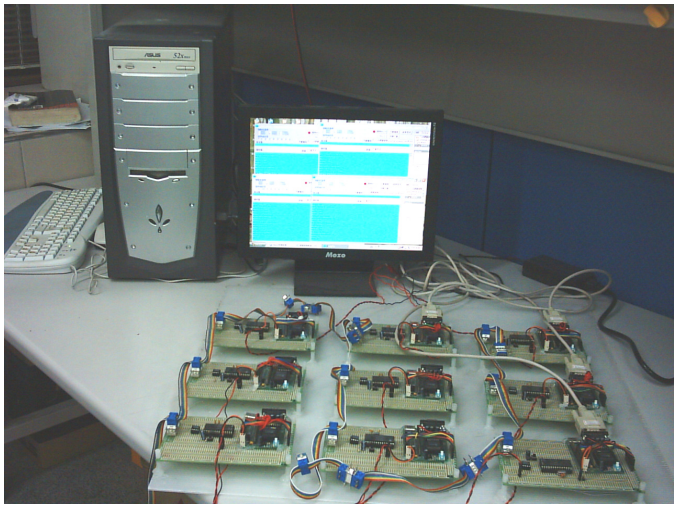


圖 5.3　硬體實驗設備圖

5.1.2 程式流程

硬體實驗的設計如圖 5.4 所示，由 master node 發出開始訊息(start message)，當傳輸端 node 接收到 start message 之後，就能同時開始傳輸所設定的訊息量，每個傳輸端的 node 都有相對應的接收端的 node 以利於傳輸效能指標的運算。當接收端收到所有的傳輸資料後，將所得的傳輸效能指標透過 RS232 傳回電腦。從 master node 到傳輸端以及從傳輸端到接收端的訊傳輸都是透過同一條 CAN BUS。接收端的 Node 將傳輸資料傳回電腦，是由每個接收端的 8051 透過 RS232 傳回電腦。

在實驗的過程中由 master node 發出 start message 的目的，是為了能夠使所有的節點能在同一時間開始傳送接收訊息，以方便在計算訊息的傳送延遲時間。當實驗結束之後，接收端的節點會將個別的傳輸結果傳回 PC。最後再將所有的接收端的節點所傳回來的資料，計算傳輸效能指標。之所以每一個傳輸節點都有一個接收端的節點相對應，是因為方便紀錄訊息的傳輸狀況，如訊息的延遲時間、錯過 deadline 的訊息、訊息沒有傳送出去的狀況。

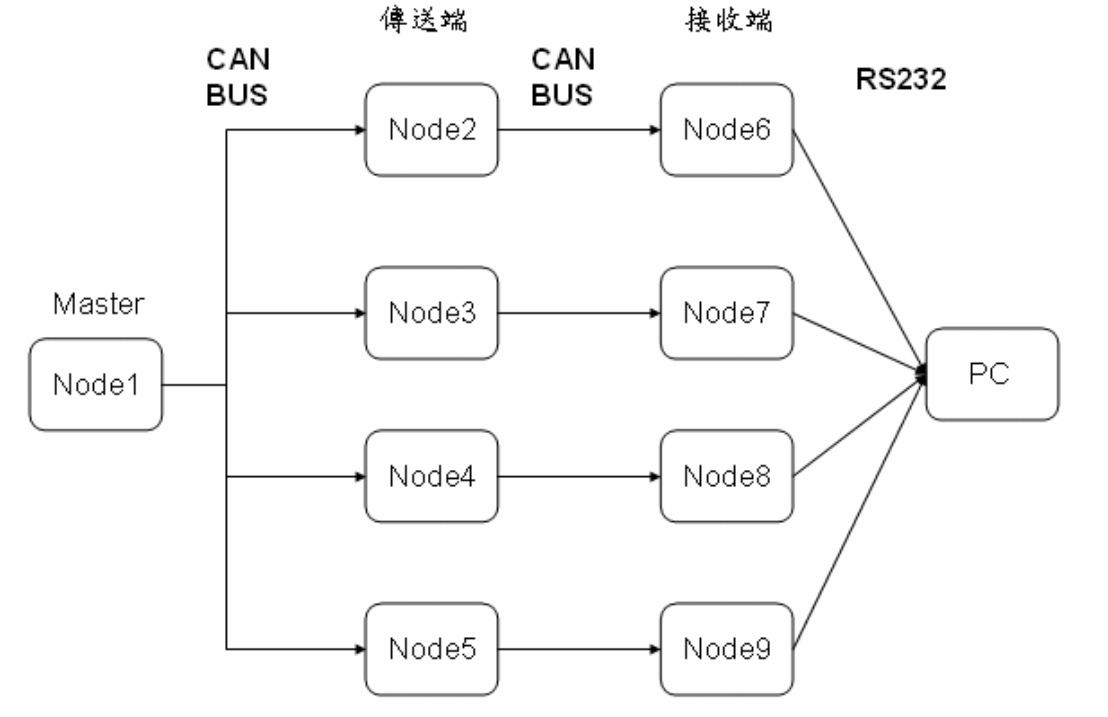


圖 5.4 訊息傳輸方向圖

傳輸端得流程圖如圖 5.5 所示，當收到 start message 之後，傳輸節點開始傳送所需要傳送的訊息，直到傳輸完所設定需要傳送的訊息。

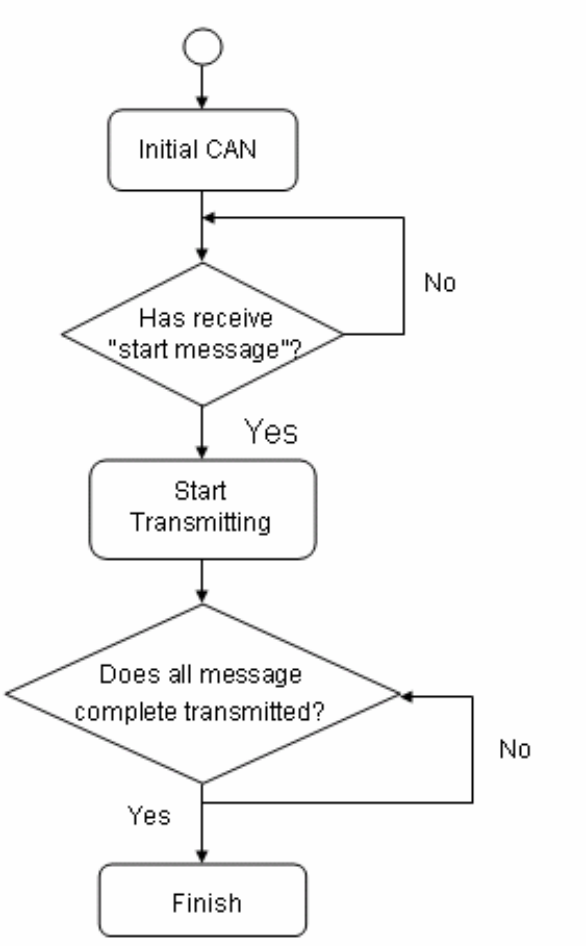


圖 5.5 程式傳輸端流程圖

接收端的程式流程圖如圖 5.6 所示，當接收到第一個訊息之後，開始計算三項傳輸指標:average transmitting time，missing deadline percentage，lost message percentage。當所有的訊息都接收到之後，將所得的三項傳輸指標的訊息傳回電腦。

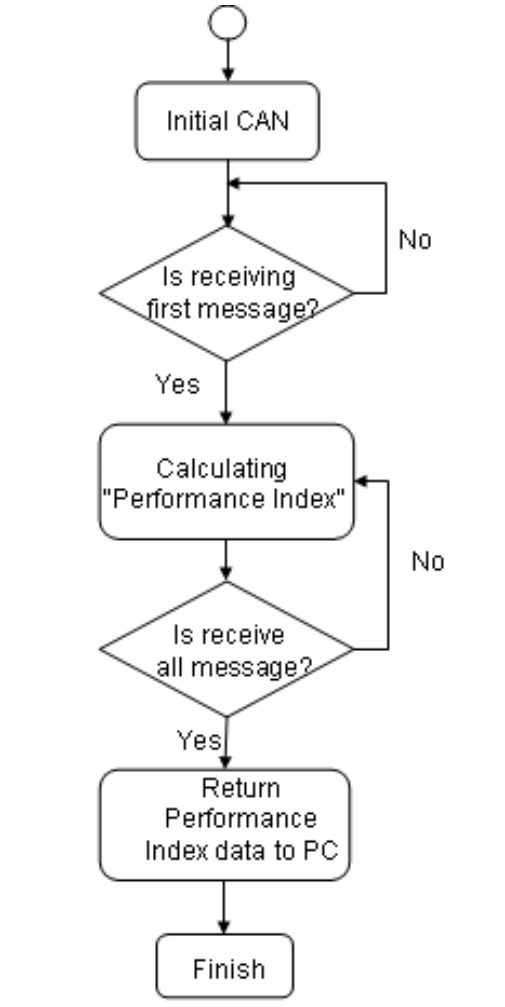


圖 5.6 程式接收端流程圖

5.2 硬體實驗問題

本節中將說明當應用 EDF 於硬體實驗時，所遇到的問題。原本 EDF 的中心理論是希望在訊息傳送不出去的時候，將訊息改以更高優先權，使訊息得以傳送出去。但由於 Stand-alone CAN controller SJA1000 的限制，訊息一旦正式開始傳送後，直到傳送結束無法中斷。此時可以停止 SJA1000 的方式是重新啟動 SJA1000，但若 BMS 上沒有連續 11 個 recessive bits(Bus free)，則 SJA1000 無法重新開始運作。在實驗進行中，BUS 上若一直有訊息在傳送，就無法達到重新啟動 SJA1000 的功能。因此無法在 SJA1000 的 CAN Controller 下實現 EDF。

但本論文中所提的 AAP 則沒有這個問題，AAP 的調整並不是即時的，是經過一段時間的累積。在訊息傳輸之前先設定好訊息的優先權，一旦設定之後，經過幾次的傳送，觀察新設定的仲裁欄的傳輸表現，才能再次更改優先權。因此在 SJA1000 中就沒有中斷正在傳輸訊息的狀況發生，所以能夠正常運作。

5.3 實驗數據

實驗是在傳輸速度125*Kbits/s*，總傳輸時間1秒鐘的狀況下進行。以不同的頻寬負載量以量測average transmitting time，missing deadline percentage，lost message percentage藉此比較其效能。實驗中使用的排程法有:RM、DM、AAP、RM+AAP與DM+AAP。

表5.1至表5.7分別為頻寬負載量在30%，50%，70%，90%，100%，110%，120%進行實驗的實驗數據。(a)表是所有Node的傳輸資料，總共有四個Node在傳送資料，傳輸資料包括:訊息仲裁欄(Identifier)，訊息的資料長度(data length)，訊息的傳送週期(period)，訊息的截止期限(deadline)。(b)表是模結果依據average transmitting time，missing deadline percentage，lost message percentage三項指標分析的數據。

表5.1(a) 傳輸速率125 *Kbits/s*時，30%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	10.2	4.2
2	0x40	16	6.2	1.2
3	0x60	16	4.2	1.7
4	0x80	16	5.9	3.9

表5.1(b) 傳輸速率125 *Kbits/s*時，30%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	224.504	211.216	240.826	196.667	239.501	222.831
Missing Deadline Perctntage(%)	1.349	1.199	1.499	4.049	1.499	1.349
Lost Message Percentage(%)	0	0	0	0	0	0

表5.2(a) 傳輸速率125 *Kbits/s*時，30%傳輸訊息量的硬體實驗結果

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	5.6	2.6
2	0x40	16	4.2	1.2
3	0x60	16	2.6	1.6
4	0x80	16	4.8	2.8

表5.2(b) 傳輸速率125 *Kbits/s*時，50%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	223.702	191.954	218.244	222.584	200.666	217.625
Missing Deadline Perctntage(%)	0.99	0.693	0.99	0.99	0.792	0.99
Lost Message Percentage(%)	0	0	0	0	0	0

表5.3(a) 傳輸速率125 *Kbits/s*時，70%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	4.1	2.1
2	0x40	16	2.3	1.3
3	0x60	16	1.8	0.9
4	0x80	16	3.1	1.1

表5.3(b) 傳輸速率125 *Kbits/s*時，70%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	258.231	240.302	250.128	213.297	380.367	270.884
Missing Deadline Perctntage(%)	2.536	0.963	1.028	3.174	1.6698	1.028
Lost Message Percentage(%)	1.220	0.963	0.064	0.835	0	0

表5.4(a) 傳輸速率125 *Kbits/s*時，90%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	3.0	1.5
2	0x40	16	1.8	0.8
3	0x60	16	1.4	0.9
4	0x80	16	2.5	1.6

表5.4(b) 傳輸速率125 *Kbits/s*時，90%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	254.740	354.493	286.842	367.867	340.288	331.053
Missing Deadline Perctntage(%)	1.767	4.362	1.790	3.149	5.719	2.002
Lost Message Percentage(%)	9.586	5.891	7.938	9.636	6.590	7.738

表5.5(a) 傳輸速率125 *Kbits/s*時，100%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	3.3	1.3
2	0x40	16	1.7	1.2
3	0x60	16	1.2	0.7
4	0x80	16	2.0	1.0

表5.5(b) 傳輸速率125 *Kbits/s*時，100%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	323.322	394.068	281.588	292.850	305.025	343.356
Missing Deadline Perctntage(%)	7.025	5.398	3.792	6.476	1.451	4.185
Lost Message Percentage(%)	18.075	15.872	15.917	18.075	7.014	16.187

表5.6(a) 傳輸速率125 *Kbits/s*時，110%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	2.6	1.6
2	0x40	16	1.5	1
3	0x60	16	1.1	0.6
4	0x80	16	2.1	0.9

表5.6(b) 傳輸速率125 *Kbits/s*時，110%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	275.004	354.877	343.177	197.032	251.107	292.552
Missing Deadline Perctntage(%)	5.774	6.652	4.922	3.337	4.423	3.080
Lost Message Percentage(%)	23.964	22.897	23.307	22.527	23.923	24.359

表5.7(a) 傳輸速率125 *Kbits/s*時，120%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	2.3	1.3
2	0x40	16	1.3	0.9
3	0x60	16	1.1	0.6
4	0x80	16	1.8	0.8

表5.7(b) 傳輸速率125 *Kbits/s*時，120%傳輸訊息量的硬體實驗結果

	Original	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time(μs)	381.546	378.887	361.267	287.409	307.204	300.543
Missing Deadline Perctntage(%)	8.124	8.289	7.407	6.443	6.984	6.200
Lost Message Percentage(%)	29.600	29.487	29.187	29.736	22.743	29.299

5.4 實驗結果分析

5.4.1 RM 與 DM 的比較

RM 與 DM 的不同在於：RM 以週期的長短安排訊息的仲裁欄;DM 以 deadline 的長短安排訊息的仲裁欄。將 5.3 節 Original，RM 與 DM 的實驗結果依據 BUS load 與三項指標（average transmitting time，missing deadline percentage，lost message percentage）的關係整理成圖 5.7。由圖 5.7(b)可知 DM 在降低 missing deadline 有顯著的效果。由圖 5.7(c)可知 RM 與 DM 都可以降低 lost message 的狀況發生，RM 改善的幅度比 DM 略大。

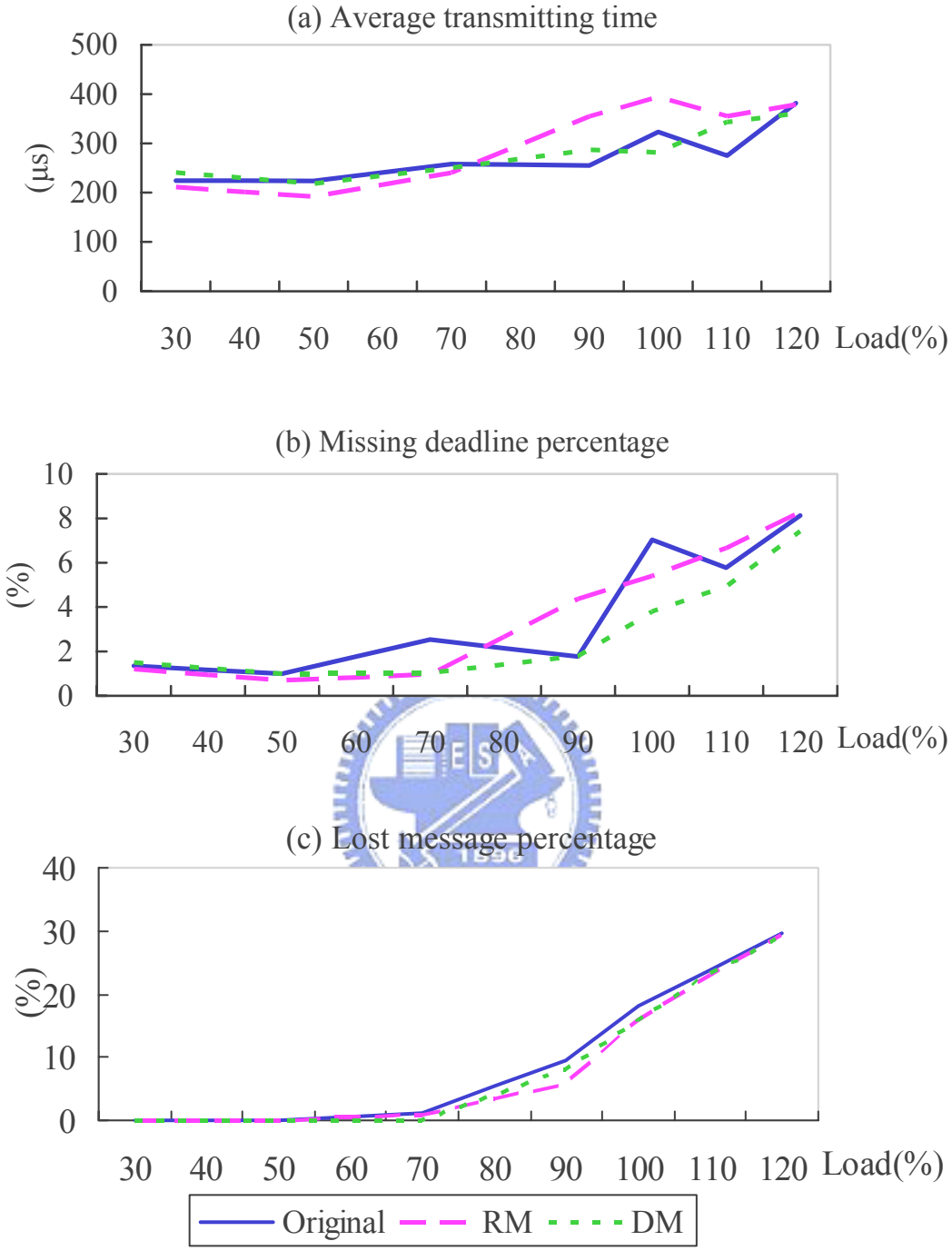


圖 5.7 RM，DM 與 Original 的傳輸效能比較

5.4.2 Active adjustable priority (AAP)

AAP 是一種動態調整訊息仲裁欄的方式。由 5.2 節可知 EDF 因為訊息傳送時無法在 SJA1000 的架構下中斷運作，而 AAP 則是以一段時間為基準，更改之後傳送的訊息的仲裁欄。將 5.3 節 Original 與 AAP 的實驗結果

依據 BUS load 與三項指標的關係整理成圖 5.8。由圖 5.8 所示 AAP 在高負載時，三項指標都有明顯的改善。在其他的情況下則沒有明顯的效果。

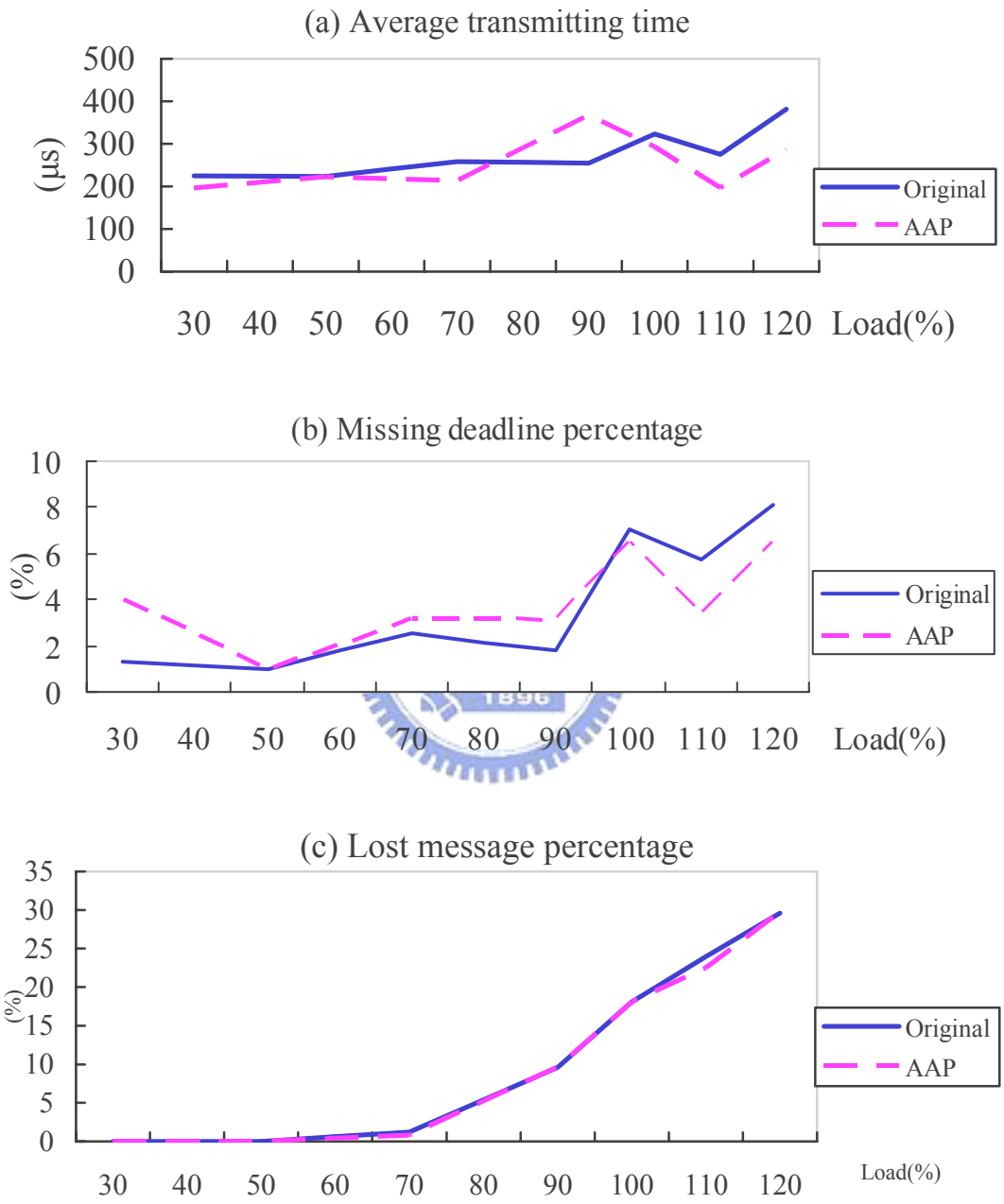


圖 5.8 Original 與 AAP 的比較

5.4.3 RM+AAP

如圖 5.9(c)可知 RM+AAP 對於降低 lost message percentage 有卓越的效果，效果比單純的 RM 或是 AAP 好。

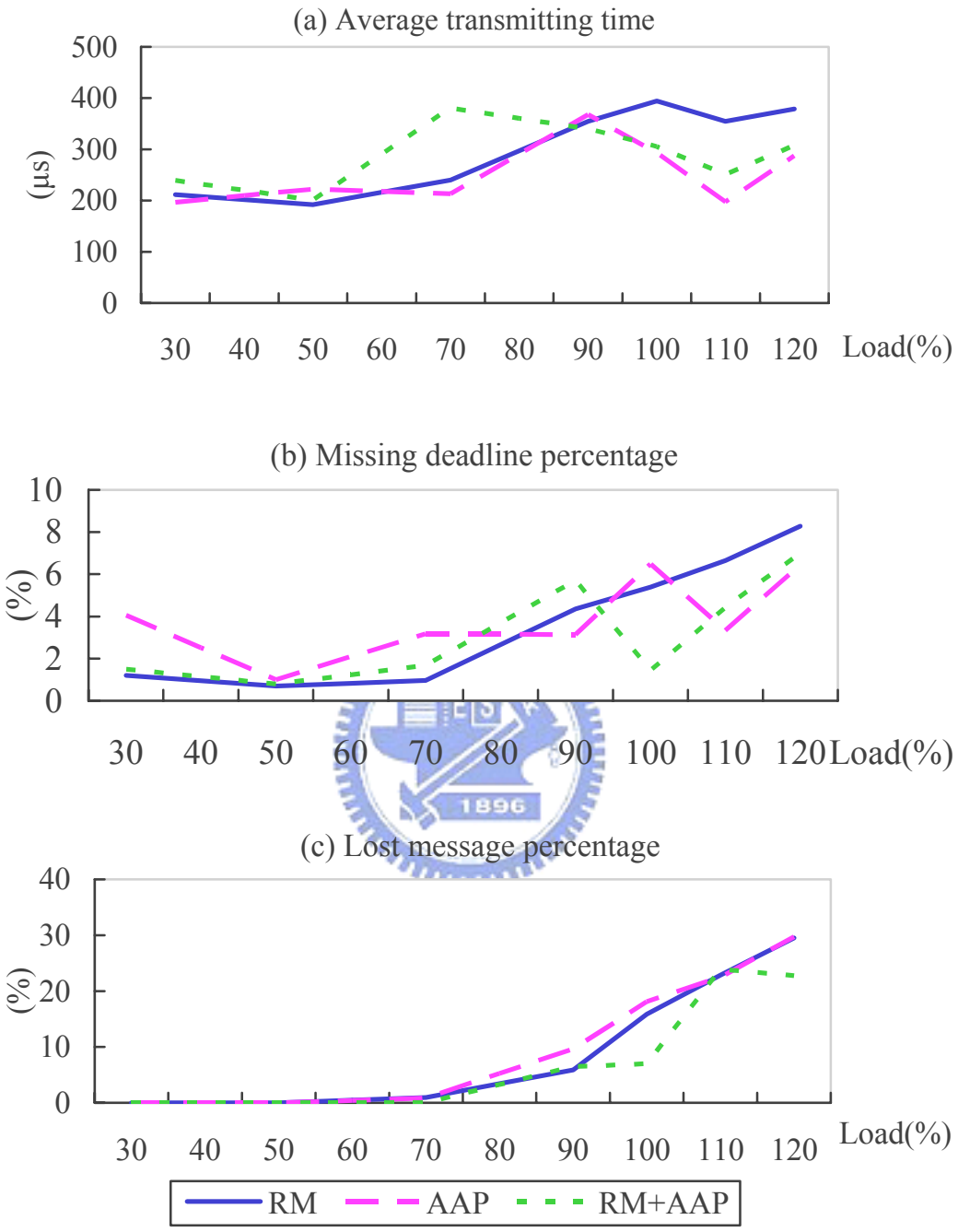


圖 5.9 RM，AAP 與 RM+AAP 的比較

5.4.4 DM+AAP

DM、AAP 與 DM+AAP 比較的結果如圖 5.10 所示。DM＋AAP 在降低 lost message percentage 的效果和 DM 差不多，兩者的效果都比 AAP 好。

但對於 missing deadline percentage 如圖 5.10(b)所示，DM 的效果最好，但 DM+AAP 在高負載時比起 DM 降低 missing deadline percentage 的效果更佳，而 AAP 在降低 missing deadline percentage 的效果沒有 DM 與 DM+AAP 好。

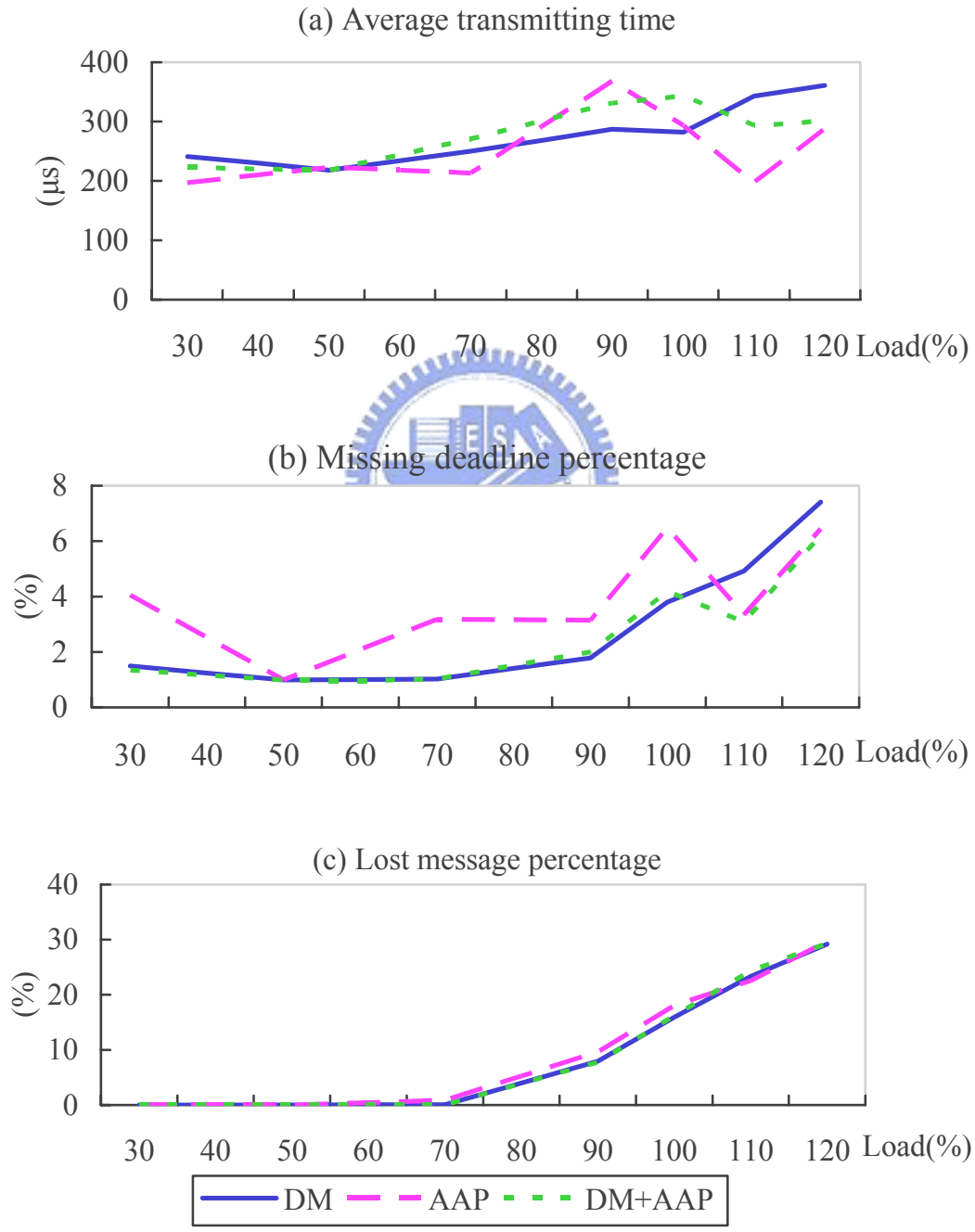


圖 5.10 DM，AAP 與 DM+AAP 的傳輸效能比較

5.4.5 RM+AAP 與 DM+AAP 的比較

圖 5.11 是由 5.3 節 Original、RM＋AAP 與 DM＋AAP 的實驗結果依據 BUS load 與三項指標的關係整理而成。由此圖可知若以 lost message percentage 為比較標準，則以 RM＋AAP 的對降低 lost message percentage 貢獻最大;以 missing deadline percentage 為比較標準，DM＋AAP 大致上有較佳的效果。

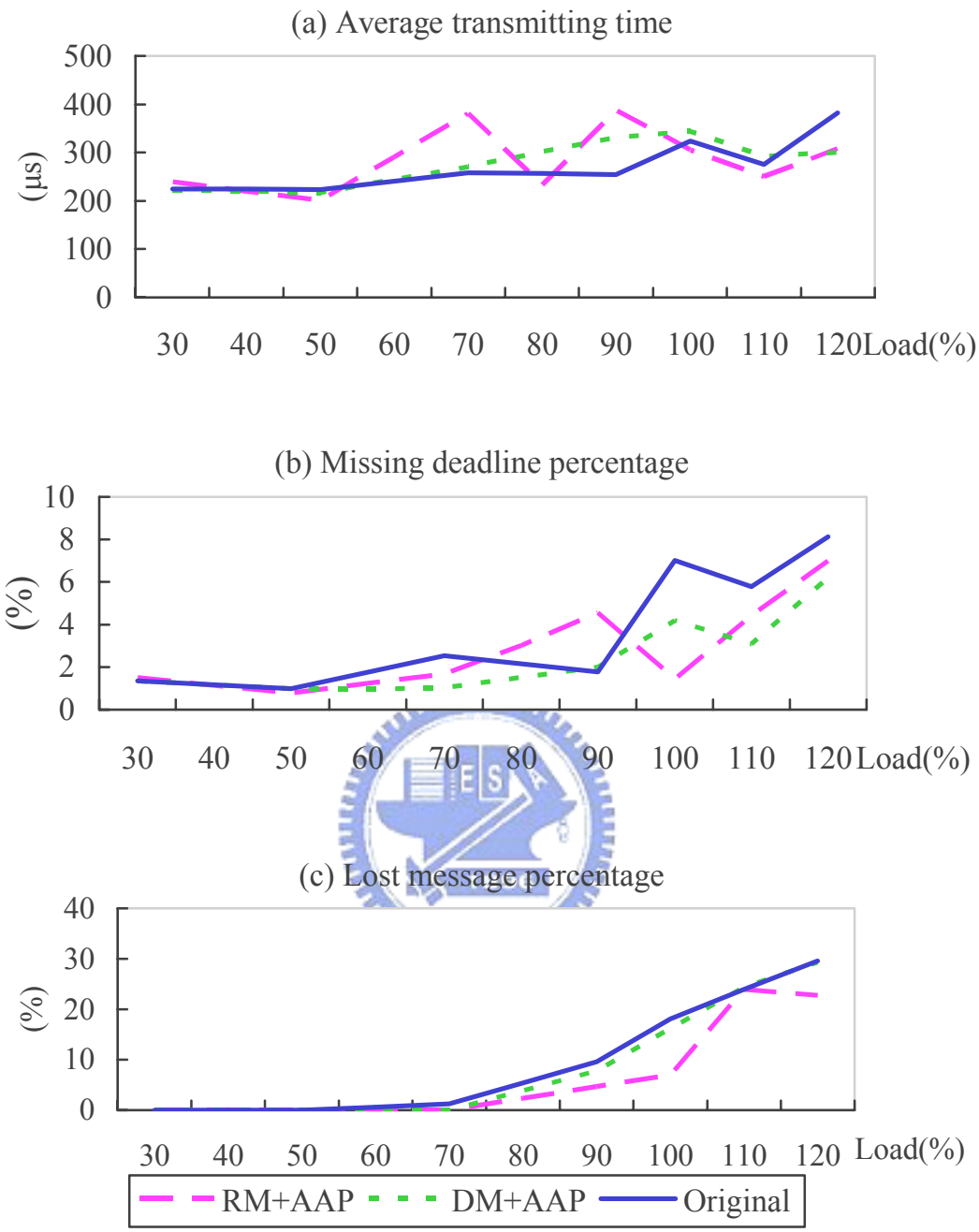


圖 5.11 Original、RM+AAP 與 DM+AAP 的比較

5.4.6 實驗結果重複性驗證

（1）將 RM＋AAP 在傳輸速率 125 *Kbits/s*，負載量 90%時，重複實驗十次得到表 5.8。

表5.8(a) 傳輸速率125 *Kbits/s*時，90%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	3.0	1.5
2	0x40	16	1.8	0.8
3	0x60	16	1.4	0.9
4	0x80	16	2.5	1.6

表5.8(b) 傳輸速率125 *Kbits/s*時，90%傳輸訊息量的十次硬體實驗結果

No.	1	2	3	4	5
Average Transmitting Time(μs)	340.288	331.004	304.797	334.552	343.700
Missing Deadline Perctntage(%)	5.719	5.446	5.443	5.387	5.556
Lost Message Percentage(%)	6.590	6.490	6.440	6.390	6.540
No.	6	7	8	9	10
Average Transmitting Time(μs)	316.323	313.943	330.373	344.178	305.350
Missing Deadline Perctntage(%)	5.559	5.455	5.559	5.493	5.446
Lost Message Percentage(%)	6.590	6.640	6.590	6.390	6.490

由表 5.8 計算平均可得：

average transmitting time(μs): 326.4508±21.6539

missing deadline percentage(%): 5.5063±0.2127

lost message percentage(%): 6.506±0.116

由上述的結果與表 5.4(b)與其他排程理論在 90%負載下的結果相比較，實驗的誤差並不影響排程理論比較的結果。且實驗結果有一定的重複性，其誤差約在±5%內。

（2）將訊息負載量 70%時，AAP 實驗時間 8 秒內的的區段變化整理成表 5.9。

表5.9(a) 傳輸速率125 *Kbits/s*時，70%的傳輸訊息量

Node	Identifier	data length (*bits*)	period (*ms*)	deadline (*ms*)
1	0x20	16	4.1	2.1
2	0x40	16	2.3	1.3
3	0x60	16	1.8	0.9
4	0x80	16	3.1	1.1

表5.9(b)　AAP在傳輸速率125kbits/s時，70%傳輸訊息量8sec的硬體實驗結果變化

time	0~1 sec	1~2 sec	2~3 sec	3~4 sec
Average Transmitting Time(μs)	213.297	195.345	224.443	189.785
Missing Deadline Perctntage(%)	3.174	3.617	3.325	3.814
Lost Message Percentage(%)	0.835	0.824	0.823	0.867
time	4~5 sec	5~6 sec	6~7 sec	7~8 sec
Average Transmitting Time(μs)	235.323	211.436	225.637	197.178
Missing Deadline Perctntage(%)	3.339	3.435	3.045	2.987
Lost Message Percentage(%)	0.834	0.789	0.824	0.809

由表 5.9 計算區段平均變化可得：

$$\text{average transmitting time}(\mu s): 211.555 \pm 23.736$$
$$\text{missing deadline percentage}(\%): 3.342 \pm 0.472$$
$$\text{lost message percentage}(\%): 0.826 \pm 0.041$$

由上訴的實驗結果，得知在不同的時間區段，實驗結果的誤差較大，但從表 5.9 可知，實驗結果沒有明顯發散的現象發生。

5.5 討論

由 5.4 節的分析結果，依據 average transmitting time，missing deadline percentage，lost message percentage 的表現整理出表 5.10。

表 5.10 硬體實驗綜合整理

	RM	DM	AAP	RM+AAP	DM+AAP
Average Transmitting Time	**	****	**	**	***
Missing Deadline Perctntage	**	****	**	***	*****
Lost Message Percentage	****	***	**	*****	****

[註] *worst ***fair *****best

本章中將排程理論在硬體實驗中驗證，EDF 在實際的硬體架構下不可行，其他 RM、DM 與 AAP 則可實際應用在硬體上。另外也將 RM、DM 與 AAP 結合並實際應用在硬體上。由表 5.9 做出下面的幾點結論:

1. DM 對於降低 missing deadline percentage 有很顯著的效果，對於 lost message percentage 也能有所改善。
2. DM+AAP 的結合比 DM 更能進一步的在高負載時降低 missing deadline percentage。
3. RM 雖然有改善 missing deadline percentage 的效果，但降低的效果沒有 DM 好。而 RM 對於改善 lost message percentage 有很好的效果。
4. RM+AAP 比起 RM 更能改善 lost message percentage，其效果是所有排

程方法中最好的。

5. 綜合上述幾點，RM+AAP 的整體效能最好，對於降低 lost message percentage 有最好的效果。DM+AAP 對於 missing deadline percentage 有最佳的效果。

第六章 結論

制定 deadline 的目的是為了系統的穩定，因為每個訊息所含的意義不同，對於傳送的時間要求也不同。如果訊息是作為回饋控制，未能及時傳遞時，會造成系統不穩定，有的訊息只是普通的資料，訊息較晚送達較沒問題。在軟體的實驗結果 RM 與 DM 對於改善 missing deadline 有不錯的效果，但 EDF 以及 AAP 則沒有效果。但以硬體的實驗結果而言，RM、DM、RM+AAP 與 DM+AAP 對於改善 missing deadline 的效果都不錯，其中以 DM+AAP 的效果最好。

在系統的觀點來看，若在傳輸中的遺失訊息的多寡，影響系統運作的穩定度，遺失的訊息越多，系統的運作也容易出問題。在硬體結果顯示 RM、DM、RM+AAP 與 DM+AAP 對於改善 lost message 的效果都不錯，但以 RM+AAP 比其他的排程法所遺失的訊息最少。因此若希望訊息在傳輸的過程中，減少訊息傳送不出去的現象發生，選擇以 RM+AAP 為主的排程法較佳。

在軟體的模擬和硬體的實驗有多處不同的結果，因為 EDF 在硬體實驗中受限於 SJA1000 的限制無法運作，所以在硬體實驗中動態調整的方法只有 AAP，AAP 單獨使用在軟體模擬或是硬體實驗效果都很差。在軟體模擬中 RM+AAP 與 DM＋AAP 的效果雖然比 AAP 好，但比單純的 RM 與 DM 差。但是 RM+AAP 與 DM+AAP 在硬體實驗中，比單純的 RM 與 DM 效果好，其原因可能是因為模擬軟體的架構問題，在模擬 CAN Node 傳輸訊息都是透過多工的執行緒，而不論是 EDF 或是 AAP 動態的調整帶傳輸

訊息的優先權都必須透過執行緒來調整，因此比起 RM 與 DM 執行緒必須多占 CPU 的資源，導致其訊息傳輸延遲或遺失的比較較多。

另外軟體模擬與硬體實驗結果的趨勢相比較，軟體模擬結果的趨勢走向較為平滑，硬體實驗常有跳動的現象，原因可能是因為硬體實驗的訊息組數較少，當訊息的組合不佳，才會有高負載表現比低負載好的現象發生。以網路控制系統而言，如果訊息是作為回饋控制，未能及時傳遞時，會造成系統不穩定，因為 DM+AAP scheme 能達成最少的 missing deadline 發生，因此在這種系統架構下選用 DM＋AAP 最能達成系統穩定。而訊息若是作為控制命令，控制命令如果遺失，系統無法做出回應動作造成系統有錯誤發生，因為 RM+AAP scheme 能達成最少的 lost message 發生，因此因此選用 RM＋AAP 可以使系統發生錯誤的機會最少。

附錄 A　CAN 實驗操作步驟

在附錄 A 中所引用的資料來源是 Philips 公司 Stand-alone CAN controller SJA1000 的 Data Sheet 與 Application Note [8][24]。

◆ CAN 節點架構圖

SJA1000 是 Standard-alone CAN controller 無法獨立運作，必須以 Host Controller 設定 SJA1000，並透過 Transceiver 上傳訊息到 CAN BUS 上。其架構如圖一所示。在 SJA1000 中訊息的傳送是由 Host Controller 透過 Interface Management Logic 寫入 Transmit Buffer 再由 CAN Core Block 傳送到 Transceiver 到 CAN BUS 上。訊息的接收則是 Transceiver 接收到訊息，透過 CAN Core Block 到 Acceptance Filter 判斷是否要接收訊息，若訊息通過 Acceptance Filter 傳到 Receive FIFO，再透過 Interface Management Logic 傳到 Host Controller。

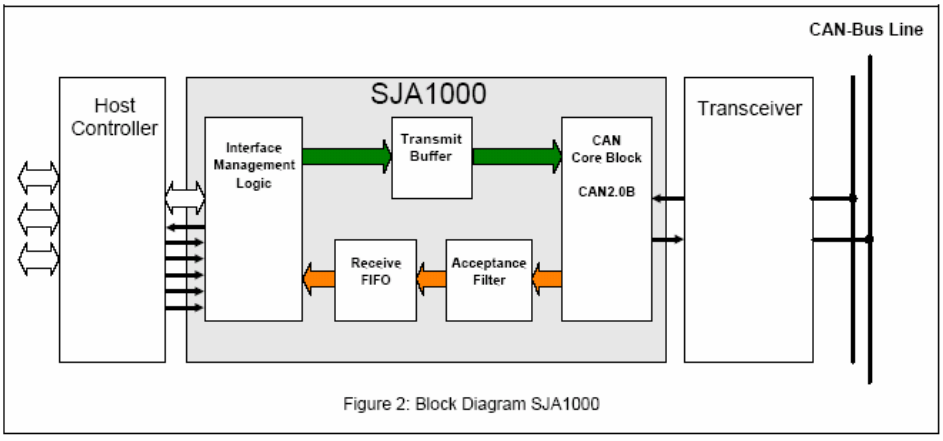


圖 A.1 CAN 節點架構圖 [8]

在本實驗設計中，節點是由 Host Controller 8051、Standard-alone CAN Controller SJA1000 與 Transceiver PCA82C251 構成，其接線圖如圖 A.2 所示。

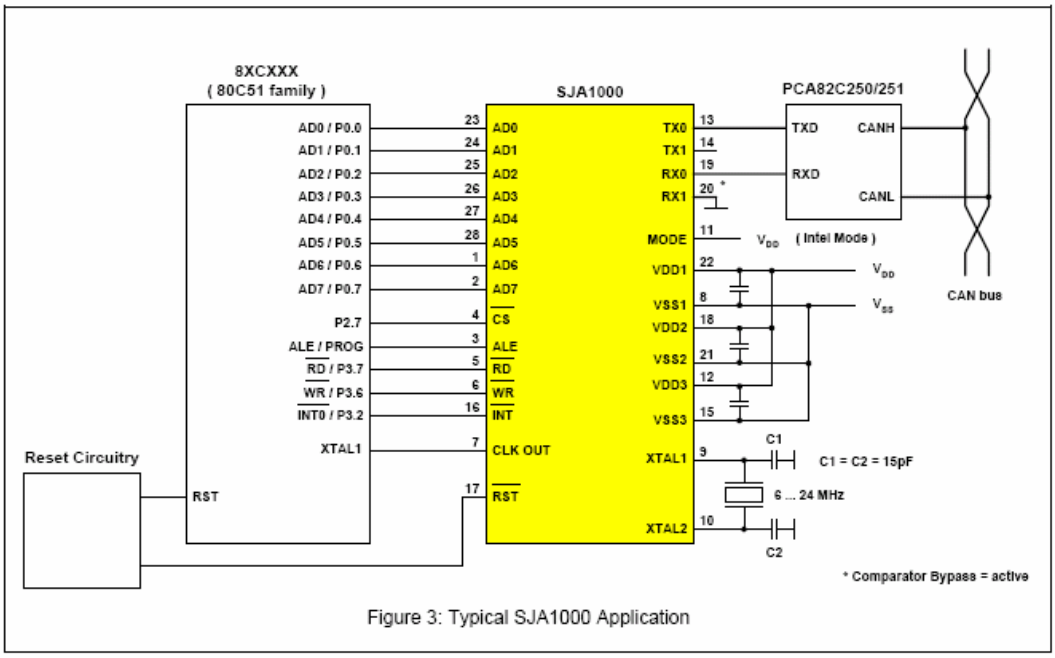


Figure 3: Typical SJA1000 Application

圖 A.2 CAN 節點接線圖 [8]

◆ CAN 資料欄框

在 CAN BUS 線上傳送資料的欄框，如圖 A.3 所示包含:欄框起始(Start of Frame)、仲裁區(Identify Field)、控制區(Control Field)、資料區(Data Field)、循環多餘碼區(CRC Field)、確認區(Acknowledgment Field)與欄框終止(End of Frame)。

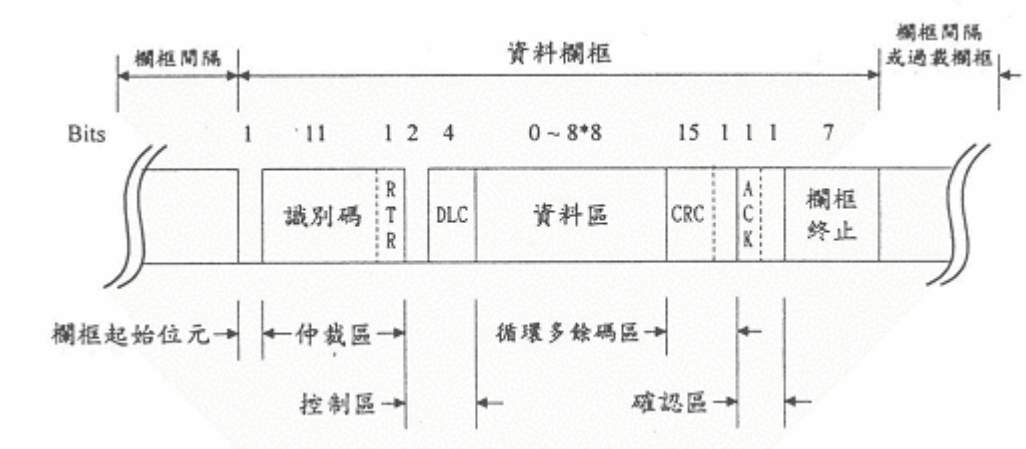


圖 A.3 CAN 資料欄框

◆ Acceptance Filter

資料欄框的接收與否是透過 Acceptance Filter 作判斷。Acceptance Filter 是由兩個暫存器:ACR 與 AMR 所控制。ACR 是 Acceptance Code register 設定要接收的訊息 Identifier。AMR 是 Acceptance Mask register 設定 Acceptance Filter 的遮罩。若 AMR 的位元設定為 1，則 ACR 的對應位元不論事 0 或 1 都可通過 Acceptance Filter，反之若 AMR 的位元設定為 0，則只有與 ACR 對應位元相同者，可通過 Acceptance Filter。Example 1 是在 PeliCAN Mode、Single Filter Mode 下的設定，其設定如圖 A.4 所示。

Example 1:

Let us assume, that the same 64 Standard Frame messages as described in the example on page 18 have to be filtered in PeliCAN mode.

This can be done using one long filter (Single Filter Mode).

The Acceptance Code Registers (ACRn) and Acceptance Mask Registers (AMRn) contain:

n	0	1 (upper 4 bits)	2	3
ACRn	01XX X010	XXXX	XXXX XXXX	XXXX XXXX
AMRn	0011 1000	1111	1111 1111	1111 1111
accepted messages (ID.28..ID.18, RTR)	01xx x010	xxxx		

("X" = irrelevant, "x" = don't care, only the upper 4 bits of ACR1 and AMR1 are used)

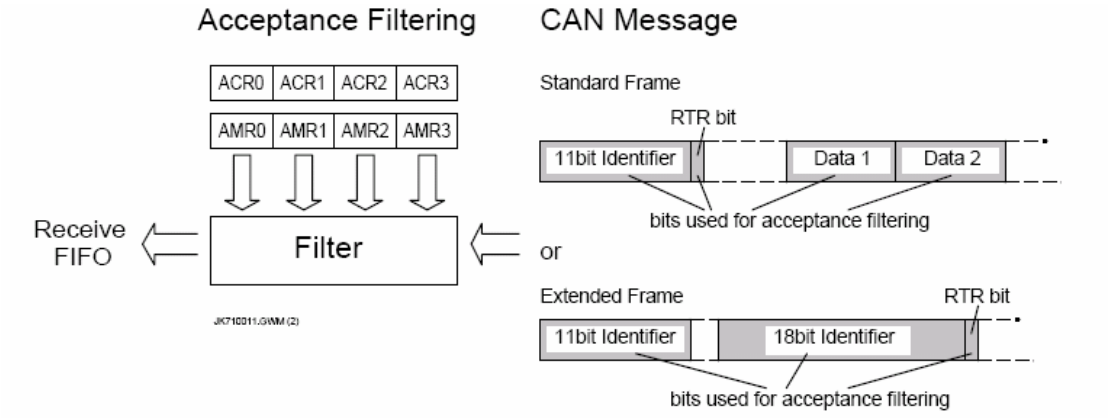


圖 A.4 PeliCAN Mode、Single Filter Mode 下 Acceptance Filter 的設定 [8]

Example: Receive ID: 0000 0000 001

```
AcceptCode0Reg   = 0x00;
AcceptCode1Reg   = 0x2F;
AcceptCode2Reg   = 0xFF;
AcceptCode3Reg   = 0xFF;
AcceptMask0Reg   = 0x00;        //only receive 0x00 0x2F
AcceptMask1Reg   = 0x1F;        //RTR don't care
AcceptMask2Reg   = 0xFF;
AcceptMask3Reg   = 0xFF;
```

Example: all pass filter

```
AcceptCode0Reg   = don't care
AcceptCode1Reg   = don't care
AcceptCode2Reg   = don't care
AcceptCode3Reg   = don't care
AcceptMask0Reg   = 0xFF;
AcceptMask1Reg   = 0xFF;        //RTR don't care
AcceptMask2Reg   = 0xFF;
AcceptMask3Reg   = 0xFF;
```

◆ Arbitration

若有多個資料欄框同時在 CAN BUS 上傳送，則決定傳送的訊息是 Arbitration Field 較小者優先權越高。其操作如圖 A.5 所示。

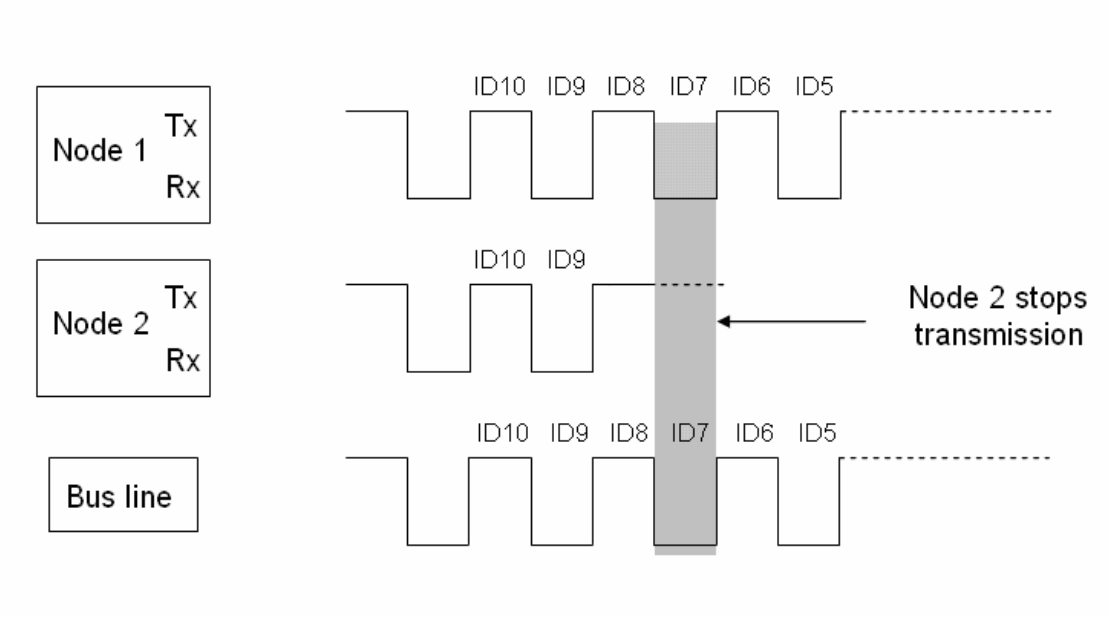


圖 A.5 位準仲裁圖

◆ SJA1000 的操作

■ SJA1000 有五種設定模式:Mode0(Reset Mode)、Mode1(Listen Only Mode)、Mode2(Self Test Mode)、Mode3(Acceptance Filter Mode) 與 Mode4(Sleep Mode)，五種操作模式的功能如表 A.1 所示，選擇何種操作模式是由 Mode control register 控制。

Example:

```
ModeControlReg = 0x01 ;        //Enter reset mode
ModeControlReg = 0x08;         // Enter Acceptance Filter Mode
                                  Select Single Filter
```

表 A.1 bit interpretation of Mode register [24]

BIT	SYMBOL	NAME	VALUE	FUNCTION
MOD.7	–	–	–	reserved
MOD.6	–	–	–	reserved
MOD.5	–	–	–	reserved
MOD.4	SM	Sleep Mode; note 1	1	sleep; the CAN controller enters sleep mode if no CAN interrupt is pending and if there is no bus activity
			0	wake-up; the CAN controller wakes up if sleeping
MOD.3	AFM	Acceptance Filter Mode; note 2	1	single; the single acceptance filter option is enabled (one filter with the length of 32 bit is active)
			0	dual; the dual acceptance filter option is enabled (two filters, each with the length of 16 bit are active)
MOD.2	STM	Self Test Mode; note 2	1	self test; in this mode a full node test is possible without any other active node on the bus using the self reception request command; the CAN controller will perform a successful transmission, even if there is no acknowledge received
			0	normal; an acknowledge is required for successful transmission
MOD.1	LOM	Listen Only Mode; notes 2 and 3	1	listen only; in this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully; the error counters are stopped at the current value
			0	normal
MOD.0	RM	Reset Mode; note 4	1	reset; detection of a set reset mode bit results in aborting the current transmission/reception of a message and entering the reset mode
			0	normal; on the '1-to-0' transition of the reset mode bit, the CAN controller returns to the operating mode

■ 在程式運作過程中如果要知道SJA1000的狀態，可由Status register得知，Status register的代表意思如表A.2所示。如要得知SJA1000的狀況，可油下列的Example來取得各種Status的狀態。

Example:

```
if (StatusReg&0x01){}      //SR0:receiver buffer status, 1:full, 0:empty
if (StatusReg&0x04){}      // SR2:transfer buffer status, 1:full, 0:empty
```

表 A.2 bit interpretation of Status register [24]

BIT	SYMBOL	NAME	VALUE	FUNCTION
SR.7	BS	Bus Status; note 1	1	bus-off; the CAN controller is not involved in bus activities
			0	bus-on; the CAN controller is involved in bus activities
SR.6	ES	Error Status; note 2	1	error; at least one of the error counters has reached or exceeded the CPU warning limit defined by the Error Warning Limit Register (EWLR)
			0	ok; both error counters are below the warning limit
SR.5	TS	Transmit Status; note 3	1	transmit; the CAN controller is transmitting a message
			0	idle
SR.4	RS	Receive Status; note 3	1	receive; the CAN controller is receiving a message
			0	idle
SR.3	TCS	Transmission Complete Status; note 4	1	complete; last requested transmission has been successfully completed
			0	incomplete; previously requested transmission is not yet completed
SR.2	TBS	Transmit Buffer Status; note 5	1	released; the CPU may write a message into the transmit buffer
			0	locked; the CPU cannot access the transmit buffer; a message is either waiting for transmission or is in the process of being transmitted
SR.1	DOS	Data Overrun Status; note 6	1	overrun; a message was lost because there was not enough space for that message in the RXFIFO
			0	absent; no data overrun has occurred since the last clear data overrun command was given
SR.0	RBS	Receive Buffer Status; note 7	1	full; one or more complete messages are available in the RXFIFO
			0	empty; no message is available

- 要使 SJA1000 運作必須透過 Command register 來操作，Command register 有五種操作法:CMR0(Transmission Request)、CMR1(Abort Transmission)、CMR2(Release Receive Buffer)、CMR3(Clear Data Overrun)、CMR4(Self Reception Request)。Command register 五種操作模式所代表的功能如表 A.3 所示。

Example:

```
/*a message want to transmit, set the Transmission Request*/
CommandReg = 0x01;   //CMR0:Transmission Request
                     //1:a message will be transmitted
/*after receive a message, release Receive Buffer*/
CommandReg = 0x04;   //CMR2:Release Receive Buffer
                     // 1:released, 0:no action
```

表 A.3 bit interpretation of Command register [24]

BIT	SYMBOL	NAME	VALUE	FUNCTION
CMR.7	–	reserved	–	–
CMR.6	–	reserved	–	–
CMR.5	–	reserved	–	–
CMR.4	SRR	Self Reception Request; notes 1 and 2	1	present; a message shall be transmitted and received simultaneously
			0	– (absent)
CMR.3	CDO	Clear Data Overrun; note 3	1	clear; the data overrun status bit is cleared
			0	– (no action)
CMR.2	RRB	Release Receive Buffer; note 4	1	released; the receive buffer, representing the message memory space in the RXFIFO is released
			0	– (no action)
CMR.1	AT	Abort Transmission; notes 5 and 2	1	present; if not already in progress, a pending transmission request is cancelled
			0	– (absent)
CMR.0	TR	Transmission Request; notes 6 and 2	1	present; a message shall be transmitted
			0	– (absent)

- Clock divider register 用來選擇 SJA1000 所使用的 CAN 模式 (1:PeliCAN, 0:BasicCAN)，選擇內部或外部震盪器，以及 Clock out frequency。其功能如表 A.4 所示。

Example:

```
ClockDivideReg = 0xC8 ;        //Peli CAN mode, Rx0 is active Rx1
                                 ground,
                               //External CLKOUT off
                               //Clock out frequency = fosc/2
```

表 A.4 bit interpretation of Clock divider register [24]

Table 49 Bit interpretation of the clock divider register (CDR); CAN address 31

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CAN mode	CBP	RXINTEN	(0)(1)	clock off	CD.2	CD.1	CD.0

Table 50 CLKOUT frequency selection; note 1

CD.2	CD.1	CD.0	CLKOUT FREQUENCY
0	0	0	$\frac{f_{osc}}{2}$
0	0	1	$\frac{f_{osc}}{4}$
0	1	0	$\frac{f_{osc}}{6}$
0	1	1	$\frac{f_{osc}}{8}$
1	0	0	$\frac{f_{osc}}{10}$
1	0	1	$\frac{f_{osc}}{12}$
1	1	0	$\frac{f_{osc}}{14}$
1	1	1	f_{osc}

Note

1. f_{osc} is the frequency of the external oscillator (XTAL).

■ The output control register 是利用軟體來設定不同的 output driver。其功能如表 A.5 所示。

Example:

```
OutControlReg    = 0x1A ;       //normal output mode
                                //Tx0 Push-pull Tx1 Float
```

表 A.5 bit interpretation of output control register [24]

Table 46 Bit interpretation of the output control register (OCR); CAN address 8

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OCTP1	OCTN1	OCPOL1	OCTP0	OCTN0	OCPOL0	OCMODE1	OCMODE0

Table 47 Interpretation of OCMODE bits

OCMODE1	OCMODE0	DESCRIPTION
0	0	bi-phase output mode
0	1	test output mode; note 1
1	0	normal output mode
1	1	clock output mode

◆ SJA1000 bit time

要設定 SJA1000 的 bit time 要設定的暫存器: bus timing register 0(BTR0) 與 bus timing register 1(BTR1)。其計算方式如圖 A.6 所示。

Table 44 Bit interpretation of bus timing register 0 (BTR0); CAN address 6

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

$$t_{scl} = 2 \times t_{CLK} \times (32 \times BRP.5 + 16 \times BRP.4 + 8 \times BRP.3 + 4 \times BRP.2 + 2 \times BRP.1 + BRP.0 + 1)$$

where t_{CLK} = time period of the XTAL frequency = $\frac{1}{f_{XTAL}}$

Table 45 Bit interpretation of bus timing register 1 (BTR1); CAN address 7

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

TSEG1 and TSEG2 determine the number of clock cycles per bit period and the location of the sample point, where:

$$t_{SYNCSEG} = 1 \times t_{scl}$$

$$t_{TSEG1} = t_{scl} \times (8 \times TSEG1.3 + 4 \times TSEG1.2 + 2 \times TSEG1.1 + TSEG1.0 + 1)$$

$$t_{TSEG2} = t_{scl} \times (4 \times TSEG2.2 + 2 \times TSEG2.1 + TSEG2.0 + 1)$$

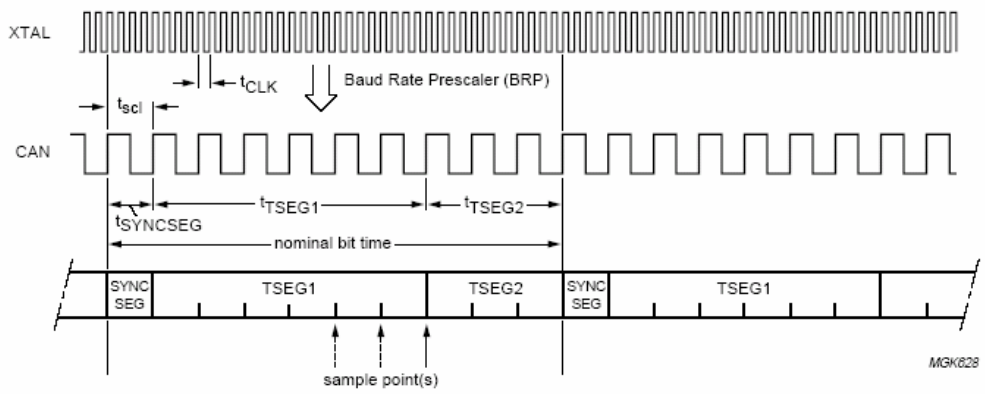


圖 A.6 bit time 的計算方法 [24]

Example 1 : 500kbits/sec, BTR0=0x01, BTR1=0x18, $f_{XTAL} = 12MHz$

$$t_{CLK} = \frac{1}{f_{XTAL}} = \frac{1}{12\times10^6} \text{ (sec)}$$

$$t_{scl} = 2\times t_{CLK} \times (32\times BRP.5+16\times BRP.4+8\times BRP.3+4\times BRP.2+2\times BRP.1+BRP.0+1)$$

$$= 2\times\frac{1}{12\times10^6}\times(32\times0+16\times0+8\times0+4\times0+2\times0+1+1)$$

$$= \frac{1}{6\times10^6} \text{ (sec)}$$

$$t_{SYNCSEG} = \frac{1}{6\times10^6}$$

$$t_{TSEG1} = t_{scl} \times (8\times TSEG1.3+4\times TSEG1.2+2\times TSEG1.1+TSEG1.0+1)$$

$$= \frac{1}{6\times10^6}(8\times1+4\times0+2\times0+0+1)$$

$$= \frac{9}{6\times10^6} \text{ (sec)}$$

$$t_{TSEG2} = t_{scl} \times (4\times TSEG2.2+2\times TSEG2.1+TSEG2.0+1)$$

$$= \frac{1}{6\times10^6}(4\times0+2\times0+1+1)$$

$$= \frac{2}{6\times10^6} \text{ (sec)}$$

normal bit time $= t_{SYNCSEG} + t_{TSEG1} + t_{TSEG2}$

$$= \frac{1}{6\times10^6}\times(1+9+2)$$

$$= \frac{12}{6\times10^6}$$

$$= 2\times10^{-6} \text{ sec}$$

transmitting rate $= \frac{1}{2\times10^{-6}}$

$$= 500kbits/\text{sec}$$

◆ 初始化 SJA1000 的設定流程

The host controller has to configure the following registers of the control segment of the SJA1000 in reset mode:

- Mode Register (in PeliCAN mode only), selecting the following modes of operation for this application
  – Acceptance Filter mode
  – Self Test mode
  – Listen Only mode
- Clock Divider Register, defining
  – if the BasicCAN or the PeliCAN mode is used
  – if the CLKOUT pin is enabled
  – if the CAN input comparator is bypassed
  – if the TX1 output is used as a dedicated receive interrupt output
- Acceptance Code and Acceptance Mask Registers
  – defining the acceptance code for messages to be received
  – defining the acceptance mask for relevant bits of the message to be compared with corresponding bitsof the acceptance code
- Bus Timing Registers, see also [6]
  – defining the bit-rate on the bus
  – defining the sample point in a bit period (bit sample point)
  – defining the number of samples taken in a bit period
- Output Control Register
  – defining the used output mode of the CAN bus output pins TX0 and TX1 Normal Output Mode, Clock Output Mode, Bi-Phase Output Mode or Test Output Mode
  – defining the output pin configuration for TX0 and TX1 Float, Pull-down, Pull-up or Push/Pull and polarity

Example: CAN initial 程式碼

```
ModeControlReg = 0x01 ;        // Enter reset mode
ClockDivideReg = 0xC8 ;        //Peli CAN mode, Rx0 is active Rx1 ground
                               //External CLKOUT off
//ACR|AMR
    AcceptCode0Reg   = 0xFF;
    AcceptCode1Reg   = 0x00;
    AcceptCode2Reg   = 0xFF;
    AcceptCode3Reg   = 0xFF;
```

```
AcceptMask0Reg   = 0x00;
AcceptMask1Reg   = 0x1F;
AcceptMask2Reg   = 0xFF;
AcceptMask3Reg   = 0xFF;

// External Oscillator 12MHz, bit-Rate 250k bps
BusTiming0Reg    = 0x00 ;
BusTiming1Reg    = 0x18 ;

OutControlReg    = 0x1A ;       //normal output mode
                                //Tx0 Push-pull Tx1 Float
ModeControlReg = 0x08;          // Single Filter
```

◆ Transmit Message 設定流程

- The host controller has to check the "Transmit Buffer Status" flag (TBS) of the Status Register, if a new message can be place replaced into The Transmit Buffer.
- Write TxFrameInfo, TxIddetify, TxBuffer to the Transmit Buffer.
- Set Transmit Request.
- Wait Transmit Complete.

Example: Transmit 程式碼

```
void CAN_TRAN(Byte ID, Byte CANData)
{
    Byte Tmp1,Tmp2;
    if( StatusReg & 0x04)    // Transmit Buffer Status released
    {
        Tmp1=ID>>3;// ID.28 ID.27 ID.26 ID.25 ID.24 ID.23 ID.22 ID.21
        Tmp2=ID<<5;// ID.20 ID.19 ID.18     X     X     X     X     X
        TxFrameInfo = 0x02 ;    // FF RTR X X DLC.3 DLC.2 DLC.1 DLC.0
        TxIdentify1 = Tmp1 ;
        TxIdentify2 = Tmp2 ;
        TxBuffer01 =    CANData ;
        TxBuffer02 =    CANData ;
```

```
            CommandReg = 0x01;    //CMR0:Transmission Request    1:a
                                  //message will be transmitted
            while ( !(StatusReg & 0x08) );    //wait complete transimt
        }
    }
```

◆ Receive Message 的設定流程

- The host controller has to check the “Receive Buffer Status” flag (RBS) of the Status Register. Is The Receive Buffer is full?
- Receive data from The Receive Buffer.
- After take the receive data, release The Receive Buffer.

Example: Receive 程式碼

```
void CAN_RECE(void)
{
 Byte Tmp1,Tmp2;
 if(StatusReg&0x01)       //SR0:receiver buffer status, 1:full, 0:empty
 {
      Tmp1=RxIdentify1<<3;
      Tmp2=RxIdentify2>>5;
      CANID=Tmp1|Tmp2;
      UartByteHex(CANID);         //output receive CANID
      CommandReg = 0x04;          //CMR2:Release Receive Buffer, 1:released,
                                    0:no action
 }
}
```

參考文獻（Reference）

[1] Road Vehicles-Interchange of Digital Information- Controller Area Network (CAN) for High-Speed Communication. ISO 11898, 1st ed., 1993.

[2] K. Etschberger, *Controller Area Network*, Germany: IXXAT Press, 2001.

[3] F. L. Lian, J. R. Moyne, and D. M. Tilbury, “Performance evaluation of control networks: Ethernet, ControlNet and DeviceNet,” *IEEE Contr. Syst. Mag.*,vol. 21, no. 1, pp. 66-83, Feb, 2001

[4] J. Schill, **“**An overview of the CAN protocol,” *Embedded System Programming*,vol. 10, no. 9, pp. 46-62, 1997.

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

[6] T. Fuhrer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther, Robert Bosch GmbH, “Time Triggered Communication on CAN,” *Proceedings 7th International CAN Conference*, pp. , 2000.

[7] F. Hartwich, et al, ”CAN Network with Time triggered Communication,” *Proceedings of ICC’2000*, Amsterdam, The Netherlands, 2000.

[8] P. Hank, and E. JÖhnk, Systems Laboratory Hamburg Germany, “Application Note : SJA1000 Stand-alone CAN Controller(AN7076)”, 1997.

[9] F. Hartwich, T. Fhrer, R. Hugel, B. M.ller, Robert Bosch GmbH, “Timing in the TTCAN Network,” *Proceedings 8th International CAN Conference*, Las Vegas, 2002.

[10] C. Liu, and J. Layland, “Scheduling Algorithms for Multi-programming in a Hard Real-Time Environment,” *Journal of ACM*, vol. 20, no. 1, pp. 46-61, Jan. 1973.

[11] J. Lehoczky, L. Sha, and Y. Ding, “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior,” Proceedings of the IEEE Real-Time Systems Symposium, pp.166-171, 1989.

[12] K. M. Zuberl, and K. G. Shin, “Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications,” *Proceedings of Real-Time Technology and Applications Symposium*, pp.240-249, May 1995.

[13] K. M. Zuberl, and K. G. Shin, “Scheduling Messages on Controller Area Network for Real-Time CIM Applications,” *IEEE Transactions on Robotics and Automation*,vol. 13, no. 2, pp.310-314, April 1997.

[14] K. M. Zuberl, and K. G. Shin, “Design and Implementation of Efficient Message Scheduling for Controller Area Network,” *IEEE Transactions on Computers*,vol. 49, no. 2, pp.182-188, February 2000.

[15] M. D. Natale, “Scheduling the CAN Bus with Earliest Deadline Techniques,” *in Proc. 21st IEEE Real-Time Systems Symp.*, Orlando, FL, pp.259-268, Nov. 2000.

[16] P. Pedreiras, and L. Almeida, “EDF Message Scheduling on Controller Area Network,” *Computing & Control Engineering Journal*,vol. 13, no. 4, pp.163-170, August 2002.

[17] L. Almeida, R. Pasadas, J.A. Fonseca, “Using the Planning Scheduler in Real-Time Fieldbuses: Theoretical Model for Run-Time Overhead,” Factory *Communication Systems Proceedings of WFCS’97*, Barcelona, Spain, pp.103-109, 1997.

[18] L. Almeida, R. Pasadas, J.A. Fonseca, “Using a Planning Scheduler to Improve Flexibility in Real Time Fieldbus Networks,” *Control Engineering*

Practice,vol. 7, pp. 101-108, 1999.

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

[20] R. Dobrin, and G. Fohler, "Implementing Off-line Message Scheduling on Controller Area Network (CAN)," 8th conference on *IEEE Emerging Technologies and Factory Automation*, vol. 1, pp.241-245, 2001.

[21] K. Tindell, A. Burns, and A.J. Wellings, "Calculating Controller Area Network (CAN) Message Response Times," *Control Eng. Practice*,vol. 3, no. 8, pp.1163-1169, 1995.

[22] H. A. Hansson, and T. Nolte, "Integrating Reliability and Timing Analysis of CAN-Based Systems," *IEEE Transactions on Industrial Electronics*,vol. 49, no. 6, pp.1240-1250, December 2002.

[23] Port GmbH, Halle, "User Manual CANopen Library," CANopen Library Version 4.2, 2001.

[24] Philips Semiconductors, "SJA1000 Stand-alone CAN Controller Data Sheet," Product Specification File under Integrated Circuits IC18, Jan. 2000.