EPCIO 即時性運動控制函式庫介紹與實作

Introduction and Implementation of EPCIO Real-Time Motion Control Library

工業技術研究院　機械所　機電控制整合部　李桂銘

摘要

由於 Windows 並非是一個即時性的作業系統，所以使用 PC-Based 運動控制器搭配 Windows 用於工業控制的環境時，面對高精高速的需求，往往因爲即時性不足而功敗垂成。有鑑於此，一些 Third-party 的協力廠商針對即時性的問題已經提供解決方案，改善了 Windows 即時性不足的缺陷，在 PC-Based 的運動控制器奠定了更穩固的基礎。

本文將介紹利用工研院機械所研發的 PC-Based 完整解決方案 EPCIO[1]，結合即時性子系統 RTX，開發具即時性的運動控制函式庫。

關鍵字

EPCIO(Exquisite Position Control and Input/Output)、RTX(Real Time extension)、硬即時(Hard Real-Time)、PC-Based

前言

PC-based 的運動控制器雖然問市已久，但在工業控制領域當中，能佔有一席之地，其原因在於 PC 具有多種周邊介面、模組化的特性，另外有豐富的軟體資源及自由軟體(Open source)作爲後盾，再加上 PC 的設計日益精進的情況下，以往可能造成系統不穩定的因素，如震動、高溫、雜訊等等，目前都已經得到良好的解決。

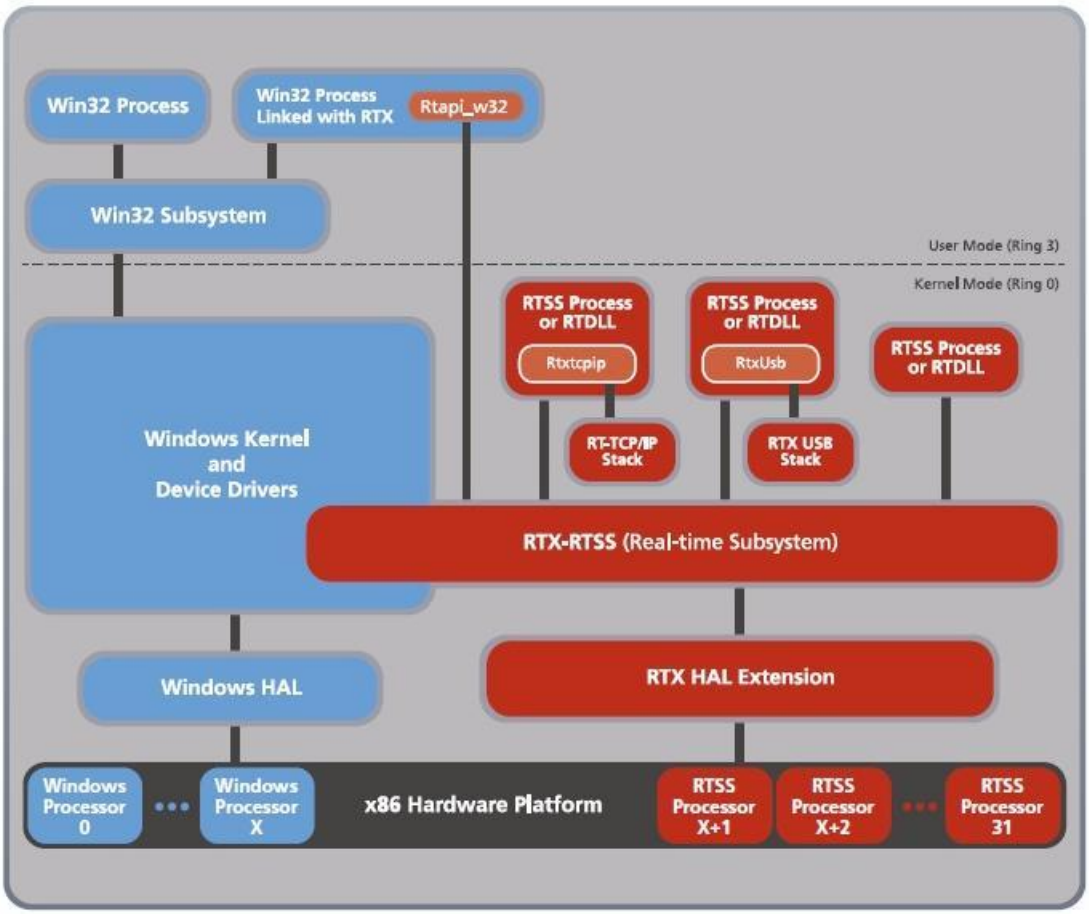
以作業系統而言，微軟的 Windows 系列的作業系統，在目前市佔率仍居於龍頭的地位，包含 Windows 2000、Windows XP、Windows XP Embedded、WinCE 到目前的 Windows 7 等等；許多的應用根據以上作業系統去做開發，而在即時性需求較高的系統，如工業控制、軍事、航太、交通、醫療等，也有越來越多的應用是基於 Windows 上開發，主要是 Windows 作業系統具有以下的優勢：(1)眾多熟悉 Win32 API 的開發人員。(2)直覺的人機介面及各類通訊介面。(3) PC 開放性

的架構。(4)大量的開發工具。企業無論在開發或使用都可以節省許多資源，包含人員的訓練及開發經費。Windows 挾帶了廣大的使用者，以及眾多熟悉 Win32 API 的軟體工程師，在 PC-based 的運動控制方案當中仍是主要的選擇。

但也由於 Windows 的設計是以個人電腦爲出發，並非以工業使用做爲考量，所以在即時性的表現在並未能滿足”Hard Real-Time”的需求，充其量只能算是”Soft Real-Time”的系統。爲了解決即時性的問題，Third-party 的廠商基於 Windows 的架構，開發出具有即時性的子系統，例如 IntervalZero 的 RTX(Real Time Extension)、Radisys corporation 的 InTime 等等，在此子系統的環境下所執行的行程，將具有低於 1ms 等級的”Hard Real Time”能力，Timer 的精準度可以到達 1us，使得以往在 PC-Based 無法達到的高階應用，透過即時性的子系統得以實現。

RTX 介紹[2][3]

RTX 是美國 IntervalZero 公司開發的，根據 Windows 系統的提出的硬即時的解決方案，RTX 並不是一個獨立的作業系統，它是 Windows 上的一個擴充子系統(Extension Subsystem)，RTX 本身並不會更改 Windows 本身排程的機制，而是加入一個即時的硬體抽象層(HAL)，透過上述的方式，使得運行在 RTSS(Real-Time SubSystem)底下的執行緒所擁有的優先權，高於運行在 Windows 底下的執行緒。



圖一 RTX 核心架構圖[4]

上圖一爲 RTX 的核心架構圖，RTX 提供 Clock、Timer、中斷管理、I/O、記憶體存取等機制，確保任務(task)即時的可靠性；另外提供的物件包含 RTSS Process、RTSS Thread、System Memory、Port I/O 等，所有 RTSS 提供的物件皆能直接與 HAL 溝通，也因此具備直接驅動周邊硬體的能力。在 RTX 2009 之後的版本，有支援對稱多處理組態(SMP: Symmetric Multi-Processing)，分爲兩種操作模式 (1)Shared Mode: RTSS 子系統與 Windows 共用一個 CPU 核心(2)Dedicated Mode:系統核心分割成某些由 RTSS 子系統專用，而其餘的由 Windows 專用。

目前主流的 CPU 都是多核心的，RTX 充分利用 SMP 的優點，支持多核心 CPU；在 Dedicated Mode 當中，可以指定 RTX 程序運行在哪些核心上，以 8 核心的 CPU 爲例，Windows 必須使用其中 1 個核心，RTX 可以獨佔剩餘的 1 至 7 個核心，Windows 本身是看不見被 RTX 獨佔的核心。

另外 RTX 對於網路的部分，提供了 RT-TCP/IP 的通訊協定，對於新一代全數位伺服網路而言，更具一大優勢。

PCI 協定[5][6]

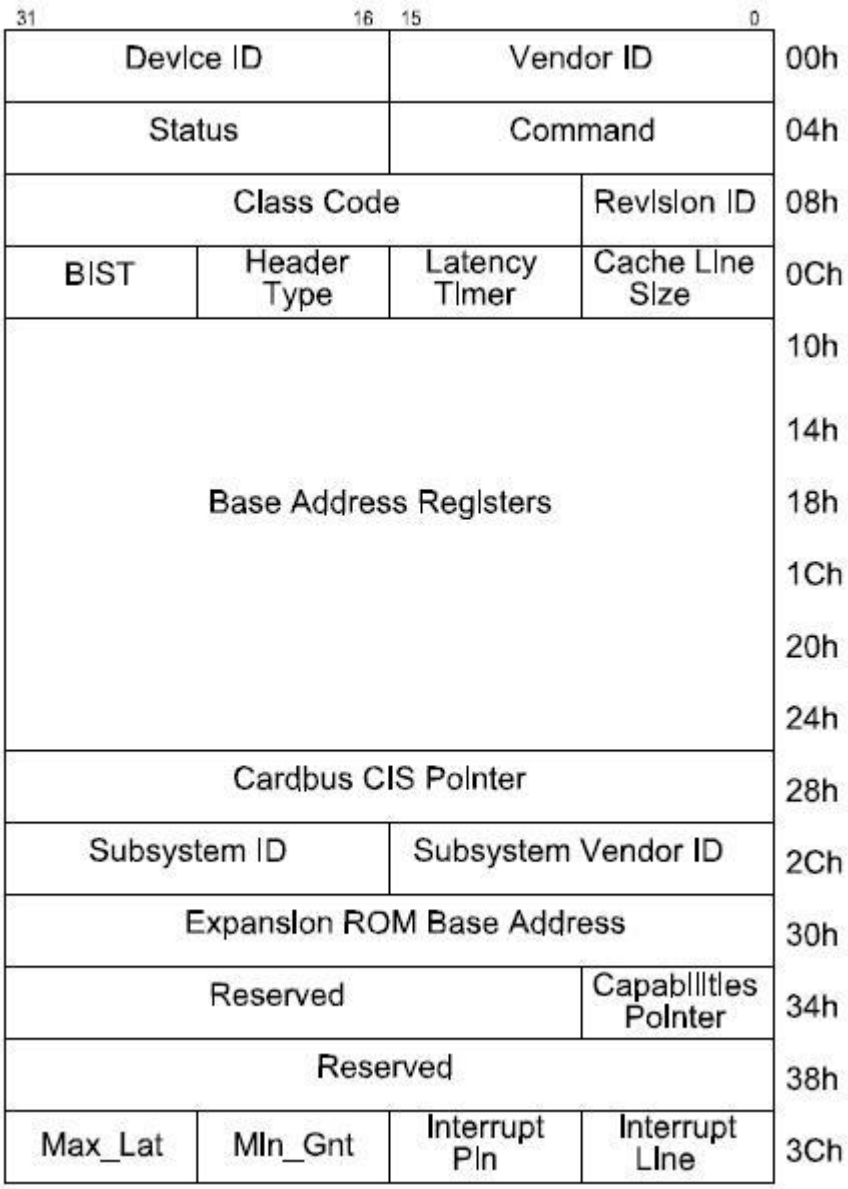
EPCIO 系列的運動控制卡有分爲：

1. ISA 介面(EPCIO-601、EPCIO-605、EPCIO-400、EPCIO-405)
2. PCI 介面(EPCIO-6000、EPCIO-6005、EPCIO-4000、EPCIO-4005)

要撰寫 EPCIO 運動控制卡驅動程式，就必須瞭解底層的匯流排協定，ISA 匯流排在現今主流的主機板上已經很少見了，所以底下就針對 PCI 協定做簡介。

PCI(Peripheral Component Interconnect) 匯流排是 PC 上最基本的匯流排，傳輸速率可達 133MB/s，許多匯流排也是掛在 PCI 匯流排上，如 USB 匯流排，是透過 USB HOST 裝置掛在 PCI 匯流排上。

PCI 匯流排定義了一組規格暫存器(PCI Configuration)，只要是 PC 端的系統軟體符合 PCI 的軟體規範，即可達成即插即用的功能(PnP)，PCI Configuration Register 基本上是一個 64 dword 的組態空間(Configuration Sapce)，其格式分爲 Header Type 0, Header Type1, Header Type2，大多數的 PCI 元件 Configuration Register 皆屬於 Header Type 0，下圖二爲 Type0 的配置空間圖，詳細資料可參考 PCI 規格書[7]。



圖二 PCI 配置空間

由上圖二可以得到以下資訊：

(1) Vendor ID (製造商識別碼)：

由 PCI SIG(PCI Special Interest Group)分配用以識別該 PCI device 製造商，若欲取得專用的 Vendor ID，則需加入 PCISIG 會員。若爲 FFFF 爲非法廠商 ID，用來判斷 PCI 裝置是否存在。

(2) Device ID (裝置識別碼)

由製造商自行指派。作業系統根據 Vender ID、Device ID 來找到相對應的驅動程式。

(3) Command (指令暫存器)

PC 與 PCI 間溝通的設定。

(4) Status (狀態暫存器)

表示目前資料傳輸狀態。

(5) Revision ID (版本識別碼)

由設備製造商自行指派，用以識別產品版本。

(6) Class Code (類別碼)

設定產品類別，需依規格做設定。

(7) Header Type

設定 PCI Configuration 之 Type，以及表示 PCI device 是單功能或多功能裝置。

(8) Base Address (基底位址)

基底位址用來存放 PCI 裝置硬體的記憶體位址，或者使用 I/O 空間的第一個位址，PCI 規範提供一種機制，使得 I/O 與記憶體分開，如果最低的位元爲 0，表示該基底位址表指向的是一個記憶體空間，如果是 1，則指向的是一個 I/O 空間。

(9) Interrupt Pin: (中斷腳位)

設定 PCI device 使用哪一條中斷請求線，其值爲 01~04 對應到 PCI device 中斷信號。

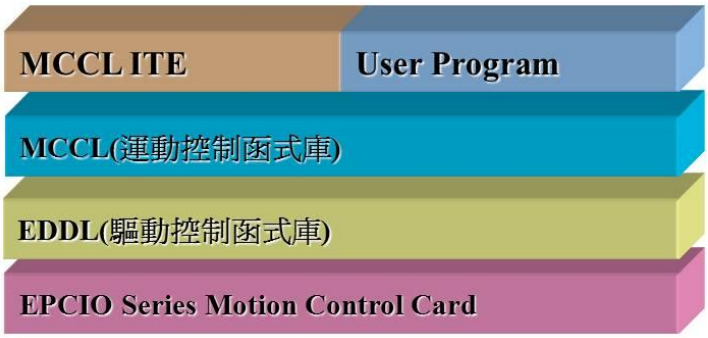
(10) Interrupt Line:

當 PCI 裝置有要求 PC 做中斷配置時，此暫存器由自動組態軟體填入系統所配置的中斷號碼(IRQ)，其中 00h~0fh 分別表示 PC 中斷控制器上的 IRQ0~IRQ15 中斷線。

當一個 PCI 裝置安裝於 PC 上，電腦在開機時系統軟體會檢查 PCI 上面的 Vendor ID，如果不是 FFFFh，則視爲該插槽上有 PCI 裝置，接著會去讀取 PCI 元件上的 Configuration Register，獲取資源需求資訊，例如 Memory range、I/O range，Interrupt number 等等。系統自動組態軟體會分配沒有衝突的資源，給該 PCI 元件，並且將分配到的 Memory、I/O、IRQ 等資源填入 Configuration Register，PCI 元件則依照 Configuration Register 內的 Memory 或 I/O 資源分配資訊去做解碼，讓 PCI 元件上的 Local Memory 可以對應到 PC 端所分配到的系統資源。

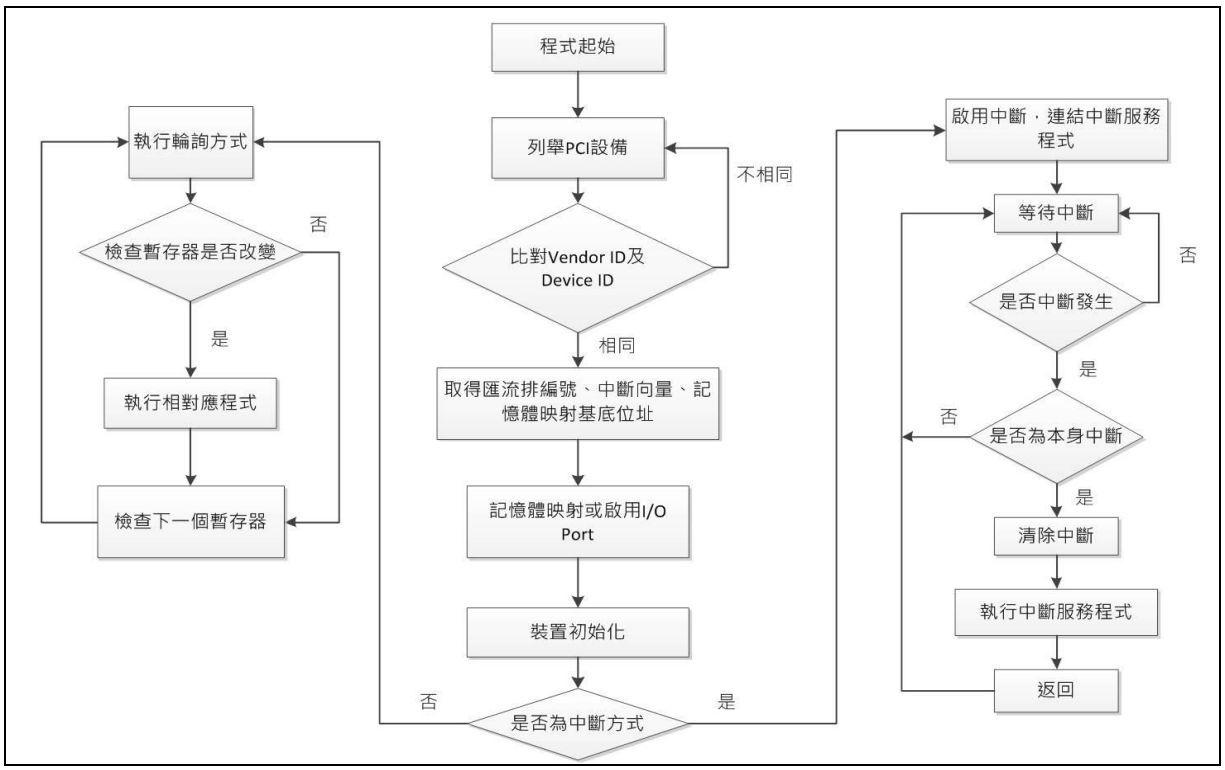
應用程式如果想控制 PCI 元件時，就必須取得此 PCI 元件上 Memory 或 I/O 的 base address 及 IRQ number 等資源分配資訊，接著再透過 Memory 或 I/O port 控制該 PCI 元件，最後則是處理 PCI 元件所發出的中斷要求。

實作部分



圖三 EPCIO 功能模組圖

圖三是 EPCIO 運動控制技術的功能模組圖，最底層是硬體的部分，往上依序是驅動控制函式庫(EDDL: EPCIO Device Driver Library)、運動控制函式庫(MCCL: Motion Control Command Library)，最上層則是使用者程式、介面的部分。爲了使 EPCIO 運動控制技術具有更良好的即時性，主要的工作就是將驅動控制函式庫及運動控制函式庫移植到 RTSS 之中，RTX 針對 PCI 及 ISA 匯流排有提供其相對應的操作 API，底下就針對 PCI 裝置的部分，做進一步的介紹。



圖四 開發流程

整體開發流程如上圖四所示，分爲以下幾個步驟，

(1) 列舉 PCI 裝置，這個步驟可以透過累加匯流排編號(Bus)、裝置編號(Device

Number)、功能編號(Function Number)進行。RTX 提供 RtGetBusDataByOffset 的函式，可以取得目前所輪詢到的 PCI 裝置資訊。

(2) 接著比對 Vendor ID 及 Device ID，看看是否就是我們所要找的裝置；如果正確的話，將匯流排編號、IRQ 編號(Interrupt Request)、記憶體基底位址等所需的資訊取回，不是的話則輪詢下一個 PCI 裝置。

(3) 根據 PCI 裝置的設計，會分成記憶體映射(Memory Map)或 Port I/O 存取：

I. 如果是記憶體映射的方式，必須呼叫 RtTranslateBusAddress 這個函式，去做基底位址轉換系統位址的動作，接著宣告一個記憶體空間後，可以得到一個虛擬位址，最後呼叫 RtMapMemory 去映射系統位址與虛擬位址。

II. 如果是 Port I/O 的方式，則呼叫 RtEnablePortIo 啓用直接 I/O 存取，接著透過 RtReadPortUchar、RtWritePortUchar…等函式去對 I/O 做存取。

(4) 執行 PCI 裝置初始化，這個步驟必須參考 PCI 裝置的硬體操作使用說明，一般而言都會有 BYTE、WORD、DWORD 等三種類型的指標，根據基底位址平移多少，去映射每個暫存器所代表的功能；不同 PCI 裝置所對應到的基底位址也可能是不一樣的，必須參考硬體的操作使用說明才能得知。在完成初始化之後硬體才能真正運作。

(5) 驅動程式的工作模式主要分爲輪詢(Polling)及中斷(Interrupt)的方式。

I. 輪詢：輪詢的方式很簡單，就是週期性地去檢查某些特定的暫存器，這些暫存器可能會因硬體行爲而改變，如果發現某個暫存器的值被改變了，則進行相對應的處理；否則的話，則檢查下一個需要檢查的暫存器，週而復始。

II. 中斷：首先必須啓用 PCI 裝置的中斷功能，接著根據步驟(2)所得到的匯流排編號、IRQ 編號，以及呼叫 RTX 所提供 RtAttachInterrupt 或 RtAttachInterruptVectorEx 函式進行中斷服務程序串接。根據 RTX 的使用手冊說明，其中 RtAttachInterrupt 串接 RTSS 的中斷服務程序，RtAttachInterruptVectorEX 則是串接 Win32 的中斷服務程序。中斷服務程序成功掛載成功之後，便會等到中斷真正觸發時才會執行；中斷服務程序需要判斷中斷是否爲共享(shared)，如果是自己的中斷則繼續處理，否則應該要傳遞給下一個 PCI 裝置。

(6) 在結束 PCI 驅動程式時需要關閉中斷，可呼叫 RTX 提供的 RtReleaseInterruptVector 或 RtReleaseInterrupt，接著可以呼叫 RtUnmapMemory 釋放記憶體，或呼叫 RtDisablePortIo 停用 Port I/O 存取。

在實際開發時，移植驅動控制函式庫因爲牽涉到 PCI 驅動程式，這部分需要參考上述的步驟去修改，上層運動控制函式庫的移植相較之下就較爲單純。函式庫的產生可分爲 RTSS DLL 以及 RTDLL 兩種方式，RTSS DLL 可以使用隱式連結(Implicitly Link)的方式，RTDLL 只能支援顯式連結(Explicit Link)的方式，也

就是在 Run-time 的時候才能取得函式的位址。兩者函式庫各有其優缺點，使用者可根據需求再進行開發。

在環境的設置的部分，首先必須將 EPCIO 運動控制卡，加入到 Rtx Driver 底下，如下圖五所示：

圖五 將 EPCIO 運動控制卡加入 Rtx Driver

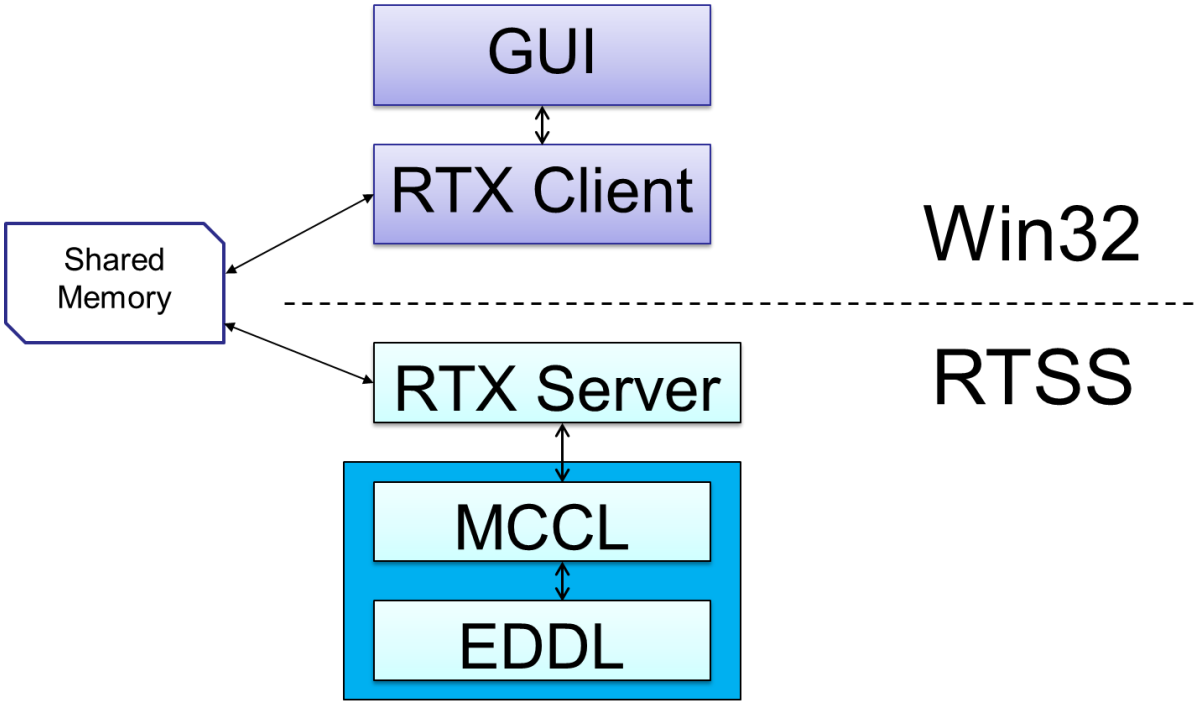
接著確認 IRQ 是否可以使用?是否與其他裝置分享?如圖六所示：

圖六 RTX PCI 裝置屬性

驗證部分

至於軟體驗證的部分，由於 RTSS 的程式是在 Kernel Mode(Ring 0)底下執行，而一般的程式是在 Win32(Ring 3)上面執行，爲了讓 RTSS 的程式可以與一般程式做溝通，RTX 提供了共享記憶體(Shared Memory)的機制來實現，另外 RTSS 內部的程式與程式間的溝通，也都是透過共享記憶體來進行。

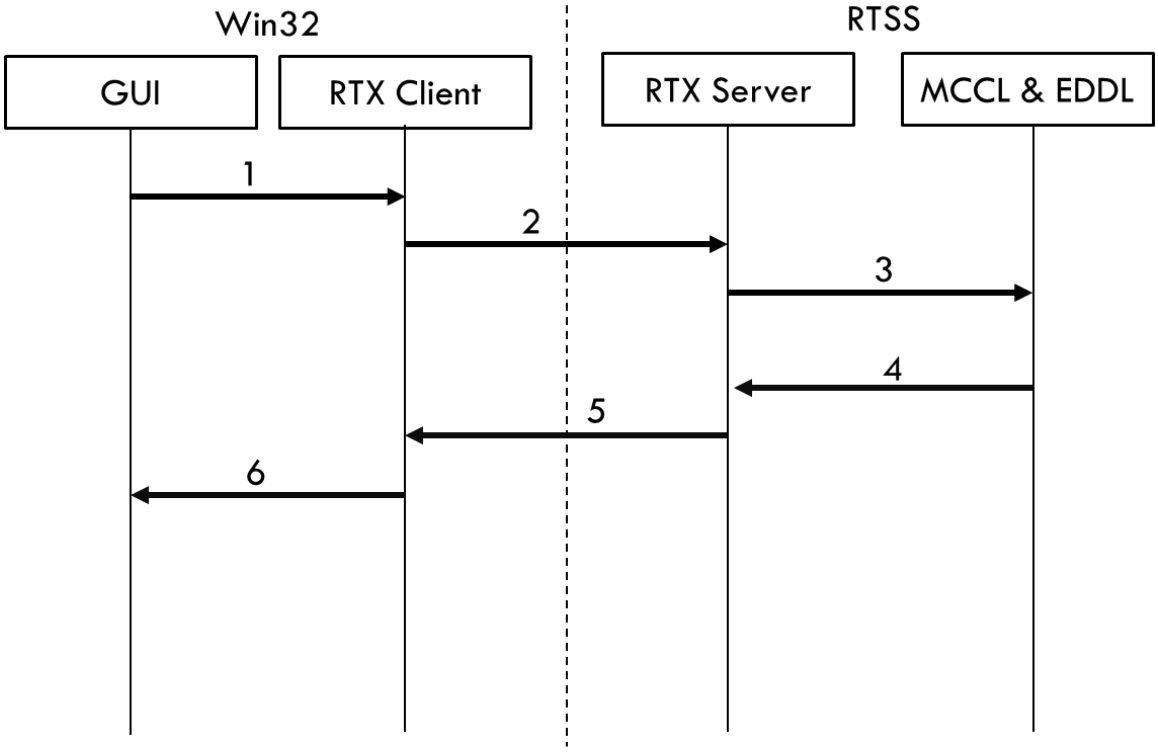
我們在移植驅動控制函式庫及運動控制函式庫之後，爲了方便在使用者模式(User Mode)下做測試與驗證，另外開發了 Client-Server 的架構，如下圖七所示。溝通的方式便是透過共享記憶體。



圖七 Client-Server 架構

其中 RTX Server 運行在 RTSS 底下，RTX Client 則是運行在 Win32，雙方透過共享記憶體進行資料的交換。RTX Server 透過 RTX 提供的函式 RtCreateSharedMemory 去新增一塊共享記憶體，這個共享記憶體由 RTX 負責管理，我們可以宣告成一個資料結構，方便之後的存取。接著 RTX Client 透過 RtOpenSharedMemory，去開啓相同名稱的共享記憶體，如果新增與開啓都成功的話，那 RTX Server 與 RTX Client 皆可以對這塊記憶體做操作，但也由於雙方同時間都可以存取，所以會面臨到資料同步的問題，相關的同步問題可以靠 Mutex、Semaphore、Event 等機制來解決。

一開始將所有 MCCL 的函式另外定義成簡單的指令表，這個指令表在 RTX Client、RTX Server 各有一份；RTX Client 主要的功能就是對原本的 MCCL 的函式進行覆寫，將 MCCL 函式轉換成 MCCL 指令，RTX Client 把指令送至 RTX Server 之後，RTX Server 透過查詢指令表，便可以轉譯成實際的 MCCL 函式。對於 Win32 上層的程式而言，所使用的函式名稱相同，但實際是透過了 Client-Server 的架構將指令傳達到 RTSS 底下的 MCCL。



圖八 系統流程圖

整個系統的流程就如上圖八所示，在初始階段的時候，RTX Server 會新增一個共享記憶體並且將自己停住，等待 Client 發送命令。之後步驟描述如下：

1. 使用者從 GUI 執行 MCCL 函式。
2. RTX Client 將 MCCL 函式轉成 MCCL 的指令，將相關的參數填入共享記憶體之後，將控制權交給 RTX Server，並且將自己停住。
3. RTX Server 取得控制權之後，從共享記憶體去解譯 RTX Client 所填入的指令及相關參數，再交由 MCCL 往下執行。
4. MCCL 將執行的結果回傳給 RTX Server。
5. RTX Server 將執行結果填回共享記憶體，再把控制權交給 RTX Client，之後等待下一回合。
6. RTX Client 取得控制權，將存放在共享記憶體的執行結果再往上傳遞給 GUI。

以”測試”的觀點來看，上述的 Client-Server 的架構是相當實用的，使用者如

果先前有撰寫 EPCIO 運動控制卡的相關程式，只需要更換函式庫，便可以使用 RTX 版本的運動控制函式庫，程式的部分不需要做大幅度的更動。但要真正發揮 RTX 的效能，應該要盡量減少整個軟體當中 Win32 與 RTSS 之間的耦合度，例如運動命令的下達、包含 Callback Function 的處理，應該要在 RTSS 底下去實現，Win32 部分可能只單純做狀態顯示、緊急關閉等功能。

結論

目前 PC 的發展朝著多核心、高速來發展，再搭配即時性的子系統，使得以往對於即時性需求較高的應用在 PC 上都得以實現。以工業控制領於而言，目前包括研華科技、凌華科技、泓格等大廠，都有相關的產品導入 RTX。PC-Based 的控制技術應用於即時性的子系統上，以不增加硬體成本的條件前提，使得性能可以往上提升，無疑是對業者是一大福音。

必須提到的一點是：由於 RTX 的程式需要執行在核心模式底下，往往因爲程式設計缺陷或種種原因，造成系統突然崩潰，這點是程式設計師必須相當注意的地方，因爲在工控領域當中，"可靠度"往往是被擺在第一位的。另外在 Windows 的環境下，想運行 RTX 就必須擁有 RTX Runtime，這部分也會增加成本的支出。

就未來趨勢來看，Soft Motion 將扮演著重要的角色，泛用運動控制市場上各種不同開放工業網路協定，包含 EtherCAT、SSCNET、MECHATROLINK 等等，也正蓬勃的發展當中。以 PC 的架構也需要具備有硬即時的能力，才能應付快速的資料交換，以 Windows 作業系統加上即時性的子系統，將對於 Soft Motion 提供一個良好的解決方案。

參考文獻

[1] http://www.epcio.com.tw/
[2] http://www.intervalzero.com/
[3] 李朝修，以即時網路爲基的控制系統研製-以串並聯機構爲實現例，碩士論文，國立成功大學，製造工程研究所，民 96 年 7 月
[4] http://intervalzero.com/assets/RTX-Architecture2.jpg
[5] 張盈喬，PCI Bus 運動控制模組技術分析，機械工業 217 民 90 年 04 月，頁 114-124
[6] 張帆 史彩成，Windows Device Driver programming 驅動程式設計，博碩文化出版，民 98 年 6 月
[7] PCI Local Bus Specification (Rev 2.2), PCI SIG