CCL Java Card CPU Design

摘要

Java 在網際網路的熱潮中切入，成為網路上目前最受重視的語言。Java 技術可以應用於智慧卡(smart card)上，而智慧卡內部需要一個可以直接執行 Java 的處理機。CCL Java Card CPU 支援 Java Card 虛擬機器(Java Card Virtual Machine，JCVM)所定義的 bytecode，具有管線式的執行架構。本文將介紹 CCL Java Card CPU，說明它各模組的細部設計及相互關係。

關鍵詞：智慧卡 (smart card)，Java Card 虛擬機器(Java Card Virtual Machine，JCVM)，管線式處理機(Pipelined Processor)，Java 虛擬機器(Java Virtual Machine， JVM)，資料堆疊(data stack)

1. 發展緣由

Java 相關技術因其具跨平台、動態連接，及內建之安全性考量等特色，非常符合網際網路分散式運算特性[1]。Java 語言所撰寫的程式是透過一個明確定義的虛擬系統架構來執行的，這個系統稱為 Java 虛擬機器(Java Virtual Machine， JVM)[2]。此外，由於電子商務的逐漸盛行，智慧卡(Smart Card)的市場商機大幅浮現，因此 Sun Microsystems 在 1996 年主動又推出的Java Card 虛擬機器(Java Card Virtual Machine，JCVM) ，其主要目的是希望 Java 語言能夠應用到 Smart Card 的環境中。JCVM 在基本架構上與 JVM 類似，最大的差異在於 JCVM 中並不支援浮點數(floating number)以及長整數(long)的資料型態，並加入了一種新的短整數(short)型態，其主要原因是為了儘量減少以 Java 發展的Smart Card 應用程式所佔的記憶體；此外，JCVM 中 bytecode 所做運算的基本單元為 16-bits 而不是JVM 中所定義的 32-bits。實現 Java 虛擬機器的方法有三：第一種方法是以解譯(interpreting)的方法來執行 bytecode，即 Java 虛擬機器每次處理及解譯一個 bytecode 指令，這是目前最常用的方法。它的優點是構建容易，缺點是解譯執行效能不佳。第二種方法是利用一個即時編譯器(just-in-time compiler)，每次執行時動態的將 Java bytecode 編譯成目的處理機的目的碼(native code)。它的優點是執行效能約可比前者提昇五到十倍，但編譯器需要額外用到約 2M 位元組的記憶體空間。最後一種方法是以 bytecode 為目的碼，設計一個 Java 處理機，用它來直接執行 bytecode。這種方法的好處為執行效能最佳，也不需要額外的記憶體，但是設計一顆 Java 處理機有一定的複雜度，而且需要提供後續的系統軟體和發展環境等支援。SME(Sun Micro Electronics，Java 的起源地)曾

提出 Java 硬體解譯的產品為 PicoJava，PicoJava 是一個處理機核心(core)，SME 將這個處理機核心授權給各大廠商設計各種的應用產品，以擴大 Java 產品的市場及應用領域，目前全球有許多 Java 晶片的設計廠商如 Imsys，PSC，aJile，advancel 等。

CCL Java Card CPU 具有 6 個 pipeline stage 的 RISC core，具有中斷(Interrupt)的處理機制、支援單步執行偵錯(Single Step Trace Debug)以及執行微指令 (micro-code) 的順序器(Sequencer)機制；並且整合週邊以及外建 8K on-chip 記憶體。目前 CCL Java Card CPU 總共支援了 245 的指令，這 245 個指令包含了 185 個 JCVM specification 2.1 本身所定義的 bytecode，以及我們自己所定義的 60 native 指令。在這 245 個指令中，有 66 個指令因為其運算相當覆雜，若是直接使用硬體來實作會佔用很大的晶片面積，因此採用發生軟體中斷的方式來實作這些指令，當 CCL Java Card CPU 偵測到這些指令時，它會發出軟體中斷，然後由相對應的中斷服務常式來完成該指令的執行；另外有 45 個指令則透過順序器的支援以硬體執行，其他的 134 個指令則可以直接由硬體執行。

本文之組織架構說明如下：第二節介紹 CCL Java CPU 之主要架構與特色；第二節的各小節介紹每個單元的細部設計；第三節介紹總結與未來方向。

2. 架構

CCL Java Card CPU 和週邊整合在同一 chip 上如圖 2-1 所示。週邊的單元包括： Bus interface unit(BIU)負責對外記憶體的存取，中斷控制單元(Interrupt controller，INTC)負責處理中斷服務要求，計時器(Timer)負責計數並發出中斷，Serial input output interface(SIO)支援 7816input/output，而 UART 是為了除錯的考量。

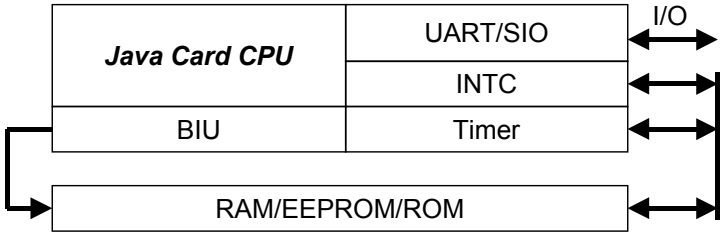

圖 2-1 Java Card Chip 方塊圖

CCL Java Card CPU 為一個 6-stage pipelined RISC core，這六個 stages 依序為：Instruction Fetch (IF)、Instruction Decoding (ID)、Register/Stack Access (REG)、Execution (EX)、Memory Access (MEM)、及 Write Back (WB)。由 7 個單元(Unit)所組成：指令擷取單元(Instruction Fetch Unit，IFU)、解碼單元(Decoding Instruction Unit，DIU)、堆疊暫存器單元(Stack Register Unit，SRU)、執行單元

(Execution Unit，EXU)、記憶體存取單元(Memory Access Unit，MAU)、控制與中斷處理單元(Pipeline Control Unit，PCU)、匯流排介面單元 (Bus Management Unit，BMU) 。以下分別就各功能方塊之功能、輸入輸出、設計考量提出說明。

2.1 指令擷取單元

指令擷取器如圖 2.1.1 所示，主要負責自指令記憶體中根據 PC 指定的位置一次讀取 2 位元組的資料，存於容量為 8 位元組的指令緩衝器(Instruction Buffer)中，再由控制邏輯(Control)記錄指令長度於指令長度暫存器(Instruction Length Buffer)，調整指令擷取及後續的執行順序。預先解碼表(Predecode Table)可以查出指令的種類；指令的種類及指令長度則傳到下一級的解碼單元使用。

此指令擷取器是分離式(Decoupled)的架構，亦即將有效指令送到解碼器進行解碼的動作和從指令記憶體讀取指令的動作可各自獨立進行，只要仍有有效指令即可送至解碼器；反之，若管線必須停止，但指令緩衝器中仍有空位置，控制邏輯送出 Request 訊號至匯流排界面，並送出要存取指令之位址。而當記憶體以 Ready 送回表示指令資料已讀回後，再將傳回的2 位元組存至指令緩衝器中適當的位置。

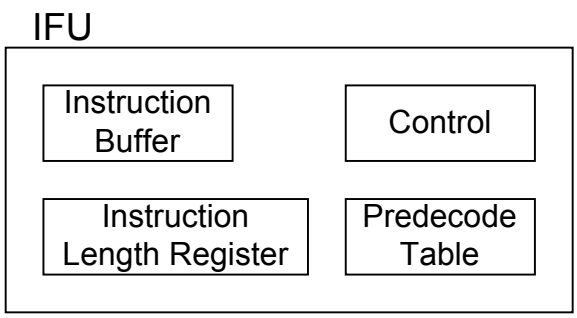

圖 2.1.1 指令擷取器方塊圖

2.2 指令解碼單元

圖2.2.1中所顯示的是解碼器內部主要的部分，包括解碼邏輯(Instruction Decoder)、管線暫存器(Pipeline Regiesters)、順序器(Sequencer)和控制邏輯(Control Unit)。解碼邏輯的功能就是根據指令碼(Opcode)產生相對應的訊號存於管線暫存器中，這些訊號包括被解碼指令的PC、運算元暫存器位置、結果暫存器位置、和控制以下各級管線的控制訊號。順序器主要是針對需要多個脈衝週期才能完成的指令，以 CCL Java Card CPU 的設計而言，只有 multi-cycle instruction 會透過順序器來執行，complex instruction 則以軟體仿真(software emulation)的方式執行。對於multi-cycle instruction 或是串列化指令而言，它們第一個週期的訊號還是由解碼器產生，而接下來所的幾個週期則由順序器

來產生各個週期所需要的訊號，等該指令所有的動作都被執行完之後，解碼訊號就不再由順序器來產生，而是切換由解碼邏輯來產生下一個指令所需要的解碼訊號，所謂的串列化指令是指那些必須全部執行完畢後才可以讓後進入管線的指令開始執行的的指令。

控制邏輯主要在選擇解碼邏輯的輸出訊號或是順序器的輸出訊號後並將之送到管線暫存器，對於只要一個週期即可執行完成的指令，控制邏輯會選擇解碼邏輯的輸出訊號，而對於 multi-cycle instruction 或是串列化指令而言，控制邏輯會在第一個週期選擇解碼邏輯的輸出訊號，而接下來所的幾個週期則會選擇順序器的輸出訊號，一直要到該指令執行完後才會再度選擇解碼邏輯的輸出訊號。控制邏輯亦會產生相對於指令的 exception bit，這些 exception bit 都會繼續往後面的 stage 傳送，最後才由控制與中斷處理單元決定要不要產生相對應的中斷。

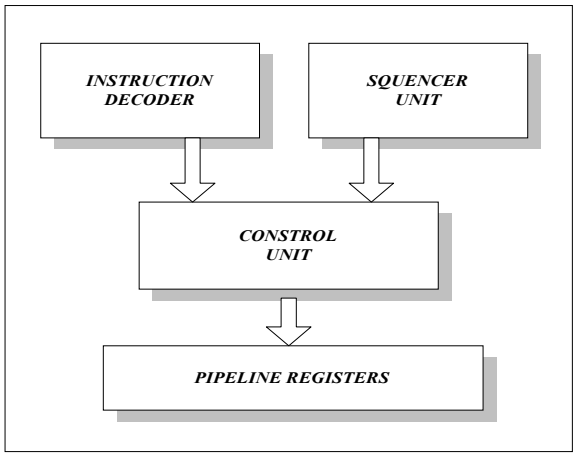

圖 2.2.1 解碼器方塊圖

2.3 堆疊暫存器單元

CCL Java Card CPU 原始的虛擬機器 JCVM 是以堆疊機器(stack machine)為基礎的。堆疊機器是假設系統在執行 Java bytecodes 過程中，資料利用一個 first-in-last-out 的資料堆疊(data stack)來做儲存的機構，指令執行時系統會自動將資料堆疊最前端的運算元取出作相對應的運算，並將運算結果放回資料堆疊去。為了快速讀寫資料堆疊內部的資料，資料堆疊以 on-chip memory 的方式來設計，記憶體的空間為 8k Bytes。當 CCL Java Card CPU 在執行 native 指令過程中，除了 laod/store 指令可以直接存取記憶體資料(包括 off-chip memory)，其他運算都需要透過暫存器。

堆疊暫存器單元主要功能是從 on-chip

memory 或暫存器中讀資料，方塊圖如圖 2.3.1 所示。每個週期由位址解碼器(Address Decoder) 解碼來自於解碼器的運算元位址及控制訊號，最多可以從資料堆疊讀出兩個運算元；或是從暫存器讀取兩個運算元，以及寫入一個暫存器運算元。此外堆疊暫存器單元還負責做資料迴送(data forwarding)，資料危障(data hazard)和堆疊暫存器忙碌(SRU busy)的檢查。資料迴送檢查在管線執行中檢查執行單元所要的運算元，其最新的數值是否尚未寫回，檢查結果將決定是否做資料迴送；並且影響執行單元取得運算元的資料路徑。資料堆疊所佔用的記憶體和 native load/store 指令所操作的記憶體彼此之間並不支援資料迴送。資料危障則是檢查 load 指令的下一個指令是否會使用到此讀取值。若是，由於管線架構的限制，控制與中斷處理單元會在此讀取指令後動態插入一個 NOP(no operation)指令以防止資料危障。堆疊暫存器忙碌發生原因是因為同一個週期堆疊暫存器和記憶體存取單元最多需要三個 ports 來做運算元的讀取；而 on-chip memory 只有兩個 ports，此時堆疊暫存器必須等待一個週期讓記憶體存取單元先做完。

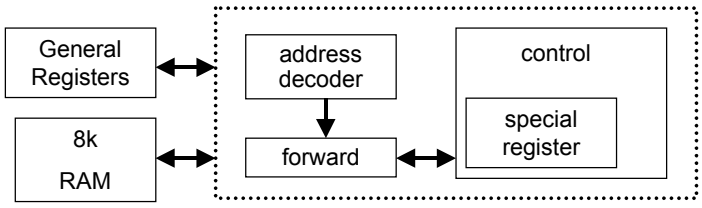

圖 2.3.1 堆疊暫存器方塊圖

2.4 執行單元

執行單元包含算術邏輯單元（ALU）、算術旗標電路（Arithmetic Flags）、跳躍控制電路（Branch Control）、跳躍目的位址運算(Branch Address Adder)。其方塊圖參照圖 2.4.1。算術邏輯單元負責執行所指定的算術和邏輯運算。跳躍控制電路處理分支指令的結果；跳躍目的位址運算計算跳躍目的位址。算術旗標電路更新旗標(Carry、Zero、Sign、Overflow)。執行單元除了支援 Java bytecode 所定義的 16 位元運算外，還支援 20 位元的加減法、20 位元的左右移 1 位指令、16 位元的左右移一位、左右旋轉指令及左右旋轉一位指令。在做移位或旋轉時會即時將 carry flag 納入，而 JVM 本身所定義的 shift 指令不會。

為了省電算術邏輯單元增加了資料輸入開關，當算術單元中的某個單元不被使用則將輸入清為零，以使其內部電路沒有電位的改變達到省電的效果。

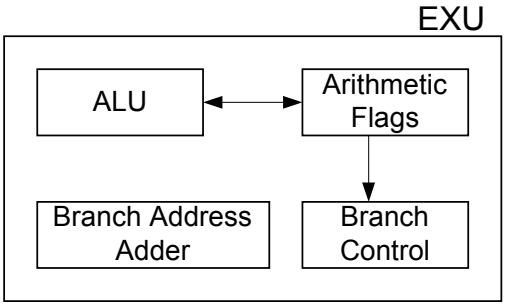

圖 2.4.1 EXU 方塊圖

2.5 記憶體存取單元

記憶體存取單元主要是根據執行單元的要求，向匯流排介面單元提出使用匯流排的請求，作記憶體的讀寫動作。另外也提供資料路徑，接受控制與中斷處理單元的控制作中斷的處理。當發生中斷時，控制與中斷處理單元會產生中斷號碼要求記憶體存取單元抓回相對應中斷服務常式的起始位址。

由於 CCL Java Card CPU 對外界匯流排的寬度是 32 bits，所以 load/store 指令對記憶體作存取一次最多可存取 4 個 bytes 資料。不論從記憶體讀取 1 或 2 或 4 個 byte(s)的資料，讀取時都是一次讀取 4 個 bytes 資料。在讀取 1 或 2 byte(s)資料時會先作 zero extension 或是 sign extension，之後才會傳回給 CPU。若要寫入資料到記憶體時，會有一組訊號線用來指示現在在匯流排上的4 個 bytes 資料中那些是有效的。對 2 或 4 bytes 資料作讀寫動作時可能會發生 misalign 的情形，即當讀寫的位址不是在 4 個 bytes 的分界時就是 misaligned access。當發生 misalign 的情形時，不論讀或是寫外界記憶體都需發動兩次的記憶體讀或寫的動作。在這兩次讀寫的過程中，pipeline 是暫停不動的，等到對記憶體兩次讀寫完成之後，才會恢復動作。圖 2.5.1 和 2.5.2 中顯示的是讀寫位址與 4 bytes boundary 之間的關係，圖 2.5.1 中位址 3 的時候和圖 2.5.2 中位址在 1、2、3 的時候是 misalign 記憶體存取。

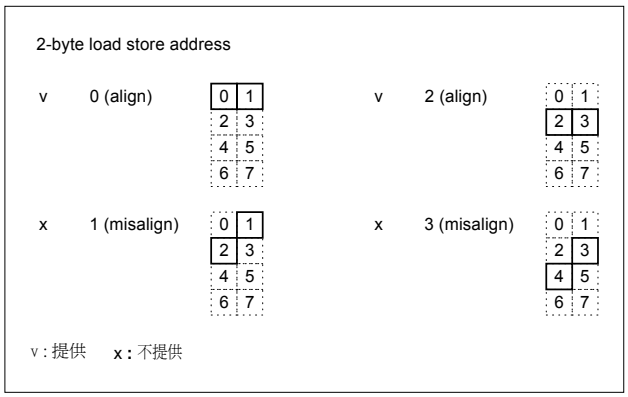

圖 2.5.1　2 bytes 記憶體讀寫中之 misalignment

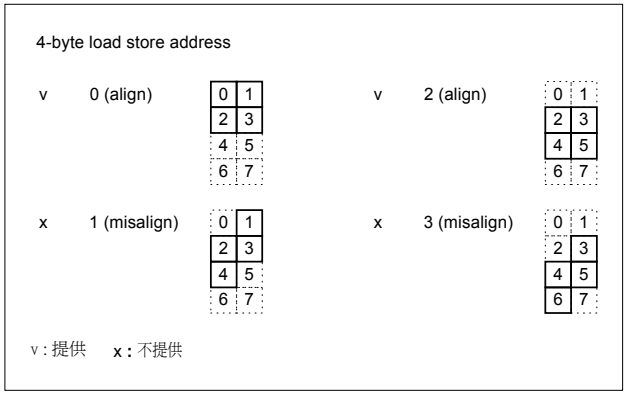

圖 2.5.2　4 bytes 記憶體讀寫中之 misalignment

2.6 控制與中斷處理單元

控制與中斷處理單元，負責各級管線間之工作協調與處理執行過程中所發生之中斷情形，目的是使得處理機內部平行操作之單元和諧工作。其控制主要包括各管線級之進行或停止，中斷狀況仲裁，及指令執行流程控制。控制單元在每週期控制各級管線之前進或停止，在正常執行狀況下，每一週期各級管線會前進一級以便處理一個新的指令，但是當指令緩衝區內之資料不足無法解碼，或記憶體存取尚未完成時，管線級會被暫時停止；資料危障發生時，控制與中斷處理單元會在管線級插入一個 NOP 後。在對執行指令各階段所產生之中斷狀況及外部中斷處理上，會維持指令的精確中斷，因各類之中斷狀況可以即時發生，此控制單元亦負責仲裁決定中斷處理之優先次序，決定最高優先處理的中斷後產生適當的清空管線級及暫停管線級訊號，存入系統狀態；當即時有多個中斷或例外發生時，控制單元會選擇其中優先權最高的來處理，其餘的須等到此中斷或例外處理完成後，控制單元又會選擇其中優先權最高的來處理。

在決定指令執行流程上，有四種可能情形：指令循序執行、有分支情形發生、有中斷情形發生要處理、及處理機電源啟動時。當指令循序執行時，此單元要決定下一次指令擷取之起始位址。當有分支情形發生，此單元除要將指令擷取之起始位址設定為分支目標位址外，亦要清除管線中之未完成指令及調整處理機狀態，以便執行正確位址之指令。當有中斷情形發生時，處理機內部之處理與分支情形發生時類似，只不過分支目標位址自存於中斷向量表(vector table)決定。當電源啟動時，此單元讓指令從一固定位址開始擷取與執行。

2.7 匯流排界面單元

匯流排界面單元負責 CCL Java Card CPU 與外界輸出入裝置和記憶體的溝通工作，所有的 CPU 控制或資料訊號都是透過匯流排界面單元，在特定的時脈週期時作輸出或輸入。由於 CCL Java Card CPU 的對外匯流排只有一條，而卻有兩種 requests 可能即時需要使用到匯流排，分別是記憶體存取(memory access)以及指令擷取(instruction fetch)，因此匯流排界面單元必須在兩個 bus requests 即時發生時做一個適當的仲裁，以決定要讓那一個 request 獲得匯流排的使用權。一般而言兩個 requests 即時發生時，優先權是記憶體存取大於指令擷取，兩者依序執行完後才接受下一個 bus requests，以滿足需求。此外，為了達到省

電的要求，CCL Java Card CPU 另外加入了 sleep mode；目前當 CCL Java Card CPU 執行到 xhalt 指令時會進入 sleep mode，在 sleep mode 底下，CCL Java Card CPU 會暫時停止整個的動作，一直到有外部中斷要求（external interrupt）來叫醒CPU 時才繼續動作。

為了支援匯流排使用優先權的仲裁以及 sleep mode，在匯流排界面單元中設計了一個 Finite State Machine（FSM），該 FSM 有四個 states，在不同的state 時，兩種 request 有不同的優先權。

斷旗號時可以依據 TCNR 設定決定是否重新由 TSR 自動裝載(auto-reload)計數值於 DCR。TSR 暫存器之值不會因重新載入之動作而有所改變，除非再次寫入 TSR 更改其值。

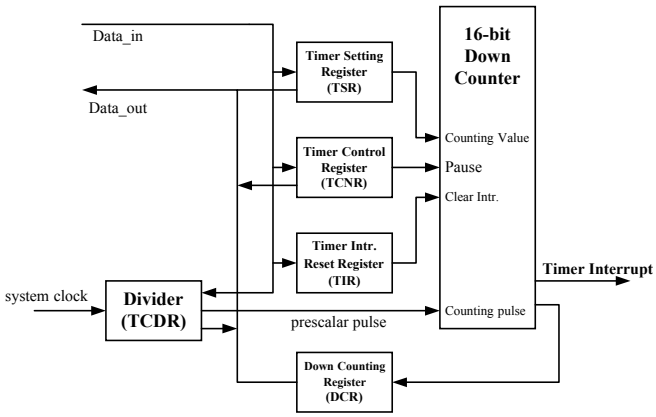

圖 2.8.1 計時器方塊圖

2.8 計時器

Java Card CPU 有 2 個 16 位元的計時器，即計時器 0 及計時器 1，如圖 2.8.1 所示。這二個計時器具有一個可預除 1~16 的前置預除器，預除值經由 TCDR 設定。計時需在計數值被載入(TSR)方才被致能開始計時之動作，計數過程中可以透過讀取計數暫存器(DCR)來得知目前向下計數值，或利用寫入計時控制暫存器(TCNR)使計時器暫停(pause)向下計數。當計時暫存器所有的位元從1 變成 0 時(亦即溢位發生時)，此時計時中斷旗號"INTR"將被設定為"1"，欲解除此旗號可透過寫入計時中斷清除暫存器(TIR)來結束其動作。在解除中

2.9 Memory Controller(BIU)

BIU 主要負責 ROM 及 FLASH 記憶體的存取，當 CPU 核心發出記憶體存取後，BIU 解碼後如果存取位置為 ROM 或 FLASH 時，便啟動適當的機制，適當的時機對 ROM 及 FLASH 發出適當的訊號，並分別對 ROM 及 FLAH 提供等待時間的控制，可由程式控制何時 ROM 及 FLASH 的資料備妥。由於 CPU 核心的資料寬度為 16 位元而 ROM 的介面為 8 位元，BIU 當需提供 8 位元轉 16 位元及 16 位元轉 8 位元的介面 。由於 Card 的應用屬低耗電，小面積，慢速的特性，而匯流排屬高耗電的單位，在設計的階段及規格的製訂上除儘量

減少邏輯閘的運用，在機制的運作方考慮儘量減少匯流排的邏輯變化。

2.10 中斷控制單元

中斷控制單元是用來收集周邊裝置的中斷訊號；其方塊圖如圖 2.10.1 所示，包括了 UART、計時器和外部（IRQ）的中斷訊號。假如此時周邊裝置發出中斷訊號，而且此周邊裝置相對於中斷遮蔽暫存器（Interrupt Mask Register）中的位元沒被遮蔽，中斷控制單元會發出“HINTR”訊號給 CPU，通知 CPU 有中斷發生。

此單元中有三個暫存器，中斷起因暫存器（Interrupt Cause Register）用來儲存周邊裝置發出的中斷訊號。一般時候只能讀不能寫。除了當外部發生中斷訊號（由 1 變 0），而且中斷要求形式暫存器（IRQ Mode Register）是 falling-edge 方式，需要靠 CPU 以 write-one-clear 的方式去清掉此中斷訊號，否則外部中斷要求相對於中斷起因暫存器的位元會一直為 1。中斷遮蔽暫存器是可讀寫的，通常程式初始化時，會將所有周邊裝置相對於中斷遮蔽暫存器的位元打開，不然即使此周邊裝置發出中斷，中斷控制單元方不會發出“HINTR”訊號給 CPU。

中斷要求形式暫存器負責設定外部中斷要求的形式，在此單元方有兩種形式：一為 falling-edge 的方式，二為 low-level 的方式。falling-edge 的方式是若外部中斷要求從 1 變 0，那外部中斷要求相對於中斷起因暫存器的位元會一直為 1，除非 CPU 以 write-one-clear 的方式，即 CPU 寫 1 到外部中斷要求相對於中斷起因暫存器的位元，才能清掉此中斷訊號。另一種 low-level 方式是當外部中斷要求為 0 時，外部中斷要求相對於中斷起因暫存器的位元設為 1；當外部中斷要求為 1 時，會清掉外部中斷要求相對於中斷起因暫存器的位元。

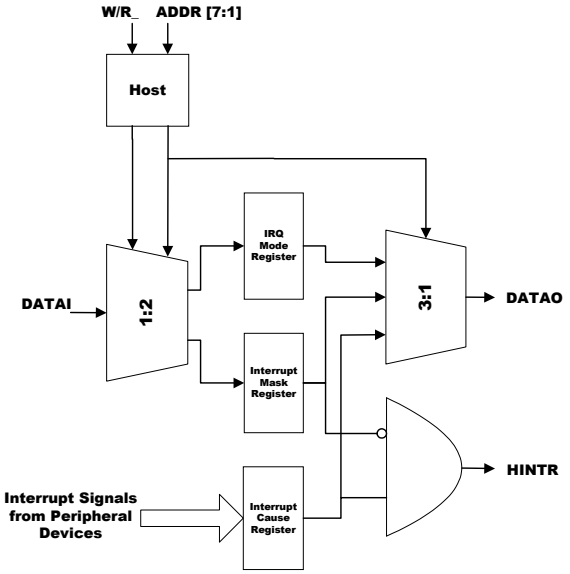

圖 2.10.1 中斷控制單元方塊圖

3. 總結

美國運通公司(American Express)將在最近推出的「藍」(Blue)卡和花旗銀行

（Citibank）即將推出的企業簽帳與認証卡中，已決意採用 Java Card 技術。花旗銀行的卡片不只可用來刷卡購物，也可當作電子身分識別證使用，讓使用者表明身分、登入電腦、加密通訊訊息，並進行電子文件的數位簽證。[3]由此可見，智慧卡(smart card)的應用已經逐漸普遍；未來相關的技術可能打入小型電子裝置市場或是較高階的應用。CCL Java Card CPU 可以用於智慧卡內部的主要處理器。未來可能的發展方向包括：低耗電設計，支援加解密設計等等。

參考文獻

[1] 江自強、肖希聖、許家彰，"網路電腦的明日之星--JAVA處理器 ，" CCL Technical Journal ，57期，1997， pp.3-12.

[2] Tim Lindholm and Frank Yellin， The Java Virtual Machine Specification， Addison Wesley， 1996.

[3] CNET新聞專區：Wednesday, June 7 2000, http://www.taiwan.cnet.com/

[4] J. Michael O'Connor， Marc Tremblay，"picoJava-I: The Java Virtual 機器 In Hardware，" IEEE Micro， March/April，1997， pp. 45-53.

[5] 馬瑞良、江自強、汝世安、黃弘一、肖希聖，"CCL Java Chip I 微架構設計 ，" CCL Technical Journal ，67 期，1998， pp.3-8.