DMA e PCI Bus Mastering

G. Lettieri

16 Marzo 2022

1 Direct Memory Access

Abbiamo visto, fino ad ora, due modalità di trasferimento dati da un dispositivo alla memoria, o viceversa:

- a "controllo di programma";
- tramite interruzioni.

Nella prima modalità il software controlla periodicamente che il dispositivo sia pronto leggendo un qualche registro di stato, quindi opera il trasferimento con una o più operazioni di lettura da registri del dispositivo, seguite da scritture in memoria, o viceversa. Nella modalità con interruzioni il trasferimento avviene nello stesso modo, ma è iniziato solo quando il dispositivo segnala la propria disponibilità tramite una richiesta di interruzione. La modalità a controllo di programma è più veloce di quella a interruzioni, ma, come sappiamo, è complicata da programmare se si vuole che il processore possa fare anche altro nei momenti in cui il dispositivo non è pronto; entrambe le modalità prevedono comunque un coinvolgimento del processore, che deve eseguire le istruzioni di lettura e scrittura, e comportano che i dati vengano scambiati sul bus due volte: una volta tra RAM e CPU e una volta tra CPU e dispositivo.

La modalità DMA (Direct Memory Access), che è la terza e ultima modalità di trasferimento dati, prevede invece che sia direttamente il dispositivo ad eseguire le necessarie operazioni di lettura o scrittura sulla RAM, una volta istruito dal software. In generale, il software vorrà eseguire un trasferimento dati dal dispositivo verso un buffer in RAM (operazione di ingresso o lettura) o da un buffer in RAM verso un dispositivo (operazione di uscita o scrittura). Supponiamo che il buffer si trovi all'indirizzo b e sia grande n byte, occupando dunque gli indirizzi $[b, b + n)$. L'indirizzo b e il numero n devono essere comunicati al dispositivo, insieme a tutte le altre eventuali informazioni necessarie a definire il trasferimento richiesto. Da quel momento in poi il dispositivo si preoccuperà di eseguire autonomamente le operazioni in RAM, al proprio ritmo. Ovviamente il dispositivo deve essere dotato di un sommatore che gli permetta di calcolare da solo gli indirizzi necessari a partire da b, e di un contatore che decrementi n ogni volta che è stato completato un trasferimento. Quando tutti i byte sono stati

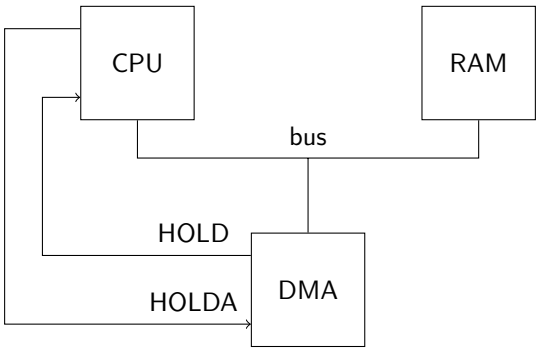

Figura 1: DMA, architettura di base.

trasferiti, il dispositivo segnalerà il completamento dell'operazione settando opportunamente un suo registro di stato e, tipicamente, inviando una richiesta di interruzione.

Dal punto di vista hardware possiamo considerare l'architettura schematizzata in Figura 1, dove abbiamo per il momento eliminato cache, MMU e bus PCI, ricreando l'architettura tipica dei primi PC IBM, o dei minicomputer PDP della DEC. Il dispositivo in grado di operare in DMA è collegato direttamente al bus dove si trovano anche la CPU e la RAM. Questo gli permette di dialogare con la RAM usando lo stesso protocollo già usato dalla CPU. Dal momento che il bus è condiviso, però, un solo dispositivo alla volta può pilotarlo: o la CPU, o il dispositivo DMA. L'accesso è arbitrato tramite i collegamenti HOLD/HOLDA fra dispositivo e CPU:

1. il dispositivo mantiene normalmente i suoi piedini di uscita in alta impedenza;
2. ogni volta che il dispositivo vuole eseguire un trasferimento sul bus, addiva HOLD;
3. in risposta, la CPU termina l'eventuale trasferimento in corso (che può essere anche nel mezzo di una istruzione), mette i suoi piedini di uscita in alta impedenza e attiva HOLDA;
4. il dispositivo attiva i suoi piedini di uscita ed esegue il trasferimento, quindi rimette le uscite in alta impedenza e disattiva HOLD;
5. la CPU disattiva HOLDA, attiva i suoi piedini e riprende il suo normale funzionamento.

In pratica. la CPU dà la precedenza al DMA nell'accesso al bus. La tecnica è chiamata "cycle stealing", in quanto il DMA ruba cicli di bus alla CPU. Si noti che, mentre è in corso il trasferimento in DMA, la CPU è comunque in grado di eseguire una eventuale istruzione già prelevata, che non preveda accessi in memoria.

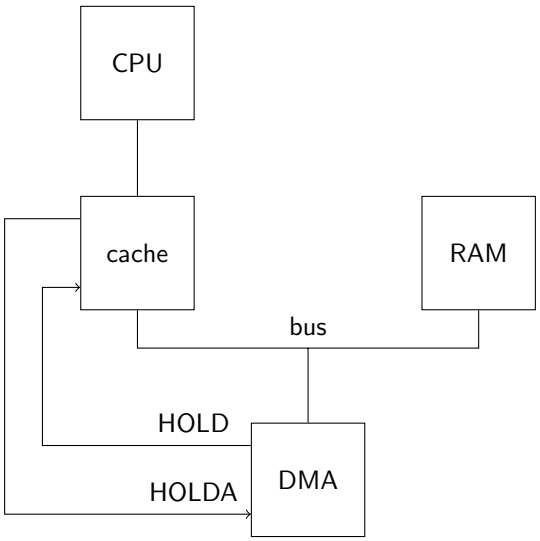

Figura 2: Interazione tra DMA e cache della CPU.

Il meccanismo HOLD/HOLDA funziona se c'è un unico dispositivo in grado di operare in DMA. Ci limitiamo a questo caso perché, come vedremo, il bus PCI ci permette di aggirare il problema senza introdurre altri meccanismi.

1.1 Interazione con la cache

Ora prendiamo in considerazione l'esistenza della cache. Notiamo subito che, come mostrato in Figura 2, i segnali HOLD e HOLDA devono ora collegare dispositivo DMA e controllore cache, in quanto è quest'ultimo, e non la CPU, ad essere collegato direttamente al bus con la memoria. A parte questo, l'handshake per il possessso del bus resta lo stesso.

L'introduzione della cache porta un grande vantaggio, ma anche alcune complicazioni. Il vantaggio è che è molto più probabile che la CPU riesca ad eseguire istruzioni mentre è in corso una operazione di DMA, perché può trovare istruzioni e operandi in cache.

Le complicazioni nascono dal fatto che le operazioni in DMA potrebbero coinvolgere parti di RAM che erano state precedentemente copiate in cache. Nel caso di cache con politica *write-back* la cache potrebbe anche contenere modifiche che non sono ancora state ricopiate in RAM. Esaminiamo i problemi, e le possibili soluzioni, partendo dal caso più semplice e passando poi a quelli più complessi. In ogni caso assumiamo che il software, per tutta la durata del trasferimento, non acceda al buffer coinvolto nel trasferimento. Ci interessa soltanto, dunque, lo stato della cache al momento di inizio e di fine del trasferimento.

1.1.1 Cache con politica *write-through*

Questo è il caso più semplice, perché all'inizio del trasferimento tutte le cacheline eventualmente presenti in cache contengono lo stesso valore delle corrispondenti cacheline in RAM. Questo implica che non ci sono problemi nel caso di operazione di uscita in DMA (direzione da RAM a dispositivo). Nel caso di operazione di ingresso in DMA (direzione da dispositivo a RAM), invece, bisogna assicurarsi che tutte le cacheline coinvolte (anche parzialmente) nel trasferimento vengano o rimosse dalla cache, o aggiornate, altrimenti il contenuto della cache, a fine trasferimento, non sarebbe consistente con il contenuto della RAM. Il software, dunque, potrebbe leggere valori "vecchi" scambiandoli per nuovi. Il problema è che l'accesso diretto alla memoria rende non più vera l'assunzione su cui abbiamo basato il funzionamento della cache, cioè che la RAM non cambia valore se non per le scritture che arrivano dalla CPU.

Il problema può essere risolto automaticamente in hardware, oppure essere demandato al programmatore software. Nel caso di soluzione completamente hardware, dobbiamo fare in modo che il controllore cache osservi *tutte* le possibili sorgenti di scritture in RAM. Grazie al bus condiviso, possiamo sfruttare il fatto che il controllore cache può osservare cosa sta accadendo durante l'accesso diretto alla RAM, ricevendo in ingresso le linee di controllo e di indirizzo. Se le linee di controllo identificano una operazione di scrittura, può usare il contenuto delle linee di indirizzo per eseguire una normale ricerca all'interno della cache (del tutto analoga a quella eseguita quando è la CPU a chiedere un indirizzo). Nel caso di *hit*, il controllore può autonomamente provvedere a invalidare la corrispondente cacheline. Alla fine del trasferimento, se il software accede al buffer, i dati dovranno essere prelevati dalla RAM, che contiene i valori aggiornati. Se il controllore riceve in ingresso anche le linee dati può aggiornare la cacheline, invece di invalidarla. La soluzione hardware appena illustrata viene detta *snooping* (ficcanasare), in quanto il controllore cache "ficca il naso" in quello che sta facendo il DMA.

Ci sono sistemi (per esempio quelli basati su processori ARM, comuni negli smartphone) in cui il problema non è risolto automaticamente in hardware e il programmatore deve risolverlo in software. Per farlo dispone di alcune istruzioni che gli permettono di interagire con il controllore cache, tipicamente per invalidare un intervallo di indirizzi. Il software deve eseguire queste istruzioni, specificando l'intervallo di indirizzi coinvolto nel trasferimento (nel nostro caso, gli indirizzi da b a $b + n$ escluso) subito prima che il trasferimento abbia inizio, o, equivalentemente, subito dopo che il trasferimento sia terminato.

1.1.2 Cache con politica *write-back* e trasferimento di intere cacheline

Consideriamo ora cache con politica *write-back*, ma assumiamo che ogni trasferimento in DMA coinvolga *intere* cacheline. Nel nostro caso stiamo assumendo che b sia allineato alla cacheline e n sia multiplo della dimensione di una cacheline.

Nel caso di cacheline non "dirty", cioè che contengono in cache lo stesso valore che si trova nella corrispondente cacheline in RAM, ricadiamo negli stessi casi già visti e possiamo adottare soluzioni analoghe. Le cacheline dirty, invece, ci creano un nuovo problema nel caso di uscita in DMA (che, ricordiamo, implica una direzione da RAM a dispositivo). In questo caso la cache contiene la versione più aggiornata della cacheline che il DMA deve leggere, e sarebbe un errore se il DMA leggesse la versione contenuta in RAM. Anche questo problema può essere risolto in hardware con un meccanismo di snooping, ma non è sufficiente lo snopping da parte del controllore cache. Infatti, è vero che il controllore cache può accorgersi che l'operazione in DMA coinvolge una cacheline dirty, ma cosa dovrebbe fare una volta scoperto ciò? Pilotare le linee dati con il valore aggiornato della cacheline comporta una corsa, in quanto anche la RAM sta vedendo la stessa operazione sul bus, e anche lei vorrà pilotare le linee dati. Tra le varie possibili soluzioni hardware, quella tipicamente adottata in questo caso prevede che sia *il dispositivo DMA* a ficcanasare nello stato della cache, prima di iniziare l'operazione di lettura in RAM. Lo snooping può essere svolto su collegamenti dedicati tra dispositivo DMA e controllore cache, bypassando dunque la RAM, oppure acquisendo il controllo del bus con il normale protocollo HOLD/HOLDA ed eseguendo una particolare operazione nota al controllore e ignorata dalla RAM (per esempio una operazione nello spazio di I/O). L'operazione di snooping deve passare al controllore cache l'indirizzo interessato dal trasferimento; il controllore deve eseguire una ricerca e, in caso di hit con cacheline dirty, può inviare lui stesso i dati al dispositivo, oppure riacquisire temporaneamente il controllo del bus e ricopiare la cacheline in RAM.

Nel caso di operazioni di ingresso in DMA (direzione da dispositivo a RAM) con cacheline dirty, c'è una ulteriore considerazione da fare. Questo caso può essere risolto con lo snooping da parte del controllore cache, come nel caso di cache write-through, ma il contenuto della cacheline dirty deve essere invalidato senza essere prima trasferito in RAM, in quanto il valore più aggiornato della cacheline deve essere quello prodotto dal dispositivo. Si noti che il controllore cache potrebbe iniziare il write back di questa cacheline (a causa di un rimpiazzamento) mentre è in corso il trasferimento DMA, ma il dispositivo non è ancora arrivato a trasferire la stessa cacheline, e dunque non ha fatto in tempo a invalidarla. Questo, per fortuna, non crea inconsistenze: la cache eseguirà il write-back, ma la cacheline in RAM verrà poi sovrascritta correttamente dal dispositivo.

Nel caso di soluzione interamente software, il programmatore può tipicamente ordinare al controllore cache di eseguire il write-back di un certo intervallo di indirizzi. Il software deve eseguire questa operazione su tutti gli indirizzi del buffer, e deve farlo necessariamente *prima* di avviare il trasferimento. Questo è ovviamente necessario per le operazioni di uscita (da RAM a dispositivo), in modo che la RAM sia correttamente aggiornata prima del trasferimento, ma è necessario anche per le operazioni di ingresso (da dispositivo a RAM) per evitare che eventuali cacheline dirty non vengano ricopiate in RAM dopo l'avvio dell'operazione, col rischio di sovrascrivere quanto già scritto dal dispositivo. Come ottimizzazione, il programmatore può disporre di una operazione che richiede

l'invalidazione *senza write-back* di un intervallo di indirizzi. Questa operazione può essere utilizzata prima di avviare un trasferimento in ingresso.

1.1.3 Cache con politica *write-back* e trasferimenti generici

Il caso più complicato è quello in cui il dispositivo deve scrivere in RAM parte di una cacheline, e questa si trova dirty in cache. La cacheline finale dovrà contenere il valore attualmente in cache per i byte non interessati dal trasferimento, e il valore scritto dal dispositivo per i rimanenti.

La soluzione interamente software cambia poco: il programmatore deve ordinare il write-back prima di avviare l'operazione, risolvendo il problema a monte. L'ottimizzazione dell'invalidazione senza write-back non può essere però usata, in quanto comporta la possibile perdita di dati nelle parti di memoria non appartenenti al buffer.

La soluzione interamente hardware può essere realizzata in vari modi, a seconda di quanto si vuol far fare al controllore cache e quanto al dispositivo. Il trasferimento in uscita (da RAM a dispositivo) non cambia sostanzialmente: il dispositivo è interessato solo a una parte della cacheline dirty, ma il contenuto da leggere è comunque interamente in cache, e si rende necessario uno snooping da parte del dispositivo. Il trasferimento in ingresso (da dispositivo a RAM), invece, può restare sostanzialmente immutato (con snopping da parte del controlore cache) solo se il controllore cache è in grado di aggiornare autonomamente la cache con i nuovi dati scritti dal dispositivo sul bus. Se, invece, il controllore è solo in grado di invalidare, il problema deve nuovamente essere risolto tramite snooping da parte del dispositivo: prima di eseguire il trasferimento parziale, il dispositivo esegue una operazione di snooping verso il controllore. In caso di hit, se il controllore risponde con un write-back, è sufficiente completare la normale scrittura in RAM. Se invece il controllore risponde inviando al dispositivo il contenuto della cacheline, il dispositivo deve combinare i dati contenuti nella cacheline con quelli da scrivere in memoria, e poi scrivere lui stesso l'intera cacheline in RAM.

In generale, un intero trasferimento che coninvole un buffer $[b, b + n)$ sarà composto, in base all'allineamento di b e al valore di n, da un possibile trasferimento di parte di una cacheline, seguito da zero o più trasferimenti di intere cacheline, e concluso da un eventuale trasferimento di parte di una cacheline. Si noti che lo snooping da parte del controllore cache è più efficiente dello snooping da parte del dispositivo, in quanto il primo avviene in parallelo con l'operazione in RAM, mentre il secondo deve essere completato prima di poter dialogare con la RAM. I trasferimenti che coinvolgono intere cacheline possono essere realizzati più efficientemente con lo snooping da parte del controllore, e solo quelli alle estremità richiedono lo snooping da parte del dispositivo. Per ottimizzare i trasferimento in ingresso è sufficiente che il dispositivo esegua il suo snooping solo nel caso di trasferimenti parziali, mentre il controllore cache esegua il suo in ogni caso.

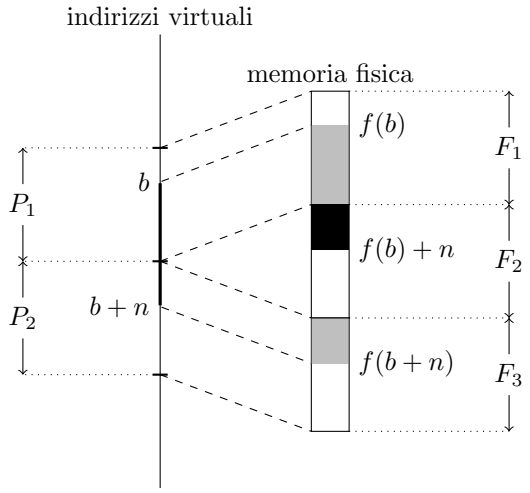

Figura 3: Un esempio con un buffer $[b, b+n)$ che attraversa due pagine (P_1 e P_2) mappate su due frame non contigui (F_1 e F_3).

1.2 Interazione con la memoria virtuale

Come sappiamo, tra la CPU e la cache troviamo la MMU, che traduce gli indirizzi virtuali (usati dal software) in indirizzi fisici (usati sul bus). Il dispositivo DMA, come collegato in Figura 2, non interagisce in alcun modo con la MMU e può utilizzare soltanto indirizzi fisici. Supponiamo che il software voglia eseguire un trasferimento in DMA da un buffer che si trova agli indirizzi *virtuali* $[b, b+n)$ verso un dispositivo (o viceversa). Saranno necessari i seguenti accorgimenti:

- al dispositivo andrà comunicato l'indirizzo *fisico* corrispondente a b, sia $f(b)$, e non l'indirizzo virtuale b;
- se l'intervallo $[b, b+n)$ attraversa più pagine che non sono tradotte in frame contigui, il trasferimento deve essere spezzato in più trasferimenti in modo che ciascuno di essi coinvolga solo frame contigui;
- la traduzione di tutti gli indirizzi in $[b, b+n)$ non deve cambiare mentre il trasferimento è in corso.

Tutti e tre i punti sono diretta conseguenza del fatto che il dispositivo non ha accesso alla traduzione da indirizzi virtuali a fisici. Consideriamo il primo punto: se comunicassimo b al dispositivo, questo lo userebbe come se fosse un indirizzo fisico, andando a leggere o scrivere in parti della memoria che non c'entrano niente con il buffer (tranne nel caso particolare in cui $b = f(b)$). Consideriamo il secondo punto e supponiamo che $[b, b+n)$ attraversi due pagine, P_1 e P_2, mappate su due frame non contigui, F_1 e F_3 (si veda la Figura 3). Se al dispositivo comunichiamo $f(b)$ ed n, questo leggerà (o scriverà) agli indirizzi *fisici* $[f(b), f(b)+n)$, invadendo dunque il frame F_2, con effetti disastrosi. Il

trasferimento, invece, deve essere spezzato in due parti: una da $f(b)$ fino alla fine di F_1, e uno dall'inizio di F_3 fino a $f(b+n)$ escluso. Alcuni dispositivi possono essere programmati per eseguire autonomamente più trasferimenti in successione, specificando in anticipo l'indirizzo e il numero di byte di ciascun trasferimento. Anche nei dispositivi che non offrono questa funzione, comunque, è sempre possibile programmare i diversi trasferimenti uno dopo l'altro in software.

Consideriamo invece il terzo punto, immaginando di trovarci in un sistema multiprocesso che realizzi, per esempio, lo swap-in/swap-out dei processi per poter eseguire più processi di quanti ne possano entrare in RAM. Supponiamo che un processo P_1 avvii un trasferimento in DMA verso un suo buffer privato e, mentre il trasferimento è in corso, il sistema decida di eseguire lo swap-out di P_1 per caricare un altro processo P_2 al suo posto. Il dispositivo DMA è ignaro del cambiamento e continuerà leggere o scrivere agli indirizzi fisici precedentemente occupati dal buffer di P_1 e ora occupati da parti della memoria di P_2, di nuovo con effetti disastrosi.

2 PCI Bus Mastering

La Figura 4 mostra un esempio di architettura con bus PCI. Come sappiamo, sul bus PCI ogni dispositivo (più preciamente, ogni funzione all'interno di ogni dispositivo) può comportarsi da "iniziatore" di una transazione. I dispositivi che sono in grado di essere iniziatori di transazioni sono detti *bus master*, e devono essere dotati dei collegamenti REQ/GNT verso l'arbitro, che ha il compito di coordinare l'accesso al bus tra i vari bus master. In Figura 4 abbiamo tre dispositivi collegati al bus PCI: il ponte, che è sempre bus master (in quanto deve iniziare le transazioni PCI per conto della CPU), un altro dispositivo bus master, e un terzo dispositivo che non è bus master. Solo il ponte e il secondo dispositivo hanno i collegamenti REQ/GNT con l'arbitro.

Dal punto di vista software l'operazione si svolge come se il dispositivo bus master fosse collegato direttamente al bus principale; in particolare, il software dialoga soltanto con il dispositivo per comunicargli, per esempio, l'indirizzo (fisico) del buffer il numero di byte da trasferire e la direzione del trasferimento. Dal punto di vista harware, però, tutto il trasferimento avviene per tramite del ponte. Ogni volta che il dispositivo vuole iniziare un trasferimento da o verso la RAM, inizia una transazione di memoria sul bus PCI, all'indirizzo opportuno. Il ponte è configurato in modo da comportarsi come obiettivo per tutte le transazioni di memoria con indirizzi appartenenti alla RAM che si trova sul bus principale. Il ponte, quindi, risponde alla transazione iniziata dal bus master e, in caso di trasferimento da RAM a dispositivo, esegue le necessarie operazioni di lettura DMA sul bus principale; in caso di trasferimento dal dispositivo alla RAM, copia temporaneamente i dati in arrivo dal dispositivo in un buffer interno e, in parallelo, inizia le necessarie operazioni di scrittura in DMA verso la RAM. Sul bus principale il ponte si comporta esattamente come un normale dispositivo DMA, e per lui valgono tutte le considerazioni che abbiamo svolto

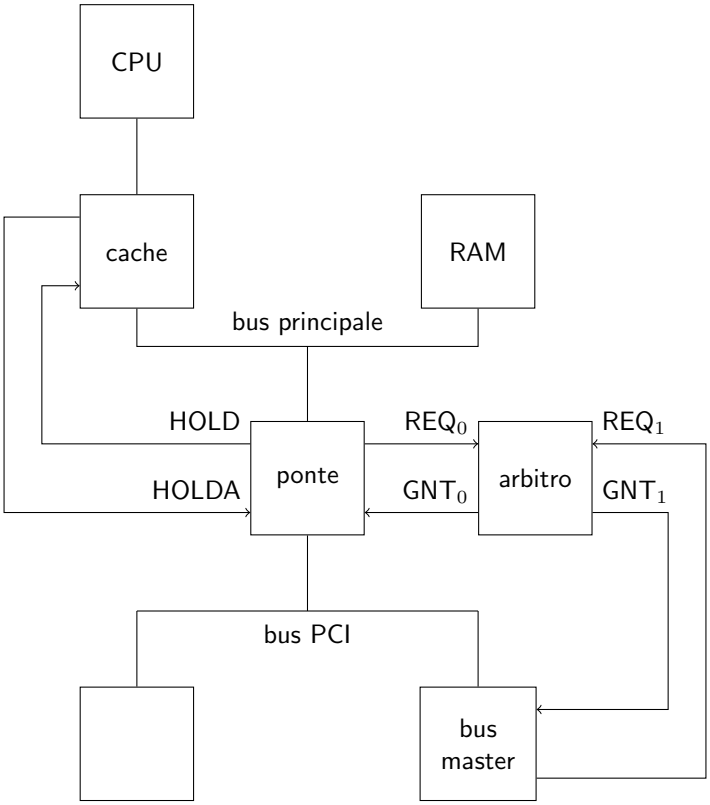

Figura 4: Bus Mastering PCI.

fino ad ora, in particolare per quanto riguarda il protocollo HOLD/HOLDA e le interazioni con la cache.

Si noti che il bus master può utilmente sfruttare la possibilità, offerta dallo standard PCI, di eseguire transazioni con più di una fase dati, in quanto conosce in anticipo il numero di byte da trasferire.

2.1 Interazione con il meccanismo delle interruzioni

La presenza della bufferizzazione intermedia nel ponte crea un problema con i trasferimenti in ingresso (da dispositivo a RAM) e il meccanismo delle interruzioni. In particolare, quando il dispositivo ha trasmesso al ponte l'ultimo byte richiesto, invierà una richiesta di interruzione per segnalare il completamento del trasferimento. Ci troviamo a questo punto di fronte ad una corsa: da una parte la richiesta di interruzione, che passa dal controllore delle interruzioni e arriva alla CPU, dall'altra i byte destinati alla memoria, che passano dal ponte: è possibile che la richiesta di interruzione arrivi e sia accettata prima che il ponte sia riuscito a trasferire tutti i byte in RAM; il software potrebbe quindi accedere erroneamente al buffer quanto ancora parte dei dati non sono stati aggiornati.

Anche questo problema può essere risolto in hardware o in software. Per risolverlo in software si può sfruttare il fatto che il ponte gestisce in modo strettamente FIFO tutti i trasferimenti di dati dal bus PCI verso il bus principale. La routine di interruzione, allora, prima di permettere l'acesso al buffer, potrebbe eseguire una lettura di un registro del dispositivo bus master. La risposta a questa lettura verrebbe accodata nel ponte *dopo* l'ultimo trasferimento di dati del bus master, e dunque l'istruzione di lettura verrebbe completata nella CPU necessariamente dopo che il ponte ha finito di trasferire i dati in RAM. Si noti che, dal momento che il dispositivo bus master aveva inviato una richiesta di interruzione, è probabilmente già previsto che la routine di interruzione debba leggere un qualche registro del dispositivo, per segnalare che la richiesta è stata servita: in tal caso, quest'unica lettura assolve entrambi i compiti.

Per risolvere il problema interamente in hardware, invece, si crea in genere un collegamento di handshake tra il controllore delle interruzioni e il ponte. Prima di inoltrare una qualunque richiesta di interruzione al processore, il controllore chiede l'OK al ponte; quest'ultimo non dà l'OK fino a quando non ha finito di trasferire tutti i dati ricevuti fino a quel momento. Un'altra soluzione, più moderna, prevede che le richieste di interruzione non viaggiono su linee separate, ma siano inoltrate come speciali transazioni sul bus PCI stesso (Message Signaled Interrupts). In questo modo le richieste si accodano naturalmente ai dati e non ci sono problemi di corse.