Il bus PCI

G. Lettieri

30 Aprile 2024

1 Il bus di espansione

Come molti altri computer dell'epoca e come i suoi discendenti di oggi, anche il PC AT era *espandibile*. In questo contesto l'aggettivo "espandibile" significa che il sistema contiene un *bus di espansione* con degli *slot* in cui possono essere inserite delle *schede di espansione* che possono aggiungere funzionalità al sistema: schede video, schede audio, schede di rete, etc. Così come per altri computer dell'epoca (come per esempio l'Apple II) queste schede di espansione potevano essere create e vendute da aziende indipendenti, senza una preventiva approvazione dell'IBM. Questa mancanza di centralizzazione ha creato una serie di scomodità per gli utenti delle schede di espansione. Per focalizzare le idee, concentriamoci sul problema seguente: le schede hanno registri e/o memoria interna a cui devono essere assegnati indirizzi nello spazio di I/O e/o di memoria affinché il software vi possa accedere. Nel PC AT, però, sono le schede stesse a riconoscere gli indirizzi dei propri registri, senza nessuna regola stabilita *a priori*. Questo può andar bene se qualcuno coordina la produzione delle schede, ma è un problema se le schede sono prodotte da aziende indipendenti e concorrenti. È chiaro che se l'azienda produttrice della scheda A e l'azienda produttrice della scheda B scelgono indirizzi sovrapposti per i loro registri, la scheda A e la scheda B non possono essere usate contemporaneamente nello stesso PC. Inoltre, a causa di queste possibili sovrapposizioni, il software non ha un modo affidabile per scoprire se una certa scheda è installata o meno: tutto ciò che può fare e provare a scrivere e leggere qualcosa dagli indirizzi dei registri di quella scheda per vedere se ottiene ciò che si aspetta, ma così facendo rischia di inviare comandi ad un'altra scheda che per caso usa quegli stessi indirizzi.

2 Lo standard PCI

Non solo le schede di espansione erano prodotte da aziende indipendenti: dal momento che il PC IBM era costruito con parti standard, aziende concorrenti dell'IBM crearono infiniti cloni "compatibili" del PC stesso, in grado di far girare lo stesso software e installare le stesse schede di espansione, malgrado i tentativi dell'IBM di arginare il fenomeno per vie legali. Il bus del PC AT venne chiamato ISA, per *Industry Standard Architecture*. Era uno standard di fatto

In alto compaiono CPU e RAM, collegate da un bus locale a un ponte ospite-PCI. Questo ponte collega il sistema al bus PCI principale. Sul bus PCI principale sono presenti: un dispositivo PCI collegato direttamente, un ponte PCI-PCI che porta a un secondo bus PCI, e un altro ponte PCI che porta a un bus non PCI, indicato in modo generico come ISA, ATA, USB, ecc. Sul secondo bus PCI sono collegati un altro dispositivo PCI e un dispositivo PCI multifunzione, rappresentato con due funzioni interne, funzione 0 e funzione 1. Il messaggio della figura è che il PCI non è solo un singolo bus piatto: da un bus PCI si può arrivare ad altri bus PCI tramite ponti, e i dispositivi possono essere semplici oppure multifunzione.

Figura 1: Esempio di architettura con bus PCI

che sostanzialmente descriveva il bus del PC AT, con tutti i suoi limiti. Per risolvere i problemi del bus ISA occorreva definire un nuovo standard e sperare che tutti vi aderissero. Ne furono proposti diversi, ma l'unico che ebbe successo fu lo standard *Periferal Component Interconnect* (PCI) proposto dall'Intel nel 1992. Lo standard PCI stabilisce quali sono i collegamenti del bus e il protocollo che le schede devono usare per comunicare. Inoltre, definisce tre spazi di indirizzamento: lo spazio di memoria, lo spazio di I/O e uno *spazio di configurazione*. I primi due spazi sono del tutto analoghi a quelli che già conosciamo e sono quelli che il software deve utilizzare, nel funzionamento a regime, per dialogare con le periferiche connesse al bus. Resta, ovviamente, il vincolo che i registri delle periferiche occupino indirizzi non sovrapposti, ma questo viene garantito nel seguente modo:

- le periferiche che rispettano lo standard non possono scegliere autonomamente gli indirizzi dei propri registri, ma devono contenere dei comparatori programmabili in modo che questi indirizzi siano impostati dal software;
- il software può impostare questi comparatori all'avvio del sistema, tramite il nuovo spazio di configurazione.

Lo spazio di configurazione è fatto in modo tale da poter accedere a dei registri di configurazione che ogni periferica deve avere per rispettare lo standard, e di poterlo fare in modo univoco senza conflitti. Accedendo a questi registri di configurazione il software di avvio (il PCI BIOS, nell'idea dello standard) può scoprire in modo affidabile quali periferiche sono connesse al bus (ricavandone anche il codice del venditore e il tipo), di quanti indirizzi ciascuna di esse ha bisogno, e programmare di conseguenze i comparatori in modo che non ci siano sovrapposizioni.

Una peculiarità dello standard PCI è che non è legato ad un particolare processore e prevede un dispositivo detto *ponte ospite-PCI* che faccia da tramite tra il bus dove risiede il processore (detto *bus locale*) e il bus PCI (si veda la Figura 1). Un altra peculiarità dello standard è di prevedere non un unico bus,

ma un *albero* di bus la cui radice è il bus PCI a cui è collegato il ponte ospite-PCI; gli altri bus sono collegati all'albero tramite ponti PCI-PCI, fino ad un massimo di 256 bus. Ad ogni bus si possono collegare fino a 32 dispositivi, e ciascun dispositivo può contenere da una a 8 funzioni (per esempio, una singola scheda di espansione conta come un dispositivo, ma al suo interno potrebbe contenere sia la funzione video che la funzione audio). Alcune funzioni possono fare da ponte verso altri tipi di bus: avremo dunque ponti PCI-ISA, ponti PCI-ATA, ponti PCI-USB, e così via.

2.1 Operazioni di lettura e scrittura

Sul bus possono transitare principalmente operazioni di lettura o scrittura in uno dei tre spazi (I/O, memoria o configurazione). Ciascuna operazione, detta *transazione*, è iniziata da un dispositivo detto *iniziatore* che cerca di operare su un altro dispositivo detto *obiettivo*. Lo standard permette a qualsiasi dispositivo di essere, in transazioni diverse, iniziatore o obiettivo. Questo permette, tra le altre cose, di realizzare il meccanismo di accesso diretto alla memoria di cui parleremo in seguito. Per il momento possiamo pensare che l'unico iniziatore sia il ponte ospite-PCI, che inizia delle transazioni sul bus PCI per tradurre le operazioni analoghe avviate dal processore sul bus locale.

Le transazioni si svolgono in più fasi: una fase di indirizzamento e una o più fasi di scambio dati. Durante la fase di indirizzamento l'iniziatore specifica il tipo di operazione (lettura o scrittura e spazio di indirizzamento) e l'indirizzo del primo byte da leggere o scrivere. Durante questa fase tutti i dispositivi che hanno registri in quello spazio confrontano l'indirizzo specificato dall'iniziatore con quello contenuto nei propri comparatori. Al più un dispositivo troverà una corrispondenza, e quello diventa l'obiettivo della transazione. Nelle successive fasi di scambio dati avviene il trasferimento dei dati dall'iniziatore all'obiettivo (scrittura) o dall'obiettivo all'iniziatore (lettura).

I principali collegamenti del bus sono i seguenti:

FRAME# (uscita per l'iniziatore, ingresso per l'obiettivo) delimita ("incornicia") l'inizio e il termine di ogni transazione;

DEVSEL# (ingresso per l'iniziatore, uscita per l'obiettivo) attivato dall'obiettivo che riconosce uno dei propri indirizzi durante la fase di indirizzamento

C/BE#[3:0] (uscita per l'iniziatore, ingresso per l'obiettivo) codificano il tipo di operazione nella fase di indirizzamento (C[3:0]), e fungono byte-enable nelle fasi di trasferimento dati (BE#[3:0]);

AD[31:0] (ingresso/uscita per tutti) codificano l'indirizzo nella fase di indirizzamento e trasportano i dati nelle fasi di trasferimento dati;

TRDY# (*target ready*, ingresso per l'iniziatore, uscita per l'obiettivo) handshake nelle fasi di scambio dati;

IRDY# (*initiator ready*, uscita per l'iniziatore, ingresso per l'obiettivo) handshake nelle fasi di scambio dati;

La figura mostra, in funzione del tempo, i segnali CLK, FRAME#, AD[31:0], C/BE#[3:0], IRDY#, TRDY# e DEVSEL#. All'inizio l'iniziatore attiva FRAME# e mette sulle linee AD un indirizzo a. In parallelo, sulle linee C/BE# compare il codice 0110, che indica una lettura nello spazio di memoria. Un dispositivo riconosce l'indirizzo e il comando, quindi attiva DEVSEL#. Dopo la fase di indirizzo, il pilotaggio delle linee AD passa dal master al dispositivo destinatario, che mette sul bus le quattro parole di dati w_1, w_2, w_3 e w_4. Durante le fasi di trasferimento, IRDY# e TRDY# risultano attivi, quindi i dati scorrono senza attese aggiuntive. Durante l'ultimo trasferimento, l'iniziatore disattiva FRAME#, segnalando la fine imminente della transazione. Il messaggio della figura è che una lettura PCI è composta da una fase iniziale di indirizzo e comando, seguita da una o più fasi dati in cui il target restituisce i dati richiesti.

Figura 2: Esempio di transazione di lettura nello spazio di memoria, con 4 fasi di trasferimento dati.

STOP# (ingresso per l'iniziatore, uscita per l'obiettivo) serve a terminare prematuramente una transazione;

CLK (ingresso per tutti) segnale di clock; tutti i dispositivi campionano i loro segnali in ingresso sul fronte in salita di questo segnale.

I segnali il cui nome termina in "#" sono attivi bassi, mentre i nomi seguiti da quadre rappresentano un gruppo di più segnali. Come si vede, il bus è a 32 bit sia per gli indirizzi che per i dati. Venne definita una estensione a 64 bit, che non fu molto usata e poi venne superata dal PCI Express. Noi ci limiteremo alla versione a 32 bit.

Il protocollo è il seguente: l'iniziatore pilota C[3:0] e AD[31:0] con il tipo di operazione e l'indirizzo iniziale del trasferimento, quindi attiva FRAME# per segnalare l'inizio di una transazione. Se uno dei dispositivi riconosce il tipo di operazione e l'indirizzo attiva DEVSEL#. Se nessun dispositivo attiva DEVSEL# entro un certo numero di clock, l'iniziatore termina l'operazione con un errore, altrimenti si prosegue con le fasi dati. Nelle operazioni di lettura (si veda l'esempio in Figura 2) l'obiettivo pilota le linee AD[31:0] e segnala la loro validità attivando la linea TRDY#, quindi attende che l'iniziatore segnali di aver ricevuto i dati attivando IRDY#; viceversa, l'iniziatore attende che TDRY# sia attivo, quindi campiona AD[31:0] (sullo stesso fronte del clock in TRDY# attivo) e, quando è pronto per il prossimo trasferimento, attiva IRDY#. Nelle transazioni di scrittura (esempio in Fig. 3) vale l'opposto: l'iniziatore pilota le linee AD[31:0] BE#[3:0] e attiva IRDY#, quindi attende TRDY#, mentre l'obiettivo attiva TRDY# quando è pronto a ricevere e attende IRDY# per sapere quando AD[31:0] e BE#[3:0] sono validi. Ogni fase dati trasferisce al più 4 byte allineati naturalmente. Ciascuna fase dati si conclude quando sia IRDY# che TRDY# sono attivi sullo stesso fronte di salita del clock.

I segnali mostrati sono gli stessi della figura precedente: CLK, FRAME#, AD[31:0], C/BE#[3:0], IRDY#, TRDY# e DEVSEL#. All'inizio compare sulle linee AD l'indirizzo a e sulle linee C/BE# il codice 0111, che identifica una scrittura. Successivamente, nelle quattro fasi dati, sulle linee AD transitano le parole w_1, w_2, w_3 e w_4. A differenza della lettura, qui il master continua a guidare il bus dati anche durante i trasferimenti; per questo il target può attivare TRDY# più rapidamente. Le linee C/BE# durante le fasi dati riportano i segnali BE#, cioè i byte enable, che possono cambiare da una parola all'altra per indicare quali byte sono validi. Il messaggio della figura è che una scrittura PCI ha struttura simile a una lettura, ma nelle fasi dati è l'iniziatore a fornire le parole sul bus, mentre il target si limita a confermare la disponibilità a riceverle.

Figura 3: Esempio di transazione di scrittura con 4 fasi di trasferimento dati.

Il riutilizzo delle stesse linee per scopi diversi riduce i costi a scapito della velocità di trasferimento, ma questa è in gran parte recuperata dalla possibilità di eseguire più fasi dati con una singola fase di indirizzamento. In tutte le transazioni (sia lettura che scrittura) è l'iniziatore a decidere il numero di fasi trasferimento: se FRAME# è attivo quando IRDY# è attivo, l'iniziatore sta chiedendo una ulteriore fase dati. Viceversa, l'obiettivo riconosce l'ultima fase di trasferimento quando campiona IRDY# attivo con FRAME# disattivo. L'obiettivo può attivare STOP# per terminare forzatamente la transazione. Un motivo per farlo può essere il seguente: per poter eseguire più fasi dati, l'obiettivo deve poter memorizzare internamente l'indirizzo di partenza e poi incrementarlo autonomamente in ogni fase dati. Un dispositivo che non è in grado di farlo attiverà STOP# la prima volta che attiva TRDY#. Questo costringe l'iniziatore a iniziare una nuova transazione per il prossimo trasferimento.

Anche se, dal punto di vista del bus PCI, l'iniziatore delle transazioni è il ponte ospite-PCI, questo non inizia mai transazioni di sua spontanea volontà, ma solo per tradurre analoghe operazioni iniziate dalla CPU (o dal controllore cache) sul bus locale. Nella nostra macchina il ponte risponde a tutte le operazioni nello spazio di I/O e ad alcune operazioni nello spazio di memoria. Il ponte ha alcuni registri nello spazio di I/O del processore e risponde direttamente alle operazioni che li indirizzano (si veda la Sezione 2.2.2), ma per il resto traduce tutte le operazioni nelle analoghe transazioni sul bus PCI, dialogando con l'obiettivo per conto della CPU o del controllore cache. La traduzione è trasparente: le operazioni nello spazio di I/O del processore si traducono in transazioni nello spazio di I/O PCI allo stesso indirizzo, e analogamente per lo spazio di memoria. Dal punto di vista della CPU (e dunque del software) è come se i dispositivi obiettivo di queste transazioni si trovassero direttamente sul bus locale. Affinchè questo avvenga, però, i dispositivi devono essere prima configurati.

La figura rappresenta una tabella verticale, organizzata per offset a passi di 4 byte. In alto, alle posizioni iniziali, compaiono: Vendor ID e Device ID a offset 0x00, Command e Status a offset 0x04, Revision ID e Class Code a offset 0x08, altri campi di intestazione inclusi Header Type e Cache Line Size a offset 0x0c. Seguono sei registri consecutivi: Base Address Register 0 a offset 0x10, BAR1 a 0x14, BAR2 a 0x18, BAR3 a 0x1c, BAR4 a 0x20, BAR5 a 0x24. Nella parte più bassa compaiono altri registri standard, fino ai campi Intr. Line e Intr. Pin a offset 0x3c. Il messaggio della figura è che ogni funzione PCI possiede uno spazio di configurazione standardizzato, in cui si trovano identificatori, comandi, informazioni di classe, registri BAR e dati relativi agli interrupt.

Figura 4: Lo spazio di configurazione di ogni funzione PCI

2.2 Lo spazio di configurazione

Lo standard PCI prevede uno spazio di configurazione di 64 parole (da 4 byte) per ogni funzione (ogni dispositivo ha almeno una funzione). In questo spazio ciascuna funzione rende accessibili i propri registri di configurazione, seguendo lo schema illustrato in Figura 4, dove abbiamo evidenziato solo i registri che ci interessano. Non tutti i registri menzionati in Figura 4 devono essere sempre presenti, ma, con la possibile eccezione del ponte ospite-PCI, tutte le funzioni devono avere i registri che nella Figura hanno lo sfondo grigio[1].

I registri obbligatori hanno il seguente significato:

Vendor ID (lettura) un codice che identifica il produttore della funzione, assegnato ad ogni produttore da una autorità centrale[2];

Device ID (lettura) un codice che identifica la funzione, assegnato dal produttore;

Command (lettura/scrittura) permette di abilitare o disabilitare varie capacità della funzione (vedere sotto);

Status (lettura) descrive alcune capacità della funzione, più lo stato di alcuni eventi, per lo più legati al verificarsi di errori;

Class Code (lettura) un codice che identifica in modo generico il tipo di funzione;

Revision ID (lettura) numero di revisione della funzione, assegnato dal produttore;

[1] La revisione 2.2 dello standard ha aggiunto altri registri obbligatori, che non consideriamo per semplicità.

[2] Il PCI SIG (PCI Special Interest Group), `https://pcisig.com/`.

Header Type (lettura) tipo di header; deve essere 0 per tutte le funzioni che non siano ponti PCI-PCI o PCI-CardBus.

Del registro Command ci interessano solo i bit 0, 1 e 2. I bit 0 e 1 abilitano la funzione a rispondere alle transazioni nello spazio di I/O e di memoria, rispettivamente. Questi bit sono per default a zero (transazioni disabilitate) e il software di inizializzazione deve metterli a 1 dopo aver assegnato alla funzione gli indirizzi necessari nei corrispondenti spazi. Come vedremo in seguito, il bit 2 abilita la funzione a comportarsi da *bus master*, qualora ne abbia la capacità. Se una funzione non ha registri nello spazio di memoria o di I/O (o non ha la capacità di comportarsi da bus master), il corrispondente bit vale 0 e non è scrivibile.

2.2.1 Transazioni nello spazio di configurazione

Nel seguito ci limiteremo a considerare il caso di un singolo bus PCI, quello direttamente collegato al ponte ospite-PCI, che è l'unico sicuramente presente.

Tramite le transazioni nello spazio di configurazione si può accedere ai registri di configurazione di ciascuna funzione, e a nient'altro. Inoltre, solo il ponte ospite-PCI può essere iniziatore di queste transazioni.

Ogni dispositivo PCI diverso dal ponte ospite-PCI, e ogni slot di espansione, ha un ingresso chiamato IDSEL. Il ponte ospite-PCI, invece, possiede un diverso collegamento verso ciascuno di questi ingressi. Per accedere ad un particolare registro di una particolare funzione, il ponte deve selezionare il dispositivo che contiene la funzione desiderata attivandone il segnale IDSEL, e poi passargli il numero della funzione (da 0 a 7), l'offset della parola che contiene il registro (colonna destra in Figura 4) e i byte enable relativi al registro. A parte l'uso di IDSEL, le transazioni nello spazio di configurazione sono per il resto simili a quelle negli altri spazi: nella fase di indirizzamento, oltre ad attivare FRAME#, il ponte attiva l'IDSEL appropriato e usa AD[7:0] per passare l'offset (con AD[1] e AD[0] sempre a zero) e AD[10:8] per passare il numero di funzione. Se l'IDSEL attivato è effettivamente collegato ad un dispositivo (potrebbe essere collegato ad uno slot di espansione vuoto), questo deve rispondere attivando DEVSEL# e poi procedere come nelle altre transazioni. Si noti che, dal momento che lo standard impone che DEVSEL# debba essere attivato entro un limite di tempo prestabilito, il ponte dispone di un modo affidabile per riconoscere gli slot vuoti: sono quelli in cui DEVSEL# non viene attivato in tempo.

2.2.2 Accesso allo spazio di configurazione via software

Anche nel caso di transazioni nello spazio di configurazione l'iniziatore è il ponte ospite-PCI dal punto di vista del bus PCI, ma le transazioni sono in realtà iniziate dal software in esecuzione sulla CPU. Questa volta, però, la CPU non dispone di istruzioni che permettano di interagire direttamente con lo spazio di configurazione. Invece, il ponte mette a disposizione due registri nello spazio di I/O della CPU, e tramite questi registri il software può accedere indirettamente

La figura mostra una parola di 32 bit suddivisa in campi. Da sinistra a destra compaiono: un bit più significativo posto a 1, che abilita l'accesso di configurazione; un campo bus; un campo dispositivo; un campo funzione; un campo offset; gli ultimi due bit, fissati a 0. Il significato della figura è che, per accedere allo spazio di configurazione PCI, il software costruisce un indirizzo che codifica in un'unica parola il bus, il dispositivo, la funzione e l'offset del registro desiderato.

Figura 5: Configuration Address Port (CAP)

allo spazio di configurazione PCI. I due registri, entrambi a 32 bit, sono i seguenti:

Configuration Address Port (CAP) (indirizzo 0xCF8) permette di selezionare una funzione e l'offset della parola a cui si vuole accedere;

Configuration Data Port (CDP) (indirizzo 0xCFC) permette di accedere alla parola selezionata tramite CAP.

Per individuare una particolare funzione si usano tre numeri: il numero di bus, il numero del dispositivo nel bus e il numero della funzione nel dispositivo. Nel nostro caso, il numero di bus è sempre 0. Il numero di dispositivo dipende unicamente dal piedino IDSEL a cui il dispositivo è collegato, mentre il numero di funzione è deciso dal costruttore del dispositivo, con il vincolo che la funzione 0 deve essere sempre presente. In ogni caso, i tre numeri sono stabiliti *a priori* in modo univoco, ed è questo che permette allo spazio di configurazione di essere accessibile già dall'avvio del sistema.

I tre numeri e l'offset della parola a cui si vuole accedere vanno scritti in CAP nel modo illustrato in Figura 5. Una volta che CAP è stato impostato, il ponte ospite-PCI tradurrà le letture e le scritture in CDP in corrispondenti transazioni nello spazio di configurazione. In particolare, il ponte attiverà l'uscita IDSEL che corrisponde al numero di dispositivo selezionato in CAP, copierà il numero di funzione e l'offset da CAP sulle linee AD[10:0], e userà come BE#[3:0] i primi 4 byte enable del processore.

Se l'operazione in CDP è di lettura, il ponte fa anche un'altra cosa: se nessun dispositivo risponde alla corrispondente transazione di lettura nello spazio di configurazione (nessuno attiva DEVSEL# in tempo), il ponte restituisce un valore di tutti 1 al processore. La lettura di questo valore permette quindi al software di rilevare in maniera affidabile l'assenza di un dispositivo nella posizione selezionata. In particolare, dal momento che nessun produttore ha ID 0xFFFF, le funzioni PCI installate sul sistema possono essere scoperte all'avvio provando a leggere il Vendor ID da tutti i possibili numeri di dispositivo (da 0 a 31) con funzione 0 (che, ricordiamo, ogni dispositivo deve avere): la lettura di un valore diverso da 0xFFFF indica la presenza di un dispositivo, che poi può essere ulteriormente ispezionato e configurato.

La figura rappresenta un registro di 32 bit. La parte principale contiene l'indirizzo base del blocco, mentre alcuni bit bassi descrivono il tipo della regione. I bit in grigio sono indicati come non scrivibili, perché dipendono dall'allineamento richiesto dalla dimensione del blocco oppure sono riservati. Tra i bit meno significativi, il bit 0 vale 0, a indicare che il BAR riguarda spazio di memoria e non spazio di I/O. Il messaggio della figura è che un BAR di memoria non contiene un indirizzo arbitrario: alcuni bit bassi sono vincolati dal tipo di regione e dalla sua dimensione, quindi il dispositivo impone un certo allineamento dell'indirizzo base.

Figura 6: Base Address Register (BAR), caso di blocco nello spazio di memoria

Anche qui è mostrato un registro di 32 bit. La parte alta contiene l'indirizzo base del blocco, mentre diversi bit bassi sono riservati o fissati. In questo caso il bit 0 vale 1, a indicare che il BAR si riferisce a spazio di I/O. Anche in questa figura i bit grigi sono indicati come non scrivibili, perché dipendono dalla dimensione della regione o dalla codifica del tipo. Il messaggio della figura è che un BAR di I/O si distingue da un BAR di memoria già dal bit meno significativo e, come nel caso precedente, l'indirizzo base deve rispettare vincoli di allineamento.

Figura 7: Base Address Register (BAR), caso di blocco nello spazio di I/O

2.2.3 I Base Address Register (BAR)

Lo scopo principale della fase di configurazione dei dispositivi è quello di assegnare indirizzi univoci negli spazi di I/O e/o di memoria, in modo che da quel momento in poi il software possa accedere ai registri direttamente. Si tenga presente che in questo caso stiamo parlando dei registri specifici della funzione (per esempio, il registro RBR di una interfaccia di ingresso), diversi dai registri che si trovano nello spazio di configurazione. Il costruttore della scheda deve definire uno o più *blocchi* di indirizzi, ciascuno con una dimensione che sia una potenza di due, all'interno dei quali rendere disponibili i registri specifici della funzione. Il costruttore deve anche decidere, per ciascun blocco, se deve trovarsi nello spazio (PCI) di I/O o di memoria, e quale è l'organizzazione interna di ciascun blocco (a che offset si trovano i vari registri all'interno del blocco). Quello che non può decidere, però, è l'indirizzo di partenza dei blocchi. Invece, per ogni blocco, deve prevedere un Base Address Register (BAR) nello spazio di configurazione (offset 0x10–0x24 in Figura 4). Sarà il software di inizializzazione a scegliere gli indirizzi di partenza in modo che non vi siano conflitti con i blocchi delle altre funzioni.

I BAR sono organizzati come in Figura 6 e 7. Si noti che alcuni bit non sono scrivibili e servono a dare al software di inizializzazione indicazioni sul

tipo di blocco e sulla sua dimensione. In particolare, il bit 0 indica se il blocco è pensato per lo spazio di I/O (bit 0 uguale a 1) o di memoria (bit 0 uguale a 0). La dimensione del blocco è 2^b, dove b è il numero di bit che non sono scrivibili. Il software di inizializzazione può scoprire quanto vale b provando a scrivere tutti 1 nel BAR e rileggendo il risultato, per vedere quali bit sono stati effettivamente scritti. Si noti che $b \geq 4$ per i blocchi di memoria, che quindi hanno una dimensione minima di 16 byte, mentre $b \geq 2$ per i blocchi di I/O, corrispondenti ad una dimensione minima di 4 byte[3].

I blocchi possono essere assegnati solo a regioni allineate naturalmente alla loro dimensione: il software di inizializzazione sceglierà la regione e ne scriverà il numero nei bit scrivibili del BAR. In questo modo il BAR (una volta azzerati i 4 o 2 bit meno significativi) conterrà l'indirizzo di partenza del blocco.

Dopo aver assegnato una regione a un blocco, il software di inizializzazione setterà il bit 0 o 1 del registro di Comando (a seconda che il blocco sia di I/O o di memoria). Da quel momento in poi la funzione risponderà alle transazioni i cui indirizzi cadono in quella regione.

Nella macchina QEMU l'inizializzazione è fatta dal PCI BIOS, quindi viene completata prima che il nostro software parta. In teoria è sempre possibile rifarla, ma noi ci limiteremo ad accettare le impostazioni del BIOS. Tutto ciò che ci serve è sapere quali indirizzi il BIOS ha assegnato ai vari blocchi, e questo possiamo farlo leggendo i BAR.

2.3 Le interruzioni

I dispositivi PCI possono generare richieste di interruzione, ma molti dei dettagli non sono fissati dallo standard, in quanto dipendono dal particolare sistema in cui si trova il bus.

Ogni dispositivo ha fino a quattro piedini di uscita per le richieste di interruzione: INTA#, INTB#, INTC# e INTD#. Ogni funzione di un dispositivo può essere collegata ad al più uno di questi piedini (anche nessuno), e più funzioni dello stesso dispositivo possono essere collegate allo stesso piedino. Il piedino utilizzato da una funzione deve essere leggibile dal registro di configurazione chiamato "Intr. Pin" in Figura 4 (un byte, sola lettura). Un valore 0 in questo registro indica che la funzione non genera richieste di interruzione, 1 indica che usa il pin INTA#, 2 indica INTB#, etc.

Chi scrive il software che deve accedere alla funzione, però, non vuole sapere questo; piuttosto, ha bisogno di sapere a quale piedino del controllore delle interruzioni ciascuna funzione è collegata. Lo standard non lo dice, in quanto si ferma ai piedini INTA#, INTB#, etc., lasciando piena libertà al progettista su come collegare questi piedini al resto. Il progettista, però, deve garantire che il BIOS scriva questa informazione nel registro "Intr. line" (un byte, lettura/-scrittura) in Figura 4. Nel nostro caso il BIOS scrive in "Intr. line" proprio il numero del piedino dell'APIC a cui la funzione è collegata, quindi possiamo limitarci a leggere questo registro.

[3] Nei blocchi di memoria i bit 1–3 contengono ulteriori indicazioni sul tipo di memoria, che tralasciamo per semplicità; nei blocchi di I/O il bit 1 è riservato e vale sempre 0.

In alto compaiono CPU e RAM, collegate da un bus locale a un ponte ospite-PCI, identificato come North Bridge i440fx con identificativo 8086:1237. Questo ponte collega il sistema al bus PCI numero 0. Sul bus PCI 0 sono collegati: una scheda video bochsvga (dispositivo 1234:1111), un dispositivo South Bridge PIIX3 (dispositivo 1) e il collegamento dal North Bridge verso il bus PCI. All'interno del South Bridge PIIX3 sono evidenziate almeno due funzioni PCI: funzione 0, che realizza un ponte PCI-ISA con identificativo 8086:7000, e funzione 1, che realizza un ponte PCI-ATA con identificativo 8086:7010. Dal South Bridge dipendono varie periferiche: tastiera e timer, collegate alla parte PCI-ISA; hard disk, collegato alla parte PCI-ATA; APIC, mostrato come periferica accessibile tramite il South Bridge. Il messaggio della figura è che, nella macchina QEMU, le periferiche non sono tutte collegate direttamente alla CPU: il North Bridge collega CPU, RAM e PCI, mentre il South Bridge raccoglie più funzioni e fornisce l'accesso a periferiche legacy e a dispositivi di storage.

Figura 8: Architettura dei bus delle periferiche nella macchina QEMU

2.4 Il bus PCI nel PC

Nel seguito faremo riferimento all'architettura di Fig. 8, con un unico bus PCI e due ponti, uno verso il bus ISA (dove si trovano le vecchie periferiche del PC AT, come la tastiera e il timer) e uno verso il bus ATA, dove si trovano gli hard disk. Questa architettura era comune intorno agli anni 2000. Il ponte ospite-PCI veniva chiamato *North Bridge* e spesso contenva altre funzioni, come quella di controllore della memoria (la memoria era dunque collegata al North Bridge invece che al bus locale, ma noi continueremo a supporre che sia collegata direttamente al bus locale, come in Figura 8); i ponti PCI-ISA e PCI-ATA erano inglobati in un altro dispositivo che svolgeva queste e altre funzioni, chiamato *South Bridge*. I PC moderni adottano lo standard PCI Express, che ha superato il vecchio standard PCI mantenendo la compatibilià a livello software. Noi ci concentreremo solo sullo standard PCI, che è molto più semplice del PCI Express ed quello emulato dalla nostra macchina virtuale QEMU. Inoltre, conoscere lo standard PCI facilita l'eventuale studio dello standard PCI Express.