Interfacce hardware

Lezione 11 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2019-20

Indice

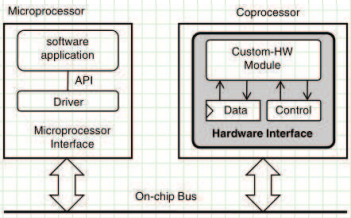
argomenti della lezione

di che si tratta:

- l'interfaccia hardware di coprocessore: funzioni e layout
- progetto dell'hardware per i dati
  - indirizzamento dei dati
  - multiplazione e masking
- progetto del controllo
  - controllo gerarchico
- mappa degli indirizzi
- insieme delle istruzioni
- esempio: un caso di accelerazione hardware
  - decisioni di progetto
  - interfaccia Avalon e modello di programmazione

funzioni dell'interfaccia hardware di coprocessore

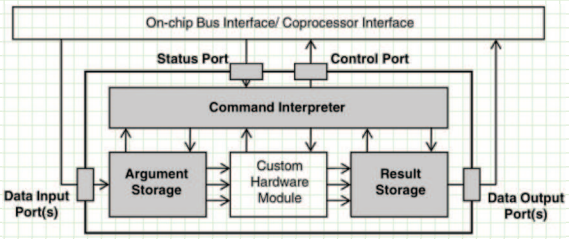
un'interfaccia hardware connette un modulo hardware custom a un bus di coprocessore o su chip
  l'interfaccia hardware pilota le porte di I/O del modulo hardware custom
il progetto dell'interfaccia hardware deve combinare la flessibilità dell'hardware custom con gli aspetti reali dell'interfaccia hardware/software
funzioni tipiche dell'interfaccia hardware:



Schaumont, Figure 12.1 - The hardware interface maps a custom-hardware module to a hardware-software interface

- *trasferimento di dati*: operazioni read/write su bus on-chip, handshake su bus di coprocessore, o anche trasferimenti ottimizzati di dati a raffiche o ad alta velocità, e.g. in DMA
- *conversione di lunghezza di parola*: dimensione e numero degli operandi del modulo hardware possono essere arbitrari, i dati per la comunicazione su bus sono limitati in entrambi gli aspetti
- *memorizzazione di operandi*: memoria locale di argomenti e parametri del modulo hardware custom: gli argomenti sono aggiornati a ogni sua esecuzione, non così i suoi parametri
- *insieme di istruzioni*: l'insieme di istruzioni custom è un aspetto chiave dell'interfaccia hardware, il suo progetto definisce la visione che il software ha del componente hardware
- *controllo locale*: realizzazione di interazioni per il controllo locale del modulo hardware custom, quali una sequenza di microoperazioni in risposta a un singolo comando software

struttura di un'interfaccia hardware di coprocessore



Schaumont, Figure 12.2 - Layout of a coprocessor hardware interface

componenti comuni di un'interfaccia hardware:

- buffer dati in ingresso: *memoria di argomenti*
- buffer dati in uscita: *memoria del risultato*
- *interprete di comandi*: controllo locale in base ai comandi software

dalla prospettiva del modulo hardware custom, è comune partizionare la collezione di porte in porte di ingresso/uscita dei dati e porte di controllo/stato

> è da notare come, in fig. 12.2, segnali di controllo e segnali dei dati siano ortogonali: il controllo fluisce verticalmente, i dati orizzontalmente

la separazione di controllo e dati è un aspetto importante del progetto poiché, nel progetto di un coprocessore, la granularità dell'interazione fra dati e controllo è scelta dal progettista

> si trattano appresso sia il *progetto dei dati* che il *progetto del controllo*

indirizzamento dei dati

proprietà di una porta dati del coprocessore: *larghezza*, *direzione* e *tasso di aggiornamento*

> estremi del tasso di aggiornamento: *parametro*, impostato solo al reset del modulo, e *argomento di funzione*, che cambia valore a ogni esecuzione del modulo hardware

per una buona corrispondenza delle effettive porte hardware alle porte dell'interfaccia custom, conviene partire dalle proprietà delle porte hardware

> e.g, un coprocessore per la funzione GCD, specificata da `int gcd(int m, int n)`, ha due porte d'ingresso e una di uscita, tutte da 32 bit, ad aggiornamento frequente

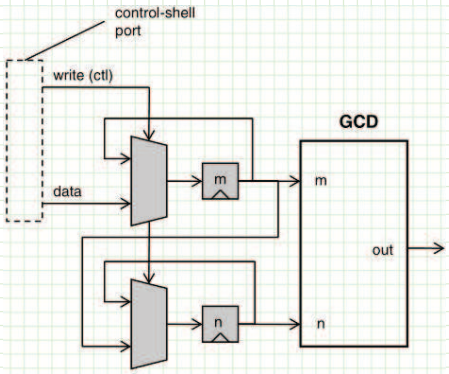
nella realizzazione di questo modulo come coprocessore mappato in memoria, le porte dell'interfaccia hardware sarebbero realizzate come registri mappati in memoria

> un approccio naturale è mappare ciascuna porta hardware custom a un distinto registro mappato in memoria, così da avere ciascuna porta indipendentemente indirizzabile dal software

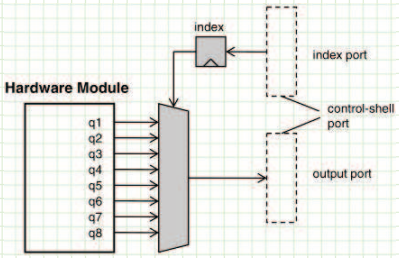
tuttavia, non sempre è possibile allocare un numero arbitrario di porte mappate in memoria nell'interfaccia hardware; in tal caso si devono multiplare le porte del modulo hardware custom sulle porte dell'interfaccia hardware

> oltre alla penuria di porte d'interfaccia, un altro motivo per la multiplazione può essere la bassa frequenza di aggiornamento di alcune porte del modulo hardware custom, sì che risulta inefficiente allocare una porta d'interfaccia separata a ciascuna di esse

si può realizzare la multiplazione in modi diversi: il primo è la *multiplazione nel tempo* delle porte del modulo hardware, la seconda con l'uso di un *registro indice* nell'interfaccia hardware



Schaumont, Figure 12.3 - Time-multiplexing of two hardware-module ports over a single control-shell port



Schaumont, Figure 12.4 - Index-register to select one of eight output ports

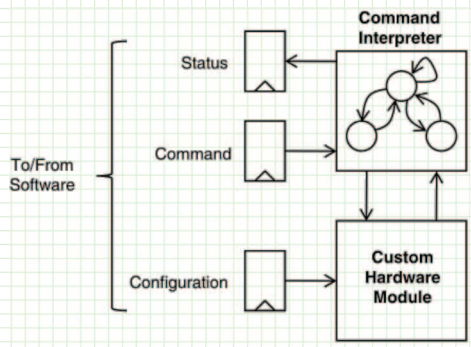
la multiplazione è anche utile per trasferire a pezzi operandi lunghi, sì che l'operando passa un pezzo alla volta con la multiplazione nel tempo

la tecnica di *masking* è adatta a operandi molto brevi, e.g. per raggruppare diverse porte da un bit del modulo hardware in una porta dell'interfaccia hardware: si usa a tal fine un *registro mask* per indicare le porte del modulo da aggiornare, e.g.: `new_hw_port = (old_hw_port & ~mask) | (upd_value & mask)`

il progetto del controllo in un coprocessore è l'insieme di attività per generare segnali di controllo e catturare segnali di stato

ne risulta un insieme di *comandi* o istruzioni, progettati ad-hoc, che possono essere eseguiti dal coprocessore

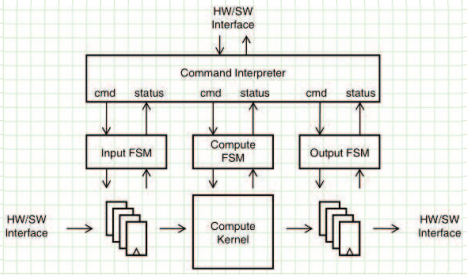
la figura 12.5 mostra una generica architettura per controllare un modulo hardware custom



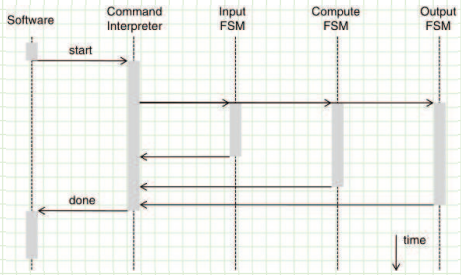
Schaumont, Figure 12.5 - Command design of a hardware interface

- un *interprete dei comandi*, al vertice del controllo nel coprocessore, accetta comandi dal software e restituisce informazione di stato
- mentre un comando è una operazione di controllo immediata, una *configurazione* è un valore che avrà effetti sull'esecuzione del coprocessore per un tempo prolungato, anche su più comandi

la figura 12.6 mostra l'architettura di un coprocessore che può realizzare la sovrapposizione di comunicazione e calcolo, come illustrato in figura 12.7



Schaumont, Figure 12.6 - Hierarchical control in a coprocessor



Schaumont, Figure 12.7 - Execution overlap using hierarchical control

l'interprete analizza ciascun comando dal software e lo suddivide in una combinazione di comandi per le FSM del livello inferiore

> e.g., per un coprocessore che abbia un banco di registri indirizzabili nel buffer d'ingresso o di uscita, l'indirizzo del registro può essere incorporato nel comando proveniente dal software

per l'effettiva sovrapposizione dell'esecuzione si deve organizzare il *pipelining* delle azioni delle FSM, con adattamento dell'interprete dei comandi ai piani di esecuzione delle FSM del livello inferiore

modello di programmazione = progetto del controllo + progetto dei dati

> il *modello di programmazione*, cioè la vista software di un modulo hardware, include una collezione delle aree di memoria usate dal modulo hardware custom e una definizione dei comandi (o istruzioni) accettati dal modulo

la *mappa degli indirizzi* riflette l'organizzazione di elementi di memoria del modulo hardware accessibili al software in lettura e in scrittura; il suo progetto dovrebbe procedere dal punto di vista del progettista del software piuttosto che dell'hardware, perciò:

- un dato indirizzo di memoria dovrebbe sempre riferirsi allo stesso registro hardware, indipendentemente dal tipo di operazione, lettura o scrittura, su di esso
- per default tutti i registri mappati in memoria dovrebbero essere accessibili in lettura e scrittura; in alcuni casi sono giustificati registri solo in lettura, e.g. registri di stato dell'hardware o dati di segnali campionati; tuttavia, sono molto rari i casi in cui si giustifica l'uso di un registro solo in scrittura
- la mappa degli indirizzi dovrebbe rispettare l'allineamento del processore; e.g., estrarre i bit 5–12 da una parola di 32 bit è più complicato che estrarre il secondo byte della stessa parola

il progetto di un buon insieme di istruzioni è un problema difficile, che si pone al progettista in termini di bilanciamento tra flessibilità ed efficienza

il problema dipende fortemente dalla funzione del modulo hardware custom

ecco alcune generiche linee-guida per il progetto:

- si possono distinguere tre classi di istruzioni: comandi one-time, comandi on-off, configurazioni; la loro miscela influisce sul comportamento generale del modulo hardware e dovrebbe essere mirata a minimizzare l'entità dell'interazione di controllo fra driver software e modulo hardware
- progettare la sincronizzazione tra software e hardware a più livelli di astrazione, cioè non solo al livello del trasferimento di dati ma anche a quello algoritmico
- un altro problema di sincronizzazione si pone quando più utenti software condividono uno stesso modulo hardware; lo si può risolvere serializzando l'uso del coprocessore o realizzando un commmutatore di contesto nel modulo hardware
- infine, la progettazione del reset va considerata con attenzione: un esempio di debolezza del progetto si ha quando per inizializzare un modulo hardware è necessario un reset generale di sistema; è sensato definire una o più istruzioni per l'inizializzazione e per il reset del modulo

in una recente esercitazione è stata presentata una realizzazione software del calcolo del delay di una traiettoria di Collatz di dato inizio

realizzazioni hardware della stessa funzione sono state oggetto di precedenti esperienze di laboratorio

e.g. la terza esperienza di laboratorio ne produce una descrizione in VHDL

le misure di prestazione condotte sulla realizzazione software mostrano che essa assorbe quasi tutto il tempo di esecuzione del programma

problema: accelerare l'esecuzione del programma usando la realizzazione hardware della funzione suddetta

una prima alternativa da valutare: integrare la funzione hardware come istruzione custom o come coprocessore mappato in memoria?

la seconda opzione appare migliore, per almeno due ragioni:

- la prima opzione è *bloccante*
- il volume di dati trasferiti in ogni interazione è molto piccolo

altre decisioni di progetto dipendono da questa prima decisione, come segue

interfaccia Avalon e modello di programmazione per il caso in esempio

la descrizione VHDL del circuito di calcolo della funzione va incorporata in un componente dotato di interfacce Avalon per i segnali di Clock, Reset e di Avalon MM Slave, sì da ricevere il dato iniziale da un'operazione di scrittura e fornire il risultato in risposta a un'operazione di lettura

- trasferimenti di dati *multiciclo* sono possibili grazie al segnale Avalon `waitrequest`, impostato dallo slave per differire la risposta a una richiesta di lettura o scrittura per un numero arbitrario di cicli

indirizzamento del coprocessore: poiché le operazioni di scrittura (del dato iniziale) e lettura (del risultato) avvengono in tempi diversi e hanno la stessa dimensione del dato, un solo indirizzo è sufficiente

- per semplicità conviene usare i segnali Avalon a 32 bit `writedata`, `readdata` nell'interfaccia hardware per questo indirizzo, con conversione interna a 16 bit per le corrispondenti porte interne di I/O del circuito di calcolo della funzione

driver software: si possono definire due macro e una funzione per l'interfaccia software di accesso al bus: `DC_RESET(d)`, `DC_START(d,x0)`, `unsigned int delay(d)`, dove `d` è l'indirizzo assegnato al coprocessore

- queste idee di progetto saranno sviluppate nella prossima esercitazione

riferimenti

letture raccomandate:

- Schaumont (2012), Cap. 12, Sez. 12.1–12.3.1, 12.4

per ulteriore consultazione:

- Schaumont (2012), Cap. 12, Sez. 12.3.2
- Avalon® Interface Specifications, Ch. 1–3, MNL-AVABUSREF, Intel Corp., 2019.10.08