Principî di comunicazione HW/SW

Lezione 09 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2016-17

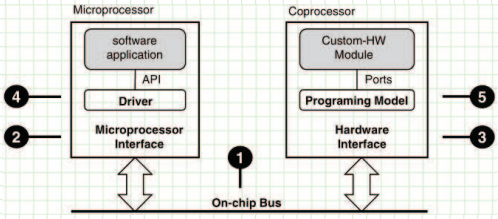
Indice

argomenti della lezione

di che si tratta:

- componenti dell'interfaccia hardware/software
- il problema della sincronizzazione: concetti e dimensioni
- schemi di sincronizzazione
  - sincronizzazione con semafori
  - sincronizzazione con handshake
  - trasferimenti di dati bloccanti e non bloccanti
- fattori-limite di prestazione: computazione o comunicazione
- accoppiamento stretto o lasco

l'interfaccia hardware/software



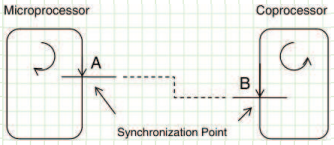
Schaumont, Figure 9.1 - The hardware/software interface

la figura mostra uno schema dei costituenti di un'interfaccia hardware/software

la funzione dell'interfaccia hardware/software è connettere l'applicazione software al modulo hardware custom; questo obiettivo coinvolge cinque elementi:

1. *bus su chip*: sia *condiviso* o *punto-punto*, trasporta dati fra microprocessore e modulo hardware custom
2. *interfaccia di microprocessore*: hardware e firmware che permette a un programma software di 'uscire' dal microprocessore, e.g. mediante istruzioni di coprocessore o di accesso a memoria
3. *interfaccia hardware*: gestisce il protocollo del bus su chip e rende disponibili i dati al modulo hardware custom mediante registri o memoria dedicata
4. *driver software*: incapsula lxale transazioni fra hardware e software in chiamate di funzione, converte strutture dati del software in strutture adatte alla comunicazione hardware
5. *modello di programmazione*: presenta un'astrazione dell'hardware all'applicazione software; per realizzare tale conversione l'interfaccia hardware può richiedere memoria e controlli agguntivi

il problema della sincronizzazione



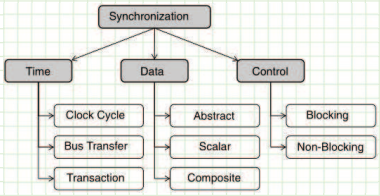
Schaumont, Figure 9.2 - Synchronization point

sincronizzazione: l'interazione strutturata di due entità parallele, altrimenti indipendenti

in figura 9.2, la sincronizzazione garantisce che il punto A nella sequenza di esecuzione del microprocessore è legato al punto B nel flusso del controllo del coprocessore

la sincronizzazione è necessaria per la comunicazione fra sottosistemi paralleli: ogni *parlante* deve avere un *ascoltatore* per essere sentito

- e.g., in un sistema dataflow, attori hardware e software devono sincronizzarsi sui loro trasferimenti di token
- anche quando un canale dataflow è realizzato mediante una memoria FIFO, la necessità di sincronizzare non viene meno, poiché la FIFO ha capacità finita, dunque il trasmittente deve attendere quando la FIFO è piena, mentre il ricevente deve attendere quando la FIFO è vuota



Schaumont, Figure 9.3 - Dimensions of the synchronization problem

tre dimensioni *ortogonali* del problema della sincronizzazione:

- *tempo*: granularità temporale delle interazioni
- *dati*: complessità strutturale dei dati trasferiti
- *controllo*: relazione tra i flussi di controllo locali

sincronizzazione con un semaforo

semaforo: una primitiva di sincronizzazione S per il controllo di accesso a una risorsa astratta condivisa, mediante le operazioni:

$P(S)$: prova l'accesso, attendi se $S=0$, altrimenti $S \leftarrow 0$

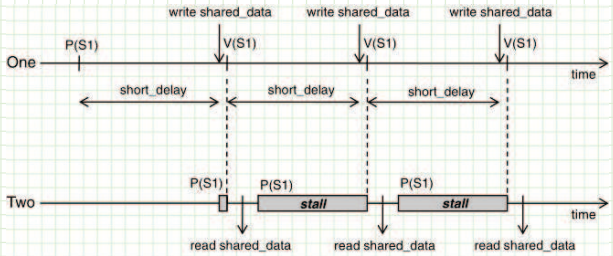
$V(S)$: rilascia la risorsa, $S \leftarrow 1$

```
int shared_data;
semaphore S1;

entity one {
 P(S1);
 while (1) {
  short_delay();
  shared_data = ...;
  V(S1);              // synchronization point
 }
}

entity two {
 short_delay();
 while (1) {
  P(S1);              // synchronization point
  received_data = shared_data;
 }
}
```

Schaumont, Listing 9.1 - One-way synchronization with a semaphore



Schaumont, Figure 9.4 - Synchronization with a single semaphore

punti di sincronizzazione: quando la prima entità esegue la $V(S1)$, così sbloccando l'altra entità

questo schema funziona nell'ipotesi che la seconda entità sia più veloce a leggere il dato di quanto lo sia la prima a scriverlo

si assuma invece il contrario, e.g. spostando la chiamata di funzione `short_delay()` dal ciclo while nella prima entità a quello nella seconda ...

in generale, nello scenario *produttore/consumatore*, entrambe le entità possono dover attendere l'un l'altra

sincronizzazione con due semafori

la situazione di ritardi variabili può essere gestita in uno schema con due semafori

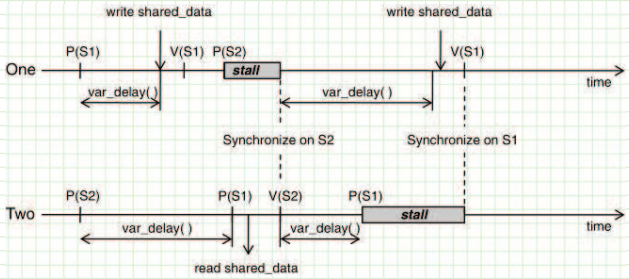
- si usa S1 per sincronizzare la seconda entità e S2 per sincronizzare la prima

```
int shared_data;
semaphore S1, S2;

entity one {
 P(S1);
 while (1) {
  variable_delay();
  shared_data = ...;
  V(S1);  // synchronization point 1
  P(S2);  // synchronization point 2
 }
}

entity two {
P(S2);
 while (1) {
  variable_delay();
  P(S1);  // synchronization point 1
  received_data = shared_data;
  V(S2);  // synchronization point 2
 }
}
```

Schaumont, Listing 9.2 - Two-way synchronization with two semaphores



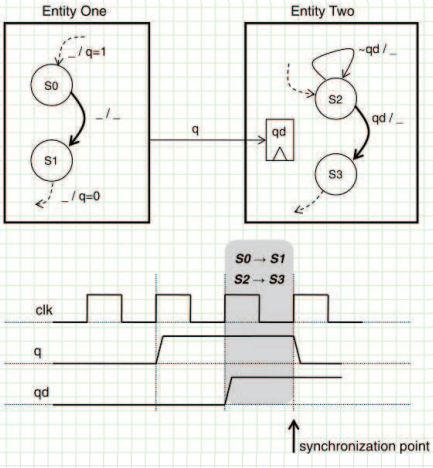
Schaumont, Figure 9.5 - Synchronization with two semaphores

la figura 9.5 illustra il caso in cui:

- alla prima sincronizzazione la prima entità è più veloce della seconda e si ha la sua sincronizzazione con il semaforo S2, laddove
- alla seconda sincronizzazione è più veloce la seconda entità e se ne ha la sincronizzazione con il semaforo S1

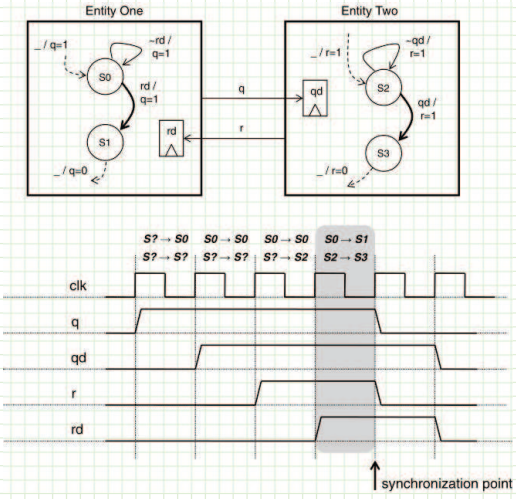
sincronizzazione con handshake

nei sistemi paralleli un semaforo centralizzato non è sempre fattibile; una comune alternativa è l'*handshake*: un protocollo di segnalazione basato su livelli di segnale



Schaumont, Figure 9.6 - One-way handshake

l'handshake unidirezionale ha lo stesso limite della sincronizzazione con un semaforo, la soluzione è:



Schaumont, Figure 9.7 - Two-way handshake

trasferimenti di dati bloccanti e non

se un'entità giunge troppo presto a un punto di sincronizzazione, dovrebbe restare in attesa fino a quando non si verifichi la condizione appropriata, o dovrebbe procedere con altro?

- in un trasferimento di dati *bloccante* il flusso di esecuzione del software o hardware è sospeso fino a quando il trasferimento è completo
  - e.g., se il software realizza il trasferimento mediante chiamate di funzioni, allora il rientro dalle chiamate si ha solo quando il trasferimento è completo
- in un trasferimento di dati *non bloccante* il flusso di esecuzione del software o hardware non si sospende, ma il trasferimento può fallire
  - una funzione software che realizzi un trasferimento di dati non bloccante dovrà fornire un flag di stato aggiuntivo che sia osservabile

sia gli schemi con semafori che con handshake discussi prima realizzano un trasferimento di dati bloccante

- l'uso di queste primitive per un trasferimento di dati non bloccante richiede che l'esito dell'operazione di sincronizzazione sia osservabile senza doverla eseguire

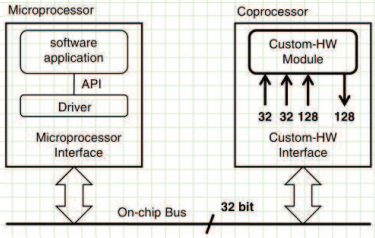
fattori-limite di prestazione

l'accelerazione del calcolo è spesso la motivazione per il progetto di hardware custom

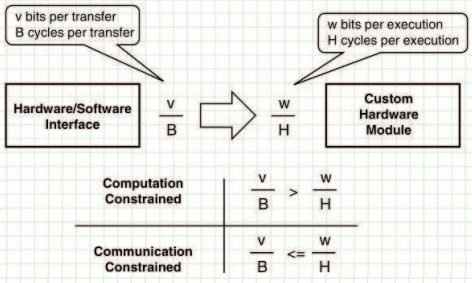
- tuttavia, anche l'interfaccia hardware/software influenza la prestazione del *sistema* che ne risulta

occorre valutare anche i vincoli di comunicazione!

- e.g., si assuma che il modulo HW custom in fig. 9.8 richieda 5 cicli di clock per il calcolo del risultato, con un totale di 320 bit di trasferimento di dati per esecuzione: può tale sistema procedere a un tasso di 320/5 = 64 bit per ciclo?



Schaumont, Figure 9.8 - Communication constraints of a coprocessor



Schaumont, Figure 9.9 - Communication-constrained system vs. computation-constrained system

il numero di cicli di clock per esecuzione del modulo hardware custom sono correlati al suo *hardware sharing factor* (HSF) $=_{def}$ numero di cicli di clock tra eventi di I/O consecutivi

Architecture	HSF
Systolic array processor	1
Bit-parallel processor	1–10
Bit-serial processor	10–100
Micro-coded processor	>100

Schaumont, Table 9.1 - Hardware sharing factor

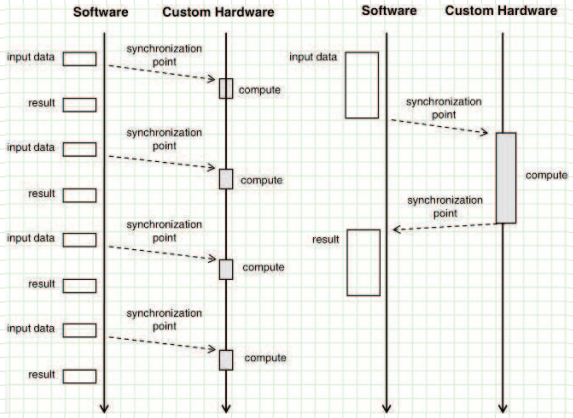
accoppiamento stretto o lasco

l'*accoppiamento* indica il livello di interazione tra flussi di esecuzione nel software e in hardware custom

stretto = frequenti sincronizzazioni | scambi di dati

lasco = il contrario

l'accoppiamento correla sincronizzazione e prestazione



Schaumont, Figure 9.10 - Tight coupling versus loose coupling

Factor	Coprocessor interface	Memory-mapped interface
Addressing	Processor-specific	On-chip bus address
Connection	Point-to-point	Shared
Latency	Fixed	Variable
Throughput	Higher	Lower

Schaumont, Table 9.2 - Comparing a coprocessor interface with a memory-mapped interface

esempio: differenza tra

interfaccia di coprocessore: sta su una porta dedicata del processore

interfaccia mappata in memoria: sta sul bus di memoria del processore

N.B.: un alto grado di parallelismo nell'insieme del progetto può essere più facile da ottenere con uno schema di accoppiamento lasco piuttosto che stretto

riferimenti

letture raccomandate:

Schaumont (2012) Cap. 9, Sez. 9.1-9.4