Comunicazione HW/SW, sistemi di bus on-chip

Lezione 09 di Sistemi dedicati

Docente: Giuseppe Scollo

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

Indice

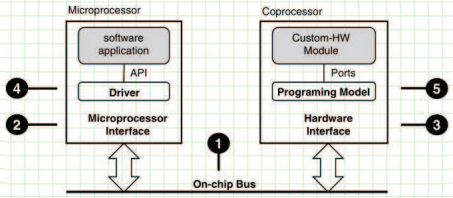
1. Comunicazione HW/SW, sistemi di bus on-chip
2. argomenti della lezione
3. l'interfaccia hardware/software
4. il problema della sincronizzazione
5. sincronizzazione con semaforo
6. sincronizzazione con due semafori
7. sincronizzazione con handshake
8. trasferimenti di dati bloccanti e non
9. fattori-limite di prestazione
10. accoppiamento stretto o lasco
11. standard di bus su chip
12. componenti di bus
13. realizzazione fisica di bus su chip
14. diagrammi di temporizzazione nei bus
15. definizione di un bus generico
16. corrispondenza di bus standard al bus generico
17. riferimenti

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
- alcuni standard di bus su chip
- componenti e realizzazione fisica di un bus su chip
- diagrammi di temporizzazione nei bus
- astrazione di alcuni bus standard in una definizione di bus generico

l'interfaccia hardware/software



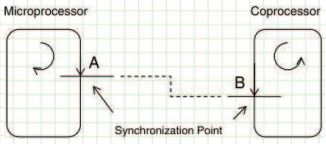
Schaumont, Figure 9.1 - The hardware/software interface

la figura mostra uno schema dei costituenti di un'interfaccia HW/SW

la funzione dell'interfaccia HW/SW 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 le 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 aggiuntivi

il problema della sincronizzazione



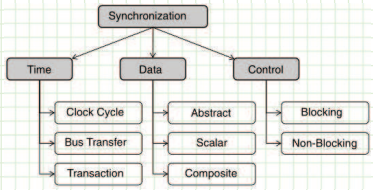
Schaumont, Figure 9.2 - Synchronization point

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:

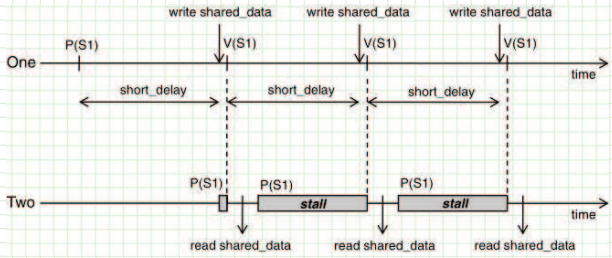
P(S): prova l'accesso, attendi se $S=0$, altrimenti $S \leftarrow 0$

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

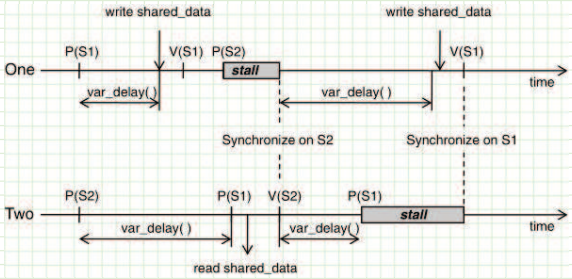
si usa S1 per sincronizzare la seconda entità e S2 per sincronizzare la prima

```
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



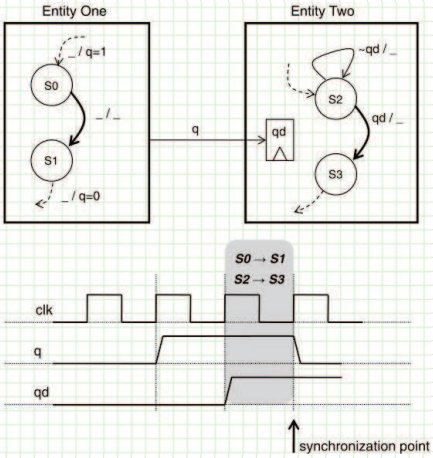
Schaumont, Figure 9.5 - Synchronization with two semaphores

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

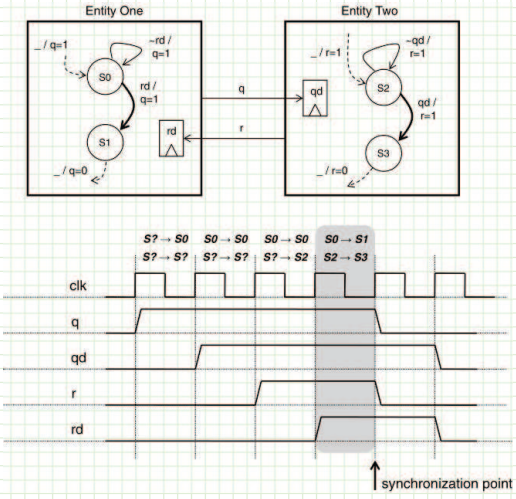
sincronizzazione con handshake

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 quelli 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

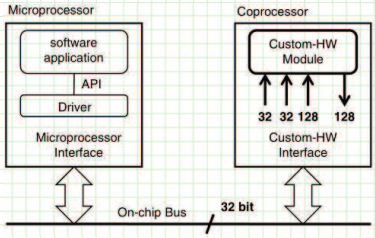
fattori-limite di prestazione

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

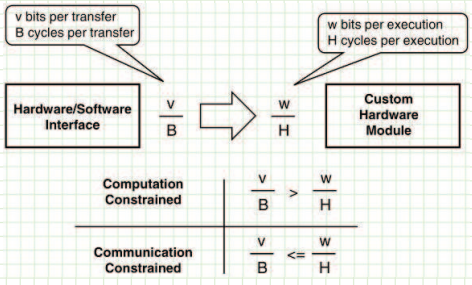
- tuttavia, anche l'interfaccia hardware/software influenza la prestazione del *sistema* che ne risulta

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 320 bit di trasferimento di dati in totale per esecuzione: può il 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

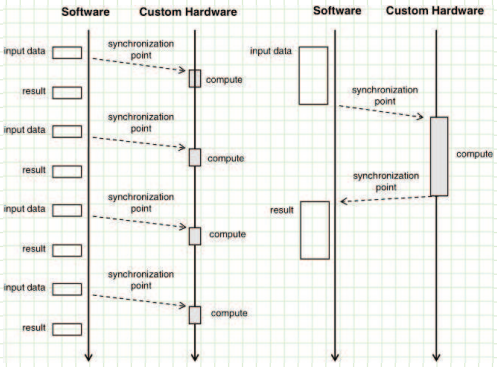
accoppiamento stretto o lasco

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

standard di bus su chip

quattro famiglie di standard di bus su chip, fra le più in uso:

- *AMBA* (Advanced Microcontroller Bus Architecture): famiglia di sistemi di bus usati da processori ARM
- *CoreConnect*: sistema di bus per processori PowerC di IBM
- *Wishbone*: sistema di bus open-source proposto dalla SiliCore Corporation, usato da molti componenti hardware open-source, e.g. quelli del progetto OpenCores
- *Avalon*: sistema di bus per applicazioni SoC dei processori Nios II di Intel

due classi principali di configurazioni: *condivisi* e *punto-punto*

varianti ulteriori per velocità, interfaccia, topologia ecc., v. tabella 10.1

si considerano appresso un bus generico condiviso e uno punto-punto, astraendo caratteristiche comuni a tutti questi

Bus	High-performance shared bus	Periferal shared bus	Point-to-point bus
AMBA v3	AHB	APB	
AMBA v4	AXI4	AXI4-lite	AXI4-stream
CoreConnect	PLB	OPB	
Wishbone	Crossbar topology	Shared topology	Point to point topology
Avalon	Avalon-MM	Avalon-MM	Avalon-ST

Legenda

AHB	AMBA highspeed bus
APB	AMBA peripheral bus
AXI	advanced extensible interface
PLB	processor local bus
OPB	onchip peripheral bus
MM	memory-mapped
ST	streaming

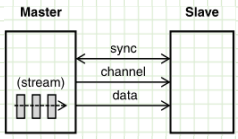
Schaumont, Table 10.1 - Bus configurations for existing bus standards

componenti di bus

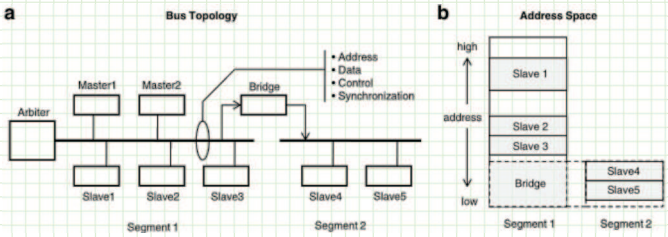
un bus condiviso su chip consiste tipicamente di più *segmenti*, connessi da *ponti*; ogni transazione è iniziata da un *master*, a cui risponde uno *slave*; se questi stanno su segmenti diversi, il ponte agisce da slave su uno dei segmenti e da master sull'altro, effettuando la *traduzione degli indirizzi*

quattro classi di segnali di bus:

- *dati*: linee dati separate per lettura e scrittura
- *indirizzi*: la decodifica può essere centralizzata o locale negli slave
- *comandi*: per distinguere lettura da scrittura, spesso qualificate da più segnali
- *sincronizzazione*: clock, distinti per segmenti diversi, e altri, quali: segnali di handshake, time-out ecc.



Schaumont, Figure 10.2 - Point-to-point bus



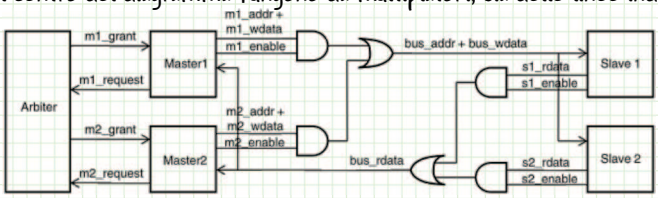
Schaumont, Figure 10.1 - (a) Example of a multi-master segmented bus system. (b) Address space for the same bus

un bus punto-punto è una connessione fisica dedicata fra un master e uno slave, per trasferimenti di dati in flussi (*stream*) illimitati

- non ha linee di indirizzo, ma può averne per canale logico nel caso di multiplazione del bus fisico per più flussi di dati
- sincronizzazione simile al protocollo di handshake visto prima

realizzazione fisica di bus su chip

la figura 10.3 mostra la disposizione fisica di un tipico segmento di bus su chip, con due master e due slave, dove le porte AND e OR al centro del diagramma fungono da multiplatori, sia delle linee indirizzo che delle linee dati



Schaumont, Figure 10.3 - Physical interconnection of a bus. The *__addr, *__wdata, *__sdata signals are signal vectors. The *__enable, *__grant, *__request signals are single-bit signals

convenzione sui nomi dei segnali di lettura/scrittura di dati:

scrittura di un dato significa il suo invio da master a slave
lettura di un dato significa il suo invio da slave a master

l'arbitraggio del bus assicura che solo un componente per volta piloti qualsiasi linea del bus

una convenzione sui nomi aiuta a dedurre funzionalità e connettività di linee di bus dai loro nomi

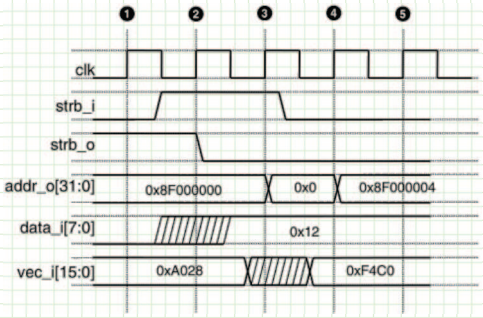
per esempio, può essere molto utile per leggere un diagramma di temporizzazione, o per visualizzare la connettività in una netlist (testuale) di un circuito

il nome di un pin di un componente rifletterà la funzionalità del pin; segnali di bus, creati da interconnessioni di pin di componenti, seguiranno anch'essi una convenzione, per evitare confusione fra segnali simili

e.g., in figura 10.3, ciascuno dei due componenti master ha un segnale wdata; per distinguerli, il nome dell'istanza del componente è il prefisso del nome di segnale del bus (e.g. m2_wdata)

diagrammi di temporizzazione nei bus

per la natura inerentemente parallela di un sistema di bus, i diagrammi di temporizzazione sono in ampio uso per descrivere le relazioni temporali fra segnali di bus



Schaumont, Figure 10.4 - Bus timing diagram notation

il diagramma in figura 10.4 mostra la notazione per descrivere le attività in un bus generico per cinque cicli di clock

- i segnali fanno riferimento al fronte di salita del clock, mostrato in alto
- i segnali di input in un ciclo di clock hanno il valore acquisito *prima* del suo fronte di inizio
- i segnali di output hanno il valore prodotto in un ciclo di clock *dopo* il suo fronte di fine

questi diagrammi sono molto utili per descrivere le attività su un bus in funzione del tempo
sono anche una parte di documentazione centrale al progetto di un'interfaccia HW/SW

definizione di un bus generico

la tabella 10.2 elenca i segnali di un sistema di bus generico, astraendo da sistemi specifici

Signal name	Meaning
clk	Clock signal. All other bus signals are references to the upgoing clock edge
m_addr	Master address bus
m_data	Data bus from master to slave (write operation)
s_data	Data bus from slave to master (read operation)
m_rnw	Read-not-Write. Control line to distinguish read from write operations
m_sel	Master select signal, indicates that this master takes control of the bus
s_ack	Slave acknowledge signal, indicates transfer completion
m_addr_valid	Used in place of m_sel in split-transfers
s_addr_ack	Used for the address in place of s_ack in split-transfers
s_wr_ack	Used for the write-data in place of s_ack in split-transfers
s_rd_ack	Used for the read-data in place of s_ack in split-transfers
m_burst	Indicates the burst type of the current transfer
m_lock	Indicates that the bus is locked for the current transfer
m_req	Requests bus access to the bus arbiter
m_grant	Indicates bus access is granted

Schaumont, Table 10.2 - Signals on the generic bus

corrispondenza di bus standard al bus generico

la tabella 10.3 mostra la corrispondenza di alcuni segnali del bus generico a segnali equivalenti dei bus CoreConnect/OPB, AMBA/APB, Avalon-MM e Wishbone

generic	CoreConnect/OPB	AMBA/APB	Avalon-MM	Wishbone
clk	OPB_CLK	PCLK	clk	CLK_I (master/slave)
m_addr	Mn_ABUS	PADDR	Mn_address	ADDR_O (master)
				ADDR_I (slave)
m_rnw	Mn_RNW	PWRITE	Mn_write_n	WE_O (master)
m_sel	Mn_Select	PSEL		STB_O (master)
m_data	OPB_DBUS	PWDATA	Mn_writedata	DAT_O (master)
				DAT_I (slave)
s_data	OPB_DBUS	PRDATA	Mb_readdata	DAT_I (master)
				DAT_O (slave)
s_ack	Sl_XferAck	PREADY	Sl_waitrequest	ACK_O (slave)

Schaumont, Table 10.3 - Bus signals for simple read/write on Coreconnect/OPB, ARM/APB, Avalon-MM and Wishbone busses

riferimenti

letture raccomandate:

- Schaumont, Ch. 9, Sect. 9.1-9.4
- Schaumont, Ch. 10, Sect. 10.1

letture per ulteriori approfondimenti:

- *Avalon® Interface Specifications*, MNL-AVABUSREF, Intel Corp., 2019.10.08
- Schaumont, Ch. 10, Sect. 10.2-10.4