INTERFACCE MEMORY-MAPPED

Grazia Pagano

DMI, Seminario di Sistemi dedicati

8 Gennaio 2018

INTERFACCE MEMORY-MAPPED

- Introduzione
- Registro mappato in memoria
- Mailbox con handshake
- FIFO queues
- Protocolli di handshake
- Memoria condivisa

INTRODUZIONE

Un'interfaccia mappata in memoria permette di allocare spazio degli indirizzi di un processore per poter far comunicare l'hardware con il software.

La memoria è parte importante per il software e a livello del linguaggio di programmazione è possibile, come vedremo, accedervi mediante i puntatori

REGISTRO MAPPATO IN MEMORIA

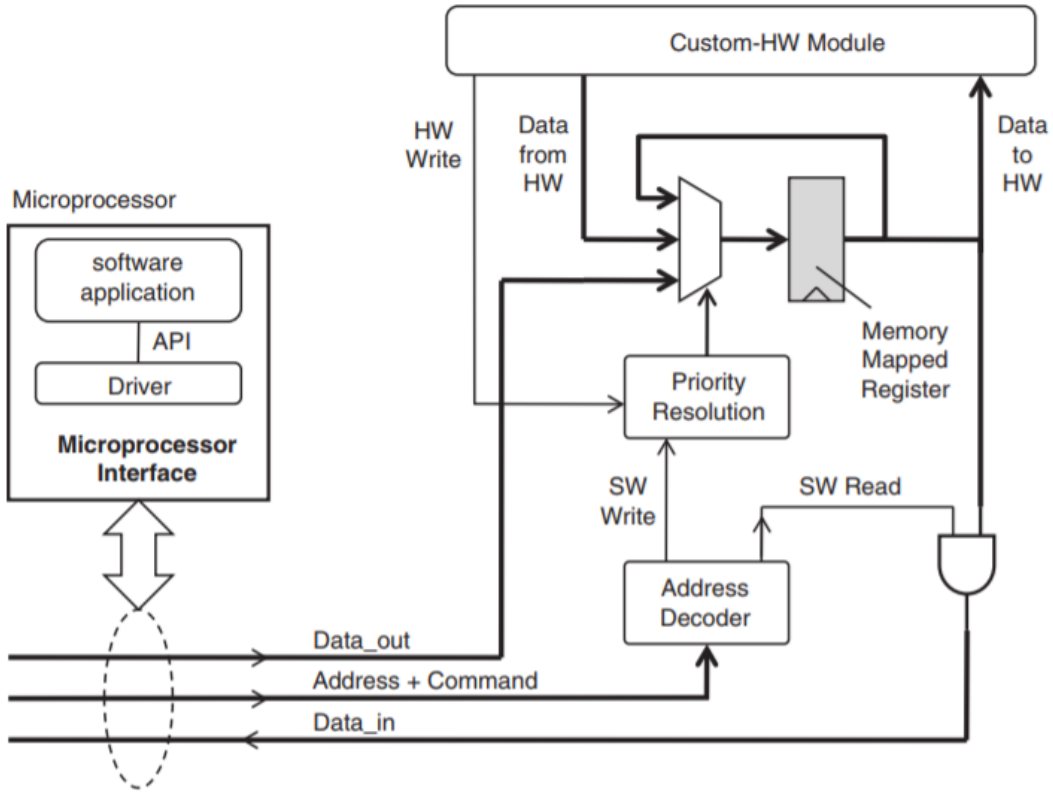

Fig. 11.1 A memory-mapped register

- Un'interfaccia mappata in memoria può essere realizzata semplicemente con un registro che può essere letto/scritto attraverso un bus.
- Accessibile con operazioni di trasferimento nel bus su specifici indirizzi (singolo indirizzo o range di indirizzi).
- L'indirizzo ed il rispettivo comando vengono analizzati da un Decoder che genera un segnale di lettura/scrittura per il registro ad ogni singolo valore di indirizzo.

REGISTRO MAPPATO IN MEMORIA

Essendo che il registro mappato in memoria lavora come una risorsa condivisa tra hardware e software, se questi effettuassero, nello stesso ciclo di clock, operazioni di scrittura si potrebbero verificare dei **conflitti**.

- NO sequenzialità
- Priorità

REGISTRO MAPPATO IN MEMORIA

Come poter accedere a un registro:

Nel **software** è possibile accedere mediante puntatori come nell'esempio:

```
volatile unsigned int *MMRegister = (unsigned int *) 0x8000;

// write the value '0xFF' into the register
*MMRegister = 0xFF;

// read the register
 int value = *MMRegister;
```

Nel **hardware** attraverso funzioni che permettono la scrittura nei registri.

REGISTRO MAPPATO IN MEMORIA

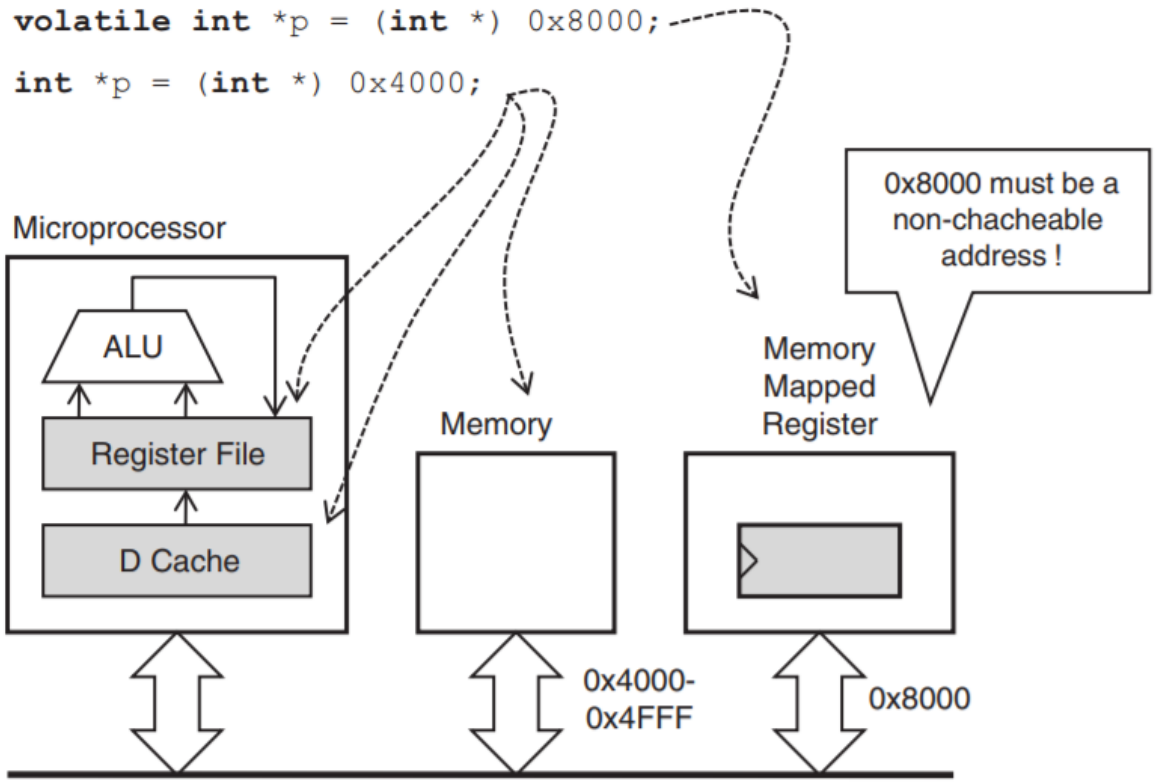

Fig. 11.2 Integrating a memory-mapped register in a memory hierarchy

Perché un puntatore volatile?

- Con l'utilizzo di un puntatore normale, il compilatore tenterà di minimizzare il numero di operazioni alla memoria principale comportando che il valore puntato può apparire in differenti locazioni nella gerarchia di memoria (memoria principale, cache e registro del processore).

- Con l'utilizzo di un puntatore volatile, il compilatore eviterà di mantenere una copia nel registro del processore. Ciò è importante perché il valore puntato può essere aggiornato anche da un modulo hardware non controllato dal microprocessore.

REGISTRO MAPPATO IN MEMORIA

Attenzione:

I puntatori di tipo volatile non sempre impediscono che l'indirizzo venga memorizzato nella cache.
Vediamo due approcci per poter evitare ciò:

1. Gli indirizzi di memoria che includono un registro mappato in memoria possono essere allocati in un'area di memoria non-cacheable.

Cache configurabile

2. L'uso di specifiche istruzioni cache-bypass che evitano l'accesso alla memoria cache

MAILBOX

Problema: con un registro non si ha riscontro sullo stato (es: scrittura/lettura)

Una mailbox con il meccanismo di handshake si propone come estensione di un registro mappato in memoria. Attraverso il settaggio di un flag è possibile avere un riscontro sullo stato.
Es: Software invia dati all'hardware impostando 'mailbox full' il flag, l'hardware legge il valore nel registro solo dopo aver letto il flag e dopo la lettura cancellerà il valore del flag così che il software potrà procedere con un ulteriore trasferimento.

MAILBOX

Una mailbox con handshake può essere realizzata con l'uso di tre registri.

Supponiamo che il software sia il '**sender**' e l'hardware sia il '**receiver**'. Dopo che il software scrive i dati, il registro *req* verrà incrementato; l'hardware, modellato da una macchina a stati finiti, scansiona *req* e non appena *req* assume un valore alto, legge i dati ed incrementa il registro ack come risposta. Quando entrambi i registri hanno un valore alto possono essere resettati di nuovo.

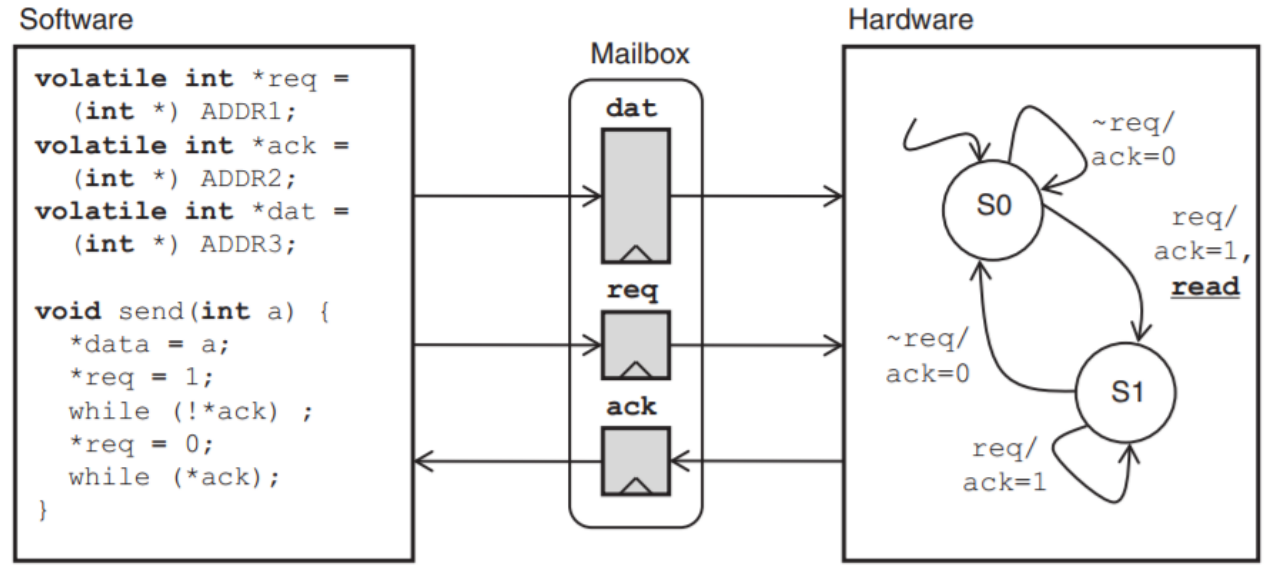

Fig. 11.3 A mailbox register between hardware and software

Il protocollo si compone di 4 fasi: *req up, ack up, req down, ack down.*
Vi sono due punti di sincronizzazione: o quando sia *req* che *ack* assumono valore alto o quando entrambi assumono valore basso permettendo così di adattarsi alla velocità del più lento.

MAILBOX

La mailbox basata su registri comporta un elevato overhead in termini di costi e prestazioni e la frequente sincronizzazione tra software ed hardware porta a due svantaggi:

1. Richiede una interazione a granularità fine tra i flussi di esecuzione di hardware e software a causa delle quattro fasi di handshake con due sincronizzazioni.
2. Richiede trasferimenti aggiuntivi sul bus per via delle frequenti sincronizzazioni.

Possibili soluzioni:

- FIFO queue
- Memoria condivisa

FIRST-IN FIRST-OUT QUEUE

In una FIFO queue è possibile scrivere diversi token in successione in modo rapido e leggerli con regolarità in modo costante. Infatti, il ruolo principale è quello di memorizzare token extra nell'operazione di scrittura e rilasciarli gradualmente nell'operazione di lettura.

È possibile implementarla con il meccanismo di handshake con due paia di segnali request/acknowledge, un paio per controllare la scrittura ed un paio per controllare la lettura.

Una FIFO può assumere lo stato empty, full o non-empty che può essere valutato comparando il valore del puntatore di lettura e quello di scrittura.

PROTOCOLLI DI HANDSHAKE

Il meccanismo di handshake richiede l'utilizzo di protocolli adatti per l'implementazione di esso.
Nell'esempio in Fig. 11.3 il software usa il protocollo master e l'hardware usa il protocollo slave.

L'interfaccia slave aspetta un segnale di richiesta, a cui reagisce e, in risposta, setta un segnale di acknowledge.
Ad ogni protocollo slave deve corrispondere un protocollo master.

Una FIFO con puntatori di lettura e scrittura avrà due interfacce slave. Si può anche costruire una FIFO con un'interfaccia slave di input per la scrittura e un'interfaccia master di output per la lettura.

Questo meccanismo permette di poter comporre più FIFO in cascata in modo da realizzarne uno più grande.

PROTOCOLLI DI HANDSHAKE

Vediamo il seguente esempio:

Si usa una FIFO con una singola locazione di memorizzazione ed è implementata come un registro.
Quest'ultimo viene aggiornato e controllato da una macchina a stati finiti che usa segnali di I/O request/acknowledge.
Un'interfaccia master può essere collegata ad un'interfaccia slave, altrimenti il protocollo non funziona.

PROTOCOLLI DI HANDSHAKE

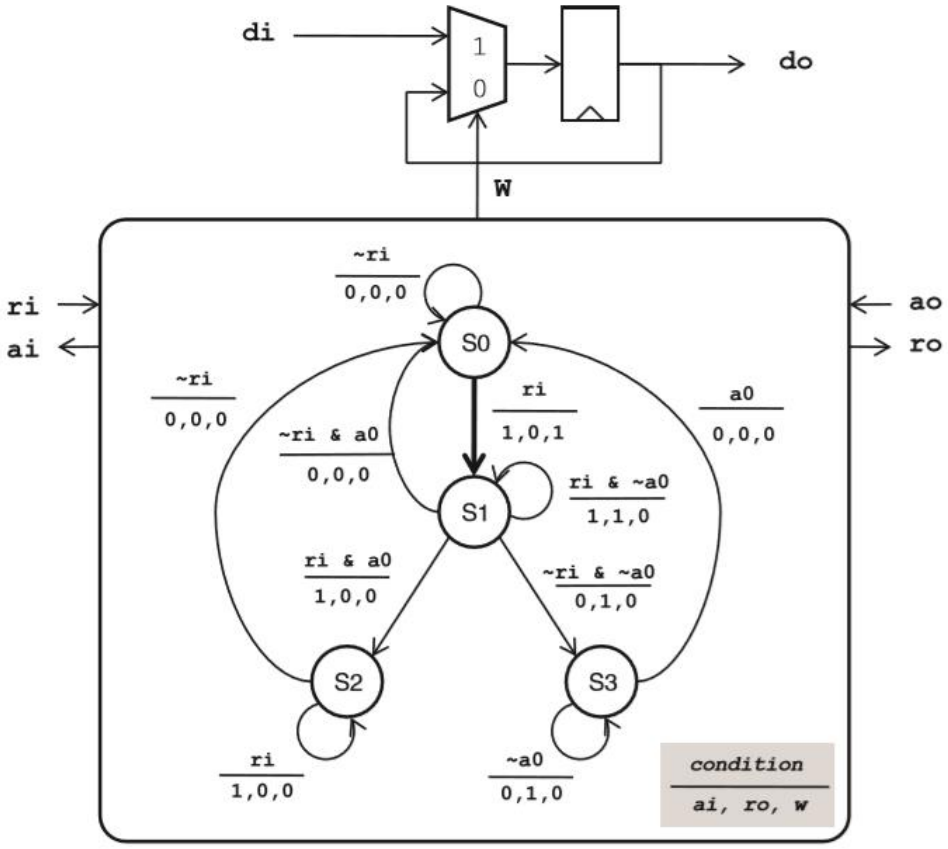

Fig. 11.5 A one-place FIFO with a slave input handshake and a master output handshake

Opera come segue (riportando parte del testo Schaumont):

<<Initially, the FSM is in state S0, waiting for the input request signal to be set. Once it is set, it will write the value on the input port into the register and transition to state S1. In state S1, the FSM sets the request signal for the output port, indicating that the FIFO stage is non-empty. From state S1, three things can happen, depending upon which handshake (input or output) completes first. If the input handshake completes (ri falls low), the FSM goes to state S3. If the output handshake responds (a0 raises high), the FSM goes to state S2. If both handshakes complete at the same time, the FSM directly goes back to S0.>>

MEMORIA CONDIVISA

È possibile usare un singolo handshake anche per controllare l'accesso ad una parte della memoria. Questo è il caso di utilizzo di una memoria condivisa.
Vediamo con un esempio:

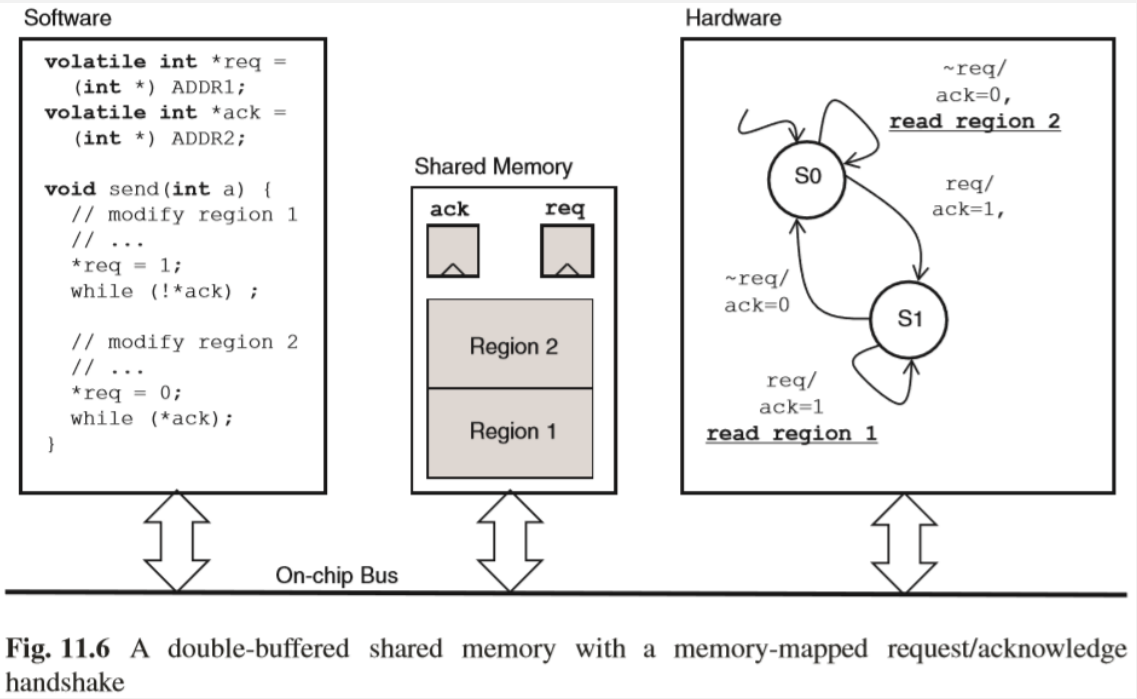

Fig. 11.6 A double-buffered shared memory with a memory-mapped request/acknowledge handshake

Un modulo di memoria è combinato con due registri mappati in memoria che implementano un handshake a doppio senso. La memoria è divisa in due regioni e viene usato un protocollo per poter accedervi. Il protocollo si compone di due fasi: nella prima le modifiche sono consentite nella regione 1 della memoria, mentre nella seconda sono consentite nella regione 2. In questo modo si avranno dati consistenti.

LETTURE

Testo di riferimento ed immagini:

Schaumont, Ch. 11(11.1.1-11.1.5)