1 Lezione del 03-12-24

1.1 Interfacce parallele

1.1.1 Interfacce di ingresso senza handshake

Iniziamo a vedere le interfacce parallele di **ingresso** senza handshake. Queste scambiano interi byte col processore, attraverso un solo registro. Non c'è quindi nessuna linea di indirizzo, l'intero registro va direttamente al processore su una linea dati. Abbiamo poi la linea di select `/s` e la linea di I/O read `/ior`.

Il registro in uscita è forchettato da una tri-state in modo da mantenere ad alta impedenza l'uscita del registro RBR quando il processore non sta leggendo dall'interfaccia. Inoltre, non vorremo nemmeno che il registro RBR si trovi a leggere dati quando il processore non sta leggendo.

Possiamo quindi ricavare, dal select e l'I/O read attraverso una porta NOR, un segnale di enable sia per la porta tri state che per il registro: quando entrambi sono bassi, si legge dal lato dispositivo dell interfaccia all'interno del registro, e si restituisce l'uscita del registro al processore.

Notiamo che l'aggiornamento dell'RBR avviene al *fronte di salita* dell'enabler (quindi di `/ior`), quindi una volta sola per ogni lettura.

1.1.2 Interfacce di uscita senza handshake

L'interfaccia parallela di **uscita** senza handshake è simile: si ha sempre un solo registro, TBR, che memorizza le linee dati in entrata qundo sono entrambi bassi il select e l'I/O write (`/iow`), cosa ricavata attraverso un'altra porta NOR.

Notiao che in questo caso TBR campiona sul *fronte di discesa* dell'enabler (quindi di `/iow`), anziché di salita.

1.1.3 Interfacce di ingresso/uscita senza handshake

Le intefacce di ingresso/uscita senza handshake si costruiscono unendo due interfacce, una di ingresso e una di uscita (e appunto dette **sottointerfacce** di ingresso e uscita), e selezionando l'interfaccia giusta attraverso un bit di indirizzo (`a0`). Il select per la selezione viene generato da una semplice rete combinatoria che prende in ingresso il `/s` globale e il bit di indirizzo, con tabella di verità: Dove `/si` è il select della sottointerfaccia

/s	/a0	/si	/so
1	-	1	1
0	0	0	1
0	1	1	0

di ingresso, e `/so` il select della sottointerfaccia di uscita. Come vediamo dall'esempio, solitamente si mette a indirizzo **pari** la porta di *ingresso*, e a indirizzo **dispari** la porta di *uscita*.

Notiamo che un montaggio alternativo si ottiene ignorando il bit di indirizzo e accedendo direttamente alle due sottointerfacce con un unico select. A questo punto sarà compito del processore discriminare fra `/ior` e `/iow` per selezionare la sottointerfaccia desiderata.

1.1.4 Interfacce di ingresso con handshake

Ricordiamo la visione funzionale delle interfacce di ingresso con handshake: dovremo avere i registri RBR e RSR, da cui ricaviamo il flag FI lato processore, mentre lato dispositivo dovremo avere sia le linee di ingresso dati che le linee di handshake, che assumiamo essere in forma `/dav` e `rfd`.

L'interfaccia si implementa quindi come una combinazione di una rete combinatoria, come avevamo visto per le interfacce senza handshake, per la generazione degli enable, e una rete sequenziale per la gestione degli handshake.

Il processore potrà accedere in lettura sia al RBR che al RSR: questo si discrimina attraeverso un bit di indirizzo (`a0`). Si ha quindi la rete combinatoria per la generazione degli enable, dove `eb` è l'enable del buffer e `es` è l'enable del registro di stato:

/s	/ior	/a0	/es	/eb
0	0	0	1	0
0	0	1	0	1
...			0	0

sintesi rete sequenziale

Notiamo che, nella sintesi, si ritarda il clock dell'interfaccia rispetto a quello del processore per evitare condizioni di *corsa* all'interfaccia.

1.1.5 Interfacce di uscita con handhsake

L'interfaccia di uscita è realizzata in modo analogo: includiamo un registro TSR che contiene il flag FO lato processore, il registro TBR, e lato dispositivo le linee `/dav` e `rfd`, dove però è l'interfaccia, e non più il processore, a fare da produttore.

La struttura interna sarà del tutto simile a l'interfaccia di ingresso, con una parte combinatoria che si occupa degli enable e una parte sequenziale che si occupa degli handshake. La parte combinatoria obbedirà alla tabella di verità:

/s	/ior	/iow	/a0	/es	/eb
0	0	1	0	1	0
0	1	0	1	0	1
...				0	0

Notiamo che compare comunque la linea di uscita in quanto vogliamo leggere lo stato del TSR.

sintesi parte sequenziale

1.1.6 Interfacce di ingresso/uscita con handshake

Possiamo combinare le interfacce con handhshake viste finora in un unica interfaccia di ingresso/uscita. Combineremo il registro di stato in un unico registro RTSR dotato dei flag FO e FI, e useremo un bit di indirizzo (`a0`) per distinguere fra le porte di ingresso e uscita.

1.2 Interfacce seriali

Le interfacce seriali trasmettono informazioni un bit a volta. Noi ne consideriamo una versione semplificata, l'interfaccia seriale start/stop.

Dal punto di vista fisico, un interfaccia seriale trasporta informazioni su due linee:

- La linea di **massa**, che funge da riferimento;
- La linea **segnale**, che porta appunto il segnale riferito a massa.

Dal punto di vista logico, invece, a interessarci sarà solo la linea segnale. Questa trasporta informazioni *half duplex*, cioè da un trasmettitore a un ricevitore (la comunicazione nelle due direzioni richiede quindi due linee).

Ora, se il segnale è rappresentato da una sequenza di **marking** (1, linea in tensione) e **spacing** (0, linea a massa) trasmessi con un periodo T, il problema diventerà sincronizzare trasmettitore e ricevitore in modo che possano comunicare efficientemente.

Quello che ci interesserà stabilire sarà quindi il **tempo di bit** T e la **trama** del segnale. Definiamo trama una sequenza di bit che rappresenta un frammento di comunicazione: iniziamo la trama con uno spacing (**bit di start**), e seguiamo poi con un numero che va da 5 a 8, stabilito in precedenza, di **bit utili** (solitamente LSB). Infine, trasmettiamo un marking per segnalare la fine della trama (**bit di stop**).

I bit di start e di stop rappresentano un **overhead**: una trama di n bit utili è lunga almeno $n+2$ per segnalare inizio e fine della trama. Avremo quindi che la velocità netta della linea è $\frac{n}{n+2}$, che è comunque più efficiente di usare un clock secondario o effettuare un handshake per ogni bit. se vuoi riguarda

Converrebbe quindi usare n arbitrariamente lunghi: purtroppo questo è fattibile fino ad un certo limite superiore, in quanto i clock di trasmettitore e ricevitore saranno necessariamente leggermente differenti in frequenza, ergo sul lungo termine si andrebbe ad accumulare un'errore troppo grande.

Infatti, l'impedenza dalla linea di trasmissione determinerà uno "smussamento" del segnale, motivo per cui preferiremo campionare ogni bit trasmesso nella posizione più centrale possibile rispetto alla sua forma d'onda. Questo significherà che, posto T il periodo del clock del trasmettitore, e $T + \Delta T$ il periodo del clock del ricevitore, vorremo:

$$n \cdot \Delta T \leq \frac{T}{2} \implies \frac{\Delta T}{T} \leq \frac{1}{2n}$$