1 Lezione del 07-11-24

1.1 Reti sequenziali sincronizzate

Le reti sequenziali sincronizzate (RSS), a differenza della asincrone (RSA), non si aggiornano per la sola variazione degli ingressi, ma per l'arrivo di un determinato segnale periodico, che chiamiamo **clock**.

Il clock è un segnale con forma d'onda periodica, di frequenza $\frac{1}{T}$ periodo, e *duty cicle* (ciclo di lavoro) $\frac{\tau}{T}$ intorno al 50%. Solitamente l'evento di sincronia delle reti sequenziali sincronizzate è il **fronte di salita** del clock.

1.2 Registri

Un registro a W bit è una collezione di W D flip-flop positive edge-triggered, che hanno:

- W ingressi d_i e W uscite q_i separate (in verità ricordiamo che troviamo sempre q e $\overline{q}$ negata, noi riporteremo solo la prima per semplicità);
- Un ingresso p in parallelo a tutti gli ingressi p_i dei singoli D flip-flop.

Si ha che p funge da **segnale di sincronizzazione** (effettivamente il nostro *clock*). Consideriamo quindi le variabili di entrata e di uscita di un registro come due singole variabili a più bit, $d_{W-1}\_d_0$ e $q_{W-1}\_q_0$.

1.2.1 Pilotaggio di registri

Per il corretto pilotaggio di un registro gli ingressi d_i devono essere stabili intorno al fronte di salita del clock, per un tempo T_{setup} prima e T_{hold} dopo il fronte stesso. L'uscita cambia dopo, come avevamo visto per i D flip-flop, un tempo $T_{prop} > T_{hold}$.

Tutto cio che accade in ingresso fra due istanti di sincronizzazione è irrilevante e non viene memorizzato.

Il registro *memorizza* lo stato di ingresso al **fronte di salita**. Gli stati di ingresso fra due fronti di salita adiacenti possono essere identici, adiacenti o non adiacenti: è irrilevante in quanto, come abbiamo detto, l'aggiornamento accade soltanto nelle condizioni di stabilità intorno al fronte di salita del clock.

Dopo il fronte di salita, le uscite cambiano il loro valore dopo T_{prop}.

Possiamo quindi aggiornare la nostra definizione di RSS come *collezione di registri e reti combinatorie*, montati arbitrariamente, purchè non ci siano anelli di retroazione di reti combinatorie (costituirebbero reti sequenziali asincrone). I registri hanno tutti lo stesso clock in comune, e possono formare anelli, in quanto abbiamo visto dal loro pilotaggio, questo non genera problemi.

1.2.2 Regole di pilotaggio per RSS

Dato l'i-esimo fronte di salita del clock al tempo t_i, lo stato di ingresso ai registri dovrà essere stabile, dalle loro regole di pilotaggio, nell'intervallo $[t_i - T_{setup}, t_i + T_{hold}]$. Non potrò quindi scegliere periodi T del clock piccoli a piacere: dovrò lasciare tempo ai registri di produrre nuovi valori (in tempo T_{prop}), e alle reti combinatorie di elaborare tali valori coi loro tempi di ritardo interni,e quindi di propagarsi nuovamente fino ai registri.

Definiamo, nello specifico, i ritardi:

- $T_{in\_to\_reg}$: il tempo di attraversamento massimo della catena di sole reti combinatorie che da uno degli ingressi della rete all'ingresso di un registro;
- $T_{reg\_to\_reg}$: il tempo di attravarsamento massimo fra l'uscita e l'ingresso di un registro;
- $T_{in\_to\_out}$: il tempo di attraversamento massimo fra un ingresso e un uscita dell'intera rete;
- $T_{reg\_to\_out}$: il tempo di attraversamento massimo fra l'uscita di un registro e un uscita della rete.

Dobbiamo introdurre poi i tempi T_{a_monte} e T_{a_valle}, cioè i tempi necessari all'utente della rete per, rispettivamente, **modificare** gli ingressi e **leggere** le uscite. Questi formano due ulteriori vincoli di pilotaggio in ingresso e in uscita.

Queste variabili di temporizzazione daranno vita ad un sistema di 4 diseguaglianze. Vediamole nel dettaglio:

- $T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_reg} + T_{setup}$
  Questa diseguaglianza assicura che un registro abbia tempo T_{hold} di immagazinare il valore dello scorso ciclo, l'utente esterno abbia tempo T_{a_monte} di modificare l'ingresso della rete, e che questo ingresso abbia tempo di arrivare ai registri $T_{in\_to\_reg}$ prima del tempo di setup T_{setup} degli stessi, che sappiamo essere necessario perchè al clock seguente i registri memorizzino effettivamente il valore (dopo T_{hold}, e lo replichino dopo T_{prop});
- $T \geq T_{prop} + T_{reg\_to\_reg} + T_{setup}$
  Questa diseguaglianza assicura che il valore generato dai registri possa propagarsi dopo T_{prop}, arrivare ai registri stessi in tempo $T_{reg\_to\_reg}$ per il T_{setup} necessario perche lo memorizzino. In sostanza, è come la precedente ma riferita alle uscite dei registri anziché dell'utente;
- $T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_out} + T_{a\_valle}$
  Questa diseguaglianza assicura che la rete abbia tempo di aggiornarsi dopo un ingresso dell'utente (T_{a_monte}), e restituire il risultato per un tempo che basti all'utente per leggere l'uscita (T_{a_valle}). Nello specifico, sappiamo che l'utente non proverà a modificare gli ingressi della rete prima del T_{hold} necessario ad aggiornare i registri al positive edge del clock, e quindi impiegherà un tempo T_{a_monte} per farlo. A questo punto, il segnale di uscità dovra viaggiare almeno dall'ingresso all'uscita, quindi si dovrà aspettare un tempo $T_{in\_to\_out}$, e infine resterà il tempo T_{a_valle} perchè l'utente abbia modo di effettuare la lettura. Notiamo che questa legge si rende necessaria in quanto un aggiornamento degli ingressi può comportare un aggiornamento delle uscite *prima* che i registri ne rispondano. In altre parole, reti di questo tipo non sono automaticamente **trasparenti**;
- $T \geq T_{prop} + T_{reg\_to\_out} + T_{a\_valle}$
  Quest'ultima diseguaglianza assicura che la rete abbia tempo di aggiornare le sue uscite, e quindi farle leggere all'utente (T_{a_valle}), a memorizzazione effettuata dei registri. Nello specifico, i registri otterrano il valore al ciclo corrente nel tempo compreso fra T_{setup} e T_{hold} centrato sul positive edge dello scorso clock, e quindi si adegueranno dopo un tempo T_{prop} rispetto al positive edge stesso. Di qui in poi dovremo aspettare un tempo $T_{reg\_to\_out}$ perchè questo valore attraversi la rete

fino alle uscite, e infine il tempo T_{a_valle} perchè l'utente abbia modo di effettuare la lettura. Questa legge si rende necessaria, al contrario della precedente, sia per reti *trasparenti* che per reti **non trasparenti**, e anzi vedremo che reti non trasparenti saranno proprio i registri a fornire le uscite.

Possiamo quindi porre il sistema completo:

$$\begin{cases} T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_reg} + T_{setup} \\ T \geq T_{prop} + T_{reg\_to\_reg} + T_{setup} \\ T \geq T_{hold} + T_{a\_monte} + T_{in\_to\_out} + T_{a\_valle} \\ T \geq T_{prop} + T_{reg\_to\_out} + T_{a\_valle} \end{cases}$$

Dove, riassumendo, le prime due condizioni garantiscono che lo stato delle variabili di ingresso resti stabile negli intervalli $(-T_{setup}, T_{hold})$ centrati sui positive edge di ogni clock; la prima e la terza tengono conto del mondo esterno *a monte*, quindi in fase di scrittura; la seconda e la quarta tengono conto del mondo esterno *a valle*, quindi in fase di lettura.

In verità, avremo altri due ritardi di cui tenere conto:

- T_{sfas}: il **massimo sfasamento** fra due clock. Visto che questo viene portato a elementi diversi, a qualche registro arrivera prima e a qualche registro arrivera dopo;
- T_{reg}: se un registro è formato da $W > 1$ bit, questi non cambieranno tutti contemporaneamente: dovremmo aggiungere $T_{prop} + T_{reg} = T'_{prop}$. A questo punto, però, possiamo considerare solo $T_{prop} \leftarrow T'_{prop}$ e ignorare T_{reg}.

1.2.3 Anticipazioni sui modelli di Moore e di Mealy ritardato

Potremmo voler determinare qual'è la più vincolante fra le diseguaglianze riportate prima. Questa, chiaramente, è quella che copre il percorso più lungo, cioè la terza. Se decidiamo di vietare il percorso che copre, cioè quello diretto fra ingressi e uscite, otteniamo il cosiddetto **modello di Moore**: cioè, un modello di RSS dove non si ammettono reti combinatorie che collegano gli ingressi direttamente alle uscite.

Un'altro vincolo che potremmo voler rilassare è il quarto, nel cosiddetto **modello di Mealy ritardato**. Questo equivale a prelevare le uscite direttamente dalle uscite dei registri, cioè a eliminare il tempo $T_{reg\_to\_out}$.

1.3 Contatori

Un contatore è una RSS il cui stato di uscita può essere visto come un **numero naturale** ad n cifre in base β. Ad ogni clock, il contatore **incrementa** o **decrementa**.

Abbiamo che si può realizzare un contatore collegando un modulo sommatore a n cifre a un registro a n cifre. L'uscita del registro viene collegata in anello di retrazione a uno degli ingressi del sommatore. Impostando il C_{in} del sommatore a 1, e il suo secondo ingresso ad un'array di n generatori di costante 0, si ha un contatore **incrementatore**, cioè che incrementa il suo valore ad ogni ciclo di clock. L'equivalente **decrementatore** si può creare usando un sottrattore a n cifre invece di un sommatore.veri

Si può creare un contatore con ingresso di abilitazione (sostanzialmente una **variabile di controllo**), cioè che incrementa o decrementa solo se è alto un certo bit di controllo, collegando tale bit al carry (o al borrow) del sommatore (sottrattore).

1.4 Contatore a una cifra in base 2

Vediamo quindi come realizzare un contatore a una cifra in base $\beta = 2$. Se l'intenzione è di creare un contatore per N cifre in codifica binaria, questo rappresenterà l'elemento fondamentale (che andremo a combinare nei prossimi paragrafi, attraverso catene di **ripple carry**).

Avremo bisogno di un input, e_i oltre al clock e al reset, che rappresenterà il riporto entrante dell'incrementatore (che come abbiamo visto può fungere da variabile di controllo, in quanto lasciare e_i a 0 significa sommare 0 a un numero, quindi lasciarlo invariato), Prenderemo poi due uscite: q, cioè l'uscita vera e propria dal registro del contatore (che chiameremo `OUTR`), e e_u, il riporto uscente. Nel caso di un contatore a una cifra in base 2, il riporto uscente si riduce al valore dell'AND fra q ed e_i, ergo se q è alto e introduciamo un riporto entrante e_i, andremo al di fuori della rappresentazione possibile su un bit e dovremo passare il riporto (che possiamo anche qui intendere come segnale di controllo) al prossimo contatore della catena. L'uscita dell'incrementatore (chiamiamola a) andrà messa in `OUTR` ad ogni aggiornamento, e verrà aggiornata dai valori di q, cioè l'uscita stessa di `OUTR` (ciclo di retroazione) e e_i, attraverso la logica dell'incrementatore che riportiamo sotto forma di tabella di verità:

q	e_i	a	e_u
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Notiamo che non abbiamo mai l'uscita $1, 1$ su a, e_u, in quanto uno degli ingressi dell'incrementatore è "fissato" a 0. Possiamo quindi dare la sintesi in Verilog:

```
// un contatore a una cifra in base 2 che prende @ei come enabler, e mette
    la
// sua uscita in @q, con eventuale riporto in @eu
module b2_counter(eu, q, ei, clock, reset_);
  input clock, reset_;
  input ei;
  output eu, q;

  reg OUTR;
  assign q = OUTR;

  wire a; // l'uscita dell'incrementatore
  b2_halfadder inc (
    .x(q), .cin(ei),
    .s(a), .cout(eu)
  );

  always @(reset_ == 0) #1 OUTR <= 0;
  always @(posedge clock) if (reset_==1) #2 OUTR <= a;
endmodule
```

sfruttando la definizione già data di incrementatore (cioè un *half adder*).

1.4.1 Scomposizione in moduli di contatori

Un contatore può essere scomposto, in qualsiasi base, in una serie di contatori ad una cifra collegati a **catena di riporti** (*ripple carry*). In questo caso il registro è dato dalla com-

binazione di n registri, uno per ogni cifra (e quindi per ogni contatore), tutti sincronizzati sullo stesso clock.