1 Lezione del 30-10-24

1.1 La funzione di memoria

Finora abbiamo visto reti combinatorie, cioè **reti prive di memoria**, dove lo stato di uscita ad un istante dipende solo dallo stato di ingresso corrente. Nelle **reti sequenziali**, invece, l'uscita dipende dalla sequenza degli stati di ingresso visti dalla rete fino a quel momento. Questa memoria si implementa attraverso **anelli di retroazione**.

Prendiamo ad esempio un buffer con un anello di retrazione, cioè una linea che porta la sua uscita al suo ingresso, e che estrae in uscita q.

Questo potrà quindi esistere in due situazioni di stabilità:

- L'uscita vale 0, e va in ingresso al buffer, dove si **rigenera** (o si *autosostiene*);
- L'uscita vale 1, e va in ingresso al buffer, dove ancora una volta si rigenera e mantiene il suo valore.

La presenza del buffer è fondamentale: mantiene l'uscita q a 0 e 1, e sopratutto ci assicura di poter associare a quel punto della rete uno stato logico.

Il problema di una rete di questo genere è che è fondamentalmente inutile: non si può controllare lo stato di stabilità del buffer, a quindi non si possono immagazzinare bit diversi a tempi diversi.

1.1.1 Uscita negata

Realizziamo allora il nostro buffer, sostituendolo cone due porte NOR disposte come invertitori (quindi un doppio invertitore, che equivale al buffer). Si ha che fra le due porte NOR abbiamo il valore complementato del buffer, cioè 1 a 0 e 0 a 1. Possiamo quindi dotare la rete di un'ulteriore uscita q_N, che equivale appunto alla negazione di q. Per questo motivo, avevamo detto, nella valutazione dei livelli di logica si ignorano le porte NOT: solitamente abbiamo già un valore negato a disposizione dai registri.

1.1.2 Stato all'accensione

Ora, se all'accensione q e q_N sono discordi, la rete si troverà già in uno degli stati stabili, e lì restera. In caso contrario, se sono concordi, teoricamente ciascuna delle due uscite dovrebbe oscillare all'infinito, con un periodo pari al doppio del tempo di risposta delle porte (in due passaggi si completa un ciclo, cioè l'ingresso della prima porta torna al neutro). Nella pratica, la rete si stabilizza, in quanto il tempo delle porte sarà necessariamente diverso e quindi si creerà prima o poi una condizione analoga a prima, dove le uscite sono discordi e la rete resta stabile.

1.1.3 Latch SR

Vediamo quini come rendere pilotabile lo stato del circuito. Introduciamo due ingressi negli input (finora duplicati) delle porte NOR: in entrata alla prima porta avremo il comando S, per SET, e in entranta alla seconda porta avremo il comando R, per RESET. Questi ingressi sono *attivi alti*, cioè i comandi S e R vengono dati quando le rispettive entrate sono in tensione. Chiamiamo questa rete **latch SR**, a volte impropriamente detta *flip-flop SR*.

Vediamo il funzionamento della rete nei diversi casi di attivazione degli ingressi:

- $S = 1$, $R = 0$: si ha che la prima NOR ha un ingresso 1, ergo mette l'uscita a 0. Quindi, la seconda NOR ha un ingresso 0, ergo mette l'uscita a 1. Ci troviamo nella configurazione stabile $q = 1$, $q_N = 0$, cioè abbiamo memorizzato un bit.
- $S = 0$, $R = 1$: si ha che la seconda NOR ha un ingresso 1, ergo mette l'uscita a 0. Quindi, la prima NOR ha un ingresso 0, quindi mette l'uscita a 1. Ci troviamo nella configurazione stabile $q = 0$, $q_N = 1$, cioè abbiamo resettato un bit.
- $S = 0$, $R = 0$: l'uscita della prima NOR vale 0 se $q = 1$, e 1 se $q = 0$, quindi q_N dà semplicemente $\bar{q}$ e viceversa, e la rete conserva il valore che aveva precedentemente. Questo comportamento rende la rete **sequenziale**: nello stato di **conservazione**, cioè quello a ingressi disattivati, si ha che la rete rimane nello stato stabile S_0 o S_1 nel quale si era portata in un momento precedente nella sequenza di stati. Si può anche dire che la rete **ricorda** l'ultimo SET o RESET ricevuto. Comunque, è una rete **asincrona**, in quanto l'uscita si aggiorna subito rispetto agli ingressi (e non in sincronia ad un clock).
- $S = 1$, $R = 1$: semanticamente, questa istruzione non ha molto significato. In uno stato di pilotaggio corretto, diciamo che questo stato **non è permesso**. Se si venisse a verificare, avremmo che alla prima porta un entrata è 1, e quindi l'uscita è 0. Alla seconda porta, quindi, un'uscita sarà 1, e avremo di nuovo uscita 0. Forzeremmo quindi la rete in uno stato $q = 0$, $q_N = 0$, che non significa nulla dal punto di vista della rappresentazione in bit della memoria.

In Verilog, possiamo descrivere il latch SR come segue:

```
// un latch SR senza preset e preclear
module sr_latch(s, r, q, q_N);
  input s, r;

  reg Q;
  output q, q_N;
  assign q = Q;
  assign q_N = ~Q;

  always @(s or r) #2
    Q <= ({s, r} == 'B00) ? Q:
         ({s, r} == 'B01) ? 'B0:
         ({s, r} == 'B10) ? 'B1:
         /*  don't care  */ 'BX;
endmodule

// implementazione a porte logiche (solo rappresentativa, Verilog
// (teoricamente) non supporta i cicli di retroazione cosi' definiti)
module sr_latch_p(s, r, q, q_N);
  input s, r;

  output q, q_N;

  assign q = ~(q_N | r);
  assign q_N = ~(q | s);
endmodule
```

1.1.4 Tabella di applicazione

Per descrivere il comportamento delle reti con memoria usiamo le **tabelle di applicazione**. Queste rappresentano, a sinistra, il valore attuale della variabile e il valore successi-

vo che si vuole questa assuma, e a destra il comando necessario perchè l'uscita passi dal valore attuale a quello successivo. Nel caso del latch SR, si ha che questa è:

q	q'	s	r
0	0	0	-
0	1	1	0
1	0	0	1
1	1	-	0

1.1.5 Regole di pilotaggio

Avevamo visto le regole per le reti combinatorie:

- Siamo in **pilotaggio in modo fondamentale**: si cambiano gli ingressi solo quando la rete è a regime;
- Gli stati di ingresso consecutivi devono essere adiacenti (per evitare *race condition*).

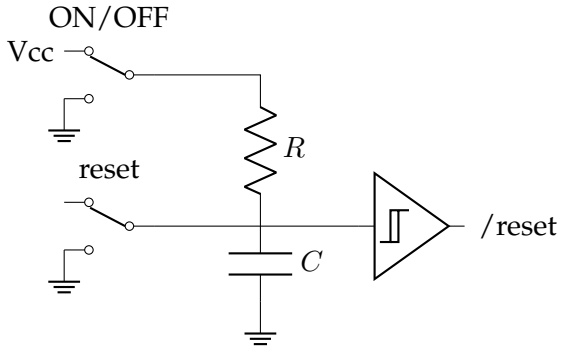
Vogliamo definire una serie di regole simili per le reti sequenziali. Abbiamo che la regola di **pilotaggio in modo fondamentale** va rispettata comunque: la rete avrà un certo **tempo di attraversamento** di cui tenere conto. Anche la seconda regola, degli **stati di ingressi consecutivi adiacenti**, è fondamentale: se non viene rispettata, si possono presentare in ingresso stati transitori spuri, e l'evoluzione delle uscite diventa imprevedebile.

Nel latch SR, però, vale che questa legge può essere violata: cioè il latch SR è robusto nei confronti di pilotaggi scorretti. Questo è il punto di forza che lo rende la rete alla base dei registri e di tutti gli elementi di memoria.

1.1.6 Lo stato iniziale

Abbiamo detto che l'SR è l'elemento alla base dei circuiti di memoria. Un SR può contenere informazioni, che corrispondono allo *stato* S_0 o S_1 in cui si trova. Si ha, però, che all'accensione il bit contenuto nell'SR è **casuale** (da quello che avevamo visto dalle modalità di pilotaggio). All'accensione di un calcolatore, si ha che alcuni elementi possono avere un valore casuale (ad esempio la RAM), altri no (ad esmpio l'instruction pointer). Si definisce quindi una **fase di reset**, distinta dalla **fase operativa**, cioè quella di operatività standard. Nella fase di reset si inizializzano gli elementi di memoria: notiamo che questo reset non corrisponde al comando R, di RESET, che diamo ai latch. In generale, quindi, non è vero che gli elementi di memoria contengono tutti zero all'accensione del calcolatore.

Vediamo quindi il circuito:



Abbiamo che la circuiteria di trigger è realizzata attraverso un circuito RC, fra l'interruttore ON/OFF e il pulsante di reset, con $\tau = R \cdot C \approx 0\,\mu$s, dove si collega il nodo fra R e C ad un trigger di Schmitt. Il trigger di Schmitt effettivamente "quantizza" la tensione, cioè scatta ad un valore 1 di tensione solamente quando la tensione in entranta è maggiore di una certa soglia. Abbiamo quindi che, spostando l'interruttore nella posizione ON, il circuito raggiunge il regime in un tempo $\approx \tau$, e quindi il trigger va a 1 in un tempo $\approx \tau$. Lo stesso quando si preme il pulsante di reset, il capacitore C si scarica e dobbiamo riportare nuovamente il circuito a regime, per cui abbiamo un istante $\approx \tau$ dove il trigger è a 0.

Abbiamo che l'uscita di questa rete va ad un ingresso detto /reset (che ricordiamo è distinto dai singoli reset dei latch SR), che è *attivo basso*: cioè nella fase iniziale dell'accensione, e ad ogni pressione successiva del pulsante reset, si ha che dal trigger esce per un tempo $\approx \tau$ il comando di /reset.

Per implementare effettivamente il meccanismo di reset si dota il latch SR di due ingressi aggiuntivi: /preset e /preclear, entrambi attivi bassi. Si distinguono quindi i seguenti casi:

- $/\text{preset} = /\text{preclear} = 1$: la rete si comporta come un latch SR normale;
- $/\text{preset} = 0$: la rete si trova nello stato S_1 (indipendentemente dallo stato di s e r);
- $/\text{preclear} = 0$: la rete si trova nello stato S_0 (indipendentemente dallo stato di s e r);
- $/\text{preset} = /\text{preclear} = 0$: abbiamo, come nel caso già visto dei semplici ingressi Set e Reset, che questo stato non è permesso, e quindi non è interessante conoscere il funzionamento della rete in tale stato.

Abbiamo quindi che per inizializzare un latch SR a 1 si porta /preset a /reset, e /preclear al Vcc. Viceversa, per inizializzare il latch a 0 si porta /preset al Vcc e /preclear a /reset.

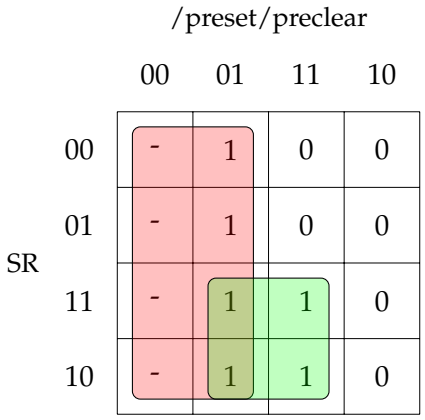
Vogliamo quindi modificare la sintesi del latch SR: conviene unirlo ad una rete combinatoria, che ha per ingresso S, R, /preset e /preclear, e in uscita z_s e z_r (che andranno in ingresso al latch vero e proprio). L'obiettivo di questa rete è di impostare i corrispondenti comandi di SET e RESET se uno fra /preset e /preclear è attivo basso, o di restituire S e R così come sono in caso entrambi siano alti.

Abbiamo, dalla sintesi con le mappe di Karnaugh, riportando i valori in coppie (z_s, z_r):

SR \ /preset/preclear	00	01	11	10
00	–	10	00	01
01	–	10	01	01
11	–	10	10	01
10	–	10	10	01

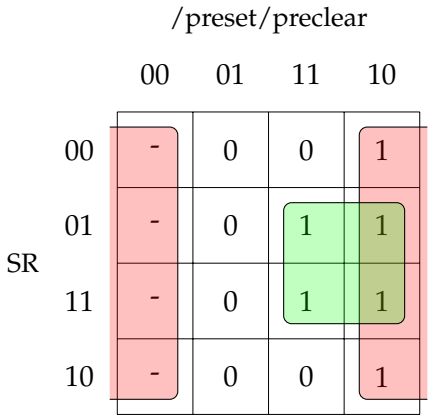
Si visualizzano i sottocubi nelle mappe presi separatamente z_s e z_r:

- z_s:



	S	R	/preset	/preclear
A	-	-	0	-
B	1	-	-	1

- s_r:



	S	R	/preset	/preclear
A	-	-	-	0
B	-	1	1	-

Da cui si ricavano le due sintesi di z_s e z_r:

$$\begin{cases} z_s = \overline{/\text{preset}} + (/\text{preclear} \cdot s) \\ z_r = \overline{/preclear} + (/\text{preset} \cdot r) \end{cases}$$

A questo punto, visto che il latch SR è realizzato a porte NOR, possiamo semplificare gli OR e i NOR in cascata: se assumiamo una NOR come una OR in serie ad una NOT, si ha che due OR equivalgono a una singola OR, ergo si possono mandare le uscite delle reti combinatorie appena sintetizzate direttamente ai NOR del latch SR, rimuovendo le OR che avremmo normalmento introdotto in una sintesi SP. Questo processo viene a volte detto *compenetrazione*.

In Verilog, possiamo quindi descrivere il latch SR aggiornato con le entrate di $/\text{preset}$ e $/\text{preclear}$ come segue:

```
// un latch SR con preset e preclear
module sr_reset_latch(preset_, preclear_, s, r, q, q_N);
  input preset_, preclear_, s, r;
  output q, q_N;

  wire s1, r1;
  assign {s1, r1} = ({preset_, preclear_, s, r} == 'B0000) ? 'BXX:
                    ({preset_, preclear_, s, r} == 'B0001) ? 'BXX:
                    ({preset_, preclear_, s, r} == 'B0010) ? 'BXX:
                    ({preset_, preclear_, s, r} == 'B0011) ? 'BXX:
                    ({preset_, preclear_, s, r} == 'B0100) ? 'B10:
                    ({preset_, preclear_, s, r} == 'B0101) ? 'B10:
                    ({preset_, preclear_, s, r} == 'B0110) ? 'B10:
                    ({preset_, preclear_, s, r} == 'B0111) ? 'B10:
                    ({preset_, preclear_, s, r} == 'B1000) ? 'B01:
                    ({preset_, preclear_, s, r} == 'B1001) ? 'B01:
                    ({preset_, preclear_, s, r} == 'B1010) ? 'B01:
                    ({preset_, preclear_, s, r} == 'B1011) ? 'B01:
                    ({preset_, preclear_, s, r} == 'B1100) ? 'B00:
                    ({preset_, preclear_, s, r} == 'B1101) ? 'B01:
                    ({preset_, preclear_, s, r} == 'B1110) ? 'B10:
                  /*({preset_, preclear_, s, r} == 'B1111)?*/'B11;

  sr_latch latch (
    .s(s1), .r(r1),
    .q(q), .q_N(q_N)
  );
endmodule

// implementazione a porte logiche
module sr_reset_latch_p(preset_, preclear_, s, r, q, q_N);
  input preset_, preclear_, s, r;
  output q, q_N;

  wire s1, r1;
  assign s1 = ~preset_ | (preclear_ & s);
  assign r1 = ~preclear_ | (preset_ & r);

  sr_latch latch (
    .s(s1), .r(r1),
    .q(q), .q_N(q_N)
  );
endmodule
```

1.2 Tabelle e grafi di flusso

Le reti sequenziali, pi+ spesso che con la tabella di applicazione, si descrivono usando **tabelle di flusso** e **grafi di flusso**.

1.2.1 Tabelle di flusso

Una tabella di flusso è una tabella che descrive come si evolvono lo stato interno e l'uscita al variare degli stati di ingresso. Ad esempio, per un latch SR, ignorando /preclear e /preset:

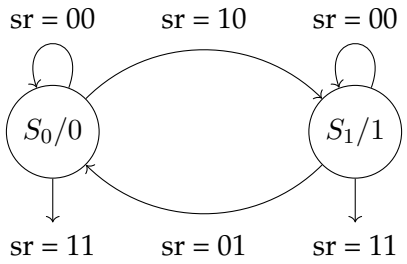
Si ha che nella tabella, le righe rappresentano gli **stati interni presenti** (SIP) e le colonne i possibili ingressi in entrata: all'intersezione fra uno stato e un ingresso si ha lo **stato interno successivo** (SIS). Si indicano con la barra ($-$) gli stati non definiti (in questo

	00	01	11	10	q
S_0	(S_0)	(S_0)	-	s_1	0
S_1	(S_1)	s_0	-	(S_1)	1

caso *non permessi*). Inoltre, l'ultima colonna indica il valore effettivo delle uscite in ogni stato (qui si è riportato solo q, e non q_N). Gli stati interni successivi cerchiati sono quelli che restano invariati dagli stati interni presenti precedenti: cioè, le coppie di stati interni presenti e ingressi che individuano uno stato interno successivo cerchiato sono coppie **stabili**.

1.2.2 Grafi di flusso

Un formalismo del tutto identico è quello del grafo di flusso: si prendono gli stati come nodi, e si disegnano archi (orientati) etichettati con gli stati di ingresso. Gli archi uscenti da un nodo simboleggiano quindi i possibili ingressi di quello stato, e entrano nei nodi che rappresentano gli stati interni successivi. Ad esempio, il grafo corrispondente alla tabella di flusso precedente è:



Notiamo come ad ogni nodo si può associare, separato dalla barra (/), l'uscita corrispondente a un dato stato interno presente. Inoltre, gli stati non definiti vengono indicati con frecce non dirette verso alcun nodo.

Questi strumenti sono utili per la descrizione e la verifica (in questo caso, sotto **ispezione**, **statica**) delle reti logiche. Nel caso di reti sequenziali, poi, la verifica **dinamica** si fa attraverso un **diagramma di temporizzazione**, cioè un grafico temporale del valore logico di ogni variabile di interesse, creato seguendo i passaggi:

1. Si decide uno stato iniziale;
2. Si attribuiscono valori agli ingressi nel tempo;
3. Si osserva l'evoluzione temporale della rete.