Roba di Elettronica Digitale

Lorenzo Rossi

AA 2019/2020

1 Dei tempi di propagazione e contaminazione

- Il tempo di contaminazione non potrà **mai** essere **superiore** al tempo di propagazione
- Restrizioni periodo di clock

$$T_{clock} \geq \sum T_{pd} + \sum T_{setup} - T_{skew}$$

- Restrizioni tempo di contaminazione

$$\sum T_{cd} \geq \sum T_{hold} + T_{skew}$$

  dal si deduce che, per un singolo flip-flop, si traduce in

$$T_{hold} \leq T_{cd} - T_{skew}$$

- Frequenza massima di funzionamento

$$f_{max} \leq \frac{1}{T_{ck_{max}}}$$

- Se un registro ha *skew* pari a ΔT, il suo ritardo di propagazione T_{pd} aumenterà di un valore pari a ΔT
- In un registro è possibile definire
  1. *Periodo di setup* T_{setup} che garantisce che il dato sia propagato attraverso il percorso di feedback prima che il master si chiuda $\rightarrow$ implica che il dato sia memorizzato
  2. *Periodo di hold* T_{hold} che garantisce che il dato sia stabile prima di permettere che il dato commuti
- A volte, i periodi di *contaminazione* e *propagazione* di un blocco logico/combinatorio sono indicati come

$$T_{cd} \leq T_{ck} \rightarrow Q \leq T_{pd}$$

2 Dei circuiti lenient

- Un circuito *non lenient* darà luogo a *glitch* nell'uscita, cioè commutazioni non volute (uscite non valide in corrispondenza di ingressi validi)
- Nella *mappa di Karnaugh,* un circuito *non lenient* mostra due *mintermini non essenziali* affacciati tra di loro
- Se la f mostra dei mintermini che **differiscono di un solo letterale negato tra di loro** allora il circuito che la implementa **non sarà lenient**
- Per garantire che un circuito sia *lenient* bisogna garantire che ci sia una linea che realizza $f(y) = 1$

3 Delle funzioni logiche realizzate con Multiplexer

Per risolvere questa categoria di esercizi bisogna guardare la tabella della verità ponendo particolare attenzione agli ingressi del tipo $100, 101$ che differiscono di un solo letterale e danno origine ad una uscita alta

4 Delle ROM

- In tutti gli esercizi trattati in aula, realizziamo le rom tramite *nMos*, quindi di fatto andiamo a realizzare le reti di *pull down* che realizzano $f(a, b, c, ...) = \bar{y}$
- Cerchiamo sempre, quindi, quale combinazione degli ingressi **abbassa, portando a massa** la linea per poi applicare *De Morgan* o l'analogo *bubble pushing*
- Affinché una rete di *pull down* funzioni, tutte le righe devono essere basse (quindi le funzioni trovate devono essere *moltiplicate* tra di loro

5 Delle FSM

Deduzione del funzionamento della macchina dal diagramma degli stati: *senza un numero massimo di stati non posso essere sicuro di aver esplorato ogni comportamento.* Solo conoscendo il numero di stati posso dedurre con correttezza il suo funzionamento. Due *FMS* si dicono **equivalenti** se e solo se ad ogni sequenza di ingressi corrispondono uguali sequenze di uscita

5.1 Macchina di Moore

- L'uscita dipende solo dallo stato attuale $output = f(stato\,attuale)$
- Se l'ingresso cambia, l'uscita non cambia
- Reagisce agli ingressi più lentamente
- Nuovo stato ed uscita sincroni
- Rispetto ad una macchina di *Mealy*, ha:
  1. Maggior numero di stati
  2. Maggiori richieste hardware
  3. Design *(progettazione)* facile

5.2 Macchina di Mealy

- L'uscita dipende dallo stato attuale e dall'ingresso attuale $output = f(stato\,attuale, ingresso\,attuale)$
- Se l'ingresso cambia, l'uscita cambia
- Reagisce agli ingressi in modo rapido
- Nuovo stato ed uscita asincroni
- Rispetto ad una macchina di *Moore*, ha:
  1. Minor numero di stati
  2. Minori richieste hardware
  3. Design *(progettazione)* più difficile

6 Delle FSM in VHDL

- Ogni diagramma degli stati dovrà prevedere uno *stato iniziale* nel quale la macchina finirà quando viene rilevato il segnale di reset
- Sarà necessario creare un *tipo enumerato* che contenga tutti i possibili stati
- Sarà necessario creare 3 processi
  1. Un *process sincrono* che abbia *clock* e *reset* nella *sensitivity list* e che si preoccupi di resettare la la macchina in corrispondenza del *reset* e di aggiornare lo stato prossimo sul fronte positivo del *clock*
  2. Un *process asincrono* che abbia il *segnale di ingresso* nella *sensitivity list* e che si preoccupi di aggiornare lo stato prossimo in funzione dello stato attuale e dell'ingresso
  3. Un *process asincrono* che abbia lo *stato corrente* nella *sensitivity list* e che si preoccupi di aggiornare l'uscita in funzione dell'ingresso. **Se la macchina è di Mealy, questo process dovrà considerare anche l'ingresso nell'assegnare l'uscita**

7 Del grafico RTL

Per disegnare un grafico RTL partendo da un codice VHDL bisogna tenere conto che

- Una *somma di due vector* diventa un *nodo sommatore* e situazioni analoghe avverranno per le altre *operazioni matematiche*
- I *while* e gli *if* diventano un *MUX*
- Le *operazioni logiche* diventano *porte logiche*
- I *signal* diventano *uno o un blocco di registri*

8 Del Bit Shift in VHDL

- Per fare il *bit-shift* con numeri *unsigned*, sarà sufficiente usare le funzioni *shift_left* e *shift_right*
- Per fare il *bit-shift* con numeri *signed*, bisognerà invece **preservare il MSB usando opportunamente l'operatore di concatenazione &**.
- Ad esempio, per uno *shift_right:*

```
signal_shifted <= signal(signal'HIGH) & signal(signal'HIGH downto 1)
```

9 Della metastabilità

- Corrisponde ed un livello logico non valido
- È uno stato di *equilibrio instabile*
- Si porta a regime in uno stato di 0 o 1 valido
- Impiega un tempo arbitrariamente lungo per andare a regime dall'istante t_d
  - La probabilità che un sistema di trovi ancora in stato metastabile tende a 0 esponenzialmente
  - Per $t \to \infty,\ P(metastabile\, a\, t) = 0$

- Il modello con cui si ricava la probabilità è il *Modello di Malthus*

$$V - V_M \cong A \cdot e^{-\frac{t}{\tau}}$$

con V tensione dell'anello e V_M tensione di Metastabilità

- Ogni sistema stabile ha almeno uno stato metastabile
- *Il ritardo aumenta l'affidabilità.* Aggiro il problema mettendo dei registri in cascata
  1. Aumentando il tempo che ci mette a propagarsi, un segnale metastabile tende a stabilizzarsi prima
  2. Aumento il tempo di *clock* → riduco la *frequenza* → riduco la possibilità di avere stati metastabili
  3. Se non posso aumentare la frequenza, aumento il numero di registri in serie
  4. *Nella pratica:* 3 Flip-Flop in serie → Probabilità nulla di metastabilità

10 Della Pipeline

- In un circuito *pipeline*, è possibile definire
  1. **Latenza**, ritardo tra quando un segnale di ingresso è valido a quando la corrispondente uscita è valida

$$Latenza = n^\circ \, linee \, isocrone \cdot \tau_{pd}$$

con T_{ck} tempo di propagazione di un registro

  2. **Throughput**, velocità con la quale le uscite corrispondenti a nuovi ingressi diventano valide

$$Throughput = \frac{1}{\tau_{pd}}$$

con τ_{pd} tempo di propagazione del blocco di pipeline più lento

- Per rendere *pipeline* un circuito, bisogna
  1. Creare una linea di *pipeline* che interseca l'uscita
  2. Ogni linea di *pipeline* deve dividere il circuito tagliando le linee di segnali in modo da essere attraversata dal segnale sempre nella stessa direzione. **Si cerca di isolare sempre la parte del circuito più lenta (quella con il maggior tempo di propagazione τ_{pd})**
  3. In corrispondenza dell'intersezione tra linea di livello e segnale, metto un *registro*
  4. Definisco il periodo di clock minimo $T_{ck_{min}}$ (quindi la $f_{ck_{max}}$) dettato dalla parte più lenta che ho isolato
- La *pipeline* **peggiora** la **latenza** ma **migliora** il **throughput**