Sistemi sincroni come macchine a stati finiti con datapath (FSMD)

Lezione 05 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2019-20

Indice

argomenti della lezione

di che si tratta:

- modelli di controllo: macchine a stati finiti (FSM)
  - FSM di Mealy, FSM di Moore
  - conversione fra i due tipi di FSM
- modelli di controllo di dataflow: FSM con datapath (FSMD)
- esempio: modello FSMD di contatore bidirezionale
- esempio: modello FSMD dell'algoritmo GCD di Euclide
- modello FSMD quale coppia di FSM allacciate
- rappresentazione datapath di FSMD
- regole di buona formazione di FSMD

macchine a stati finiti (FSM)

FSM, un comune modello del controllo per la descrizione dell'hardware (e altro):

- un insieme di stati, con un distinto stato iniziale
- un insieme di ingressi e uno di uscite
- una funzione di transizione di stato
- una funzione di uscita

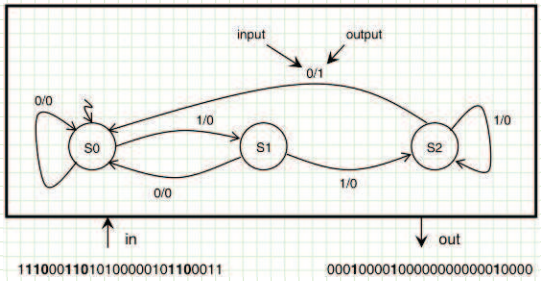
diverse varianti:

- con stati *accettanti*: un sottoinsieme distinto dell'insieme di stati
- con una funzione di uscita che dipende
  - da stato corrente e ingresso: FSM di *Mealy*
  - solo dallo stato corrente: FSM di *Moore*

i modelli FSM sono spesso presentati come grafi di transizioni etichettate

esempio: FSM riconoscitore di pattern

ecco un esempio di FSM di Mealy per il riconoscimento di un pattern binario



Schaumont, Figure 5.5 - Mealy FSM of a recognizer for the pattern '110'

l'output '1' segnala il riconoscimento di un'occorrenza del pattern nella stringa binaria in ingresso

- il riconoscimento avviene attraverso un processo incrementale: riconoscimento di prefissi iniziali del pattern via via più lunghi

  gli stati sono la *memoria* (finita) di una FSM

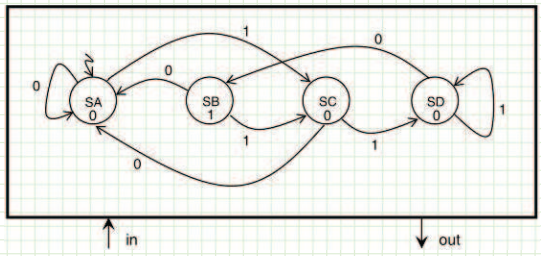
è possibile costruire una FSM di Moore equivalente, con un metodo generale

FSM di Moore equivalente a una data FSM di Mealy

una semplice procedura per convertire una FSM di Mealy in una FSM di Moore equivalente:

- stati di Moore: in corrispondenza biunivoca con le coppie distinte (s, o) dove s è uno stato di Mealy ed esiste una transizione a s etichettata con output o
- per ogni $(s, o) \leftrightarrow S$ assegnare l'output o allo stato di Moore S
- per ogni transizione $s \to s'$ etichettata i/o definire una transizione etichettata i allo stato di Moore $(s', o) \leftrightarrow S'$ dagli stati di Moore S corrispondenti a (s, x), per ogni output x
- risolvere l'eventuale ambiguità sulla scelta dello stato iniziale della FSM di Moore

questa procedura, con la corrispondenza $(s0,0) \leftrightarrow SA$, $(s0,1) \leftrightarrow SB$, $(s1,0) \leftrightarrow SC$, $(s2,0) \leftrightarrow SD$, dà la FSM di Moore equivalente per il riconoscitore di pattern '110' presentata in figura:



Schaumont, Figure 5.6 - Moore FSM of a recognizer for the pattern '110'

FSM con datapath (FSMD)

il modello FSMD combina l'elaborazione dataflow con il suo controllo, descritto da una FSM

serve a tal fine la definizione di *istruzioni* eseguibili da un datapath, sotto il controllo di una FSM

in Gezel, blocchi `sfg` (*signal flow graph*) rappresentano le istruzioni

- in tutto simili al blocco `always`, contengono espressioni eseguite solo quando la FSM lo prescrive
- le istruzioni, rappresentate dai nomi degli `sfg`, sono gli output dalla FSM al datapath
- più precisamente, a ogni ciclo di clock la FSM può prescrivere l'esecuzione (concorrente) di un *insieme* delle istruzioni del datapath

un controllore FSM è definito per uno specifico datapath:

- ne acquisisce i nomi delle istruzioni quale vocabolario di output
- le sue transizioni di stato possono essere *condizionate* da espressioni sui valori di registri del datapath

esempio: modello FSMD di un contatore bidirezionale

il contatore, qui descritto in Gezel, è inizializzato a 0, quindi alterna:

- sequenze di incrementi fino a una soglia massima, a
- sequenze di decrementi fino a una soglia minima

```
dp updown(out c :ns(3)) {
 reg a : ns(3);
 always { c = a; }
 sfg inc { a = a + 1; } // instruction inc
 sfg dec { a = a - 1; } // instruction dec
 sfg clr { a = 0; } // instruction clr
}
```

Schaumont, Listing 5.12 - Datapath for an up-down counter with three instructions

```
fsm ctl_updown(updown) {
 initial s0;
 state s1, s2;
 @s0 (clr) -> s1;
 @s1 if (a < 3) then (inc) -> s1;
                else (dec) -> s2;
 @s2 if (a > 0) then (dec) -> s2;
                else (inc) -> s1;
}
```

Schaumont, Listing 5.13 - Controller for the up-down counter

Cycle	FSM curr /next	DP instr	DP expr	a curr /next
0	s0/s1	clr	a = 0	0/0
1	s1/s1	inc	a = a+1	0/1
2	s1/s1	inc	a = a+1	1/2
3	s1/s1	inc	a = a+1	2/3
4	s1/s2	dec	a = a-1	3/2
5	s2/s2	dec	a = a-1	2/1
6	s2/s2	dec	a = a-1	1/0
7	s2/s1	inc	a = a+1	0/1
8	s1/s1	inc	a = a+1	1/2
9	s1/s1	inc	a = a+1	2/3

Schaumont, Table 5.4 - Behavior of the FSMD in Listing 5.13

la tabella accanto mostra cosa accade a ogni ciclo di clock, per i primi dieci cicli

esempio: algoritmo GCD di Euclide

l'algoritmo è un po' diverso da quello visto nell'esercitazione precedente:
terminazione con una delle due variabili uguale a zero

```
dp euclid(in m_in, n_in : ns(16);
          out gcd : ns(16)) {
 reg m, n : ns(16);
 reg done : ns(1);
 sfg init { m = m_in;
            n = n_in;
            done = 0;
            gcd = 0; }
 sfg reduce {  m = (m >= n) ? m - n : m;
               n = (n > m) ? n - m : n; }
 sfg outidle {  gcd = 0;
               done = ((m == 0) | (n == 0)); }
 sfg complete{ gcd = ((m > n) ? m : n);
               $display("gcd = ", gcd); }
}
```

```
fsm euclid_ctl(euclid) {
 initial s0;
 state s1, s2;
 @s0 (init) -> s1;
 @s1 if (done) then (complete) -> s2;
                else (reduce, outidle) -> s1;
  @s2 (outidle) -> s2;
}
```

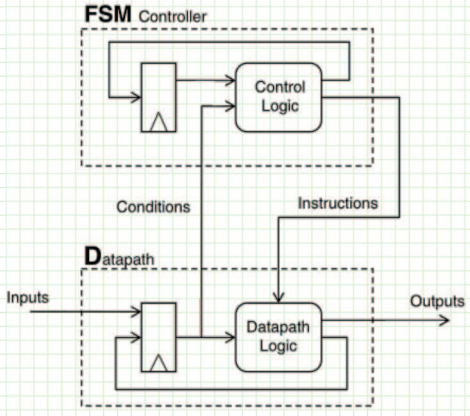
elementi di controllo introdotti con la FSM: quasi tutti quelli proposti nella precedente esperienza di laboratorio

N.B.: l'esecuzione di `reduce` e `outidle` attivata dalla FSM nello stato `s1` è *concorrente*

Schaumont, Listing 5.14 - Euclid's GCD as an FSMD

modello FSMD quale coppia di FSM allacciate

un modello dataflow può vedersi come una FSM, con spazio degli stati definito dai registri



attività delle FSM a ogni ciclo di clock:

1. entrambe: aggiornamento dello stato (registri)
2. FSM controllore: scelta del prossimo stato e delle istruzioni per la FSM datapath, determinata da stato corrente e condizioni sullo stato del datapath
3. FSM datapath: scelta del prossimo stato e output determinati da stato corrente e istruzioni selezionate dalla FSM controllore

Schaumont, Figure 5.7 - An FSMD consists of two stacked FSMs

in pratica conviene descrivere solo la FSM controllore con transizioni di stato
per non incorrere nella notoria esplosione dello spazio degli stati

rappresentazione datapath di FSMD: controllo

se un datapath, riconducibile a una FSM, può essere descritto da espressioni, questo dovrebbe essere possibile anche per la FSM controllore

infatti in qualche caso ciò può dare dei vantaggi, ma gli inconvenienti sono molti:

```
dp updown_ctl(in a_sm_3, a_gt_0 : ns(1);
              out instruction : ns(2)) {
 reg state_reg : ns(2);
 // state encoding: s0 = 0, s1 = 1, s2 = 2
 // instruction encoding: clr = 0, inc = 1, dec = 2
 always {
  state_reg  = (state_reg == 0) ? 1 :
               ((state_reg == 1) & a_sm_3) ? 1 :
               ((state_reg == 1) & ~a_sm_3) ? 2 :
               ((state_reg == 2) & a_gt_0) ? 2 : 1;
  instruction = (state_reg == 0) ? 0 :
               ((state_reg == 1) & a_sm_3) ? 1 :
               ((state_reg == 1) & ~a_sm_3) ? 2 :
               ((state_reg == 2) & a_gt_0) ? 2 : 1;
 }
}
```

Schaumont, Listing 5.15 - FSM controller for updown counter using expressions

il confronto con la descrizione dell'esempio nel Listing 5.13 mostra:

- una maggiore complessità descrittiva, e
- la necessità di introdurre una codifica numerica degli stati invece di una simbolica

per contro, descrizioni separate di controllore e datapath, entrambe mediante espressioni, possono essere combinate in una descrizione singola, il che può migliorare la prestazione

- per esempio, spesso si può ridurre la latenza di un ciclo di clock, perché le condizioni su transizioni di stato, che nel modello FSMD devono essere valutate su registri, in una descrizione singola di datapath possono essere generate e valutate nello stesso ciclo di clock
- tuttavia, come si vede appresso, a fronte di questo vantaggio emergono ulteriori inconvenienti

rappresentazione datapath di FSMD completa

```
dp updown_ctl(out c : ns(3)) {
 reg a : ns(3);
 reg state : ns(2);
 sig a_sm_3 : ns(1);
 sig a_gt_0 : ns(1);
 // state encoding: s0 = 0, s1 = 1, s2 = 2
 always {
  state = (state == 0) ? 1 :
          ((state == 1) & a_sm_3) ? 1 :
          ((state == 1) & ~a_sm_3) ? 2 :
          ((state == 2) & a_gt_0) ? 2 : 1;
  a_sm_3 = (a < 3);
  a_gt_0 = (a > 0);
  a = (state == 0) ? 0 :
      ((state == 1) & a_sm_3) ? a + 1 :
      ((state == 1) & ~a_sm_3) ? a - 1 :
      ((state == 2) & a_gt_0) ? a + 1 : a - 1;
  c = a;
 }
}
```

Schaumont, Listing 5.16 - updown counter using expressions

il confronto fra la descrizione di FSMD in due parti data nei Listing 5.12 e 5.13 con questa descrizione monolitica mostra che in quest'ultima:

- la commistione degli aspetti di elaborazione dei dati con gli aspetti di scheduling, sequenziamento e controllo, complica la *comprensibilità* del disegno complessivo
- l'integrazione monolitica di tali aspetti rende più difficile anche la *riusabilità* di parti della descrizione
  - per esempio, riuso del datapath con uno scheduling diverso

per contro, la descrizione singola offre più opportunità di ottimizzazione:

- degli assegnamenti di stato, che sono invece generati automaticamente dagli strumenti di sintesi nell'altro caso
- di applicazioni dove il controllo è poco influente, mentre molto di più lo sono requisiti di throughput elevato

per applicazioni complesse, molto strutturate, la descrizione di FSMD con descrizioni separate di FSM e datapath sembra preferibile

FSMD ben formate

una *FSMD ben formata* è una che esibisce comportamento deterministico

> una proprietà desiderabile, di cui godono anche i grafi SDF, come si è visto

per una realizzazione hardware di FSMD, comportamento deterministico significa che l'hardware è esente da situazioni di corsa

> le situazioni di corsa sono un problema tipico dei sistemi concorrenti, e l'hardware è inerentemente concorrente

un modello FSMD è *ben formato* se soddisfa le seguenti proprietà:

1. nessun registro né segnale ha più di un assegnamento in ciascun ciclo di clock
2. non esiste alcuna circolarità fra assegnamenti a segnali
3. se un segnale è usato come operando in un'espressione, deve avere un valore noto nello stesso ciclo di clock
4. tutti gli output del datapath devono avere un valore definito (assegnato) in ogni ciclo di clock

riferimenti

letture raccomandate:

> Schaumont, Ch. 5, Sect. 5.3–5.4.3, 5.6

per ulteriore consultazione:

> Schaumont, Ch. 5, Sect. 5.7