Sintesi logica: reti sequenziali

M. Favalli

Engineering Department in Ferrara

Sintesi logica per reti sequenziali sincrone

- FSM ⇒ rete sincrona
  - obbiettivi: costo, prestazioni, consumo di potenza, affidabilitá
- Tipologie di FSM
  - completamente specificata
  - non completamente specificata
- Passaggi
  - minimizzazione del numero di stati
  - codifica dello stato
  - ottimizzazione della parte combinatoria della rete

Sommario su FSM

- Una FSM é data da $\langle \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{S}, \lambda, \delta, \boldsymbol{s_0} \rangle$ dove:
  - $\boldsymbol{I}$ é l'insieme dei simboli di ingresso
  - $\boldsymbol{O}$ é l'insieme dei simboli di uscita
  - $\boldsymbol{S}$ é l'insieme degli stati
  - λ é la funzione di uscita $\lambda : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{O}$
  - δ é la funzione di stato futuro $\delta : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$
- Modello del tempo sincrono

Macchine completamente e non completamente specificate

- In una macchina completamente specificata ogni sequenza di ingresso é possibile e quindi i valori di λ e δ sono sempre specificati
- In una macchina non completamente specificata alcune sequenze sono considerate impossibili e quindi λ e δ in alcuni casi possono non essere specificate

Sintesi di FSM

- Minimizzazione del numero di stati
- Codifica dello stato
- Sintesi e codifica della rete combinatoria che realizza λ e δ
- STA per determinare la massima frequenza di clock
- Obbiettivi: area e ritardo

Il problema dell'automa minimo

- Alcuni stati possono riassumere la stessa storia passata del sistema
- Quindi il numero di stati puó essere ridotto con benefici sulla realizzazione finale della FSM
- Per FSM completamente specificate: **due stati sono equivalenti a partire da quegli stati per ogni possibile sequenza di ingresso le uscite sono le stesse**
- Per FSM non completamente specificate: **due stati sono compatibili se a partire da quegli stati per ogni possibile sequenza di ingresso le uscite sono le stesse quando specificate**

Il problema dell'automa minimo

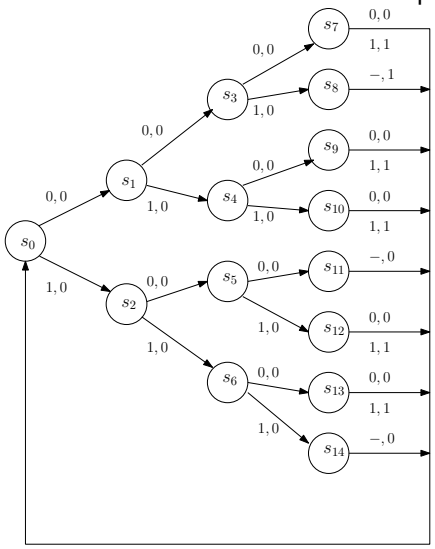
- Nel caso dell'automa completamente specificato la relazione di indistinguibilitá é una relazione di equivalenza che divide l'insieme degli stati in classi di equivalenza
- L'automa minimo é dato dalle classi massime di equivalenza e contiene uno stato per ogni classe
- Puó essere calcolato con un algoritmo iterattivo o recursivo
- Nel caso di FSM non completamente specificate la relazione di compatibilitá non é una relazione di equivalenza

Calcolo delle classi massime di equivalenza

- Partiziona l'insieme degli stati $\boldsymbol{S}$ in un insieme di classi (non di equivalenza) $\boldsymbol{\Pi}$ ciascuna delle quali contiene tutti gli stati che per ogni ingresso producono la stessa uscita
- Vengono svolti nuovi passi in cui ogni classe viene partizionata in sottoclassi ciascuna delle quali contiene tutti gli stati che per lo stesso ingresso hanno transizioni verso stati che appartengono alla stessa classe
- Quando non vengono aggiunte nuove classi, quelle rimanenti sono le classi massime di equivalenza
- É banalmente simile al procedimento grafico basato sulla tabella triangolare e ha un costo $\boldsymbol{O(|S|)^2}$

Esempio

FSM che riceve serialmente parole di 4 bit che rappresentano un numero intero senza segno trasmesso dal bit di maggior peso. La rete riconosce numeri primi portando a 1 l'uscita sull'ultimo bit di una parola.



Esempio - calcolo dell'automa minimo

$\Pi^0 = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_{11}, s_{14}\}, \{s_7, s_9, s_{10}, s_{12}, s_{13}\}, \{s_8\}$

$\Pi^1 =$
$\{s_0, s_1, s_2, s_{11}, s_{14}\}, \{s_3\}, \{s_4\}, \{s_5\}, \{s_6\}, \{s_7, s_9, s_{10}, s_{12}, s_{13}\}, \{s_8\}$

$\Pi^2 =$
$\{s_0, s_{11}, s_{14}\}, \{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}, \{s_5\}, \{s_6\}, \{s_7, s_9, s_{10}, s_{12}, s_{13}\}, \{s_8\}$

$\Pi^3 =$
$\{s_0\}, \{s_{11}, s_{14}\}, \{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}, \{s_5\}, \{s_6\}, \{s_7, s_9, s_{10}, s_{12}, s_{13}\}, \{s_8\}$

Rimangono quindi 10 stati dai quali puó essere costruio l'automa minimo

Sintesi di reti sequenziali sincrone con SIS

- L'esempio considerato verrá utilizzato per illustrare il flusso di progetto di reti sincrone utilizzando SIS
- Il primo passo consiste nel descrivere la FSM utilizzando il linguaggio **kiss**

```
.i 1
.o 1
.s 15
.r s0
0 s0 s1 0
1 s0 s2 0
0 s1 s3 0
1 s1 s4 0
0 s2 s5 0
1 s2 s6 0
0 s3 s7 0
1 s3 s8 0
0 s4 s9 0
1 s4 s10 0
0 s5 s11 0
1 s5 s12 0
```

```
0 s6 s13 0
1 s6 s14 0
0 s7 s0 0
1 s7 s0 1
- s8 s0 1
0 s9 s0 0
1 s9 s0 1
0 s10 s0 0
1 s10 s0 1
- s11 s0 0
0 s12 s0 0
1 s12 s0 1
0 s13 s0 0
1 s13 s0 1
- s14 s0 0
.e
```


Calcolo dell automa minimo con SIS

- La FSM puó essere letta con `read_kiss <nome file>`
- La FSM é simulabile con il comando `simulate`
- Per calcolare l'automa minimo si utilizza il programma `stamina` che é disponibile con SIS che si chiama con `state_minimize stamina -v 2` in modo da avere informazioni sul processo di sintesi

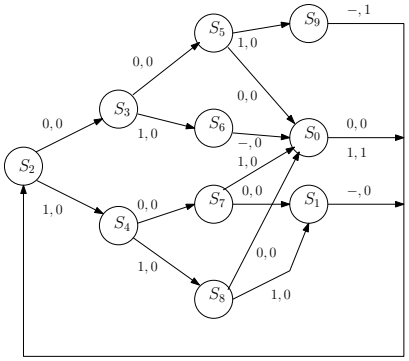
```
....
state  0: (s12,s13,s10,s9,s7)
state  1: (s14,s11)
state  2: (s0)
state  3: (s1)
state  4: (s2)
state  5: (s3)
state  6: (s4)
state  7: (s5)
state  8: (s6)
state  9: (s8)
....
Number of states in original machine : 15
Number of states in minimized machine : 10
```

Automa minimo

- Conviene salvarlo con `write_kiss <nome file>`
- Il risultato é il seguente

```
.i 1
.o 1
.p 19
.s 10
.r S2
0 S0 S2 0
1 S0 S2 1
0 S2 S3 0
1 S2 S4 0
0 S1 S2 0
1 S1 S2 0
0 S3 S5 0
1 S3 S6 0
0 S4 S7 0
1 S4 S8 0
```

```
0 S5 S0 0
1 S5 S9 0
0 S6 S0 0
1 S6 S0 0
0 S7 S1 0
1 S7 S0 0
0 S8 S0 0
1 S8 S1 0
.e
```



Codifica dello stato

- La codifica dello stato consiste nell'associare una configurazione binaria di n variabili a ciascuno stato in modo che sia $2^n >= |S|$ $n \geq \lceil |S| \rceil$
- La codifica dello stato ha un impatto sul costo finale della rete sincrona che é determinato da quello delle reti combinatorie che implementano le funzioni di stato futuro e uscita e da quello dei flip-flop
- Il problema é computazionalmente molto pesante perché il numero di possibili codifiche é proporzionale a $(2^n)!/(2^n - |S|)!$
- Per valutare l'effetto di ciascuna di quesete scelte bisognerebbe poi sintitizzare le reti che realizzano le funzioni di stato futuro e uscita
- Si utilizzano degli euristici dipendentemente dallo stile di implementazione di tali reti (a 2 livelli o multilivello)

Codifica dello stato

- Non entriamo nel dettaglio degli euristici per la codifica dello stato
- Viene mostrato solo uno dei diversi possibili principi utilizzati in tali tecniche con riferimento a una implementazione delle reti combinatorie come reti a 2 livelli
- Si cerca di produrre funzioni di stato futuro e uscita che siano copribili con il minor numero di implicanti di dimensioni il piú grandi possibile

Esempio

- Si supponga di avere questa frazione di relazione di stato futuro e uscita

stato presente	0	1
....		
s_j	$s_p, 0$	$s_q, 0$
s_k	$s_p, 1$	$s_r, 0$
....		

- s_j e s_k hanno transizioni di stato coincidenti con ingresso 0
- Se li codifico con due configurazioni a distanza Hamming unitaria posso pensare a una copertura "simbolica" che copre

stato presente		0	1
	....		
s_p	s_j	$s_p, 0$	$s_q, 0$
	s_k	$s_p, 1$	$s_r, 0$
	....		

Esempio

- Si supponga di utilizzare 3 variabili di stato per codificare lo stato
- Si assegni **100** a s_j e **110** a s_k, mentre s_p ha codifica **010**

stato presente	0	1
....		
100	**0 1 0, 0**	**0 0 1, 0**
110	**0 1 0, 1**	**1 0 1, 0**
....		

- Se poi s_q e s_r non hanno altri vincoli posso assegnarli a due configurazioni adiacenti, ad esempio **001** e **101**
- Si osserva come per la seconda e la terza funzione di stato futuro sia stato possibile espandere rispetto alla seconda variabile di stato presente

Codifica dello stato con SIS

- SIS puó utilizzare due programmi esterni per la codifica dello stato `jedi` e `nova`
- Si considera il primo di questi che ha diverse opzioni per la codifica delle quali ci interessano
  - `-e r`: codifica casuale binaria che utilizza il minimo numero di flip-flop
  - `-e h`: one hot encoding che utilizza un flip-flop per ciascuno stato la cui uscita é 1 se la FSM é in tale stato e 0 altrimenti
  - `-e o`: output dominant encoding (si cerca principalmente di espandere gli implicanti delle uscite)
  - provate le altre opzioni
- Si tratta quindi di leggere il file con FSM ottimizzata per poi si da il comando `state_assign jedi -e r`, il risultato é una descrizione della rete a 2 livelli in formato BLIF

File BLIF

```
.model min.kiss
.inputs IN_0
.outputs OUT_0
.latch    [0] y_v1   0
.latch    [1] y_v2   1
.latch    [2] y_v3   1
.latch    [3] y_v4   0
.start_kiss
.i 1
.o 1
.p 19
.s 10
.r S2
0 S0 S2 0
1 S0 S2 1
0 S2 S3 0
1 S2 S4 0
0 S3 S5 0
1 S3 S6 0
0 S4 S7 0
1 S4 S8 0
0 S5 S0 0
1 S5 S9 0
....
.end_kiss
```

latches

annotated FSM

```
.latch_order y_v1 y_v2 y_v3 y_v4
.code S0 0111
.code S2 0110
.code S3 1001
.code S4 0011
.code S5 0001
.code S6 1111
.code S7 1010
.code S8 1100
.code S1 1101
.code S9 1011
.names IN_0 y_v1 y_v2 y_v3 y_v4 [0]
1-00- 1
0--10 1
1--00 1
1001- 1
0001- 1
.names IN_0 y_v1 y_v2 y_v3 y_v4 [1]
11--- 1
--1-1 1
-1--0 1
00-0- 1
1001- 1
-1011 1
```

state encoding

next state and output SP expressions

File BLIF

```
.names IN_0 y_v1 y_v2 y_v3 y_v4 [2]
--1-1 1
00-0- 1
1-00- 1
1--10 1
0--00 1
0001- 1
-1011 1
.names IN_0 y_v1 y_v2 y_v3 y_v4 [3]
--00- 1
-1--0 1
-111- 1
1--10 1
0--10 1
.names IN_0 y_v1 y_v2 y_v3 y_v4 OUT_0
101-1 1
-1011 1
.exdc
.inputs IN_0 y_v1 y_v2 y_v3 y_v4
.outputs [0] [1] [2] [3] OUT_0
.names y_v1 y_v2 y_v3 y_v4 [0]
010- 1
00-0 1
-000 1
1110 1
```

external don't cares

```
.names y_v1 y_v2 y_v3 y_v4 [1]
010- 1
00-0 1
-000 1
1110 1
.names y_v1 y_v2 y_v3 y_v4 [2]
010- 1
00-0 1
-000 1
1110 1
.names y_v1 y_v2 y_v3 y_v4 [3]
010- 1
00-0 1
-000 1
1110 1
.names y_v1 y_v2 y_v3 y_v4 OUT_0
010- 1
00-0 1
-000 1
1110 1
.end
```

Note

- Il file BLIF é annotato con la FSM in formato `kiss`, se si simula il BLIF in SIS la FSM e la rete sincrona vengono co-simulate
- Si ha poi la descrizione delle funzioni di stato futuro e uscita come espressioni SP
- Infine si ha con `.exdc` la descrizione dei don't care per ogni variabile di stato futuro e uscita
- Queste derivano dal fatto che alcune configurazioni delle variabili di stato non sono utilizzate e possono essere utilizzate da SIS per semplificare la parte combinatoria
- Il costo della rete é pari a 69 letterali

Technology independent optimization

- Si possono minimizzare le espressioni SP con `simplify`
- Poi si puó utilizzare un qualsiasi script per l'ottimizzazione di reti combinatorie multilivello come ad esempio `script.rugged`
- Dopo questi passaggi la rete ha un costo pari a 38 letterali con i nodi che passano da 5 a 7 a causa delle estrazioni
- Il file BLIF in cui puó essere scritta la rete risultante ha un formato simile a quello delle descrizione della rete a 2 livelli contenendo oltre alla rete sincrona multilivello, anche la FSM, la codifica e i don't care esterni

Technology mapping

- La libreria deve contenere sia celle combinatorie che sequenziali
- A questo scopo si é costruita una libreria contenente `mcnc.genlib` che `mcnc_latch.genlib`

```
....
# WARNING: area and delay parameters are arbitrary.
LATCH dff 10 Q = D;
PIN D NONINV 2 999 1.0 0.1 1.0 0.1
SEQ Q ANY RISING_EDGE
CONTROL CLOCK 2 999 1.0 0.1 1.0 0.1
CONSTRAINT D 0.1 0.1

LATCH dlatch 8 Q = D;
PIN D NONINV 2 999 1.0 0.1 1.0 0.1
SEQ Q ANY ACTIVE_HIGH
CONTROL CLOCK 2 999 1.0 0.1 1.0 0.1
CONSTRAINT D 0.1 0.1
```

- Una volta letta la libreria si procede con il technology mapping `map -m 0 -s` nel caso si voglia risparmiare area

Technology mapping: risultati e file BLIF

- L'area della rete mappata risulta pari a 90 a.u. e il ritardo massimo combinatorio a 23.30 a.u., mentre il periodo di clock minimo é pari a 24.30 a.u. (tiene conto del tempo di setup e di risposta dei FF)
- Il file BLIF che puó essere scritto con `write_blif -s -n <nome file>` risulta

```
.model state_opt.kiss
.inputs a
.outputs j
.mlatch dff D=r22 Q=b NIL 0
.mlatch dff D=q22 Q=c NIL 1
.mlatch dff D=w22 Q=d NIL 1
.mlatch dff D=a23 Q=e NIL 0
.start_kiss
....
.end_kiss
.latch_order b c d e
....
.gate inv1 a=a O=g22
.gate inv1 a=c O=h22
.gate aoi22 a=g22 b=e c=h22 d=b O=i22
.gate inv1 a=d O=j22
.gate inv1 a=b O=k22
.gate nor3 a=j22 b=c c=k22 O=l22
```

```
.gate nor3 a=h22 b=b c=g22 O=m22
.gate oai21 a=l22 b=m22 c=e O=n22
.gate oai21 a=e b=g22 c=b O=o22
.gate nand2 a=j22 b=o22 O=p22
.gate nand3 a=i22 b=n22 c=p22 O=q22
.gate oai22 a=q22 b=k22 c=b d=c O=r22
.gate nor2 a=q22 b=g22 O=u22
.gate nand2 a=r22 b=d O=v22
.gate oai21 a=u22 b=r22 c=v22 O=w22
.gate oai22 a=d b=g22 c=a d=j22 O=y22
.gate aoi22 a=r22 b=y22 c=b d=c O=z22
.gate nand2 a=e b=z22 O=a23
.gate inv1 a=n22 O=j
.exdc
....
.end
```

Suggerimenti

- Si provi la stessa procedura di sintesi utilizzando le altre opzioni di codifica messe a disposizione del comando `jedi`
- Si provino anche altre FSM