Laboratorio di Architettura degli Elaboratori

Nicola Bombieri
Dipartimento di Informatica
Università di Verona

A.A. 2014/2015

In questa lezione vengono riassunti i concetti fondamentali per la modellazione e la minimizzazione dei circuiti sequenziali mediante SIS.

Concetti fondamentali sulla modellazione di circuiti sequenziali

I circuiti sequenziali sono circuiti il cui comportamento dipende non solo dai valori assunti dai segnali di ingresso nel momento presente ma anche dai loro valori negli istanti passati.

Un circuito sequenziale può essere modellato utilizzando una macchina a stati finiti (Finite State Machine - FSM) definita come una 6-upla M = (S, I, O, δ, λ, *s*) dove:

- S è l'insieme degli stati;
- I è l'insieme degli ingressi;
- O è l'insieme delle uscite;
- δ è la *funzione di stato prossimo* che ad una coppia (ingresso, stato presente) associa lo stato prossimo;
- λ è la *funzione d'uscita* che
    - ad una coppia (ingresso, stato presente) associa un valore per l'uscita (FSM di Mealy);
    - ad uno stato presente associa un valore per l'uscita (FSM di Moore);
- *s* è lo stato di reset (a volte potrebbe non essere definito).

Quando il valore delle uscite dipende sia dallo stato presente sia dagli ingressi parliamo di *FSM di Mealy*, quando invece il valore delle uscite dipende solo dallo stato presente parliamo di *FSM di Moore*. In seguito tratteremo solo FSM di Mealy.

Solitamente una FSM si rappresenta per mezzo di un grafo delle transizioni (State Transition Graph - STG) dove i nodi sono gli stati e gli archi rappresentano le transizioni da uno stato all'altro. Ad ogni transizione corrisponde un insieme dei valori di ingresso ed un insieme dei valori di uscita.

I circuiti sequenziali possono essere *sincroni* se i valori delle uscite assumono significato in corrispondenza di un evento su un segnale di sincronismo (solitamente detto clock), oppure *asincroni* se i valori delle uscite cambiano al variare degli ingressi senza tener conto del segnale di sincronismo.

Siccome il comportamento delle uscite di un circuito sequenziale dipende dai valori in ingresso anche negli istanti passati occorre che esso contenga una sorta di memoria di tale storia passata. Questa memoria è rappresentata dallo **stato**. La realizzazione pratica di un circuito sequenziale pertanto richiede la presenza di elementi di memoria per la memorizzazione dello stato. L'elettronica digitale mette a disposizione due tipi di componenti detti **latch** e **flip-flop** in grado di memorizzare il valore di un bit (stato

binario). Occorre quindi codificare tutti gli stati che può assumere il circuito mediante dei numeri binari e poi utilizzare dei latch o flip-flop per memorizzare tali numeri. **Una FSM con N stati necessita di log_2N componenti di memoria elementari.**

La codifica scelta ha un notevole impatto sulla possibile minimizzazione del circuito. Esistono diverse regole che permettono di assegnare la codifica in modo da minimizzare la logica del circuito in base a area, ritardo, consumo di potenza, ecc.

Di seguito vengono riassunti i passi da eseguire per modellare, minimizzare e mappare su una libreria tecnologica un circuito sequenziale di cui sia nota una descrizione informale.

1. Rappresentare il diagramma degli stati.
2. Estrarre la tabella degli stati dal diagramma degli stati
3. Minimizzare gli stati della FSM mediante l'algoritmo di Paull-Unger (tabella delle implicazioni).
4. Assegnare una codifica agli stati.
5. Estrarre la tabella delle transizioni a partire dalla tabella degli stati e dalla codifica scelta.
6. Estrarre la tabella delle eccitazioni da cui è possibile estrarre la logica combinatoria che descrive la funzione di stato prossimo e la funzione di uscita.
7. Effettuare la minimizzazione delle logica combinatoria.
8. Effettuare il mapping tecnologico.

Per approfondimenti sulle operazioni sopra elencate si faccia riferimento ai capitoli 5, 6 del libro "Progettazione digitale" (2a edizione) di F. Fummi, C. Silvano, M. Sami.

Modellazione e minimizzazione di circuiti sequenziali in SIS

Per modellare un circuito sequenziale in SIS è necessario definire la tabella delle transizioni ed eventualmente definire manualmente la codifica degli stati (NB: esiste un comando che permette di eseguire la codifica in modo automatico).

I passi sono i seguenti:

1. La tabella delle transizioni deve essere descritta all'interno della sezione delimitata dalle keyword `.start_kiss` e `.end_kiss` Le transizioni devono essere specificate come un insieme di righe che riportano in ordine: valore degli ingressi, stato presente, stato prossimo, valore delle uscite. La tabella delle transizioni deve essere preceduta da 5 righe che specificano il numero di segnali di input, il numero di segnali di output, il numero di transizioni, il numero di stati e lo stato di reset (vedi esempio sotto).

2. Dopo la tabella delle transizioni (dopo `.end_kiss`) possono essere riportate le istruzioni necessarie per definire la codifica degli stati qualora non si decida di farla definire a SIS in modo automatico. La keyword da utilizzare per definire la codifica è `.code` seguita dal nome dello stato e dalla sua codifica binaria.

3. Dopo aver modellato il circuito, è possibile procedere con la minimizzazione degli stati. Una volta caricato il file .blif con il comando `read_blif`, la

minimizzazione degli stati si esegue con il comando `state_minimize stamina`.

4. Generazione logica funzioni δ e λ:

   -) se il file contiene già la codifica binaria degli stati (`.code`) allora occorre generare le funzioni δ e λ con il comando `stg_to_network`;

   -) altrimenti occorre assegnare automaticamente gli stati con il comando `state_assign jedi` (che genera anche le funzioni δ e λ). Attenzione che prima occorre minimizzare gli stati e poi farne l'assegnazione.

5. Eseguire la minimizzazione delle funzioni δ e λ, ad esempio lanciando lo script `script.rugged` come visto per la minimizzazione dei circuiti combinatori.

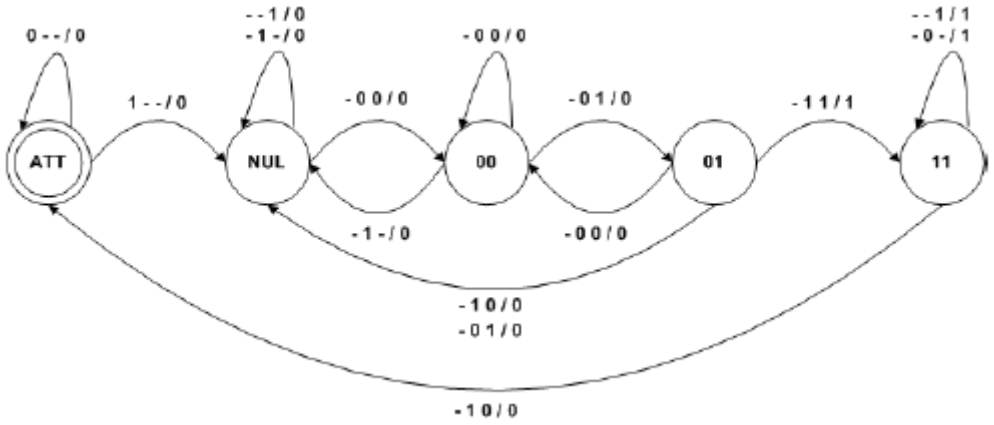
Eseguendo `print_stats` dopo `stg_to_network` oppure `state_assign jedi` si noterà che viene valorizzato il campo `latches` che indica il numero di elementi di memoria utilizzati nel circuito per memorizzare lo stato.

Riassumendo, la codifica degli stati si può effettuare con una delle seguenti modalità:

Con **codifica manuale** degli stati:	Con **codifica automatica** (SIS) degli stati
1. Tabella transizioni (STG) con keyword `.start_kiss - .end_kiss` 2. Minimizzazione stati con comando `sis>state_minimize stamina` 3. Codifica manuale stati con keyword `.code` 4. Generazione funzioni uscita e stato prossimo con comando `sis>stg_to_network` 5. Minimizzazione logica combinatoria delel funzioni uscita e stato prossimo con es. `script.rugged`	1. Tabella transizioni (STG) con keyword `.start_kiss - .end_kiss` 2. Minimizzazione stati con comando `sis>state_minimize stamina` 3. Codifica automatica stati e generazione funzioni uscita e stato prossimo con comando `sis>state_assign jedi` 4. Minimizzazione logica combinatoria delle funzioni uscita e stato prossimo con es. `script.rugged`

Esempio

Si consideri il circuito sequenziale che è in grado di riconoscere la sequenza di ingresso 00 01 11, (ingresso IN a 2 bit). Il circuito è attivo ed inizia ad analizzare i valori dell'ingresso IN quando l'ingresso START passa da 0 a 1. Nello stesso ciclo di clock in cui viene riconosciuta la sequenza 00 01 11, l'uscita OUT passa da 0 a 1. OUT rimane a 1 fino a quando gli ingressi assumeranno il valore 10; momento in cui il circuito viene nuovamente posto in attesa che il segnale START passi da 0 a 1. La seguente figura illustra il grafo di transizione degli stati.



Nello stato ATT, il circuito è in attesa che il segnale di start commuti da 0 a 1. Gli altri 4 stati indicano che una porzione della sequenza è stata riconosciuta. Si noti che se nello stato 01 viene applicato l'input 00, l'automa anziché portarsi nello stato NUL si sposta nello stato 00 riconoscendo l'inizio di una sequenza valida. Scegliendo di codificare gli stati come ATT = 000, NUL = 001, 00 = 010, 01 = 011 e 11 = 100, si riporta di seguito la tabella di verità della funzione dello stato prossimo e dell'uscita OUT. Lo stato attuale è codificato mediante i bit `a2 a1 a0`, mentre lo stato prossimo è codificato con i bit `s2 s1 s0`.

a2	a1	a0	START	IN1	IN0	s2	s1	s0	OUT
0	0	0	0	-	-	0	0	0	0
0	0	0	1	-	-	0	0	1	0
0	0	1	-	0	0	0	1	0	0
0	0	1	-	-	1	0	0	1	0
0	0	1	-	1	-	0	0	1	0
0	1	0	-	0	0	0	1	0	0
0	1	0	-	0	1	0	1	1	0
0	1	0	-	1	-	0	0	1	0
0	1	1	-	1	1	1	0	0	1
0	1	1	-	0	0	0	1	0	0
0	1	1	-	1	0	0	0	1	0
0	1	1	-	0	1	0	0	1	0
1	0	0	-	1	0	0	0	0	0
1	0	0	-	-	1	1	0	0	1
1	0	0	-	0	-	1	0	0	1

La rappresentazione nel formato blif è la seguente:

```
.model automa
.inputs START IN1 IN0
.outputs OUT

.start_kiss
.i 3            #numero di segnali di ingresso
.o 1            #numero di segnali di uscita
.s 5            #numero di stati
.p 15           #numero di transizioni
.r ATT          #stato di reset

#tabella delle transizioni
#(ingressi, stato presente, stato prossimo, uscita)
0-- ATT ATT 0
1-- ATT NUL 0
--1 NUL NUL 0
-1- NUL NUL 0
-00 NUL 00 0
-00 00 00 0
-1- 00 NUL 0
-01 00 01 0
-00 01 00 0
-10 01 NUL 0
-01 01 NUL 0
-11 01 11 1
--1 11 11 1
-0- 11 11 1
-10 11 ATT 0

.end_kiss

#codifica degli stati.
#E' opzionale perché può essere calcolata automaticamente
#tramite il comando state_assign jedi
.code ATT 000
.code NUL 001
.code 00 010
.code 01 011
.code 11 100

.end
```

Una volta caricato il file blif con il comando `read_blif`, è possibile creare le funzioni per lo stato prossimo e per l'output con il comando `stg_to_network` in quanto l'assegnazione degli stati è già stata fatta manualmente.

Comandi utili di SIS

`read_blif`	Carica la descrizione blif del circuito
`simulate i0 i1 i2 …`	Simula il circuito in base ai valori forniti per gli ingressi. Esecuzioni successive del comando considerano lo stato in cui il circuito si è portato dopo l'ultima esecuzione
`print_stats`	Visualizza informazioni sul circuito
`write_blif`	Visualizza la descrizione blif del circuito.
`write_eqn`	Visualizza le equazioni booleane corrispondenti ai nodi del circuito
`stg_to_network`	Costruisce le funzioni di stato prossimo e di uscita a partire dalla tabella delle transizioni e dalla codifica degli stati
`state_assign jedi`	Usa l'algoritmo jedi per effettuare automaticamente la codifica degli stati; costruisce anche le funzioni di stato prossimo e di uscita
`state_minimize stamina`	Usa l'algoritmo stamina per minimizzare gli stati della FSM
`write_kiss`	Visualizza la tabella delle transizioni

Esercizi

Esercizio 1: Realizzare il circuito sequenziale (2 ingressi, 2 uscite) corrispondente alla seguente tabella degli stati. Sia A lo stato di reset.

	00	01	10	11
A	D/10	D/00	A/11	D/11
B	B/1-	A/--	B/-1	C/--
C	C/0-	H/-1	I/-0	A/-0
D	A/10	A/-0	D/11	A/11
E	G/10	G/--	C/-1	I/1-
F	H/--	H/-0	A/01	B/10
G	A/10	G/-0	H/00	D/11
H	D/1-	G/-0	H/00	A/-1
I	F/--	F/11	E/00	A/01

Minimizzare il numero degli stati e ottimizzare la logica combinatoria.

Esercizio 2: Realizzare un circuito sequenziale con due ingressi binari in grado di riconoscere la sequenza 00 11 00 01. Si verifichi il comportamento del circuito durante il riconoscimento della sequenza 00 11 00 11 00 01.

Esercizio 3: Su due linee vengono trasmessi serialmente dei numeri binari a partire dal bit meno significativo (un bit ogni colpo di clock). Si progetti un sommatore seriale in grado di fornire in uscita, per ogni colpo di clock relativo a ciascun bit trasmesso, il corrispondente bit di somma. Minimizzare il numero degli stati e ottimizzare la logica combinatoria. nei 2 casi:

a) codifica manuale degli stati;

b) codifica automatica degli stati.