1 Sintesi delle reti sequenziali sincrone e asincrone

Corso di Architetture dei Sistemi di Elaborazione
prof. Antonino Mazzeo

1.1 Progetto di una rete sequenziale

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

1. Mediante un diagramma temporale in cui sono poste in evidenza le principali sequenze in ingresso al sistema che si intende realizzare e quelle desiderate di uscita, identificare gli stati della rete.

2. Compilare la relativa tabella primitiva (una tabella ridondante nel numero degli stati e che ha nelle righe gli stati/uscite se macchina di Mealy e nelle colonne gli ingressi) strutturata con un unico stato stabile in corrispondenza di ogni riga e nelle colonne la sequenza di stati transienti che per un assegnato ingresso portano allo stato stabile (eventualmente rappresentare la rete con un diagramma degli stati).

3. Derivare dalla tabella primitiva la descrizione in formato kiss da, includere successivamente in un unico file .blif di descrizione del SIStema da realizzare. Leggere in SIS il file .blif, precedentemente editato con un editor di testo, dando il comando:

   ```
   sis> read_blif <nome_file>.blif
   ```

4. Procedere con la minimizzazione degli stati della FSM (operazione che a mano è effettuata mediante l'algoritmo di Paull-Unger a partire dalla tabella primitiva completa delle transizioni e dello stato delle uscite) eseguendo in SIS il programma STAMINA che implementa un metodo di minimizzazione dello spazio degli stati di tipo euristico per macchine incompletamente specificate, dando il comando:

   ```
   sis> state_minimize stamina
   ```

5. Effettuare la codifica degli stati al fine di poter effettuare il mapping del grafo STG (State Transition Graph) che descrive la macchina a stati finiti nella netlist che ne caratterizza la specifica implementazione logica. Il mapping è ottimo nel senso che l'algoritmo è in grado di calcolare il codice binario ottimo per ciascuno degli stati simbolici della macchina. A partire da un tale codice, si crea per ciascuno dei bit

che lo compongono un latch, generando in tal modo un'implementazione a livello logico della rete. SIS mette a disposizione due programmi di codifica degli stati: JEDI e NOVA. Il primo, JEDI, è un programma di codifica generale degli stati, utilizzabile anche per la codifica degli stati di ingresso e uscita, per reti multilivello. Il secondo, NOVA, è particolarmente indicato per macchine a stati basata su PLA, ma può anche essere utilizzato per implementazioni multilivello. Eseguire, pertanto l'uno o l'altro dei due seguenti comandi:

```
state_assign jedi
state_assign nova
```

6. Estrarre la tabella delle transizioni a partire dalla tabella degli stati e dalla codifica scelta dando il comando:

   ```
   stg_to_network
   ```

7. Effettuare la minimizzazione della logica combinatoria eseguendo lo script standard seguente:

   ```
   source script
   ```

8. Eseguire la semplificazione dei nodi minimizzando le funzioni booleane di ciascuno essi, dando il comando:

   ```
   full_simplify
   ```

9. Estrarre, basandosi sull'analisi degli stati irraggiungibili, i don't care della rete sequenziale dando il comando:

   ```
   extract_seq_dc
   ```

10. Rieseguire la semplificazione dei nodi minimizzando le funzioni booleane di ciascuno essi, dando il comando:

    ```
    full_simplify
    ```

11. Rieseguire l'estrazione di ulteriori don't care ed eseguire lo script rugged

    ```
    extract_seq_dc
    source script.rugged
    ```

12. eseguire il retime

    ```
    retime -n
    ```

13. Leggere la libreria tecnologica che include anche i latch

    ```
    sis> rlib2.genlib
    ```

14. Effettuare il mapping tecnologico

    ```
    sis> map -s
    ```

2 Rete Sequenziale asincrona

```
#-------------------------------------------------------
  # Esercizio rete asincrona sintetizzato con SIS
  # Descrizione: progettare una rete asincrona con due ingressi x1 e x2
  # e una uscita z, tale che l'uscita assume il valore 1 solo se gli ingressi
  # sono entrambi alti (x1=x2=1) e la penultima transizione di livello sia
  # avvenuta sulla linea x1, successivamente l'uscita z torna al valore 0
  # non appena x1=x2=0.
  # File : asyncrona1.blif
  # Autore: Antonino Mazzeo
  #<Lo ssvolgimento con il metodo manuale è in esercizio 3 di Aldo Esposito>
  #-------------------------------------------------------
  .model
  .inputs x1 x2
  .outputs z
  .start_kiss
  .i 2
  .o 1
  .s 7
  .p 19
  .r q1
  00 q1 q1 0
  01 q1 q3 0
  10 q1 q2 0
  00 q2 q1 0
  10 q2 q2 0
  11 q2 q4 -
  00 q3 q1 0
  01 q3 q3 0
  11 q3 q7 0
  01 q4 q6 -
  10 q4 q5 -
  11 q4 q4 1
  00 q5 q1 0
  10 q5 q5 0
  11 q5 q7 0
  00 q6 q1 0
  01 q6 q6 0
```

```
11 q6 q4 -
01 q7 q6 0
10 q7 q5 0
11 q7 q7 0
.end_kiss
.end
SIS> set autoexec print_stats
SIS> read_blif ./esercizi/async_ese3.blif
async_ese3.blif pi= 2 po= 1 nodes= 1 latches= 0
lits(sop)= 0 #states(STG)= 7
SIS> state_minimize stamina
Running stamina, written by June Rho, University of Colorado at Boulder
Number of states in original machine : 7
Number of states in minimized machine : 3
async_ese3.blif pi= 2 po= 1 nodes= 1 latches= 0
lits(sop)= 0 #states(STG)= 3
SIS> state_assign jedi
Running jedi, written by Bill Lin, UC Berkeley
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 12 #states(STG)= 3
SIS> stg_to_network
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 12 #states(STG)= 3
SIS> extract_seq_dc
number of latches = 2 depth = 3 states visited = 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 12 #states(STG)= 3
SIS> source script.rugged
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 12 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 12 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
```

```
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
SIS> write_kiss
.i 2
.o 1
.p 12
.s 3
.r S1
00 S0 S1 0
01 S0 S1 0
10 S0 S0 0
11 S0 S0 0
00 S1 S1 0
01 S1 S2 0
10 S1 S2 0
11 S1 S2 -
00 S2 S1 0
01 S2 S2 0
10 S2 S0 -
11 S2 S2 1
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
SIS> write_eqn
Warning: only combinational portion is being written.
INORDER = x1 x2 LatchOut_v2 LatchOut_v3;
OUTORDER = [10] [11] z;
[10] = !x1*LatchOut_v3 + !x1*!x2;
[11] = !x2*z + LatchOut_v3*![10];
z = x1*!LatchOut_v2*!LatchOut_v3;
Don't care:
INORDER = LatchOut_v3 LatchOut_v2;
OUTORDER = LatchIn_v4.0 LatchIn_v4.1 z;
```

```
dc[25] = LatchOut_v3*LatchOut_v2;
LatchIn_v4.0 = dc[25];
LatchIn_v4.1 = dc[25];
z = dc[25];
async_ese3.blif pi= 2 po= 1 nodes= 3 latches= 2
lits(sop)= 11 #states(STG)= 3
SIS>
```

3 Rete sequenziale sincrona

3.1 Semaforo

Si progetti come rete sequenziale sincrona l'unità di controllo di un semaforo con priorità che controlla il traffico di auto nelle due direttrici principali Nord-Sud (NS) ed Est-Ovest (EO). Il controllo dispone di due ingressi e due uscite. Gli ingressi, se posti a 1 indicano la presenza di auto in ciascuna delle due direttrici, le uscite pilotano le due luci Verde e Rosso (in questa versione non si utilizza il giallo) del semaforo. Il controllo opera in modo semplificato e descritto dalla seguente tabella delle transizioni di stato secondo il modello di Mealy:

Stato/Ingressi	00	01	10	11
verde_ns (Vns)	Vns/10	Veo/01	Vns/10	Vns/10
verde_eo (Veo)	Veo/01	Veo/01	Vns/10	Vns/10

e a cui corrisponde il seguente diagramma degli stati:

Il controlo semaforico sarà progettato utilizzando gli strumenti di sintesi presenti nel pacchetto SIS di Berkeley.

Per prima cosa si predispone con un editor di testi il file di specifica che chiameremo semaforo.blif (il formato .blif è discusso nel manuale d'uso di SIS).

Il file conterrà le seguenti linee:

Figura 3.1: diagramma degli stati del semaforo secondo Mealy

```
#------------------------------------------------------------------------
# Esempio tratto dal testo F.Fummi, M.G.Sami, C.Silvano,
#"Progettazione Digitale", MacGrawHill
# cap. 6.6.1 semaforo a 2bit
# File : semaforo.blif
#------------------------------------------------------------------------
.model SEMAFORO
.inputs TRAFFICONS TRAFFICOEO
.outputs LUCENS LUCEEO
.start_kiss
.i 2
.o 2
.s 2
.r VERDENS
.p 5
00 VERDENS VERDENS 10
01 VERDENS VERDEEO 01
1- VERDENS VERDENS 10
0- VERDEEO VERDEEO 01
1- VERDEEO VERDENS 10
.end_kiss
.end
```

Il modello del semaforo è caratterizzato dall'elenco presente nei .inputs e .outputs, mentre la tabella delle transizioni e delle uscite è racchiusa fra .start_kiss ed .end_kiss, preceduta dall'indicazione del numero di ingressi (.i), delle uscite (.o), del numero di stati (.s), dallo stato di reset (.r) e dal numero di linee della tabella (.p).

Di seguito si mostra l'output prodotto da SIS al terminale a seguito dei comandi

1. read_blif, per il caricamento del file;
2. print_stats, per stamparne le statistiche;
3. write_blif, per la stampa del contenuto del file caricato semaforo.blif.

In particolare, il comando 2) indica che il modello immesso presenta pi=2 ingressi; po=2 uscite; una rete composta da 2 nodi a cui non sono ancora associate equazioni essendo il numero di letterali in somma di prodotti lits(sop)=0; è, infine, indicato la presenza di uno state graph con due stati #states(STG)=2.

```
MacBook-di-Antonino-Mazzeo:semaforo mazzeo$ SIS
UC Berkeley, SIS 1.3.6 (compiled 2007-12-24 16:56:48)
SIS> read_blif semaforo_1.blif
SIS> print_stats
SEMAFORO pi= 2 po= 2 nodes= 2 latches= 0
lits(sop)= 0 #states(STG)= 2
SIS> write_blif
.model SEMAFORO
.inputs TRAFFICONS TRAFFICOEO
.outputs LUCENS LUCEEO
.start_kiss
.i 2
.o 2
.p 5
.s 2
.r VERDENS
00 VERDENS VERDENS 10
01 VERDENS VERDEEO 01
1- VERDENS VERDENS 10
0- VERDEEO VERDEEO 01
1- VERDEEO VERDENS 10
.end_kiss
.latch_order
.names LUCENS
.names LUCEEO
.end
SIS>
```

Il modello così descritto può anche essere simulato nelle sue transizioni fornendo, a partire dallo stato di reset VERDENS sequenze di ingressi e controllandone le uscite con il comando `simulate`. Di seguito è mostrato l'uso di tale comando per verificare tutte le transizioni presenti nella tabella delle transizioni del semaforo:

```
SIS> simulate 0 0
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 1 0
Next state: VERDENS ((null))
SIS> simulate 1 0
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 1 0
Next state: VERDENS ((null))
SIS> simulate 1 1
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 1 0
Next state: VERDENS ((null))
SIS> simulate 0 1
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 0 1
Next state: VERDEEO ((null))
SIS> simulate 0 0
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 0 1
Next state: VERDEEO ((null))
SIS> simulate 0 1
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 0 1
Next state: VERDEEO ((null))
SIS> simulate 1 0
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 1 0
Next state: VERDENS ((null))
SIS> simulate 0 1
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 0 1
Next state: VERDEEO ((null))
SIS> simulate 1 1
Network simulation:
Outputs: 0 0
Next state:
STG simulation:
Outputs: 1 0
Next state: VERDENS ((null))
SIS>
```

Ciascun passo della simulazione indica le uscite della rete (network) associate al grafo delle transizioni di stato, che in questa fase non è stato ancora realizzato, per cui le uscite sono poste a zero, le uscite previste dal grafo e lo stato prossimo. Una volta generata la network con il comando `stg_to_network`, si mostrerà che le due simulazioni coincideranno.

Prima di procedere alla costruzione della rete, occorre codificare gli stati della macchina, cosa che può essere fatta o con una assegnazione casuale dei codifici o con una codifica ottima. In SIS sono presenti due strumenti per la codifica ottima automatica, Jedi e Nova, applicabili mediante appositi comandi. Stante la semplicità dell'esempio trattato e al fine di mostrare il funzionamento dei vari strumenti adoperati per la sintesi, si procederà con una codifica casuale dei codici agli stati e, inizialmente, non sarà effettuata la minimizzazione della parte combinatoria della rete. Di seguito è riportato il codice aggiornato con le linee per la codifica degli stati posti dopo la sezione kiss e posto nel file semaforo_2.blif

```
#-------------------------------------------------------------------------
# Esempio tratto dal testo F.Fummi, M.G.Sami, C.Silvano,
#"Progettazione Digitale", MacGrawHill
# cap. 6.6.1 semaforo a 2 bit
# File : semaforo_2.blif
#-------------------------------------------------------------------------
.model SEMAFORO
.inputs TRAFFICONS TRAFFICOEO
.outputs LUCENS LUCEEO
.start_kiss
.i 3
.o 3
.p 10
.s 4
.r VERDENS
.p 5
00- VERDENS VERDENS 110
01 VERDENS VERDEEO 01
1- VERDENS VERDENS 10
0- VERDEEO VERDEEO 01
1- VERDEEO VERDENS 10
.end_kiss
.code VERDENS 00
.code VERDEEO 10
.end
```

Caricato in SIS tale nuovo modello, viene dato il comando `stg_to_network` per creare automaticamente le funzioni di stato prossimo e di uscita della rete che viene mostrata con il comando `write_blif`. Dopo il passo di codifica fatto casualmente o in modo automatico, il numero di variabili di stato può risultare ridontante, per cui si possono utilizzare i codici non utilizzati come condizionidi don't care ai fini della ottimizzazione delle reti combinatorie. All'interno di SIS non eSISte un meccanismo automatico per la scrittura nel modello della rete delle condizioni di indifferenza corrispondenti alle eventuali codifiche di stato non utilizzate. Tale operazione, pertanto, deve essere effettuata o manualmente o ricorrendo al comando `extract_seq_dc` che, utilizzando diagrammi di decisone binaria, attraversa la FSM estratta da un circuito e identifica se presenti, gli

stati non raggiungibili. Questo comando costruisce automaticamente anche le condizioni di indifferenza.

```
SIS> read_blif semaforo_2.blif
SIS> print_stats
SEMAFORO pi= 2 po= 2 nodes= 2 latches= 0
lits(sop)= 0 #states(STG)= 2
SIS> stg_to_network
SIS> write_blif
.model SEMAFORO
.inputs TRAFFICONS TRAFFICOEO
.outputs LUCENS LUCEEO
.latch [6] LatchOut_v2 0
.latch [7] LatchOut_v3 0
.start_kiss
.i 2
.o 2
.p 5
.s 2
.r VERDENS
00 VERDENS VERDENS 10
01 VERDENS VERDEEO 01
1- VERDENS VERDENS 10
0- VERDEEO VERDEEO 01
1- VERDEEO VERDENS 10
.end_kiss
.latch_order LatchOut_v2 LatchOut_v3
.code VERDENS 00
.code VERDEEO 10
.names TRAFFICONS TRAFFICOEO LatchOut_v2 [6]
01- 1
0-1 1
.names [7]
.names TRAFFICONS TRAFFICOEO LatchOut_v2 LUCENS
1-- 1
-00 1
.names TRAFFICONS TRAFFICOEO LatchOut_v2 LUCEEO
01- 1
0-1 1
.exdc
.inputs TRAFFICONS TRAFFICOEO LatchOut_v2 LatchOut_v3
.outputs [6] [7] LUCENS LUCEEO
.names LatchOut_v3 [6]
1 1
.names LatchOut_v3 [7]
1 1
.names LatchOut_v3 LUCENS
1 1
.names LatchOut_v3 LUCEEO
1 1
.end
SIS>
```

Effettuata la codifica degli stati e la costruzione del network, se si effettua la simulazione della tabella delle transizioni di stato, una volta creata la rete, i risultati di STG e

del network sono coerenti:

```
SIS> simulate 0 1
Network simulation:
Outputs: 0 1
Next state: 10
STG simulation:
Outputs: 0 1
Next state: VERDEEO (10)
SIS> simulate 1 0
Network simulation:
Outputs: 1 0
Next state: 00
STG simulation:
Outputs: 1 0
Next state: VERDENS (00)
```

Il costo della rete è ottenuto mediante il comando `print_stats` che riporta

```
SIS> print_stats
SEMAFORO       pi= 2 po= 2 nodes=  4 latches= 2
lits(sop)=  11 #states(STG)=   2
```

indicando un costo di 11 letterali per la rete combinatoria e di 2 latch per memorizzare i due stati del grafo.

Le equazioni della rete sono ottenute mediante il comando `write_eqn` di seguito riportato

```
SIS> write_eqn
Warning: only combinational portion is being written.
INORDER = TRAFFICONS TRAFFICOEO LatchOut_v2 LatchOut_v3;
OUTORDER = [6] [7] LUCENS LUCEEO;
[6] = !TRAFFICONS*LatchOut_v2 + !TRAFFICONS*TRAFFICOEO;
[7] = 0;
LUCENS = !TRAFFICOEO*!LatchOut_v2 + TRAFFICONS;
LUCEEO = !TRAFFICONS*LatchOut_v2 + !TRAFFICONS*TRAFFICOEO;
Don't care:
INORDER = TRAFFICONS TRAFFICOEO LatchOut_v2 LatchOut_v3;
OUTORDER = LatchIn_v4.0 LatchIn_v4.1 LUCENS LUCEEO;
LatchIn_v4.0 = LatchOut_v3;
LatchIn_v4.1 = LatchOut_v3;
LUCENS = LatchOut_v3;
LUCEEO = LatchOut_v3;
```