Laboratorio di Architettura degli Elaboratori

Nicola Bombieri
Dipartimento di Informatica
Università di Verona
A.A. 2010/2011

Progetto Controllore-Datapath

In questa lezione vengono riassunti i concetti fondamentali per la modellazione di un circuito descritto dall'interazione tra un controllore, realizzato mediante una macchina a stati finiti (FSM), e un datapath, realizzato mediante un insieme di unità funzionali..

Progetto di circuiti modulari in SIS

Per includere un componente all'interno di un modello blif usare le seguenti istruzioni:

```
.subckt nomecomponente parametroformale=parametroattuale …

.search nomefilecomponente.blif
```

La prima istruzione permette di includere un componente di nome "*nomecomponente*" e di collegare i suoi parametri formali con i parametri attuali. La seconda istruzione indica a SIS quale sia il file che contiene la definizione del componente "*nomecomponente*".

Unità funzionali di un datapath

Gli elementi principali di un datapath sono componenti per la memorizzazione dei dati (*registri*) e componenti per l'elaborazione dei dati (*unità funzionali*). Di seguito vengono riportati i modelli blif di alcuni di tali componenti.

1) **Registro a 1 bit**

```
.model REGISTRO
.inputs A
.outputs O
.latch A O re NIL 0
.end
```

2) **Registro a 4 bit**

```
.model REGISTRO4
.inputs A3 A2 A1 A0
.outputs O3 O2 O1 O0
.subckt REGISTRO A=A3 O=O3
.subckt REGISTRO A=A2 O=O2
.subckt REGISTRO A=A1 O=O1
.subckt REGISTRO A=A0 O=O0
```

```
.end
.search registro.blif
```

3) Sommatore

```
.model SOMMATORE
.inputs A B CIN
.outputs O COUT
.names A B K
10 1
01 1
.names K CIN O
10 1
01 1
.names A B CIN COUT
11- 1
1-1 1
-11 1
.end
```

4) Sommatore a 2 bit

```
.model SOMMATORE2
.inputs A1 A0 B1 B0 CIN
.outputs O1 O0 COUT
.subckt SOMMATORE A=A0 B=B0 CIN=CIN O=O0 COUT=C0
.subckt SOMMATORE A=A1 B=B1 CIN=C0 O=O1 COUT=COUT
.search sommatore.blif
.end
```

5) Multiplexer a 4 ingressi 1 bit ciascuno

```
.model MUX1
.inputs S1 S0 i3 i2 i1 i0
.outputs out
.names S1 S0 i3 i2 i1 i0 out
001--- 1
01-1-- 1
10--1- 1
11---1 1
.end
```

6) Multiplexer a 4 ingressi 2 bit ciascuno

```
.model MUX2
.inputs X1 X0 a1 a0 b1 b0 c1 c0 d1 d0
.outputs o1 o0
.subckt MUX1 S1=X1 S0=X0 i3=a1 i2=b1 i1=c1 i0=d1
out=o1
.subckt MUX1 S1=X1 S0=X0 i3=a0 i2=b0 i1=c0 i0=d0
out=o0
.search mux1.blif
```

```
.end
```

7) Multiplexer a 2 ingressi da 3 bit ciascuno

```
.model MUX3
.inputs A2 A1 A0 B2 B1 B0 S
.outputs O2 O1 O0

.names S A2 B2 O2
11- 1
0-1 1
.names S A1 B1 O1
11- 1
0-1 1
.names S A0 B0 O0
11- 1
0-1 1
.end
```

8) Demultiplexer a 1 bit e 4 uscite

```
.model DEMUX
.inputs S1 S0 IN
.outputs X Y Z W
.names S1 S0 IN X
001 1
.names S1 S0 IN Y
011 1
.names S1 S0 IN Z
101 1
.names S1 S0 IN W
111 1
.end
```

9) Comparatore a 4 bit

```
.model UGUALE4
.inputs A3 A2 A1 A0 B3 B2 B1 B0
.outputs O
.subckt xnor A=A3 B=B3 X=X3
.subckt xnor A=A2 B=B2 X=X2
.subckt xnor A=A1 B=B1 X=X1
.subckt xnor A=A0 B=B0 X=X0
.names X3 X2 X1 X0 O
1111 1
.search xnor.blif
.end
```

10) **“Maggiore” a 6 bit (restituisce 1 se il numero rappresentato dai primi 6 bit è maggiore del numero rappresentato dai successivi 6 bit)**

```
.model 6_gt
.inputs A5 A4 A3 A2 A1 A0 B5 B4 B3 B2 B1 B0
.outputs AgtB
.subckt xor A=A5 B=B5 X=X5
.subckt xor A=A4 B=B4 X=X4
.subckt xor A=A3 B=B3 X=X3
.subckt xor A=A2 B=B2 X=X2
.subckt xor A=A1 B=B1 X=X1
.subckt xor A=A0 B=B0 X=X0
.names A5 A4 A3 A2 A1 A0 X5 X4 X3 X2 X1 X0 AgtB
1-----1----- 1
-1----01---- 1
--1---001--- 1
---1--0001-- 1
----1-00001- 1
-----1000001 1

.search xor.blif
.end
```

11) **“Minore-uguale” a 6 bit partendo da un “Maggiore” a 6 bit (restituisce 1 se il numero rappresentato dai primi 6 bit è minore o uguale al numero rappresentato dai successivi 6)**

```
.model 6_le
.inputs C5 C4 C3 C2 C1 C0 D5 D4 D3 D2 D1 D0
.outputs CleD
.subckt 6_gt A5=C5 A4=C4 A3=C3 A2=C2 A1=C1 A0=C0
B5=D5 B4=D4 B3=D3 B2=D2 B1=D1 B0=D0 AgtB=z
.names z CleD
0 1

.search 6_gt.blif
.end
```

12) **generazione di un valore costante a 1 (di un bit)**

```
.model uno1
.outputs uscita
.names uscita
1
.end
```

13) **generazione di un valore costante a 0 (di un bit)**

```
.model zero1
.outputs uscita
.names uscita
.end
```

14) generazione del valore costante dieci (su 4 bit)

```
.model dieci4
.outputs O3 O2 O1 O0
.subckt uno1 uscita=O3
.subckt zero1 uscita=O2
.subckt uno1 uscita=O1
.subckt zero1 uscita=O0
.end
```

Modellazione di una FSMD in SIS

Per modellare una FSMD in SIS è necessario eseguire le seguenti operazioni:

1. Modellare la FSM del Controllore mediante la tabella delle transizioni e assegnare la codifica agli stati con "`state_assign jedi`" (vedere la lezione sulla "Modellazione e minimizzazione dei circuiti sequenziali").
2. Salvare il Controllore ottenuto dopo la codifica degli stati in un file diverso da quello originale usando il comando `write_blif nomefile`.
3. Modellare il Datapath come una interconnessione di componenti funzionali, quali sommatori, moltiplicatori, multiplexer, registri, ecc.
4. Inglobare il Controllore e il Datapath in un unico file blif come se fossero due componenti, collegando opportunamente i segnali di comunicazione. E' importante specificare con la direttiva `.search` il file del Controllore ottenuto dopo la codifica degli stati e non quello scritto a mano.

Esempio: Semaforo con Priorità Temporizzato

Si consideri il classico esempio di un semaforo posto all'incrocio tra una strada principale (direttrice nord sud, NS) e una secondaria (direttrice est ovest, EO). Il semaforo può assumere solo i colori verde e rosso ed è dotato di sensori che rilevano la presenza di traffico. Il circuito che controlla il semaforo ha 2 segnali di ingresso forniti dai sensori (TRAFFICONS e TRAFFICOEO) e due di uscita (LUCENS e LUCEEO) con il seguente significato:

- TRAFFICONS[1]: segnala la presenza di traffico lungo la direttrice NS assumendo il valore 1.
- TRAFFICOEO[1]: segnala la presenza di traffico lungo la direttrice EO assumendo il valore 1.
- LUCENS[1]: deve essere posto a 1 per accendere la luce verde sulla strada NS. Se viene posto a 0 si accende la luce rossa.
- LUCEEO[1]: deve essere posto a 1 per accendere la luce verde sulla strada EO. Se viene posto a 0 si accende la luce rossa.

Per evitare incidenti, il circuito di controllo deve garantire che le luci sulle strade NS e EO siano sempre accese in opposizione. Il circuito assegna priorità alla strada NS e commuta dal verde al rosso su NS solo se TRAFFICONS=0 e TRAFFICOEO=1, in caso contrario mantiene il verde su NS. In assenza di traffico sia su NS che su EO il semaforo non modifica la configurazione raggiunta. Non appena giunge traffico su NS, indipendentemente da cosa succede su EO, il semaforo assegna la luce verde a NS e la luce rossa a EO. Il controllore non commuta mai un semaforo da verde a rosso senza aver atteso 1024 cicli di clock.

Descrivere con una FSM un contatore modulo 1023 è una operazione semplice, ma richiede la definizione esplicita di 1023 stati. Si può invece immaginare di progettare una semplice unità di elaborazione con un ingresso (*INIZIO*[1]) e una uscita (*FINE*[1]) che conti 1023 cicli di clock. Questo contatore viene posto a 0 quando *INIZIO* vale 1 e pone a 1 *FINE* quando raggiunge il valore 1023.

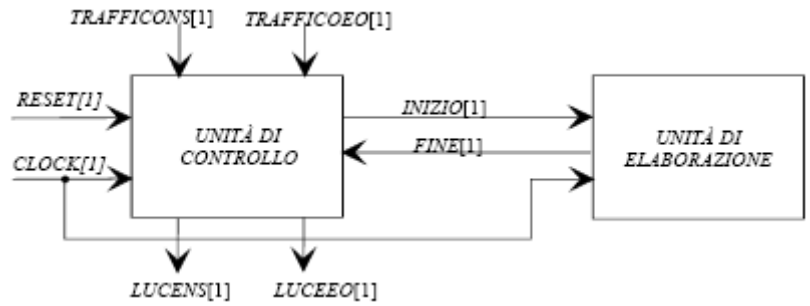


Figura1: FSMD del semaforo temporizzato.

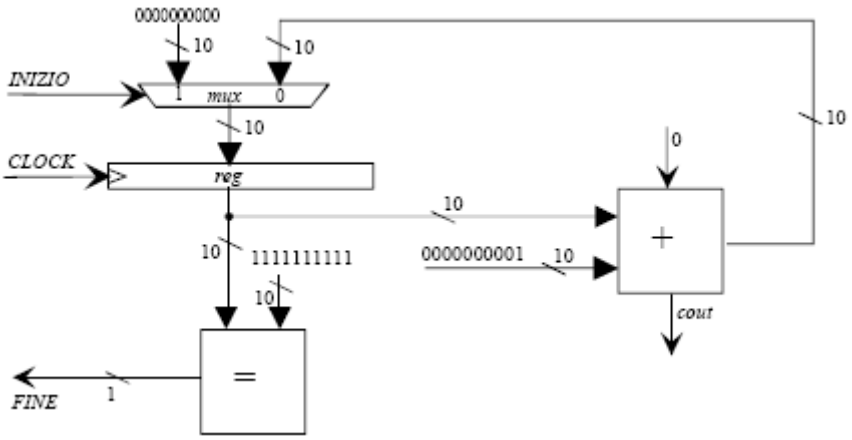


Figura2: Datapath del semaforo temporizzato.

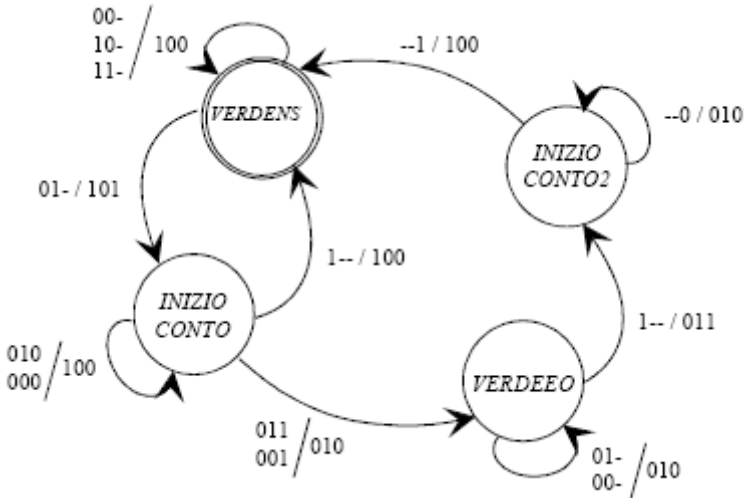


Figura3: FSM del semaforo temporizzato.

Per semplicità di simulazione, riportiamo di seguito una versione del semaforo con priorità che attende solo 8 cicli di clock invece di 1024 prima di commutare da rosso a verde.

```
################################################
######## file semaforo_temporizzato.blif ########
################################################
.model SEMAFORO_TEMPORIZZATO
.inputs TRAFFICONS TRAFFICOEO
.outputs LUCENS LUCEEO

# istanziazione dell'unita' di controllo
.subckt CONTROLLO TRAFFICONS=TRAFFICONS TRAFFICOEO=TRAFFICOEO \
FINE=FINE LUCENS=LUCENS LUCEEO=LUCEEO INIZIO=INIZIO

# istanziazione dell'unita' di elaborazione (datapath)
.subckt ELABORAZIONE INIZIO=INIZIO FINE=FINE
.end

# file che contiene il modello del controllo
.search controllo.blif

# file che contiene il modello del datapath
.search elaborazione.blif


########################################
######## file elaborazione.blif ########
########################################
.model ELABORAZIONE
.inputs INIZIO
.outputs FINE

# definizione delle costanti
.names ZERO # la costante zero

# le costanti 0, 1 e 7 rappresentate usando 3 bit
.subckt     COSTANTE0 O2=C02 O1=C01 O0=C00
.subckt     COSTANTE1 O2=C05 O1=C04 O0=C03
.subckt     COSTANTE7 O2=C08 O1=C07 O0=C06
```

```
# istanziazione del multiplexer che selezione tra 0 ed il valore
# corrente del contatore
.subckt MUX3 A2=C02 A1=C01 A0=C00 B2=S2 B1=S1 B0=S0 S=INIZIO O2=M2 \
O1=M1 O0=M0

# istanziazione del registro che memorizza il valore del contatore
.subckt REGISTRO3 A2=M2 A1=M1 A0=M0 O2=R2 O1=R1 O0=R0

# istanziazione del sommatore che incrementa di 1 il valore del
# registro ad ogni ciclo di clock
.subckt SOMMATORE3 A2=R2 A1=R1 A0=R0 B2=C05 B1=C04 B0=C03 CIN=ZERO \
O2=S2 O1=S1 O0=S0 COUT=COUT

# istanziazione del comparatore che verifica se il valore del registro
# vale 7
.subckt UGUALE3 A2=R2 A1=R1 A0=R0 B2=C08 B1=C07 B0=C06 O=FINE
.end

# modello della costante 0 su 3 bit
.model COSTANTE0
.outputs O2 O1 O0
.names O2
.names O1
.names O0
.end

# modello della costante 1 su 3 bit
.model COSTANTE1
.outputs O2 O1 O0
.names O2
.names O1
.names O0
1
.end

# modello della costante 7 su 3 bit
.model COSTANTE7
.outputs O2 O1 O0
.names O2
1
.names O1
1
.names O0
1
.end

# file che contengono i modelli del registro, del multiplexer, del
# sommatore e del comparatore
.search registro3.blif
.search mux3.blif
.search sommatore3.blif
.search uguale3.blif

#####################################
######## file controllo.blif ########
#####################################
.model CONTROLLO
.inputs TRAFFICONS TRAFFICOEO FINE
.outputs LUCENS LUCEEO INIZIO
```

```
# tabella delle transizioni
.start_kiss
.i 3
.o 3
.p 10
.s 4
.r VERDENS
00- VERDENS VERDENS 100
1-- VERDENS VERDENS 100
01- VERDENS INIZIO_CONTO 101
0-0 INIZIO_CONTO INIZIO_CONTO 100
1-- INIZIO_CONTO VERDENS 100
0-1 INIZIO_CONTO VERDEEO 010
0-- VERDEEO VERDEEO 010
1-- VERDEEO INIZIO_CONTO2 011
--0 INIZIO_CONTO2 INIZIO_CONTO2 010
--1 INIZIO_CONTO2 VERDENS 100
.end_kiss

# logica combinatoria per la funzione di stato prossimo e la funzione
# di uscita calcolata da SIS usando il comando "state_assign jedi"
.latch    [3] LatchOut_v3   0
.latch    [4] LatchOut_v4   0
.latch_order LatchOut_v3 LatchOut_v4
.code VERDENS 00
.code INIZIO_CONTO 01
.code VERDEEO 10
.code INIZIO_CONTO2 11
.names TRAFFICONS FINE LatchOut_v3 LatchOut_v4 [3]
--10 1
-011 1
0101 1
.names TRAFFICONS TRAFFICOEO FINE LatchOut_v3 LatchOut_v4 [4]
--011 1
1--10 1
0-001 1
01-00 1
.names TRAFFICONS FINE LatchOut_v3 LatchOut_v4 LUCENS
1-0- 1
--00 1
-111 1
0001 1
.names TRAFFICONS FINE LatchOut_v3 LatchOut_v4 LUCEEO
--10 1
-011 1
0101 1
.names TRAFFICONS TRAFFICOEO LatchOut_v3 LatchOut_v4 INIZIO
1-10 1
0100 1
.end
```

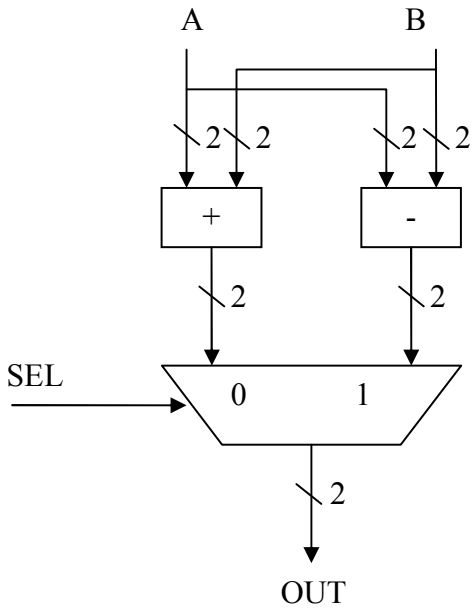
Comandi utili di SIS

`read_blif nomefile`	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_blif nomefile`	Salva su file la descrizione blif del circuito modificata da SIS;
`write_eqn`	Visualizza le equazioni booleane corrispondenti ai nodi del circuito;
`state_minimize stamina`	Usa l'algoritmo stamina per minimizzare gli stati della FSM;
`state_assign jedi`	Usa l'algoritmo jedi per assegnare automaticamente una codifica degli stati che permetta una minimizzazione del circuito e costruisce le funzioni di stato prossimo e di uscita;
`source nomefile`	Esegue la sequenza di comandi contenuta nel file `nomefile`;
`write_kiss`	Visualizza la tabella delle transizioni.

NOTA BENE.
Ogni chiamata del comando "`simulate i0 i1 i2 …`" fa avanzare il clock di un periodo. E' possibile eseguire una sequenza di comandi "`simulate i0 i1 i2 …`" salvati su un file richiamando tale file con il comando "`source nomefile`".

Esercizi

Esercizio 1: Realizzare in formato blif il seguente datapath. A e B sono due numeri positivi da 2 bit. Se SEL=0 allora OUT fornisce il valore della somma tra A e B; se SEL= 1 allora OUT fornisce il valore della differenza tra A e B.



Esercizio 2: Realizzare in formato blif la FSMD del circuito che controlla l'apertura di una cassaforte.

Gli ingressi del circuito sono:

- APRI[1]: vale 1 quando viene premuto il bottone *invio* per aprire la cassaforte. In tal caso il circuito deve controllare che il valore di NUM sia uguale a "0110" per aprire la cassaforte.
- NUM[4]: è il codice a 4 bit della combinazione.

Le uscite del circuito sono:

- APERTA[1]: viene impostato a 1 quando viene premuto il bottone *invio* e NUM=0110. Viene impostato a 0 quando viene premuto il bottone *chiudi* per richiudere la cassaforte oppure dopo che sono trascorsi 4 cicli di clock da quando la cassaforte è stata aperta.