Implementarea mașinilor cu stări finite în Verilog

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

31 octombrie 2024

Introducere

Obiective:

- Descrierea mașinilor cu stări finite în Verilog

De citit:

- Chris Fletcher: "EECS150: Finite State Machines in Verilog", [Flet08]

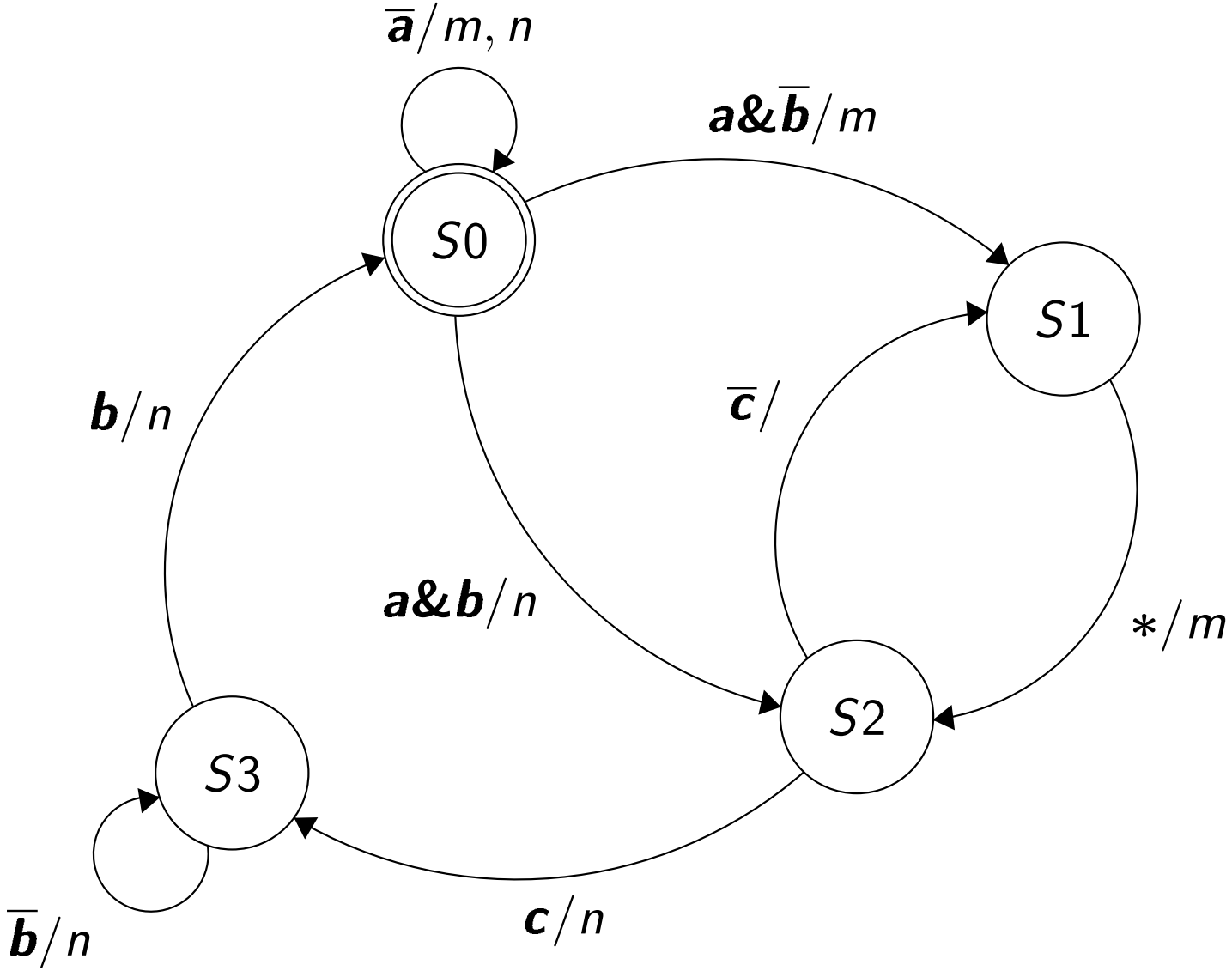
În acest material va fi prezentată o metodă de descriere comportamentală a *mașinilor cu stări finite* de tip Mealy utilizând limbajul Verilog, aceasta putând fi adaptată facil pentru automate cu stări finite de tip Moore.

Desrierea comportamentală în limbajul Verilog urmărește utilizarea setului sintetizabil al limbajului, permițând implementarea directă prin unelte de sinteză a soluției construite.

Studiu de caz

Implementarea unei mașini Mealy descrisă prin diagrama tranzițiilor de stare

Exercițiu: Să se implementeze următoarea mașină cu stări finite:



Studiu de caz (contin.)

Implementarea unei mașini Mealy descrisă prin diagrama tranzițiilor de stare

În urma *analizei* diagramei tranzițiilor de stare se pot observa următoarele:

- mașina are 3 intrări de 1 bit: *a*, *b* și *c*
- la ieșire, automatul poate activa oricare din cele 2 ieșiri de 1 bit: *m* și *n*
- sistemul se poate afla în oricare din cele 4 stări: *S0*, *S1*, *S2* sau *S3*
- starea *S0* este starea inițială (activată după inițializarea dispozitivului)

O tranziție între două stări ale mașinii este simbolizată printr-un arc, etichetat cu condiția logică de declanșare a tranziției împreună cu ieșirile activate de tranziție.

Formatul etichetei arcului de tranziție:

condiție_logică/ieșire_activată, ...

Implementarea mașinilor cu stări finite în cinci pași

Pasul 1

Se vor defini constante cu nume pentru fiecare stare a mașinii. Constantele sunt declarate utilizând cuvântul rezervat `localparam`. Fiecăre constantă de stare are o valoare distinctă, valoare reprezentată pe numărul necesar de biți (pentru o mașină având 13 stări, valorile constantelor de stare sunt reprezentate pe 4 biți)

Pentru exercițul propus, aceste constante de stare sunt definite ca în fragmentul de mai jos:

```
1 localparam S0_ST = 2'd0;
2 localparam S1_ST = 2'd1;
3 localparam S2_ST = 2'd2;
4 localparam S3_ST = 2'd3;
```

Numele celor 4 constante de stare includ sufixul `_ST`.

Implementarea mașinilor cu stări finite în 5 pași (contin.)

Pasul 2

Se definesc semnalele pentru starea curentă și pentru starea următoare. Semnalul de stare curentă, *st*, primește valori într-un bloc `always` fiind declarat de tip `reg`, pe același număr de biți folosit la reprezentarea constantelor de stare.

Semnalul pentru starea următoare, *st_nxt*, pe același număr de biți ca *st*, este generat de o logică combinațională, dependent de starea curentă *st* și de intrările automatului. Dacă *st_nxt* este generat într-un bloc `always`, el va fi declarat de tip `reg`.

Pentru problema de rezolvat, cele 2 semnale sunt definte astfel:

```
1  reg [1:0] st;
2  reg [1:0] st_nxt;
```

Implementarea mașinilor cu stări finite în 5 pași (contin.)

Pasul 3

Din diagrama tranzițiilor de stare se construiește starea următoare într-un bloc `always` combinațional. Se poate folosi instrucțiunea `case(st)` pentru tratarea tranzițiilor asociate fiecărei stări:

```
1  always @ (*)
2          case(st)
3                  S0_ST: if (!a) st_nxt = S0_ST;
4                                 else if (b) st_nxt = S2_ST;
5                                 else st_nxt = S1_ST;
6                  S1_ST: st_nxt = S2_ST;
7                  S2_ST: if (c) st_nxt = S3_ST;
8                                 else st_nxt = S1_ST;
9                  S3_ST: if (b) st_nxt = S0_ST;
10                                else st_nxt = S3_ST;
11         endcase
```

Important: Vor fi tratate prin instrucțiunile `if ... else` toate condițiile logice de intrare asociate fiecărei stări (nu vor rămâne cazuri de configurații de intrare netratate prin cod Verilog).

Implementarea mașinilor cu stări finite în 5 pași

Pasul 4

Din diagrama tranzițiilor de stare se construiesc ieșirile mașini printr-un bloc `always` combinațional. Folosind instrucțiunea `case(st)`, sunt activate ieșirile asociate fiecărei tranziții de stare:

```
1  always @ (*) begin
2          m = 1'd0;
3          n = 1'd0;
4          case(st)
5                  S0_ST: if (!a) {m, n} = 2'b11;
6                                 else if (b) n = 1'd1;
7                                 else m = 1'd1;
8                  S1_ST: m = 1'd1;
9                  S2_ST: if (c) n = 1'd1;
10                 S3_ST: n = 1'd1;
11         endcase end
```

Important: Pentru a nu include în `if ... else` cazurile când una sau mai multe ieșiri devin 0 (de ex. ramura *else* din linia 9), ieșirile sunt mai întâi inițializate la valoarea implicită (linia 2 și 3).

Implementarea mașinilor cu stări finite în 5 pași (contin.)

Pasul 5

Se actualizează starea curentă într-un bloc `always` secvențial. La fiecare front declanșator al semnalului de tact, semnalul de stare următoare devine starea curentă. De asemenea, activarea intrării (asincrone, în acest caz) de reset declanșază aducerea automatului în starea inițială.

Cu mici modificări, codul de mai jos poate fi folosit pentru implementarea oricărei mașini cu stări finite, cu condiția respectării etapelor anterioare:

```
1  always @ (posedge clk, negedge rst_b)
2    if (!rst_b)    st <= S0_ST;
3    else           st <= st_nxt;
```

Cod Verilog complet

```
1   module fsm (
2           input clk, rst_b,
3           input a, b, c,
4           output reg m, n
5   );
6           localparam S0_ST = 2'd0;
7           localparam S1_ST = 2'd1;
8           localparam S2_ST = 2'd2;
9           localparam S3_ST = 2'd3;
10          reg [1:0] st;
11          reg [1:0] st_nxt;
12          always @ (*)
13                  case(st)
14                          S0_ST:  if (!a) st_nxt = S0_ST;
15                                  else if (b) st_nxt = S2_ST;
16                                  else st_nxt = S1_ST;
17                          S1_ST:  st_nxt = S2_ST;
18                          S2_ST:  if (c) st_nxt = S3_ST;
19                                  else st_nxt = S1_ST;
20                          S3_ST:  if (b) st_nxt = S0_ST;
21                                  else st_nxt = S3_ST;
22                  endcase
23          always @ (*) begin
24                  m = 1'd0;
25                  n = 1'd0;
26                  case(st)
27                          S0_ST:  if (!a) {m, n} = 2'b11;
28                                  else if (b) n = 1'd1;
29                                  else m = 1'd1;
30                          S1_ST:  m = 1'd1;
31                          S2_ST:  if (c) n = 1'd1;
32                          S3_ST:  n = 1'd1;
33                  endcase end
34          always @ (posedge clk, negedge rst_b)
35                  if (! rst_b) st <= S0_ST;
36                  else st <= st_nxt;
37  endmodule
```

Referințe bibliografice

[Flet08] C. Fletcher. EECS150: Finite State Machines in Verilog. [Online]. Available: http://inst.eecs.berkeley.edu/~cs150/fa08/Documents/FSM.pdf (Last accessed 25/04/2016).