Lucrare de laborator nr. 10
Modelarea maşinilor cu stări finite în VHDL

1. Scopul lucrării

Însuşirea principiilor şi tehnicilor de modelare a circuitelor secvențiale de tip maşină cu stări finite în VHDL.

2. Modelarea maşinilor cu stări finite (FSM)

În domeniul circuitelor digitale, în majoritatea cazurilor, proiectanții trebuie să proiecteze circuite care efectuează anumite secvențe specifice de operații, de exemplu, controlere utilizate pentru comanda funcționării altor circuite. Maşinile cu stări finite reprezintă o metodă foarte eficientă pentru modelarea circuitelor secvențiale. Prin modelarea maşinilor cu stări finite într-un limbaj de descriere hardware şi utilizarea programelor de sinteză proiectanții se pot concentra doar asupra modelării secvenței dorite de operații fără a-şi pune probleme în privința implementării circuitului. Această opreație este lăsată programelor de sinteză.

Înainte de a scrie un model trebuie luate în considerație diferitele aspecte ale maşinilor cu stări finite. Un model bun este esențial pentru a obține un circuit corect funcțional care să îndeplinească în mod optim cerințele de proiectare. De aceea, este important să se înțeleagă principiile maşinilor cu stări finite şi să se cunoască aspectele referitoare la modelarea VHDL a acestora.

2.1 Maşini cu stări finite. Generalități.

O maşină cu stări finite (*Finite State Machine* – FSM), aşa cum indică numele, este un circuit secvențial cu mai multe stări interne. În comparație cu circuitele secvențiale obişnuite (numărătoare, regiştri de deplasare, etc), tranzițiile dintr-o stare în alta şi secvența de evenimente este mult mai complicată. Deşi diagrama bloc de bază a unui FSM este similară cu cea a unui circuit secvențial obişnuit, procedura de proiectare este diferită. Astfel, modelul unei maşini cu stări finite se construieşte plecând de la un model mult mai abstract, cum ar fi o diagramă de stări, care descrie în format grafic interacțiunile şi tranzițiile dintre stările interne.

Tipuri de maşini cu stări finite

Formal, o maşină cu stări finite este specificată de 5 elemente: stările simbolice, semnalele de intrare, semnalele de ieşire, funcția pentru stabilirea stării următoare şi funcția pentru stabilirea valorilor de la ieşire.

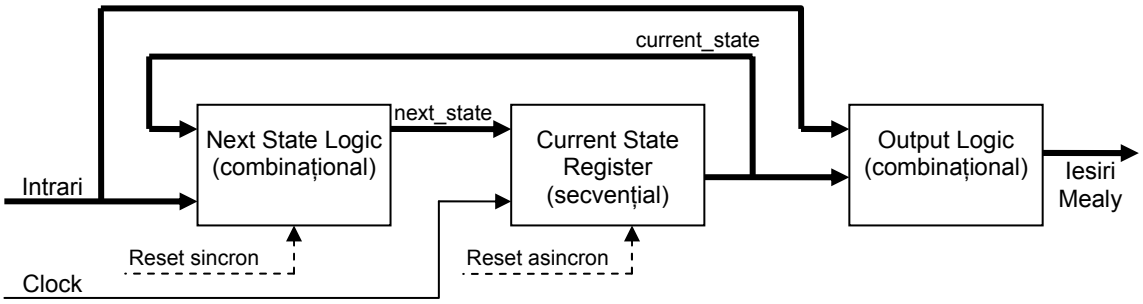
După tipul funcției pentru stabilirea valorilor de la ieşire, maşinile cu stări finite se împart în două categorii:

- *FSM cu ieşiri Moore* – în acest caz semnalele de la ieşire sunt o funcție doar de starea curentă a maşinii;
- *FSM cu ieşiri Mealy* – în acest caz semnalele de la ieşire sunt o funcție atât de starea curentă cât şi de semnalele de la intrare.

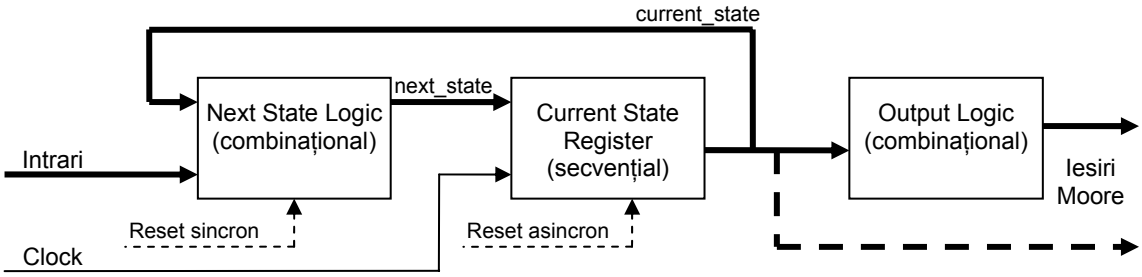
În figura 1 sunt prezentate diagramele bloc ale FSM cu ieşiri Mealy, respectiv Moore. În fiecare diagramă bloc se disting trei părți:

1. *Current State Register* – registru stare curentă. Este un bloc secvențial care constă din unul sau mai multe bistabile (flip-flop) utilizate pentru a reține („memora”) starea curentă a FSM.

2. *Next State Logic* – bloc logic combinațional utilizat pentru a genera următoarea stare (*next_state*) a FSM. Ieşirea *next_state* acestui bloc este o funcție de intrările FSM şi de starea curentă *current_state*.
3. *Output Logic* – bloc logic combinațional utilizat pentru a genera semnalele de la ieşire. În cazul maşinii Moore acest bloc poate lipsi, astfel că ieşirea corespunde stării curente a FSM.



Mealy



Moore

Figura 1 – *Diagramele bloc ale maşinilor cu stări finite cu ieşiri Mealy şi Moore*.

Observație: O maşină cu stări finite poate avea simultan şi ieşiri de ambele tipuri, Mealy şi Moore. În acest caz maşina va conține două blocuri combinaționale *Output Logic*: un bloc pentru ieşirea Mealy şi un alt bloc pentru ieşirea Moore.

Tabelul şi Diagrama de stări

O diagramă de stări este o reprezentare grafică a funcționării secvențiale a maşinii cu stări finite. Diagrama de stări constă din *noduri*, reprezentate ca cercuri, şi *arce* de tranziții unidirecționale. Un nod reprezintă o stare unică a FSM şi are asociat un nume simbolic unic. Un arc reprezintă o tranziție dintr-o stare în altă stare şi este insoțit, dacă e cazul, de condiția care determină tranziția. Condiția este o expresie logică compusă din semnalele de la intrare. O tranziție este executată la fiecare ciclu al semnalului de ceas dacă expresia logică asociată arcului are valoarea '1' logic. De reținut că într-o diagramă de stări semnalul de ceas nu este reprezentat explicit.

De asemenea, înt-o diagramă de stări sunt specificate şi valorile de la ieşire. În cazul unei ieşiri Moore, aceasta fiind o funcție de starea curentă, este plasată în interiorul sau alături de cercurile corespunzătoare stărilor. În cazul ieşirii Mealy, deoarece depinde atât de starea curentă cât şi de intrări, valoarea este plasată după expresia intrărilor asociată arcului tranziției, separat de expresie prin semnul slash ('/').

Mai jos sunt prezentate, ca exemplu, un tabel de stări şi două diagrame de stări echivalente asociate unei maşini cu 5 stări.

Tabelul 1 – *tabel de stări pentru un FSM cu 5 stări*

Intrări		Current state	Next state	Ieşiri	
A	B			Y_Me	Y_Mo
0	X	000 (ST0)	000 (ST0)	1	0
1	X	000 (ST0)	001 (ST1)	0	0
0	X	001 (ST1)	000 (ST0)	0	1
1	X	001 (ST1)	010 (ST2)	1	1
X	X	010 (ST2)	011 (ST3)	0	0
X	1	011 (ST3)	011 (ST3)	1	1
0	0	011 (ST3)	000 (ST0)	1	1
1	0	011 (ST3)	100 (ST4)	0	1
X	X	100 (ST4)	000 (ST0)	0	1

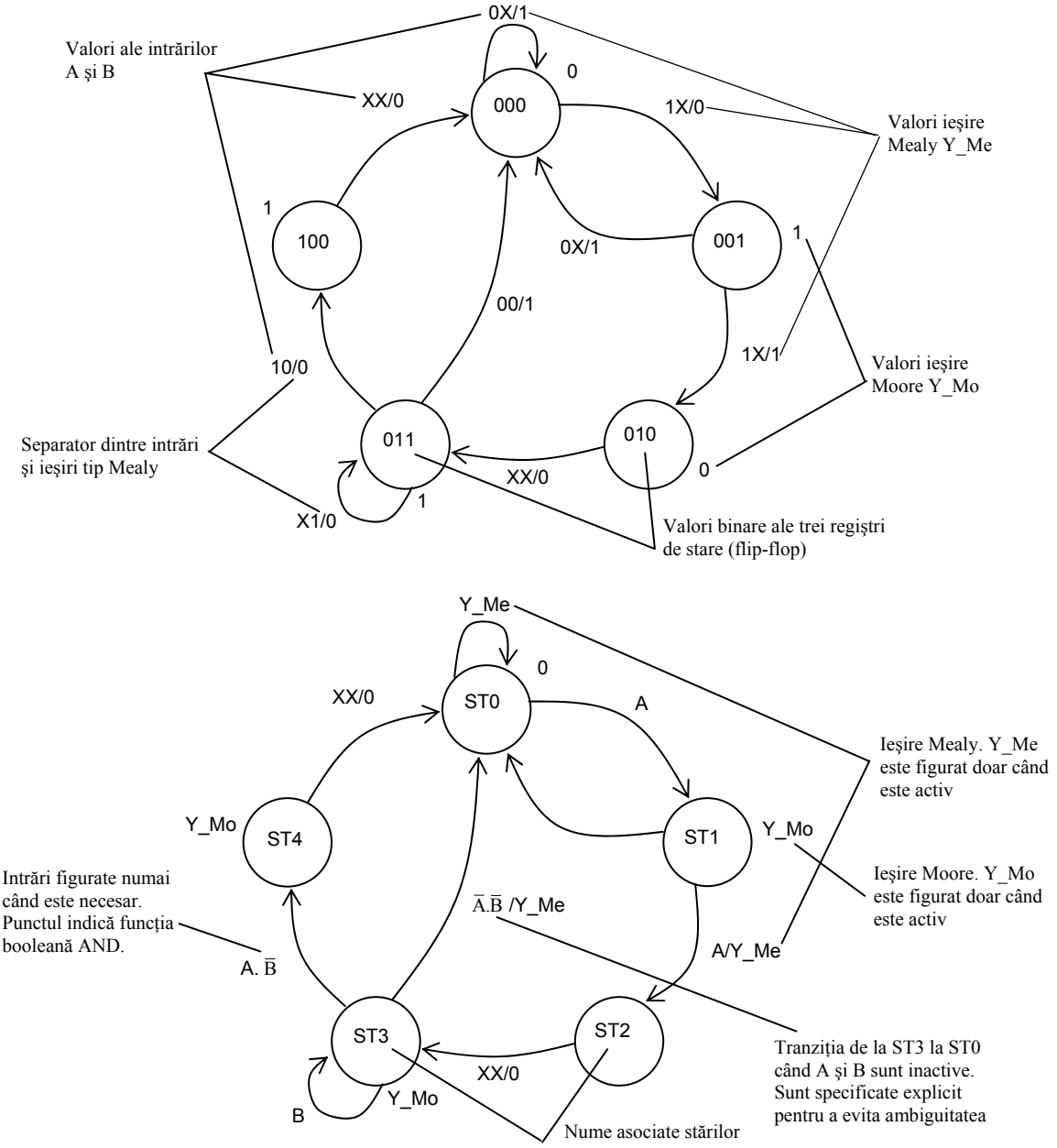


Figura 2 – *Două diagrame de stări echivalente pentru un FSM cu 5 stări*

În prima diagramă stările sunt reprezentate binar în timp ce în a doua diagramă stările sunt reprezentate prin nume. Condițiile asupra valorilor semnalelor de la intrare A şi B care

determină o tranziție sunt specificate în dreptul unui arc înaintea semnului '/'. Valorile specificate după semnul '/', dacă există, indică valorile semnalului de la ieşire care este o funcție de valorile la intrări şi de starea curentă (ieşire Mealy). Diferența majoră în cadrul celei de-a doua diagrame de stări este aceea că semnalele de la intrări şi ieşiri sunt figurate numai când sunt active, altfel ele nu mai sunt prezente în diagramă.

2.2 Stiluri de codare în VHDL a maşinilor cu stări finite

Există trei modalități diferite pentru modelarea în VHDL a aceleiaşi maşini cu stări finite. Astfel, o primă posibilitate este aceea în care codul VHDL este structurat în trei părți separate reprezentând, corespunzător, cele trei blocuri ale unui FSM (blocul current state, blocul next state şi blocul de ieşire). Pe de altă parte, codul aferent blocurilor poate fi combinat în două părți sau, mai compact, într-o singură parte de cod. Oricum, stilul de codare este independent de tipul maşinii cu stări finite.

Astfel, pentru o maşină cu stări finite pot rezulta patru arhitecturi diferite după cum urmează:

- părți de cod separate pentru blocurile current-state (CS), next-state (NS) şi output logic (OL);
- părți de cod combinat pentru CS şi NS şi cod separat pentru OL;
- părți de cod combinat pentru NS şi OL şi cod separat pentru CS
- o singură parte de cod combinat pentru toate blocurile CS, NS şi OL

Mai jos se prezintă cele patru variante de implementare în cod VHDL pentru o maşină cu stări finite a cărei diagramă de stări este prezentată în figura 3.

Din diagramă se observă că maşina cu stări finite este cu ieşire tip Moore. Maşina este prevăzută cu intrare de reset asincron (Reset) iar Ctrl este semnal de intrare de date. De asemenea, se observă că ieşirea acestui FSM este un semnal pe 3 biți. Vom nota cu Y portul de la ieşirea circuitului.

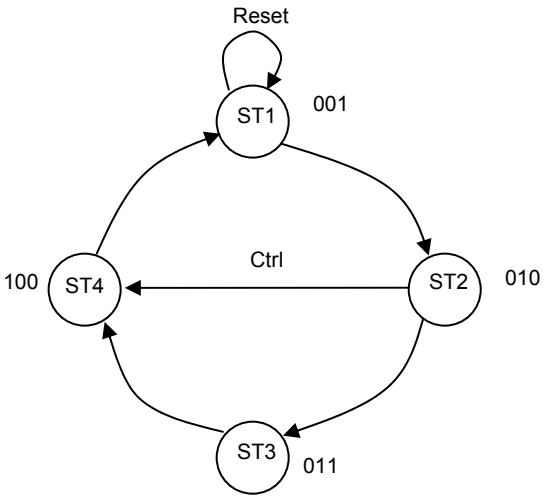


Figura 3 – Diagrama de stări pentru o maşină Moore cu patru stări

În coloana din stânga este prezentat modelul VHDL al FSM în care există cod separat pentru fiecare din blocurile NS, VS şi OL.

În coloana din dreapta este prezentată implementarea VHDL a FSM cu cod combinat pentru blocurile CS şi NS şi cod separat pentru blocul OL.

```vhdl
-- FSM cu cod separat pentru blocurile
-- NS, CS si OL

library ieee;
use ieee.std_logic_1164.all;

entity FSM_A is
   port(Clk, Reset: in std_logic;
     Ctrl: in std_logic;
     Y: out std_logic_vector(2 downto 0));
end entity FSM_A;

architecture RTL of FSM_A is
  type StateType is (ST1, ST2, ST3, ST4);
  signal CurrentState,NextState: StateType;
begin
  -- blocul next state (NS)
  COMB_NS: process (Ctrl, CurrentState)
  begin
    case CurrentState is
      when ST1 =>
        NextState <= ST2;
      when ST2 =>
        if (Ctrl = '1') then
          NextState <= ST4;
       else
         NextState <= ST3;
       end if;
      when ST3 =>
        NextState <= ST4;
      when ST4 =>
        NextState <= ST1;
      when others =>
        NextState <= ST1;
    end case;
  end process COMB_NS;

  -- blocul current state (CS)
  SEQ_CS: process (Clk, Reset)
  begin
    if (Reset = '1') then
      CurrentState <= ST1;
    elsif Clk'event and Clk='1' then
      CurrentState <= NextState;
    end if;
  end process SEQ_CS;

  -- blocul output logic (OL)
  with CurrentState select
    Y <= "001" when ST1,
         "010" when ST2,
         "011" when ST3,
         "100" when ST4,
         "001" when others;
end architecture RTL;
```

```vhdl
-- FSM cu cod combinat pentru blocurile
-- CS si NS si separat pentru OL

library ieee;
use ieee.std_logic_1164.all;

entity FSM_B is
   port(Clk, Reset: in std_logic;
     Ctrl: in std_logic;
     Y: out std_logic_vector(2 downto 0));
end entity FSM_B;

architecture RTL of FSM_B is
  type StateType is (ST1, ST2, ST3, ST4);
  signal State: StateType;
begin

  -- blocurile NS si CS
  NS_CS: process (Clk, Reset)
  begin
    if (Reset = '1') then
      State <= ST1;
    elsif Clk'event and Clk='1' then
      case State is
         when ST1 =>
            State <= ST2;
         when ST2 =>
            if (Ctrl = '1') then
               State <= ST4;
           else
              State <= ST3;
           end if;
         when ST3 =>
            State <= ST4;
         when ST4 =>
            State <= ST1;
         when others =>
            null;
      end case;
    end if;
  end process NS_CS;

  -- blocul OL
  with State select
    Y <= "001" when ST1,
         "010" when ST2,
         "011" when ST3,
         "100" when ST4,
         "001" when others;
end architecture RTL;
```

În coloana din stânga este prezentată a treia modalitate de implementare VHDL a FSM în care s-a utilizat cod combinat pentru blocurile NS şi OL (ambele blocuri fiind combinaționale) şi cod separat pentru blocul CS.

În coloana din dreapta este prezentată implementarea VHDL a FSM în care toate cele trei blocuri, NC, NS şi OL, sunt descrise într-un singur proces.

```
-- FSM cu cod combinat pentru blocurile
-- NS si OL si separat pentru CS
library ieee;
use ieee.std_logic_1164.all;

entity FSM_C is
   port(Clk, Reset: in std_logic;
     Ctrl: in std_logic;
     Y: out std_logic_vector(2 down to 0));
end entity FSM_C;

architecture RTL of FSM_C is
  type StateType is (ST1, ST2, ST3, ST4);
  signal CurrentState,NextState: StateType;
begin

  -- blocurile NS si OL
  COMB_NS_OL: process (Ctrl, CurrentState)
  begin
    case CurrentState is
      when ST1 =>
        Y<="001";
       NextState <= ST2;
      when ST2 =>
        Y<="010";
        if (Ctrl = '1') then
          NextState <= ST4;
       else
         NextState <= ST3;
       end if;
      when ST3 =>
        Y<="011";
        NextState <= ST4;
      when ST4 =>
        Y<="100";
        NextState <= ST1;
      when others =>
        Y<="001";
        NextState <= ST1;
    end case;
  end process COMB_NS_OL;

  -- blocul CS
  SEQ_CS: process (Clk, Reset)
  begin
    if (Reset = '1') then
      CurrentState <= ST1;
    elsif Clk'event and Clk='1' then
      CurrentState <= NextState;
    end if;
  end process SEQ_CS;

end architecture RTL;
```

```
-- FSM cu cod combinat pentru toate
-- blocurile NS, CS si OL
library ieee;
use ieee.std_logic_1164.all;

entity FSM_D is
   port(Clk, Reset: in std_logic;
    Ctrl: in std_logic;
    Y: out std_logic_vector(2 down to 0));
end entity FSM_D;

architecture RTL of FSM_D is
begin

  -- blocurile CS, NS si OL
  CS_NS_OL: process (Clk, Reset)
    type StateType is (ST1, ST2, ST3, ST4);
    variable State: StateType;
  begin
    if (Reset='1') then
      State := ST1;
    elsif Clk'event and Clk='1' then
       case State is
          when ST1 =>
              State := ST2;
          when ST2 =>
              if (Ctrl = '1') then
                State := ST4;
             else
                State := ST3;
             end if;
          when ST3 =>
              State := ST4;
          when ST4 =>
              State := ST1;
          when others =>
              State := ST1;
       end case;
     end if;
     case State is
        when ST1 =>   Y<="001";
        when ST2 =>   Y<="010";
        when ST3 =>   Y<="011";
        when ST4 =>   Y<="100";
        when others=> Y<="001";
     end case;
  end process CS_NS_OL;

end architecture RTL;
```

3. Modelarea VHDL al unui controler descris ca o maşină cu stări finite.

3.1 Specificații

Mai jos se prezintă modelul VHDL a unui circuit de comandă (controler) descris ca o maşină cu stări finite şi care poate fi utilizat în cadrul unui circuit de multiplicare cu întârziere-adunare. Circuitul funcționează conform descrierii următoare:

Stare	Acțiune	Starea următoare
End	if STB	-> Init:
Init	load SRA,SRB	-> Check:
	Clear ACC	
Check:	if Stop	-> End:
	If not Stop and LSB=0	-> Shift:
	If LSB=1	-> Add:
Add:	load ACC	-> Shift:
Shift:	shift SRA,SRB	-> Check:

Diagrama de stare a controlerului este prezentată în figura 3. Diagrama corespunde unei maşini cu ieşire Moore (iesirile depind doar de starile acestuia).

Multiplicatorul cu întîrziere-adunare în care va fi utilizat controlerul acceptă două intrări *A, B* pentru numerele supuse multiplicării şi conține un acumulator care este inițializat cu zero.

Atât timp cât nu toți biții intrării A au valoarea zero sînt executați următorii paşi:

1. dacă bitul cel mai puțin semnificativ al lui *A* este diferit de zero, *B* este adunat în acumulator.
2. A:=A div 2 şi B:=B*2 prin deplasarea lui A cu o poziție la dreapta şi a lui B cu o poziție la stânga.

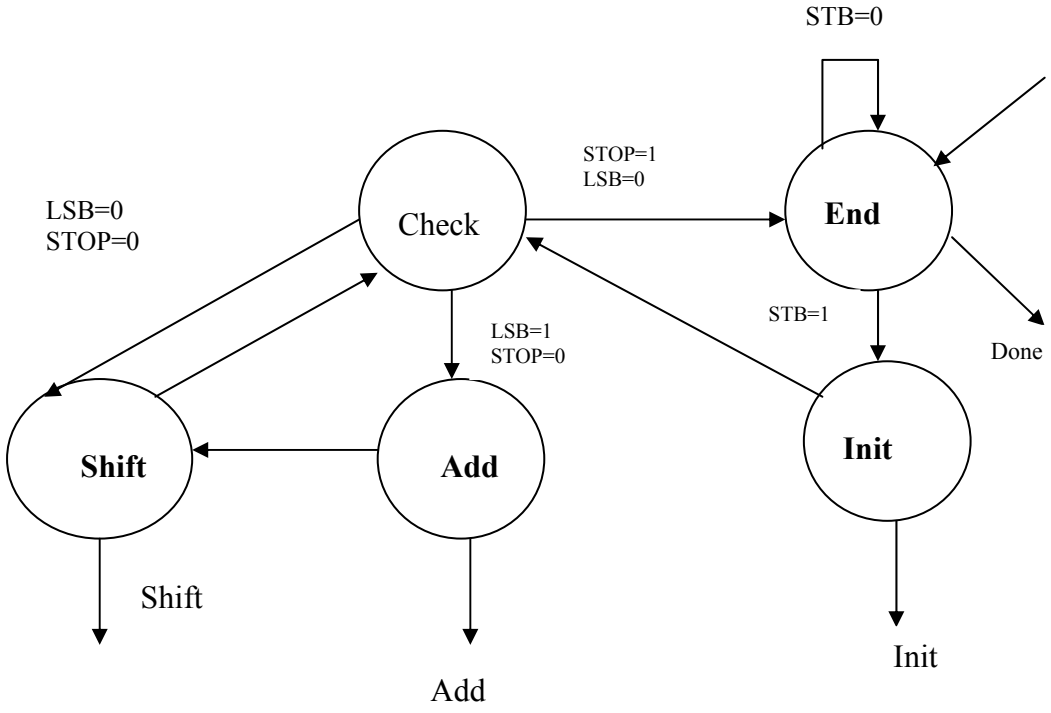


Fig.3 *Diagrama de stare a controlerului*

Cînd A=0 acumulatorul va conține produsul A*B.

Pentru un astfel de sumator avem nevoie de un automat pentru controlul functionarii. Acesta are trei intrări de date:

- STB: semnal care strobează intrarile A şi B şi inițializează multiplicatorul.
- LSB: contine bitul cel mai puțin semnificativ al lui A .
- STOP: semnal activ cind toți biții intrarii A sînt zero.

şi patru ieşiri:

- INIT: inițializează acumulatorul.
- SHIFT: deplasează A cu o poziție la dreapta şi pe B cu o poziție la stînga.
- ADD: adună valoarea curentă a lui B în acumulator.
- DONE: indică terminarea execuției operației.

Controlerul este inițial în starea “End”, stare în care este emis semnalul DONE. Când semnalul STB este activ se intră în starea “Init” care emite semnalul INIT pentru încărcarea regiştrilor A, B şi inițializarea acumulatorului.Trecem apoi în starea Check.

Dacă în starea “Check” semnalul STOP este în 1 logic (LSB=’0’) atunci se revine în starea “Init”. Altfel, dacă *LSB=’0’* se trece în starea “Shift” iar dacă LSB =’1’ se trece în starea Add.

Dacă sîntem în starea “Add” este emis semnalul ADD care determină încărcarea conținutului registrului în acumulator. Se trece apoi în starea “Shift”.

In starea „Shift” este emis semnalul SHIFT care produce deplasarea registrului *A* cu o poziție la dreapta şi a lui *B* cu o poziție la stânga. Se trece apoi în starea “Check”.

3.2 Declarația de entitate şi arhitectura modelului (fisier FSM.vhd)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;

3  entity Controller is
4     port ( STB, CLK, LSB, Stop : in Bit ;
5              Init, Shift, Add , done : out Bit );
6  end Controller;

10 architecture FSM of Controller is
12    type States  is ( InitS, Checks, AddS, ShiftS, EndS );
13    signal State : StateS := EndS ;

15 begin
16    -- parte combinationala (output logic)
19        Init <= '1' when State = InitS else '0' ;
20        Add <= '1' when State = Adds else '0' ;
21        Shift <= '1' when State = ShiftS else '0' ;
22        Done <= '1' when State = EndS else '0' ;
24    -- parte current state si next state
26 StateMachine :
28    process (CLK)
29    begin
30          if CLK'Event and CLK = '0' then
31                case State is
32                      when Inits =>
33                            State <= Checks;
34                      when Checks  =>
35                            if LSB = '1' then
36                                  State <=  AddS ;
37                            elsif Stop = '0' then
38                                  State <= ShiftS;
```

```
39                              else
40                                    State <= EndS;
41                              end if;
42                        when Adds =>
43                              State <= Shifts;
44                        when ShiftS =>
45                              State <= Checks;
46                        when Ends =>
47                              if STB = '1' then
48                                    State <= InitS ;
49                              end if;
50                end case;
51          end if;
52    end process;
54  end;
```

Corpul arhitecturii este compus din două părți: o parte cu instrucțiuni concurente de atribuire condițională care corespunde blocului Output Logic din schema bloc a maşinii Moore şi o altă parte reprezentată de instrucțiunea process care cuprinde descrierea combinată a celorlalte două blocuri: *blocul next*_state şi current_state. În cadrul arhitecturii este declarat şi utilizat tipul enumerat *States* pentru reprezentarea stărilor posibile ale controlerului. Un tip enumerat este un tip ale cărui posibile valori sînt descrise ca identificatori sau şiruri de caractere.

De exemplu, tipul predefinit *"Bit"* este decris ca un tip enumerat. Declarația tipului enumerat la linia 12 cuprinde următoarele stări: *InitS, Checks, AddS, ShiftS, EndS* . Semnalul *State* reprezintă starea registrului si este inițializat cu starea *EndS*.

Liniile 19-22 introduc un nou tip de specificatii de atribuire concurențiale : specificatii de atribuire concurente condiționale pentru a comanda cele patru ieşiri în funcție de starea curentă a maşinii. De exemplu, la linia 19 ori de cîte ori semnalul *State* este *InitS* semnalul *Init* este '1', altfel este '0'. Aceste specificatii de atribuire – ca şi cele de la liniile 20,21,22 sunt executate ori de cîte ori semnalul *State* ia noi valori.

Tranziția stărilor este implementată de proces in liniile 28-52, care sînt executate pentru fiecare front al semnalului de clock.

Specificatia *if* de la linia 30 filtrează fronturile crescătoare ale semnalului de clock. În consecință, corpul specificatiei *if* –care reprezintă restul procesului– este executat doar pe frontul căzător al semnalului de *clock*.

Pe frontul căzător al semnalului de clock specificatia *case* (linia 31) determină starea curentă şi este parcursă ramificația corespunzătoare.

Apoi, restul intrărilor în maşină sînt examinate pentru găsirea starii următoare corespunzătoare.

3.3 Testarea modelului VHDL al controlerului

Programul de test pentru automat este prezentat in continuare:

(fisier Test_FSM.vhd)

```
56  library IEEE;
57  use IEEE.std_logic_1164.all;
58  entity Test_Controller is end;

62  architecture Driver of Test_Controller is
64    component Controller
65          port ( STB, CLK, LSB, Stop : in Bit;
66                      Init, Shift, Add, Done : out Bit );
67    end component;
```

```
69    signal STB, LSB, Done , Init, Shift, Add, Stop : Bit;
70    signal CLK, ENA : Bit :='1' ;
72  begin
74    UUT : Controller port map
75          ( STB, CLK, LSB, Stop, Init, Shift, Add, Done );
77    Clock : CLK <= not (CLK and ENA ) after 10 ns;
78    Enable : ENA <= '0' after 500  ns;

80    Stimulus : process
82          variable Val : Bit_Vector ( 7 downto 0) ;
83    begin
84          wait until CLK'Event and CLK='1' ;
85          STB <= '1', '0' after 20 ns;
86          Val := "01110101" ;
87          loop
88                LSB <= Val(0) ;
89       Stop <= not  ( Val (7) or Val (6) or Val (5) or Val (4) or
                Val(3)  or Val (2) or Val(1) or Val (0) );
91                wait until CLK'Event and CLK='1' ;
92                exit when Done = '1' ;
93                if Shift = '1' then
94                      Val := '0' & Val ( 7 downto 1);
95                end if;
96          end loop;
97    wait;
98    end process;
100 end;
```

Liniile 77-78 implementeaza un circuit de ceas care ruleaza de la 0 la 500ns, cu perioada de 20 ns si factor de umplere de 50%.

Generatorul de stimuli (liniile 80-98) emuleaza caile de date ale multiplicatorului pentru a putea testa controlerul. Generatorul utilizeaza doua instrutiuni *wait* (liniile 84,91) pentru a se sincroniza cu semnalul de ceas.

Specificatiile *wait* prezinta clauza *until* urmate de o expresie logica.

Clauza *until* va influenta comportamentul specificatiei *wait* dupa cum urmeaza. Procesul ce contine specificatia *wait* este sensitiv la toate semnalele continute de clauza *until* (in acest caz *CLK*) atita timp cit specificatia *wait* este activa. Ori de cite ori apare o modificare in valoarea semnalului se evalueaza expresia.

Daca este adevarata, *wait* ia sfirsit, iar procesul isi continua executia. Daca expresia este falsa, *wait* suspenda procesul pina cind va apare un alt eveniment la care acesta este sensitiv.

Specificatiile *wait* sesizeaza fronturile crescatoare ale semnalului de ceas.

A doua specificatie *wait* este inclusa intr-o bucla infinita (liniile 87-96). Bucla este totusi controlata de o specificatie *exit* ce prezinta o expresie logica *"Done = '1' "* ce urmeaza clauzei *when***. I**n lipsa acesteia se iese neconditionat din bucla.

3.4 Aplicații

1. Copiați fişierele sursă FSM.vhd şi Test_FSM.vhd în directorul de lucru
2. Compilați fişierele sursă, efectuați simularea entității de test Test_Controller şi urmăriți cu atenție formele de undă rezultate pentru a decide dacă acestea sunt în concordanță cu specificațiile.
3. În fişierul FSM.vhd, creați o nouă arhitectură numită FSMc pentru entitatea Controller în care toate cele trei blocuri specifice unui FSM să fie cuprinse într-un singur segment de cod (într-un singur proces).
   În arhitectura Driver a entității Test_Controller, introduceți încă o instanță (cu eticheta UUTc) a componentei Controller şi adăugați în partea declarativă 2 linii

de configurare pentru instanța existentă (UUT) şi pentru noua instanță (UUTc) astfel încât să efectuați simularea controlerului în paralel cu ambele arhitecturi: FSM şi FSMc. În liniile de configurare instanței UUT i se va asocia perechea entitate-arhitectură: Controller-FSM, în timp ce instanței UUTc i se va asocia perechea Controller-FSMc

Observație: noua instanță UUTc a componentei Controller se va conecta la aceleaşi semnale de intrare ca şi instanța UUT, în schimb porturile de ieşire se vor conecta la semnale denumite `Initc, Shiftc, Addc, Donec` . Aceste semnale trebuie declarate în partea declarativă a arhitecturii.

Recompilați fişierele sursă şi repetați simularea simularea entității de test. Urmăriți dacă ieşirile corespondente de la cele două instanțe concid.

4. În fişierul FSM.vhd creați încă o arhitectură numită FSM_sep în care pentru cele trei blocuri specifice maşinii cu stări finite să existe cîte o parte separată de cod VHDL.

   Odată definită această nouă arhitetură, în arhitectura Driver a entitatii Test_Controller introduceți încă o instanță cu eticeta UUTsep a componentei Controller care va avea porturile de ieşire conectate la semnalele `Init_sep, Shift_sep, Add_sep, Done_sep` . Procedând similar ca la punctul 3, recompilați fişierele sursă, repetați simularea entității Test_Controller şi urmăriți dacă formele de undă de la ieşirile corespondente ale celor 3 instanțe sunt identice.