

```
module OR(A, B, C);  
input A, B;  
output C;  
assign C = A | B;  
endmodule
```

```
module AND(A, B, C);  
input A, B;  
output C;  
assign C = A & B;  
endmodule
```

```
module MUX(Sel, I0, I1, E);  
input Sel, I0, I1;  
output E;  
wire temp1;  
wire temp2;  
AND a1(I1, Sel, temp1);  
AND a0(I0, ~Sel, temp2);  
OR o1(temp1, temp2, E);  
endmodule
```

```
module MUX(Sel, I0, I1, E);  
input Sel, I0, I1;  
output E;  
assign E = (I1 & Sel) | (I0 & ~Sel);  
endmodule
```

```
module main();  
reg _sel, _i0, _i1;  
wire _e;  
MUX m(_sel, _i0, _i1, _e);  
initial  
begin  
    $monitor("%d", _e);  
    _i0 = 1;  
    _i1 = 0;  
    _sel = 0;  
end  
endmodule
```

```

module MUL(Clk, A, B, start, R, ack);
input Clk, start;
input [7:0]A,B;
output reg ack;
output reg[15:0]R;
reg [7:0]X,Y;
reg [1:0]stare;
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                Y <= B;
                R <= 0;
                ack <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare==1)
        if(Y==0)
            begin
                ack <= 1;
                stare <= 2;
            end
        else
            begin
                R <= R + X;
                Y <= Y - 1;
                stare <= 1;
            end
    end
    else if(stare==2)
        stare <= 0;
initial begin
    stare <= 0;
    ack <= 0;
end
endmodule

```

```

module SQRT(Clk, A, start, R, ack, mull);
input Clk, start;
input [7:0]A;
output reg ack;
output reg[7:0]R;
output reg[15:0]mull;
reg [7:0]X;
wire [15:0]n;
wire ack2;
reg start2;

```

```

reg [3:0]stare;
MUL m(Clk, R, R, start2, n, ack2);
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                R <= 0;
                ack <= 0;
                start2 <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare == 1)
        begin
            if(n > X)
                begin
                    stare <= 3;
                end
            else
                begin
                    start2 <= 1;
                    if((n > 0 && R > 0) || R == 0)
                        R <= R + 1;
                    stare <= 2;
                end
        end
    end
    else if(stare==2)
        begin
            start2 <= 0;
            if(ack2==1)
                begin
                    mull <= n;
                    stare <= 1;
                end
            else
                stare <= 2;
        end
    end
    else if(stare==3)
        begin
            R <= R - 1;
            stare <= 0;
            ack <= 1;
        end
    end
initial begin
    stare <= 0;
    ack <= 0;
end

```

```

endmodule

module ceas(c);
output reg c;
always
    #5 c <= ~c; //semnal de perioada 10
initial
    c <= 0;
endmodule

module main();
reg start;
reg [7:0]A;
wire ack;

```

```

wire [7:0]R;
wire [15:0]mull;
wire Clk;
ceas cc(Clk);
SQRT m(Clk, A, start, R, ack, mull);
initial begin
    $monitor("%d %d %d %d %d %d", $time,
        start, A, R, ack, mull);
    A <= 49;
    #4 start <= 1;
    #10 start <= 0;
    #1000 $finish();
end
endmodule

```

```

module MUL(Clk, A, B, start, R, ack);
input Clk, start;
input [7:0]A,B;
output reg ack;
output reg[15:0]R;
reg [7:0]X,Y;
reg [1:0]stare;
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                Y <= B;
                R <= 0;
                ack <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare==1)
        if(Y==0)
            begin
                ack <= 1;
                stare <= 2;
            end
        else
            begin
                if(Y[0]==1)
                    R <= R + X;
                X <= X<<1;
                Y <= Y>>1;
                stare <= 1;
            end
        else if(stare==2)

```

```

            stare <= 0;
        initial begin
            stare <= 0;
            ack <= 0;
        end
    endmodule

    module ceas(c);
    output reg c;
    always
        #5 c <= ~c; //semnal de perioada 10
    initial
        c <= 0;
    endmodule

    module main();
    reg start;
    reg [7:0]A,B;
    wire ack;
    wire [15:0]R;
    wire Clk;
    ceas cc(Clk);
    MUL m(Clk, A, B, start, R, ack);
    initial begin
        $monitor("%d %d %d %d %d %d", $time,
            start,A, B, R, ack);
        A <= 5;
        B <= 7;
        #4 start <= 1;
        #10 start <= 0;
        #100 $finish();
    end
    endmodule

```

```

module MUL(Clk, A, B, start, R, ack);
input Clk, start;
input [3:0]A,B;
output reg ack;
output reg[7:0]R;
reg [3:0]X,Y;
reg [1:0]stare;
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                Y <= B;
                R <= 0;
                ack <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare==1)
        if(Y==0)
            begin
                ack <= 1;
                stare <= 2;
            end
        else
            begin
                R <= R + X;
                Y <= Y - 1;
                stare <= 1;
            end
    end
else if(stare==2)
    stare <= 0;

```

```

initial begin
    stare <= 0;
    ack <= 0;
end
endmodule

module ceas(c);
output reg c;
always
    #5 c <= ~c; //semnal de perioada 10
initial
    c <= 0;
endmodule

module main();
reg start;
reg [3:0]A,B;
wire ack;
wire [7:0]R;
wire Clk;
ceas cc(Clk);
MUL m(Clk, A, B, start, R, ack);
initial begin
    $monitor("%d %d %d %d %d %d", $time,
start,A, B, R, ack);
    A <= 5;
    B <= 7;
    #4 start <= 1;
    #10 start <= 0;
    #200 $finish();
end
endmodule

```

```

module SQRT(Clk, A, start, R, ack);
input Clk, start;
input [15:0]A;
output reg ack;
output reg[7:0]R;
reg [15:0]X;
reg [3:0]stare;
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                R <= 0;
                ack <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare == 1)
        begin
            if(R * R > X)
                begin
                    stare <= 2;
                end
            else
                begin
                    R <= R + 1;
                    stare <= 1;
                end
        end
    end
    else if(stare==2)
        begin
            stare <= 0;
            R <= R - 1;

```

```

                ack <= 1;
            end
        initial begin
            stare <= 0;
            ack <= 0;
        end
    endmodule

    module ceas(c);
    output reg c;
    always
        #5 c <= ~c; //semnal de perioada 10
    initial
        c <= 0;
    endmodule

    module main();
    reg start;
    reg [15:0]A;
    wire ack;
    wire [7:0]R;
    wire Clk;
    ceas cc(Clk);
    SQRT m(Clk, A, start, R, ack);
    initial begin
        $monitor("%d %d %d %d %d", $time,
start, A, R, ack);
        A <= 300;
        #4 start <= 1;
        #10 start <= 0;
        #200 $finish();
    end
    endmodule

```

```

module MUL(Clk, B, A, start, R, ack);
input Clk, start;
input [63:0]B;
input [31:0]A;
output reg ack;
output reg[63:0]R;
reg [31:0]X,Y;
reg [63:0]stare;
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= B;
                Y <= A;
                R <= 0;
                ack <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare==1)
        if(Y==0)
            begin
                ack <= 1;
                stare <= 2;
            end
        else
            begin
                R <= R + X;
                Y <= Y - 1;
                stare <= 1;
            end
    end
    else if(stare==2)
        stare <= 0;
initial begin
    stare <= 0;
    ack <= 0;
end
endmodule

```

```

module POW(Clk, A, B, start, R, ack);
input Clk, start;
input [31:0]A,B;
output reg ack;
output reg[63:0]R;
reg [31:0]X,Y;
wire [63:0]n;

```

```

wire ack2;
reg start2;
reg [3:0]stare;
MUL m(Clk, R, X, start2, n, ack2);
always @(posedge Clk)
    if(stare==0)
        if(start==1)
            begin
                X <= A;
                Y <= B-1;
                R <= A;
                ack <= 0;
                start2 <= 0;
                stare <= 1;
            end
        else
            stare <= 0;
    else if(stare == 1)
        begin
            if(Y==0)
                stare <= 3;
            else
                begin
                    start2 <= 1;
                    if(ack2==1 || R == 1)
                        begin
                            R <= n;
                            Y <= Y-1;
                        end
                    end
                    stare <= 2;
                end
        end
    end
    else if(stare==2)
        begin
            start2 <= 0;
            if(ack2==1)
                stare <= 1;
            else
                stare <= 2;
        end
    end
    else if(stare==3)
        begin
            stare <= 0;
            ack <= 1;
        end
    end
initial begin
    stare <= 0;

```

```

        ack <= 0;
    end
endmodule

module ceas(c);
output reg c;
always
    #1.5 c <= ~c;
initial
    c <= 0;
endmodule

module main();
reg start;
reg [31:0]A, B;

```

```

wire ack;
wire [63:0]R;
wire Clk;
ceas cc(Clk);
POW m(Clk, A, B, start, R, ack);
initial begin
    $monitor("%d %d %d %d %d", $time,
start, A, B, ack, R);
    A <= 2;
    B <= 8;
    #1 start <= 1;
    #4 start <= 0;
    #200 $finish();
end
endmodule

```


Cursul 2

Circuite secvențiale - utilizare sincronă și asincronă

Sincron - asincron

- Sincronizare
 - circuitele lucrează după un semnal de ceas comun
 - evoluează în aceleași momente
- Asincronism
 - fiecare circuit evoluează pe baza unor comenzi provenite de la alte circuite
- pot apărea ambele moduri de comandă în același sistem

Utilizare asincronă (1)

- exemplu - comanda unui ascensor
- intrări
 - etajul destinație
 - etajul curent
- ieșiri
 - urcare
 - coborîre

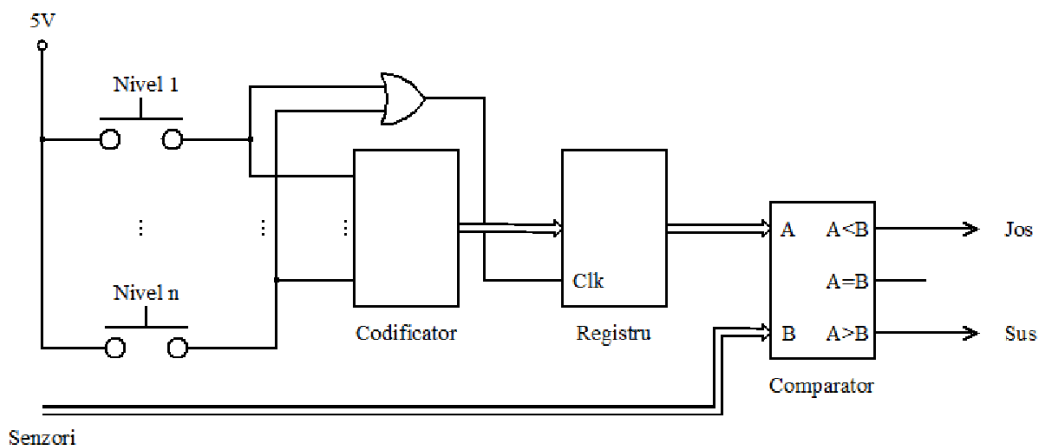
Utilizare asincronă (2)

- comanda - dată de relația dintre intrări
 - etaj destinație < etaj curent urcare
 - etaj destinație > etaj curent coborîre
 - etaj destinație = etaj curent nici o acțiune
- modificarea intrării impune modificarea automată a comenzii
- nu este necesară reținerea unei stări curente

Utilizare asincronă (3)

- sunt totuși necesare circuite secvențiale?
- controlul informațiilor de la intrări
- etajul curent - provenit de la senzori
- etajul destinație - indicat de utilizator
 - apăsarea unui buton
 - informația se poate pierde -> trebuie păstrată
 - putem folosi un registru

Utilizare asincronă (4)



Utilizare asincronă (5)

- problemă cu schema anterioară
 - este apăsat un nou buton în timp ce liftul se deplasează către etajul destinație deja stabilit
 - deci etajul destinație este schimbat - *în timpul deplasării*
- proiectarea asincronă este foarte dificilă
 - mai utilă pentru componente, nu pentru sisteme întregi

Utilizare sincronă

- este apăsat un buton corespunzător unui etaj
 - liftul este nefolosit - ne deplasăm spre acel etaj
 - liftul are deja o comandă - ignoră noua apăsare a butonului
 - răspunsuri diferite la aceeași valoare a intrării
- soluția - sistem cu stare
- comanda dată - depinde de starea curentă
 - și de ultimele informații de la intrări (senzori)

Proiectarea sistemelor secvențiale simple

Automate

- stări
- tranziții între stări
- 3 categorii de variabile
 - de intrare
 - de stare
 - de ieșire

Ecuatii de funcționare

- pornind de la valorile pentru
 - intrări
 - starea curentă
- dorim să obținem valorile pentru
 - starea următoare
 - ieșiri

Proiectare - pași (1)

- analiza problemei - foarte important!
- stabilirea variabilelor de intrare și de ieșire
- graful de fluentă
 - stările
 - tranzițiile
 - valorile variabilelor de intrare care condiționează tranzițiile
 - valorile ieșirilor corespunzătoare tranzițiilor

Proiectare - pași (2)

- matricea (tabelul) de fluentă
 - exprimare formală a grafului de fluentă
 - liniile - stările curente
 - coloanele - combinațiile de valori ale variabilelor de intrare
 - conținutul unei celule
 - starea următoare
 - valorile variabilelor de ieșire

Proiectare - pași (3)

- reducerea stărilor echivalente
 - pe baza analizei matricii de fluentă
 - 2 stări sunt neechivalente dacă
 - produc valori diferite ale ieșirilor pentru cel puțin o combinație de valori ale intrărilor
 - echivalența lor depinde de echivalența altor 2 stări, care sunt neechivalente (recursiv)
 - altfel sunt echivalente
 - stările echivalente pot fi reunite

Proiectare - pași (4)

- codificarea stărilor
 - rezultă și numărul de biți necesar pentru a reține starea (numărul variabilelor de stare)
- rescrierea matricii de fluentă cu stările în forma codificată
- aplicarea minimizării
- obținerea ecuațiilor de funcționare
 - câte o ecuație booleană pentru fiecare variabilă de stare, respectiv ieșire

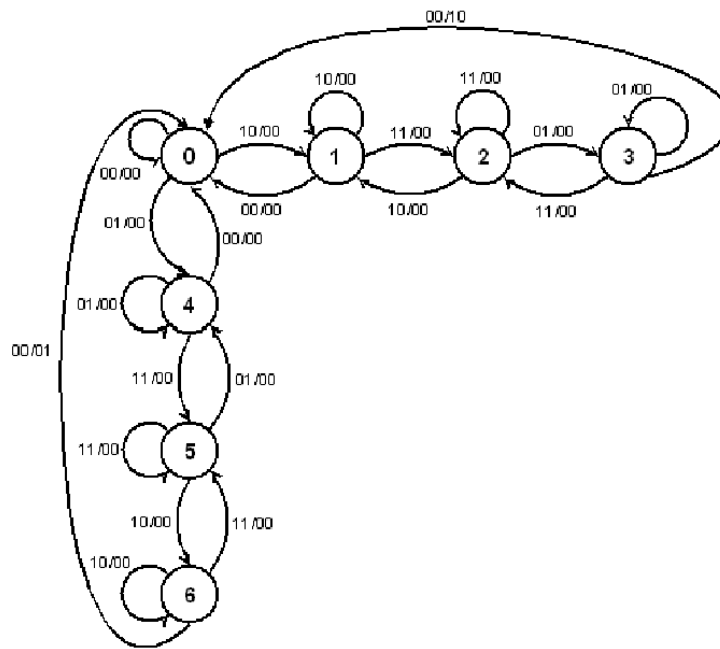
Exemplu

- problema - numărarea persoanelor care intră și ies printr-un culoar
- semnale de intrare - 2 celule fotoelectrice
- semnale de ieșire - incrementarea a 2 numărătoare (intrați, respectiv ieșiți)
- premise
 - pe culoar nu pot trece două persoane una pe lângă alta
 - mișcarea persoanelor - relativ lentă (în comparație cu automatul)

Detectarea intrărilor/ieșirilor

- secvența semnalelor de intrare
 - 00->10->11->01->00 - a intrat o persoană
 - 00->01->11->10->00 - a ieșit o persoană
- secvență incompletă - nu a intrat/ieșit nimeni
- la completarea unei secvențe, se activează ieșirea corespunzătoare
 - primește valoarea 1 pentru o perioadă de ceas

Graful de fluentă



Matricea de fluentă

	00	01	10	11
0	0/00	4/00	1/00	*/**
1	0/00	*/**	1/00	2/00
2	*/**	3/00	1/00	2/00
3	0/10	3/00	*/**	2/00
4	0/00	4/00	*/**	5/00
5	*/**	4/00	6/00	5/00
6	0/01	*/**	6/00	5/00

Stări echivalente - etapa 1

(0,1): e	(1,3): n	(2,6): (1,6) (2,5)
(0,2): (3,4)	(1,4): (2,5)	(3,4): n
(0,3): n	(1,5): (1,6) (2,5)	(3,5): (3,4) (2,5)
(0,4): e	(1,6): n	(3,6): n
(0,5): (1,6)	(2,3): e	(4,5): e
(0,6): n	(2,4): (3,4) (2,5)	(4,6): n
(1,2): e	(2,5): (3,4) (1,6)	(5,6): e

Stări echivalente - etapa 2

(0,1): e	(1,3): n	(2,6): n
(0,2): n	(1,4): n	(3,4): n
(0,3): n	(1,5): n	(3,5): n
(0,4): e	(1,6): n	(3,6): n
(0,5): n	(2,3): e	(4,5): e
(0,6): n	(2,4): n	(4,6): n
(1,2): e	(2,5): n	(5,6): e

Reducerea stărilor echivalente

- grupări posibile: (0,1), (0,4), (1,2), (2,3), (4,5), (5,6)
- nu se poate forma nici un grup de 3 sau mai multe stări
- noile stări (variantă)
 - (0,1) -> A
 - (2,3) -> B
 - (4,5) -> C
 - 6 -> D

Codificarea stărilor

- 4 stări -> 2 variabile de stare (biți)
- variantă
 - A <-> 00
 - B <-> 01
 - C <-> 10
 - D <-> 11
- ecuațiile finale depind de codificare
 - dar nu putem ști care variantă va fi mai bună

Diagrama de minimizare

x_1x_0 y_1y_0	00	01	11	10
00	00/00	10/00	01/00	00/00
01	00/10	01/00	01/00	00/00
11	00/01	**/**	10/00	11/00
10	00/00	10/00	10/00	11/00

Minimizare - $y_{1,n+1}$

$y_1y_0 \backslash x_1x_0$	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	*	1	1
10	0	1	1	1

Minimizare - $y_{0,n+1}$

$y_1y_0 \backslash x_1x_0$	00	01	11	10
00	0	0	1	0
01	0	1	1	0
11	0	*	0	1
10	0	0	0	1

Minimizare - z_1

$y_1y_0 \backslash x_1x_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	*	0	0
10	0	0	0	0

Minimizare - z_0

x_1x_0 y_1y_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	*	0	0
10	0	0	0	0

Ecuatiile de funcționare

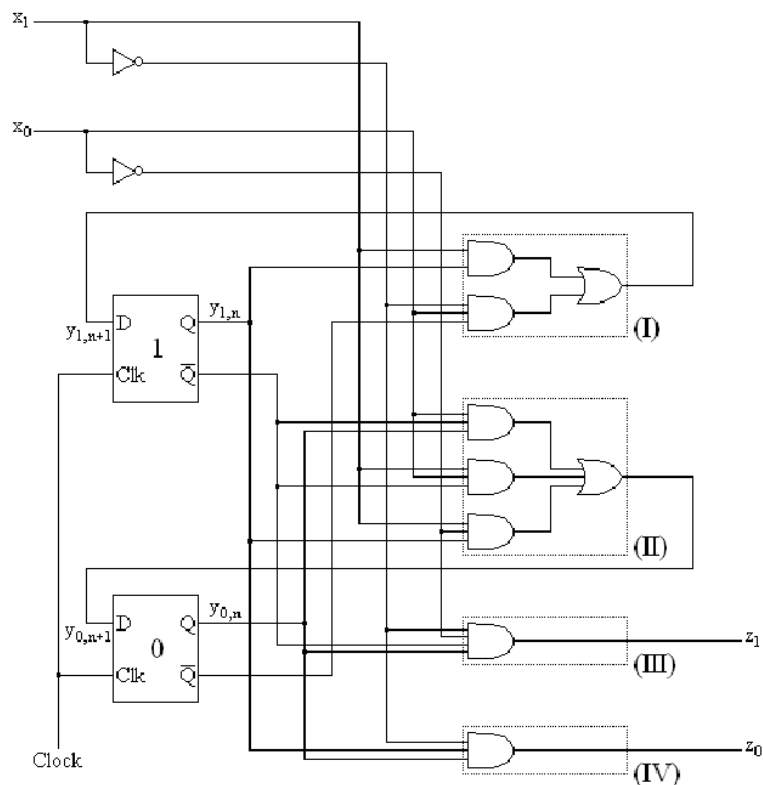
$$(I) \quad y_{1,n+1} = (y_1 \cdot x_1 + \overline{y_0} \cdot \overline{x_1} \cdot x_0)_n$$

$$(II) \quad y_{0,n+1} = (\overline{y_1} \cdot y_0 \cdot x_0 + \overline{y_1} \cdot x_1 \cdot x_0 + y_1 \cdot x_1 \cdot \overline{x_0})_n$$

$$(III) \quad z_{1,n} = (\overline{y_1} \cdot y_0 \cdot \overline{x_1} \cdot \overline{x_0})_n$$

$$(IV) \quad z_{0,n} = (y_1 \cdot y_0 \cdot \overline{x_1})_n$$

Implementarea circuitului



Cursul 3

Automate hardware

Complexitate (1)

- sistemele discutate la cursul anterior permit rezolvarea de probleme simple
 - nu utilizează nici măcar circuite combinaționale și secvențiale predefinite
 - sumatoare, comparatoare, decodoare etc.
 - regiștri, numărătoare etc.
- în general problemele care trebuie abordate sunt mai complexe

Complexitate (2)

- logica este separată în două părți
 - elementele de acționare
 - secvențiatorul
- acesta din urmă comandă elementele de acționare
- modelare
 - tot sub forma unui automat
 - dar determinarea stărilor se face în alt mod

Componente (1)

- elemente de acționare
 - circuite combinaționale și secvențiale
 - realizează operațiile cerute de algoritmul implementat
 - operațiile trebuie realizate la anumite momente
 - implementarea algoritmului
 - materializare - circuitele secvențiale primesc comenzi la anumite momente

Componente (2)

- secvențiatorul
 - automatul de control
 - trimite comenzi către elementele de acționare la momentele potrivite
 - implementare cablată - automat simplu
 - stările - decodificate
 - câte un bistabil pentru fiecare stare
 - la fiecare moment - un singur bistabil are valoarea 1

Intrările secvențiatorului

- semnale care pot fi testate
- în funcție de valoarea lor se pot lua decizii
- proveniență
 - semnale din exteriorul sistemului
 - semnale generate de elementele de acționare (comparatoare, numărătoare - terminal count, porți logice etc.)

Ieșirile secvențiatorului

- semnale de comandă către elementele de acționare
 - comenzi pentru circuitele secvențiale
 - validări pentru circuitele combinaționale
- exemple
 - semnale de resetare
 - încărcare regiștri
 - incrementare/decrementare numărătoare
 - validare decodare etc.

Descrierea algoritmului

- schema logică
- 2 tipuri de blocuri
 - testări
 - semnalele de intrare
 - acțiuni
 - comenzi către elementele de acționare

Implementarea secvențiatorului

- delimitarea stărilor
 - o stare poate include simultan testări și acțiuni
- determinarea ecuațiilor
 - variabile de stare
 - precizează starea următoare în funcție de starea curentă și de intrări
 - uzual, stările nu sunt codificate
 - câte o variabilă de stare (bistabil) pentru fiecare stare
 - ieșiri

Exemplu

- măsurarea perioadei unui semnal
- testarea repetată a valorii semnalului de intrare (INPUT)
 - automatul este mult mai rapid decât variațiile semnalului măsurat
 - determinarea duratei unei perioade
 - incrementarea unui numărător
 - cât timp intrarea este pe 0, apoi cât este pe 1
 - sau invers

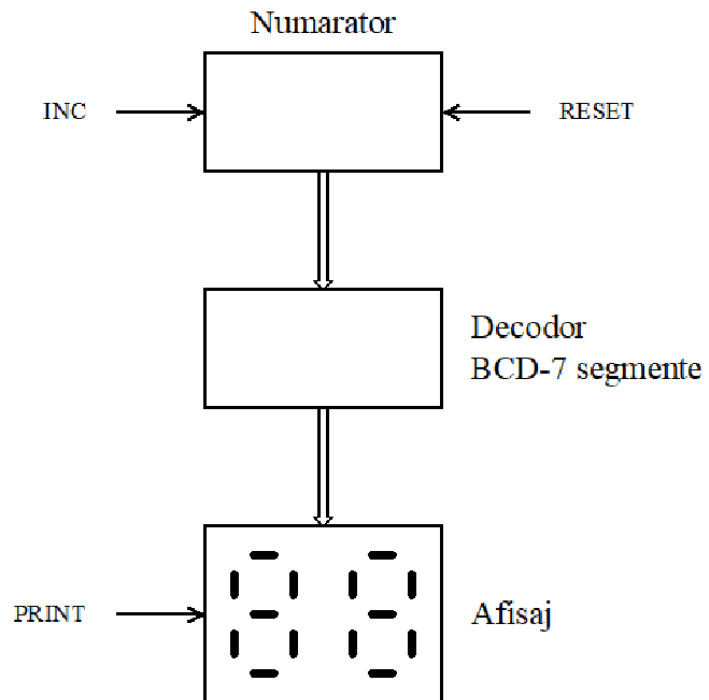
Exemplu (continuare)

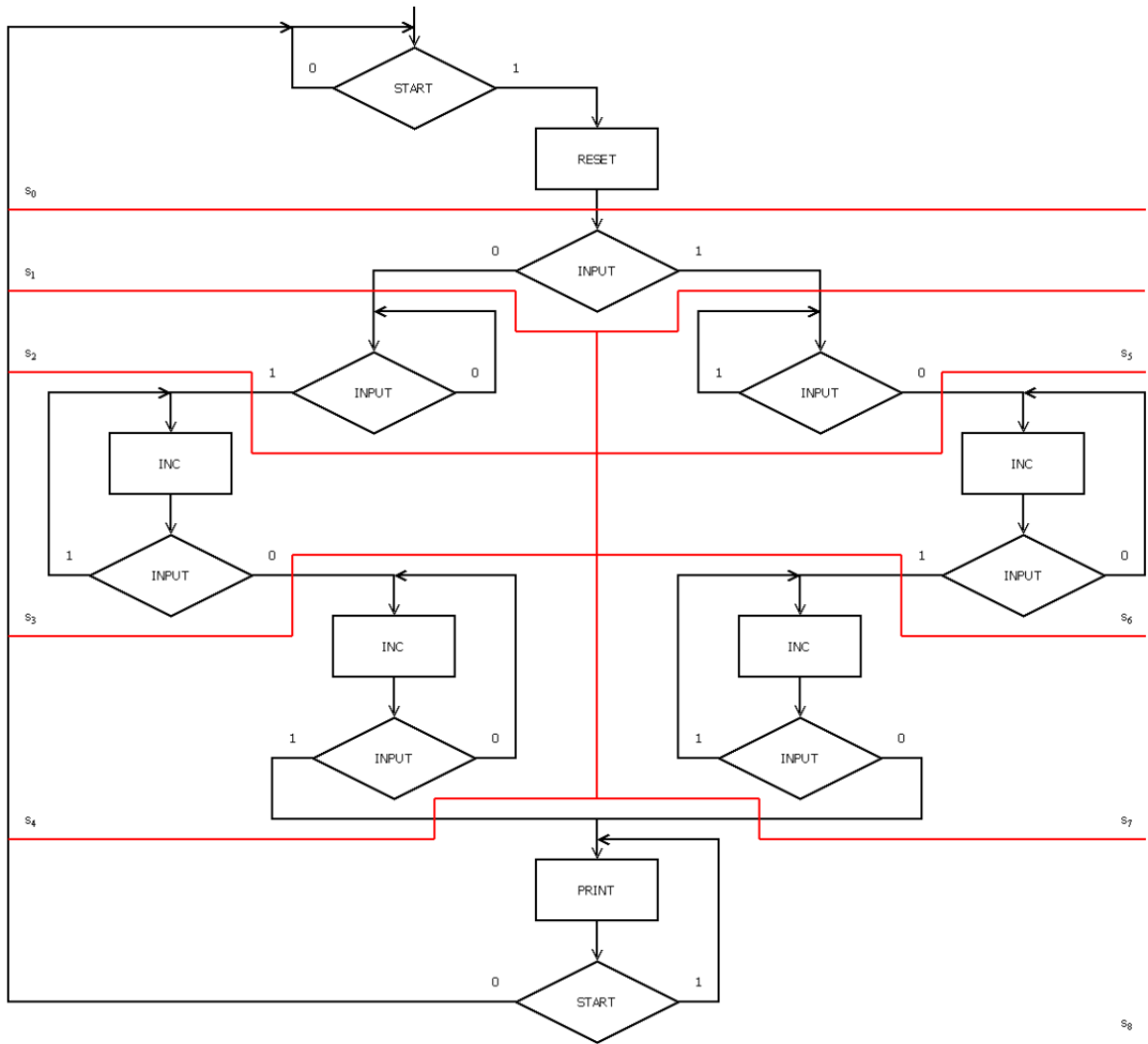
- inițial
 - se așteaptă apăsarea unui buton (START)
- final
 - afișare (PRINT)
 - se așteaptă terminarea apăsării butonului

Elementele de acțiune

- determinarea valorii - numărător
- pentru afișare de folosește un display cu 7 segmente
- conectare
- valoarea din numărător este trimisă către display
- pentru adaptarea informației - decodor BCD-7 segmente

Elementele de acțiune - schema





Ecuatii de funcționare

$$s_{0,n+1} = s_{0,n} \cdot \overline{\text{START}} + s_{8,n} \cdot \overline{\text{START}}$$

$$s_{1,n+1} = s_{0,n} \cdot \text{START}$$

$$s_{2,n+1} = s_{1,n} \cdot \overline{\text{INPUT}} + s_{2,n} \cdot \overline{\text{INPUT}}$$

$$s_{3,n+1} = s_{2,n} \cdot \text{INPUT} + s_{3,n} \cdot \text{INPUT}$$

$$s_{4,n+1} = s_{3,n} \cdot \overline{\text{INPUT}} + s_{4,n} \cdot \overline{\text{INPUT}}$$

$$s_{5,n+1} = s_{1,n} \cdot \text{INPUT} + s_{5,n} \cdot \text{INPUT}$$

$$s_{6,n+1} = s_{5,n} \cdot \overline{\text{INPUT}} + s_{6,n} \cdot \overline{\text{INPUT}}$$

$$s_{7,n+1} = s_{6,n} \cdot \text{INPUT} + s_{7,n} \cdot \text{INPUT}$$

$$s_{8,n+1} = s_{4,n} \cdot \text{INPUT} + s_{7,n} \cdot \overline{\text{INPUT}} + s_{8,n} \cdot \text{START}$$

$$\text{RESET} = s_{0,n} \cdot \text{START}$$

$$\text{PRINT} = s_{8,n}$$

$$\text{INC} = s_{3,n} + s_{4,n} + s_{6,n} + s_{7,n}$$

Cursul 4

Microprogramare

Implementarea funcțiilor booleene

- combinațional
 - minimizare
- cu ajutorul unei memorii ROM
 - intrările - biții de adresă
 - ieșirile - biții de date
 - conținutul memoriei ROM urmează direct tabelul de adevăr

Exemplu

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Adresa	Valoare
0	1
1	0
2	0
3	1
4	0
5	1
6	0
7	1

Microprogram

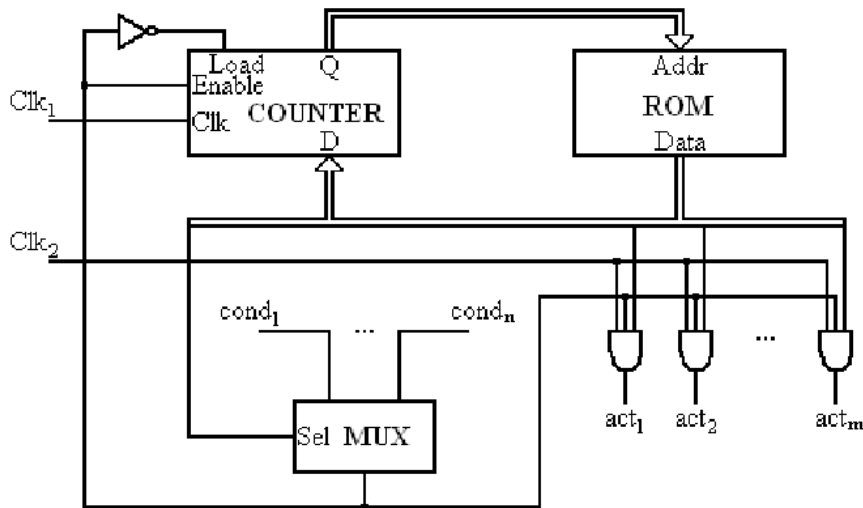
- implementarea secvențiatorului
 - memorie ROM
 - fiecare locație corespunde unei stări
 - conține informațiile necesare execuției (microinstrucțiuni)
 - numărător program
 - adresa locației corespunzătoare următoarei stări
 - actualizat prin incrementare sau salt

Microprogramare orizontală

Microprogramare orizontală

- câmpurile unei microinstrucțiuni
 - codul condiției testate
 - poate fi și 1 (+5V - întotdeauna adevărat) sau 0 (0V - întotdeauna fals)
 - semnalele de ieșire (acțiunile)
 - 1 - activat
 - în cazul în care condiția testată este adevărată
 - adresa de salt
 - în cazul în care condiția testată este falsă
 - altfel - incrementarea numărătorului program

Structura secvențiatorului



Condiții

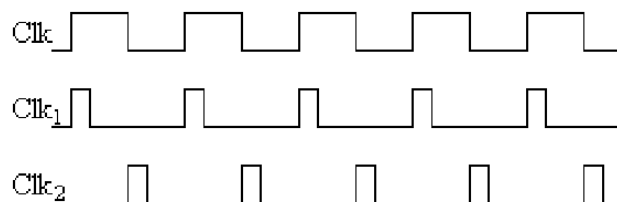
- de unde provin
 - intrări din exterior
 - ieșiri ale elementelor de acționare
- la fiecare pas se testează exact o condiție
 - selectată de multiplexor
 - care este comandat de codul microinstrucțiunii
 - în funcție de condiția testată
 - adevărat: acțiuni + incrementare
 - fals: nici o acțiune + salt

Condiții și acțiuni

- dacă pe parcursul execuției algoritmului trebuie testate atât condiția x, cât și NOT(x)
 - este necesar să avem câte o intrare în multiplexor pentru x și NOT(x)
- dacă două sau mai multe acțiuni nu apar niciodată simultan
 - în memoria ROM se poate stoca doar un cod al acțiunii activate, care este introdus într-un decodor -> mai puțini biți necesari în ROM

Ceas

- două semnale de ceas - separare în timp
 - Clk₁ - delimitarea stărilor
 - Clk₂ - validarea semnalelor de comandă (acțiuni)
 - obținute din același semnal periodic



Probleme

- delimitarea stărilor - mai puțin flexibilă decât la secvențiatorul cablat
- testare și acțiune în aceeași stare
 - dacă acțiunea trebuie efectuată numai când condiția testată are valoarea 1
 - altfel - ce valoare se trece în microinstrucțiune pentru acțiunea respectivă?
- starea următoare - uneori este mai greu de specificat

Probleme - exemplificare

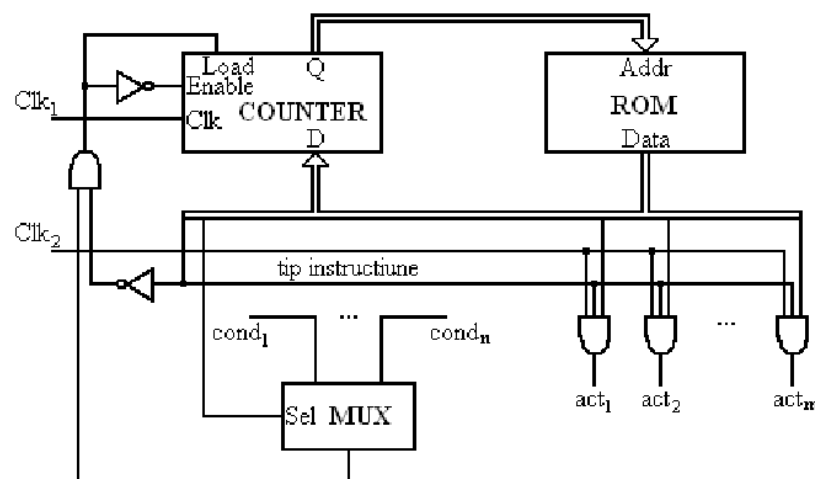
- aceeași problemă ca la cursul anterior
 - starea s_8 nu poate urma și după s_4 , și după s_7
 - similar, s_0 nu urmează după s_8
 - soluții
 - introducerea unor stări noi - nu realizează nici o acțiune, doar salt
 - duplicarea stării s_8 (una urmează după s_4 , cealaltă după s_7)
 - codificarea stărilor astfel ca s_0 să urmeze după s_8

Microprogramare verticală

Microprogramare verticală

- locațiile din ROM - dimensiuni mai mici
- 2 tipuri de microinstrucțiuni
 - de salt
 - bit de identificare al tipului (valoarea 0)
 - codul condiției de testat
 - adresa de salt
 - de acțiune
 - bit de identificare al tipului (valoarea 1)
 - acțiuni

Structura secvențiatorului



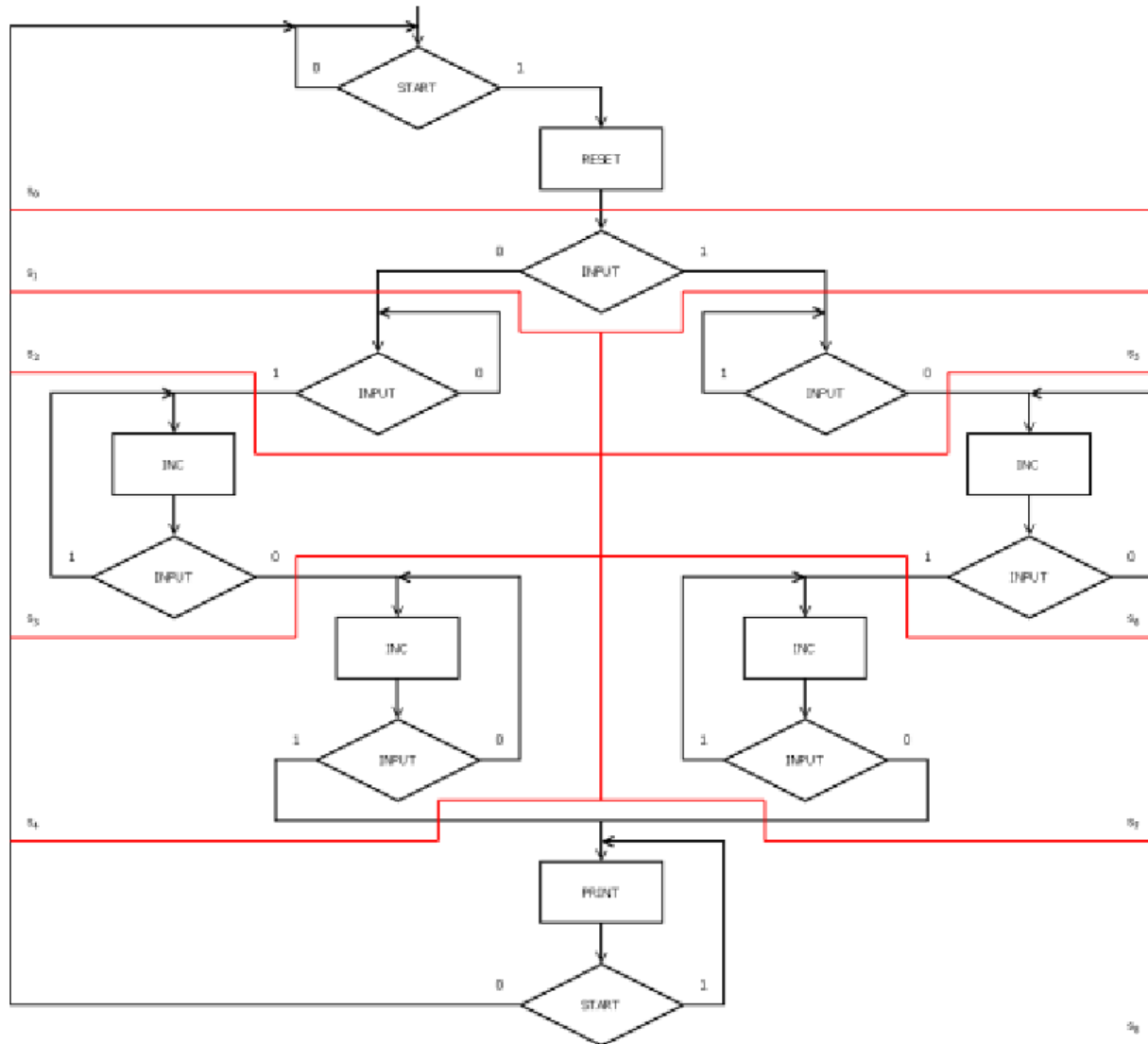
Probleme

- delimitarea stărilor - (și mai) inflexibilă
- nu se pot grupa teste și acțiuni în aceeași stare
- mai multe stări -> mai multe cicluri de ceas -> viteză mai mică

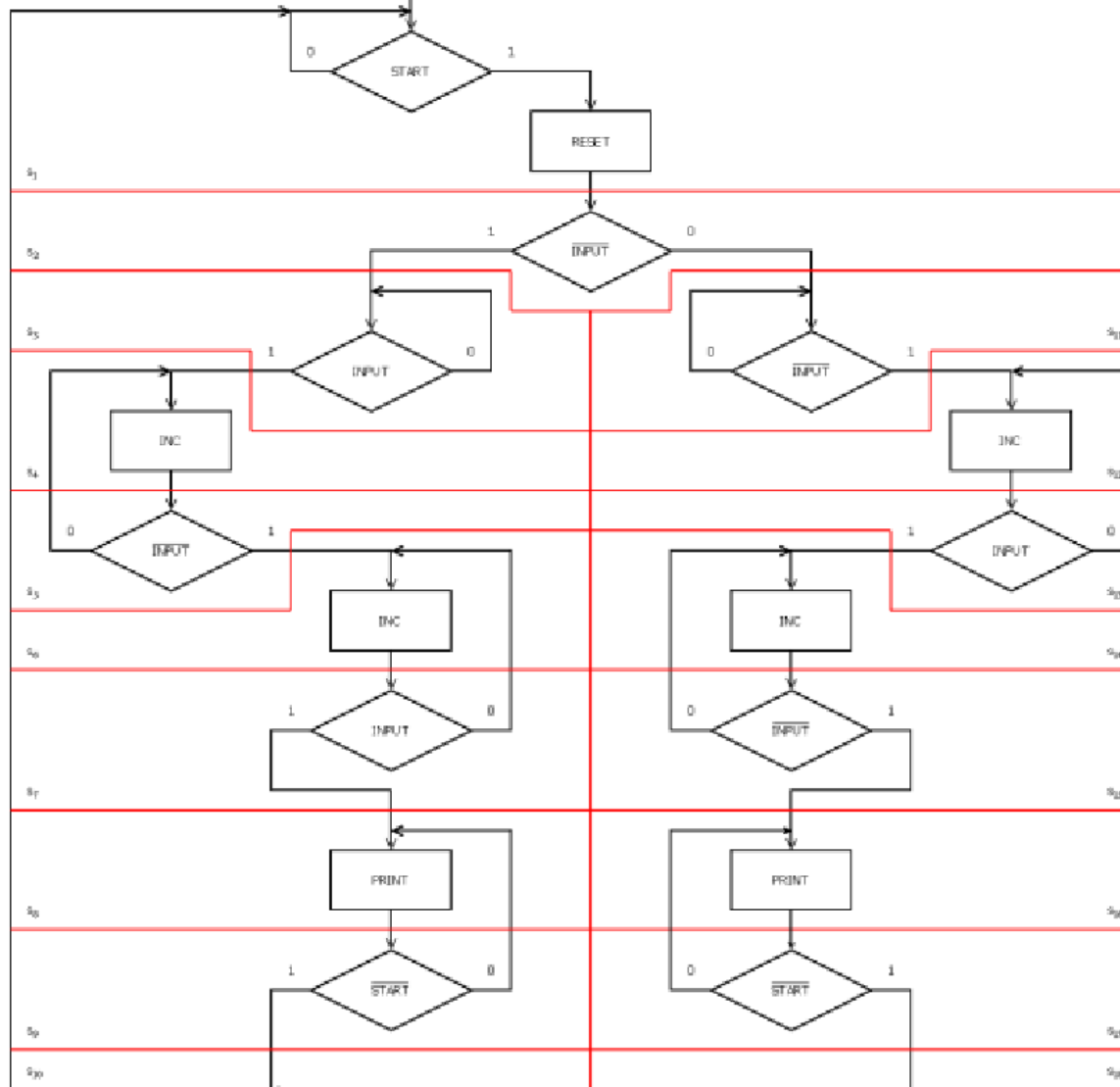
Exemplu

- măsurarea perioadei unui semnal (la fel ca la cursul 2)
- nu pot fi acțiuni și salturi simultan -> multe stări
- atenție la perioada determinată
 - înainte: numărul înregistrat x perioada ceasului automatului
 - acum: incrementările nu se mai fac la fiecare perioadă de ceas

Secvențiator cablat - stări



Microinstrucțiuni orizontale - stări



Microinstrucțiuni

- s_1 : START; RESET; s_1
- s_2 : NOT(INPUT); \emptyset ; s_{11}
- s_3 : INPUT; \emptyset ; s_3
- s_4 : TRUE; INC; \emptyset
- s_5 : NOT(INPUT); \emptyset ; s_4
- s_6 : TRUE; INC; \emptyset
- s_7 : INPUT; \emptyset ; s_6
- s_8 : TRUE; PRINT; \emptyset
- s_9 : NOT(START); \emptyset ; s_8
- s_{10} : FALSE; \emptyset ; s_1

s_{11} : NOT(INPUT); \emptyset ; s_{11}

s_{12} : TRUE; INC; \emptyset

s_{13} : INPUT; \emptyset ; s_{12}

s_{14} : TRUE; INC; \emptyset

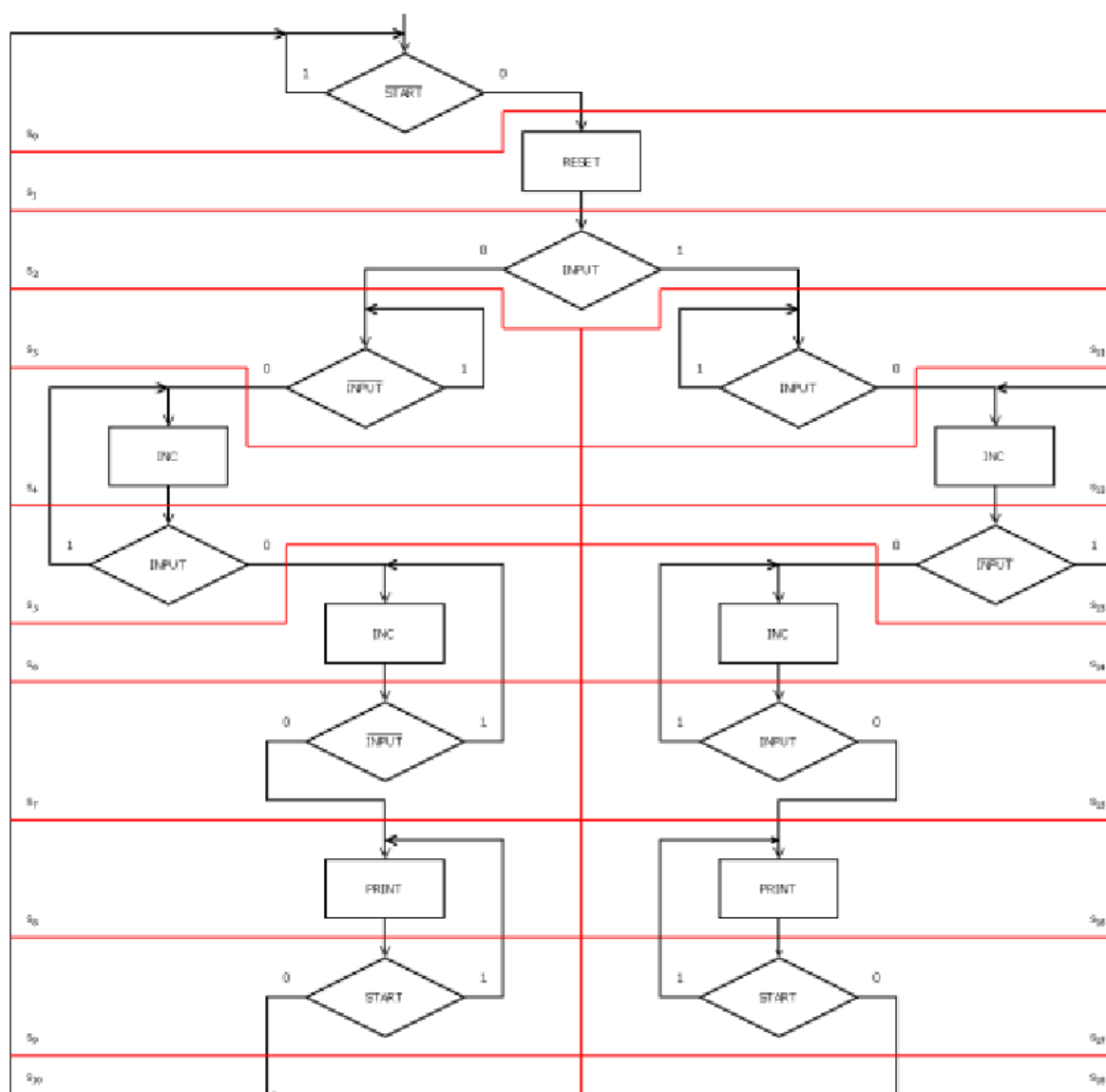
s_{15} : NOT(INPUT); \emptyset ; s_{14}

s_{16} : TRUE; PRINT; \emptyset

s_{17} : NOT(START); \emptyset ; s_{16}

s_{18} : FALSE; \emptyset ; s_1

Microinstrucțiuni verticale - stări



Microinstrucțiuni

s₀: 0; NOT(START); s₀
s₁: 1; RESET
s₂: 0; INPUT; s₁₁
s₃: 0; NOT(INPUT); s₃
s₄: 1; INC
s₅: 0; INPUT; s₄
s₆: 1; INC
s₇: 0; NOT(INPUT); s₆

s₈: 1; PRINT
s₉: 0; START; s₈
s₁₀: 0; TRUE; s₀
s₁₁: 0; INPUT; s₁₁
s₁₂: 1; INC
s₁₃: 0; NOT(INPUT); s₁₂
s₁₄: 1; INC
s₁₅: 0; INPUT; s₁₄

s₁₆: 1; PRINT
s₁₇: 0; START; s₁₆
s₁₈: 0; TRUE; s₀

Comparație (1)

- microinstrucțiuni orizontale
 - structură mai clară
 - se pot grupa mai bine stările - mai puține stări
 - test urmat de acțiune - în aceeași stare
 - acțiune urmată de test - nu se poate în aceeași stare
- microinstrucțiuni verticale
 - cuvântul de memorie - mai scurt
 - mai multe stări - mai multe locații de memorie

Comparație (2)

- în ambele cazuri, față de implementarea cablată, se pierde din flexibilitate în delimitarea stărilor
 - și implicit în eficiența automatului
- avantaj - reproiectare ușoară (dacă e necesar)
- se schimbă conținutul memoriei ROM
- în cazul cablat - refacerea în întregime a părții combinaționale

Cursul 5

Implementarea sistemelor secvențiale

Variante tehnologice (1)

- ASIC
 - *Application-Specific Integrated Circuits*
 - proiectarea - realizată în mod specific pentru un anumit circuit sau clasă de circuite
 - avantaj - optimizare pentru rezolvarea problemei abordate -> performanțe superioare
 - tipuri
 - full-custom
 - semi-custom

Variante tehnologice (2)

- circuite programabile
 - permit implementarea a diferite automate
 - în funcție de problema abordată
 - pot fi reconfigurate după necesități
 - avantaje
 - flexibilitate în proiectare și depanare
 - preț redus - datorită producției de serie
 - dezavantaj
 - nu pot asigura obținerea de performanțe maxime

Circuite programabile (1)

- arii de porți (*Gate Arrays*)
 - număr mare de celule
 - fiecare celulă - sumă (OR) de termeni produs (AND)
 - de obicei se folosesc doar o parte dintre intrări (câte sunt necesare)
 - structura corespunde cu modul de proiectare al circuitelor combinaționale
 - celulele sunt interconectate - se pot realiza circuite complexe

Circuite programabile (2)

- CPLD
 - Complex Programmable Logic Devices
 - fiecare celulă include și un bistabil
 - la ieșirea părții combinaționale
 - poate fi folosit sau nu (comportament pur combinațional)
 - poate exista și reacție înapoi - de la ieșirea bistabilului spre partea combinațională
 - modelează comportamentul secvențial

Circuite programabile (3)

- FPGA
 - *Field Programmable Gate Arrays*
 - celulele - de obicei mai simple decât la CPLD
 - partea combinațională - LUT (*Look-Up Table*)
 - implementare LUT - variante
 - multiplexor
 - memorie ROM
 - memorie RAM - mai ușor de reconfigurat

Implementare (1)

Probleme

1. proiectarea logică a circuitului
 2. implementarea fizică
 - configurarea și interconectarea celulelor
 - utilizarea circuitelor disponibile în cadrul celulelor
 - structura fizică poate diferi de cea logică
- dificil de gestionat

Implementare (2)

Soluția

- compilatoare de hardware
 - realizează implementarea fizică
 - țin cont de structura hardware concretă pe care se face

implementarea

- se folosesc limbaje dedicate de descriere a hardware-ului
 - VHDL, Verilog etc.

Limbajul Verilog

Organizare

- circuitele - implementate ca module
- moduri de descriere a circuitelor
 - structurală
 - nivel jos
 - indicarea componentelor folosite
 - comportamentală
 - nivel înalt
 - se descrie comportamentul dorit al circuitului
 - pot fi combinate

Module - structură

- nume modul
- lista parametrilor (semnale de intrare-ieşire)
- declararea tipului parametrilor
 - input
 - output
 - inout (mai rar)
- descrierea modului
 - diverse moduri

Exemplu

- proiectarea unui multiplexor 2->1
 - intrări
 - de date: I0, I1
 - de selecție: Sel
 - ieşire: E
- pot fi folosite ambele moduri de descriere, inclusiv combinații

Varianta 1

```
module MUX(Sel,I0,I1,E);  
input Sel,I0,I1;  
output E;  
assign E=(I1&Sel)|(I0&~Sel);  
endmodule
```

- assign - asignare continuă
 - orice schimbare la intrare duce la schimbarea ieşirii (combinațional)

Varianta 2

```
module MUX(Sel,I0,I1,E);  
input Sel,I0,I1;  
output E;  
wire x,y,z;  
not n0(z,Sel);  
and a1(x,I1,Sel);  
and a0(y,I0,z);  
or o0(E,x,y);  
endmodule
```

- wire - declară semnale (fire electrice) interne
- not, and, or - funcții (module) predefinite
 - primul parametru - ieşirea
 - număr variabil de intrări
- se pot utiliza și alte module definite de utilizator

Varianta 3

```
module MUX(Sel,I0,I1,E);  
input Sel,I0,I1;  
output E;  
reg E;  
always @(Sel,I0,I1)  
    E=(I1&Sel)|(I0&~Sel);  
endmodule
```

- reg - variabila poate reține o valoare între două asignări
 - de obicei secvențial (dar nu neapărat)
- always - buclă infinită
 - se execută de fiecare dată când se modifică una dintre intrările din lista asociată
 - asignarea pentru variabila E nu este continuă
 - valoarea variabilei E se schimbă doar când se execută instrucțiunea respectivă

Varianta 4

```
module MUX(Sel,I0,I1,E);  
input Sel,I0,I1;  
output E; reg E;  
always @(Sel,I0,I1)  
begin  
    if(Sel==0) E=I0;  
    else E=I1;  
end  
endmodule
```

Elemente ale limbajului Verilog

Vectori (1)

- folosiți pentru semnalele care constau din mai mulți biți
 - numere, adrese etc.
 - domeniul din care fac parte indicii - flexibil
- ```
input [7:0]x;
wire [0:7]y;
reg [10:3]z;
– toate definesc semnale pe 8 biți
```

### Vectori (2)

- pot fi accesați
  - integral (toți biții simultan)
  - bit cu bit
  - o parte din biți

```
z=x;
z[6]=y[4];
z[5:3]=x[6:4];
```

### Tablouri (1)

- similare celor din limbajele software
  - la un moment dat poate fi accesat un singur element al tabloului
- ```
wire x[3:0]; //tablou cu 4 elemente pe 1 bit fiecare  
a=x[0]; //corect  
b=x; //eroare  
c=x[2:0]; //eroare
```

Tablouri

- elementele unui tablou pot fi vectori
 - pot fi accesate direct și la nivel de bit
- ```
reg [7:0]y[15:0]; // tablou de 16 elemente, fiecare pe 8 biți
reg [7:0]z;
z=y[5];
```

### Atribuiți (1)

- utilizate în descrierile comportamentale
  - nu și pentru assign
- două tipuri
  1. blocante (=)
    - atribuirea curentă va începe doar după terminarea celei anterioare
  2. neblocaute (<=)
    - atribuirea curentă poate începe înainte de terminarea celei anterioare

### Atribuiți (2)

- a are inițial valoarea 7
- ```
a=5;  
b=a+3;  
    – b va primi valoarea 8  
a<=5;  
b<=a+3;  
    – b va primi valoarea 10
```

Întârzieri (1)

- instrucțiunile din exemplul anterior sunt raportate ca executându-se la același moment
 - indiferent de tipul de atribuire utilizat
- uneori trebuie să specificăm faptul că o instrucțiune se execută mai târziu decât cea dinaintea sa
 - și după cât timp

Întârzieri (2)

- ```
#5 a=5;
 b=a+3;
```
- a doua instrucțiune se execută la 5 unități de timp după prima
  - deci nu mai putem avea execuție în paralel

- chiar dacă se folosesc atribuiri nebloccante
- dacă nu specificăm o întârziere - similar cu #0

#### Baze de numerație

- valori exprimate în diferite baze de numerație
  - baza 2: 7'b0101110
  - baza 8: 8'o247
  - baza 10: 8'd25 sau 25 (implicit)
  - baza 16: 12'hA0F
- numărul dinaintea semnului ' exprimă întotdeauna numărul de biți (nu de cifre) al reprezentării

#### Înaltă impedanță

- valorile pe care le poate lua un semnal
  - 0
  - 1
  - x (nedeterminat)
  - z (înaltă impedanță)
- exemplu de utilizare
  - a <= 8'bzzzzzzzz;
  - valoarea z nu poate fi testată

#### Concatenări de semnale

- sintaxa: {*semnal*, *semnal*, ...}
- exemplu
  - reg [3:0]a,b;
  - reg [7:0]c,d;
  - {a[2],c} <= {d[6:1],b[2:0]};



## Cursul 6

### Proiectarea sistemelor secvențiale

#### Principii

- modelarea - în principal comportamentală
  - se pot utiliza și elemente structurale pentru unele componente
- trebuie luat în considerare semnalul de ceas
  - în mod explicit

#### Exemplu 1

```
• bistabil D - latch
module Latch(Clk,D,Q);
input Clk,D;
output Q;
reg Q;
always @(Clk,D)
if(Clk==1) Q<=D;
endmodule
```

#### Exemplu 2

- bistabil D - flip-flop
- considerăm și intrările R și S
  - asincrone (nu depind de semnalul de ceas)
  - prioritare
  - necesare pentru inițializare și alte operații
- tratarea semnalului de ceas
  - contează doar frontul crescător
  - trebuie detectat

#### Exemplu 2 - varianta 1

```
module FlipFlop(Clk,D,R,S,Q);
input Clk,D,R,S;
output Q;
reg Q;
always @(posedge Clk,D,R,S)
 if(R==1) Q<=0;
 else if(S==1)
 Q<=1;
 else if(posedge Clk)
 Q<=D;
endmodule
```

#### Exemplu 2 - varianta 1 (cont.)

- detectarea frontului crescător - *posedge*
  - front descrescător - *negedge*
- varianta NU ESTE CORECTĂ
  - frontul crescător poate fi precizat în lista de sensibilitate a structurii *always*
  - dar nu poate fi testat într-o instrucțiune *if*
  - pot fi testate doar valori ale semnalelor, nu și tranziții ale acestora

### Exemplu 2 - varianta 2

```
module FlipFlop(Clk,D,R,S,Q);
input Clk,D,R,S;
output Q;
reg Q;
always @(R,S)
 if(R==1) Q<=0;
 else if(S==1)
 Q<=1;
always @(posedge Clk)
 if(R==0 && S==0)
 Q<=D;
endmodule
```

### Exemplu 2 - varianta 2 (cont.)

- se pot folosi 2 sau mai multe structuri *always*
  - și *initial*
- toate se execută în paralel
- intrările R și S trebuie să fie prioritare
  - sunt testate în cealaltă structură *always*
- combinațiile sincron-asincron - dificil de tratat

### Automate

- două componente
  - evoluția stării
    - pe baza semnalului de ceas
  - elemente de acționare
    - descriere de obicei comportamentală
    - calculează valorile pentru starea următoare și ieșirile
- izolare - structuri *always* separate
  - modularitate - mai ușor de înțeles și controlat

### Structura de bază

```
module nume_modul(Clk, intrari, iesiri);
input Clk, intrari;
output iesiri;
reg iesiri;
reg stare_curenta, stare_urmatoare;
always @(posedge Clk)
 stare_curenta<=stare_urmatoare;
 // eventual reset, ...
always @(intrari, stare_curenta)
begin
 // calcul iesiri, stare urmatoare
end
endmodule
```

### Structura de bază (cont.)

- partea combinațională răspunde imediat la orice modificare a intrărilor sale
  - starea curentă
  - intrări
- partea secvențială permite modificarea doar pe frontul crescător al ceasului
- nu e obligatoriu ca stare\_curenta și stare\_urmatoare să fie implementate ca regiștri separați

### Exemplu

- măsurarea perioadei unui semnal
- digitizare - transformare într-un semnal cu valori doar 0 și 1
  - perioada rămâne aceeași
  - nu este sigur că valorile de 0 și respectiv 1 au aceeași durată în cadrul unei perioade
- aceleași obiective ca la cursul 4

### Varianta 1

```
module P(start,i,digit,print);
input start,i;
output reg [3:0]digit;
output reg print;
always @(start,i)
begin
 digit=0;
 print=0;
 if(i==0) begin
 while(i==0);
 while(i==1) digit=digit+1;
 while(i==0) digit=digit+1;
 print=1;
 end
 else begin
 while(i==1);
 while(i==0) digit=digit+1;
 while(i==1) digit=digit+1;
 print=1;
 end
end
end
endmodule
```

### Varianta 1 (cont.)

- problema
  - nu apare semnalul de ceas
  - la ce momente au loc toate acțiunile?
- semnalul de ceas trebuie să apară explicit ca intrare
- o instrucțiune poate fi întârziată până la apariția unui eveniment
  - ex: @(posedge clk) digit=digit+1;

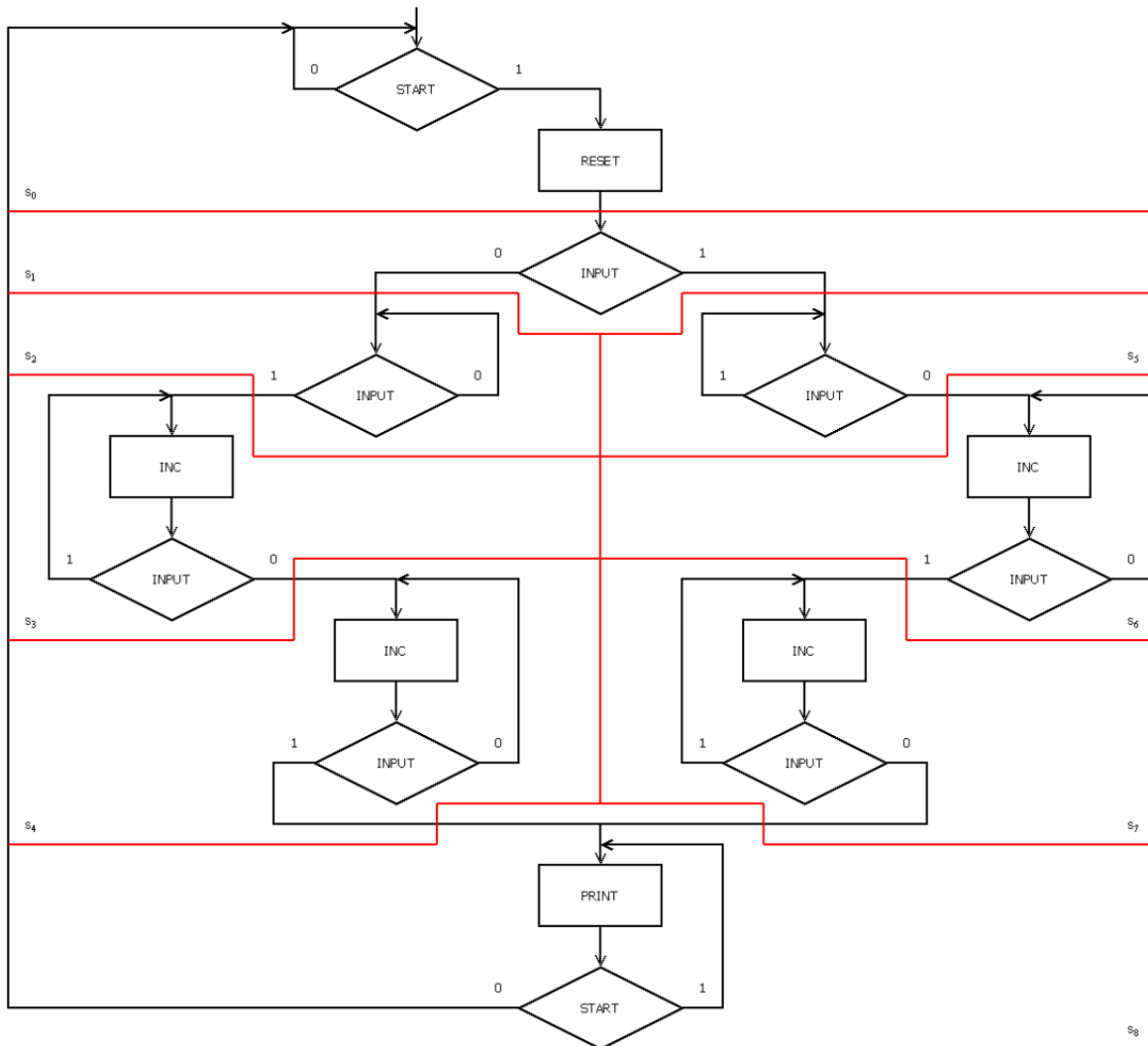
### Varianta 1 (cont.)

- o soluție ar fi deci ca toate acțiunile să fie întârziate în acest mod

```
while(i==0) @(posedge clk);
while(i==1)
 @(posedge clk) digit=digit+1;
while(i==0)
 @(posedge clk) digit=digit+1;
```
- foarte greu de gestionat

### Varianta 2

- revenim la proiectarea automatului
  - la fel ca la cursul 4
  - orientativ - ne interesează stările, nu exact semnalele de comandă
    - implementarea acestora o realizează compilatorul
    - ex: în loc de RESET vom scrie  
digit<=0; display<=0;
    - descrierea - comportamentală



### Varianta 2 - implementare

```
module P(Clk,start,i,digit,print);
input Clk,start,i;
output reg [3:0]digit;
output reg print;
reg [3:0]state;
reg [3:0]state_next;
always @(posedge Clk)
 state<=state_next;
always @(start, i, state)
 if(state==0)
 begin
 if(start==0) state_next<=0;
 else state_next<=1;
 digit<=0;
 print<=0;
 end
 else if(state==1) begin
 if(i==1) state_next<=2;
 else state_next<=5;
 end
 ... // celelalte stari
initial
 state<=0; state_next<=0;
endmodule
```

### Modelarea prin automate (1)

- limbajul Verilog este util mai ales pentru partea de implementare
- descrierea algoritmului de funcționare rămâne sarcina proiectantului
  - modelare pe bază de stări (automat)
- atenție la atribuirea de valori pentru ieșiri
  - teoretic - toate ieșirile primesc valori la fiecare stare
  - practic - atunci când se schimbă valorile

### Modelarea prin automate (2)

#### Avantaj

- separă partea de memorare a stării (secvențială) de cea combinațională
  - două blocuri always independente
    - unul actualizează starea
    - celălalt realizează calculul valorilor pentru ieșiri și starea următoare
  - descriere clară

### Modelarea prin automate (3)

#### Probleme

- la descrierea părții combinaționale
  - blocul always reacționează la modificarea unei variabile
  - nu este potrivit pentru integrarea într-un sistem secvențial

### Exemplu

```
module ex(Clk, x, y, z);
input Clk, x, y;
output z;
reg z;
//starea - cati biti sunt necesari
reg [2:0]state, state_next;
always @(posedge Clk)
 state<=state_next;
always @(x, y, state)
 if(state==0)
 if(x==1) begin
 z<=z+1;
 state_next<=0;
 end
 else state_next<=1;
 . . .
```

### Problema 1

- starea 0 - se incrementează z cât timp x are valoarea 1
  - buclă de program
- suntem în starea 0 și starea următoare este tot 0
- dacă x și y nu variază, z nu e incrementat
  - always nu detectează vreo schimbare pe nici una dintre variabile, deci nu se activează

### Problema 2

- starea 0 - apare o modificare a valorii variabilei y
- nu ar trebui să influențeze cu nimic evoluția sistemului
  - deoarece y nu intervine în starea 0
  - dar always detectează o schimbare pe una din variabile, deci se activează
  - ca urmare, z are o incrementare parazită

### Soluții (1)

- *always* să reacționeze și la schimbările semnalului de ceas
  - sau numai la frontul crescător al acestuia
  - pot apărea interferențe cu blocul *always* care se ocupă de actualizarea stării
    - atribuirile neblocante ( $\leq$ ) se execută în paralel și atunci când sunt în blocuri *always/initial* diferite
  - nu rezolvă a doua problemă

### Soluții (2)

- toate variabilele care apar în lista de sensibilitate a blocului *always* să fie testate prin instrucțiuni *if* în toate stările
  - s-ar rezolva a doua problemă
  - foarte greu de realizat practic

### Soluții (3)

- descriere structurală (de nivel jos) a părții combinaționale
  - fără blocuri *always*
  - ecuații booleene și/sau circuite predefinite
  - ar rezolva ambele probleme
  - puterea limbajului Verilog se pierde în foarte mare măsură dacă renunțăm la descrierile comportamentale

### Soluții (4)

- un singur vector de stare (în loc de doi)
- un singur bloc *always*
  - activat doar de frontul crescător al semnalului de ceas
  - toate calculele se (re)fac pe frontul crescător
  - deci toate variabilele sunt utilizate și actualizate numai pe frontul crescător
- ambele probleme sunt rezolvate

### Implementare

```
module nume_modul(Clk, intrari, iesiri);
input Clk, intrari;
output iesiri;
reg iesiri;
reg stare;
always @(posedge Clk)
 if(stare==0)
 //descriere stare 0
 //actualizare stare
 else if(stare==1)
 //descriere stare 1
 //actualizare stare
 ... //alte stari
endmodule
```

### Limitare

- pentru variabilele de intrare se iau în considerare doar valorile de la momentele când apare un front crescător al ceasului
  - dacă o variabilă de intrare este activată doar între două fronturi crescătoare consecutive, va fi ignorată
  - în general nu este o problemă
    - variabilele de intrare variază lent
    - la nevoie, intrările care provin de la alte circuite pot fi sincronizate după același semnal de ceas

### Exemplu

- calculul celui mai mare divizor comun a două numere
- algoritmul lui Euclid
- numerele sunt primite simultan din exterior
- semnale de notificare
  - *start* - numerele sunt disponibile la intrare
  - *ack* - rezultatul este disponibil la ieșire

### Implementare

```
module
 cmmdc(Clk, start, x, y, r, ack);
input Clk, start;
input [7:0]x, y;
output reg [7:0]r;
output reg ack;
reg [7:0]a, b;
reg [1:0]state;
always @(posedge Clk)
 if(state==0)
 if(start==1) begin
 a<=x;
 b<=y;
 ack<=0;
 state<=1;
 end
 else state<=0;
 else if(state==1)
 if(a==b) begin
 r<=a;
 state<=2;
 end
 else if(a>b) begin
 a<=a-b;
 state<=1;
 end
 else begin
 b<=b-a;
 state<=1;
 end
 else if(state==2) begin
 ack<=1;
 state<=0;
 end
initial
begin
 state<=0;
 ack<=0;
end
endmodule
```



## **Cursul 7**

### **Utilizarea limbajului Verilog în proiectarea hardware-ului**

#### Semnale

- elemente de bază în orice implementare fizică
- fiecare semnal poartă 1 bit de informație
- deci în mod fundamental lucrăm cu biți
  - sau cu vectori de biți

#### Tipuri de semnale (1)

- pot fi declarate în diverse moduri
  - în funcție de unghiul din care le privim
- intrare-ieșire pentru module
  - input - semnal de intrare pentru modulul curent
  - output - semnal de ieșire pentru modulul curent
  - inout - atât intrare, cât și ieșire

#### Tipuri de semnale (2)

- semnale interne modulelor
  - reg - poate memora valori preluate la anumite momente
    - uzual - implementare secvențială
  - wire - realizează legătura între intrarea unui circuit și ieșirea altuia
    - nu poate memora valori
- cele două categorii se pot suprapune

#### Tipuri de semnale (3)

- semnalele declarate ca input sunt automat de tip wire
  - nu putem impune valori unui semnal de intrare
- semnalele declarate ca output sunt de tip wire în mod implicit
  - pot fi conectate combinațional la alte semnale, de exemplu prin assign
  - le putem declara ca reg în mod explicit

#### Exemplu

```
module semnale(i, e1, e2);
input i;
output e1, e2;
reg x, e2;
assign e1=~x;
...
endmodule
```

#### Exemplu (cont.)

- semnalul i este intrare, deci nu poate fi reg
- semnalele e1 și e2 sunt ieșiri
- e2 este reg
  - poate fi modificat direct

- e1 nu este declarat reg
  - poate fi modificat prin intermediul semnalului x

#### Depinde de unde privim semnalul

- același semnal poate fi ieșire pentru un circuit (modul) și intrare pentru altul
  - situația apare de fapt foarte des
- deci și modul de declarare a semnalului poate diferi între cele două module
  - un modul îl poate modifica, celălalt îi poate doar citi valoarea

#### Exemplu

```
module iesire(s_out);
output s_out;
reg s_out;
...
s_out = 1;
...
endmodule

module intrare(s_in);
input s_in;
reg x; // semnal intern
...
x = s_in;
...
endmodule

module main();
wire s;
iesire a(s);
intrare b(s);
...
endmodule
```

#### Exemplu (cont.)

- ieșirea primului modul este conectată la intrarea celui de-al doilea
  - deci este de fapt același semnal
  - primul modul îi modifică valoarea
  - al doilea poate doar să o citească
  - la fel și modulul *main*
- în mod obișnuit nu putem modifica valoarea unui semnal de ieșire a unui modul din afara acestuia

#### Semnale inout

- reutilizarea aceluiași pini pentru intrări și ieșiri
  - la momente diferite de timp
  - util doar la module top-level
- de fapt sunt componente diferite care lucrează cu intrarea, respectiv ieșirea

– ceea ce scoatem pe ieșire nu trebuie să interfereze cu ceea ce primim pe intrare

#### Exemplu

```
module in_out(Clk,data);
input Clk;
inout [7:0]data;
reg [7:0]d,x;
assign data=d;
initial
 d<=8'bzzzzzzzz;
always @(posedge Clk)
begin
 x=data;
 #5 d<=x+1;
 #5 d<=8'bzzzzzzzz;
end
endmodule
```

#### Exemplu (cont.)

- semnalul *data* privit ca ieșire
  - legat (combi-național) la ieșirea variabilei *d*
  - deci orice modificare a valorii *d* se regăsește automat pe ieșirea *data*
  - când nu avem nimic de transmis pe ieșire, *d* trebuie să fie în înaltă impedanță
- semnalul *data* privit ca intrare
  - este citit în variabila *x* la anumite momente

#### Exemplu (cont.)

- atunci când scriem la ieșire, semnalul de la intrarea *data* trebuie să fie la rândul său în înaltă impedanță
  - provine de la alt modul
  - deci trebuie să existe un protocol de comunicare
  - sarcina proiectantului

#### Operații cu semnalele

- operatorii - sintaxă similară cu limbajul C
- operațiile pe biți - mult mai des utilizate
- unele operații pe biți se pot realiza mult mai eficient în hardware
- dacă un singur bit dintr-o variabilă are valoarea *x* (nedeterminat), toată variabila are valoarea *x*

#### Operatori de reducere

- funcții booleene de aritate 2 (sau mai mare)
  - operatori:  $|$  &  $^$
  - pot fi aplicați asupra biților dintr-un vector
  - exemplu: test dacă o variabilă are valoarea 0
- ```
reg [7:0]a;
```

```
reg b;  
b <= ~(|a);
```

Tipuri auxiliare

- integer
- real
- time
- variabilele din aceste tipuri folosite nu sunt pentru sinteza circuitelor, ci pentru
 - exprimarea operațiilor paralele
 - simulare

Șiruri de caractere

- nu există un tip dedicat
- constante - între caracterele "..."
- variabile - declarate ca reg
 - trebuie să conțină un număr suficient de biți
 - dacă sunt mai mulți - completate cu 0 la dreapta

```
reg [20*8:1]s; // 20 de octeti  
s="sir de caractere";
```

Bucle

- implementare
 - nivel jos
 - test + salt
- implementarea salturilor
 - prin stabilirea stării următoare
- structuri repetitive (while/for/repeat)
 - nu sunt destinate descrierii buclelor secvențiale
 - rol - descrierea operațiilor care se pot realiza în paralel

Exemplu

- inversarea ordinii biților într-o variabilă

```
reg [7:0]a;  
integer i;  
a<=8'b01010000;  
#5 for(i=0;i<8;i=i+1)  
    a[i]<=a[7-i];
```

Exemplu (cont.)

- atenție la precizarea întârzierilor

```
a<=8'b01010101;  
for(i=0;i<8;i=i+1)  
#5 a[i]<=a[7-i];
```

- în acest caz, fiecare atribuire din buclă se face la 5 unități de timp după cea anterioară
- nu se fac în paralel - rezultat incorect

Abordarea problemelor (1)

- se începe cu analiza problemei și cu schițarea unei soluții
 - pseudocodul este util, dar pot fi folosite și alte forme de exprimare a algoritmului
- de aici se pot extrage
 - intrările și ieșirile
 - variabilele interne
 - operațiile care trebuie efectuate

Abordarea problemelor (2)

- stabilirea stărilor și a tranzițiilor între stări
 - se realizează prin împărțirea algoritmului schițat în pași (stări)
- buclele se implementează prin salturi
 - de fiecare dată când atribuim o nouă valoare variabilei de stare, realizăm un salt
- atenție la cursul 6 - doar ultima soluție de implementare a automatului este corectă

Abordarea problemelor (3)

- un modul poate avea în același timp componente cu descriere structurală și respectiv comportamentală
 - dar acestea nu trebuie amestecate
 - descrierile comportamentale sunt materializate prin blocuri *initial* și *always*
- descrierile structurale indică în primul rând conexiunile între componente

Exemplu (1)

```
1 reg x,m;
2 wire y,z;
3 always @(posedge Clk)
4     if(stare==0) begin
5         x<=0;
6         assign y=~x;
7         and a(z,x,m);
8     end
. . .
```

Exemplu (2)

- liniile 6 și 7 sunt greșite
 - descriere structurală
 - nu trebuie declarate de fiecare dată, deoarece indică niște conexiuni permanente
- corect - plasarea lor înaintea blocului *always*
- în schimb, linia 5 este corectă
 - descriere comportamentală - indică o acțiune