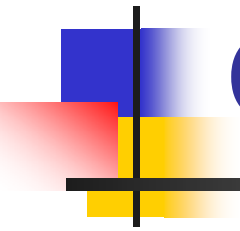
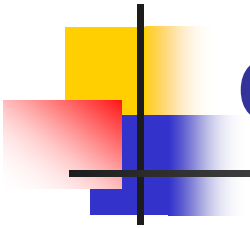


Circuite integrate digitale

Curs 10

Automate finite

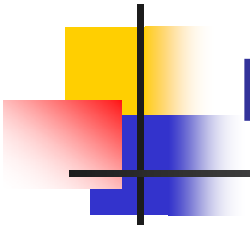


Cuprins

Proiectarea automatelor

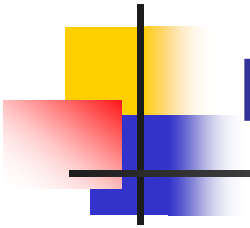
Descrierea automatelor în Verilog

Exemple



Etapele proiectării unui automat (1)

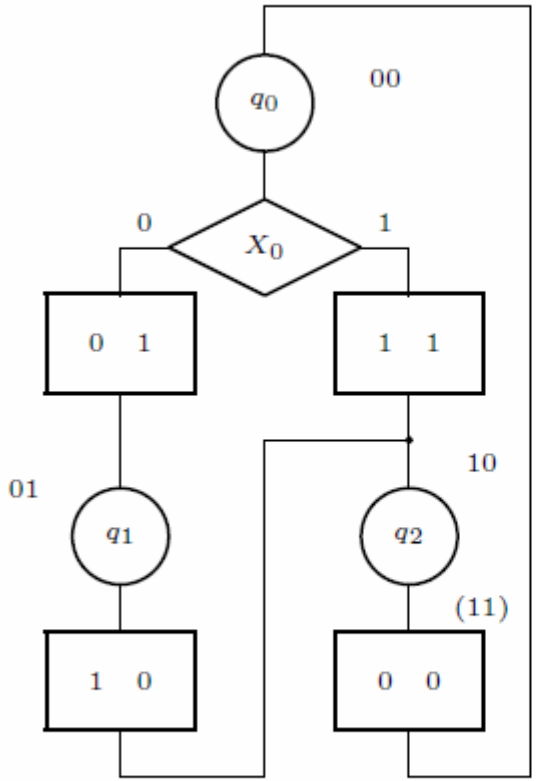
- se porneşte de la descrierea automatului şi se obţine o reprezentare (de regulă, organigramă binară sau graf)
- pe baza reprezentării şi a diferitelor cerinţe de proiectare se stabilesc:
  - mulţimile care definesc automatul (X, Y, Q)
  - tipul automatului care va fi implementat – schema de principiu – proiectarea constă în explicitarea schemei (sau descrierea ei în Verilog)
- codificarea stărilor



Etapele proiectării unui automat (2)

- deducerea funcţiilor de tranziţie a stărilor şi de calcul a ieşirii
  - se porneşte de la diagrama de referinţă pentru starea curentă
  - se completează diagrama stărilor următoare şi diagrama ieşirilor
  - se deduc funcţiile logice
  - dacă folosim Verilog, putem descrie tranziţiile
- schema finală, cu toate cerinţele

Exemplu de proiectare



Q_0 \ Q_1		
	1 1	0 1
	1 0	0 0

$Q_1\ Q_0$

Q_0 \ Q_1		
	- -	1 0
	0 0	$X_0\ X_0'$

$Q_1^+\ \ Q_0^+$

Q_0 \ Q_1		
	- -	1 0
	0 0	X_0 1

$Y_1\ Y_0$

Q_0 \ Q_1		
	0 0	1 1
	- -	X_0 1

$Q_1^+\ \ Q_0^+$

Q_0 \ Q_1		
	0 0	1 0
	- -	X_0 1

$Y_1\ Y_0$

Varianta 2

Varianta 1

$$Q_1^+ = Q_0 + X_0 Q_1'$$

$$Q_0^+ = Q_1' Q_0' X_0'$$

$$Y_1 = Q_0 + X_0 Q_1'$$

$$Y_0 = Q_1' Q_0'.$$

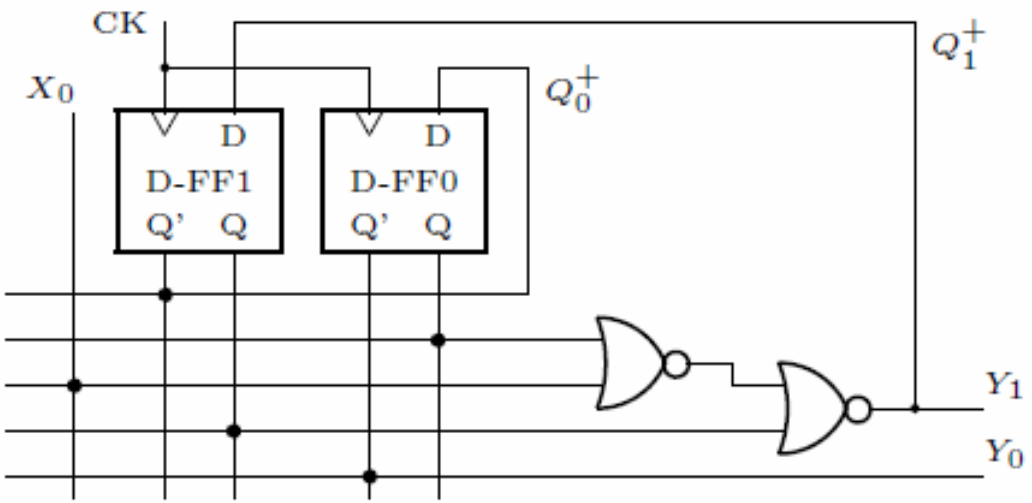
Varianta 2

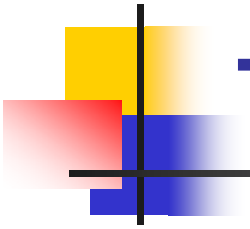
$$Q_1^+ = Q_1' Q_0 + X_0 Q_1' = (Q_1 + (Q_0 + X_0)')'$$

$$Q_0^+ = Q_1'$$

$$Y_1 = Q_1' Q_0 + X_0 Q_1' = (Q_1 + (Q_0 + X_0)')'$$

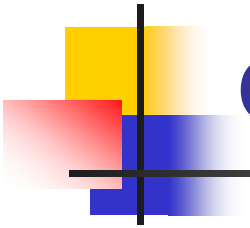
$$Y_0 = Q_0'.$$





Tema 11 (cursul 10)

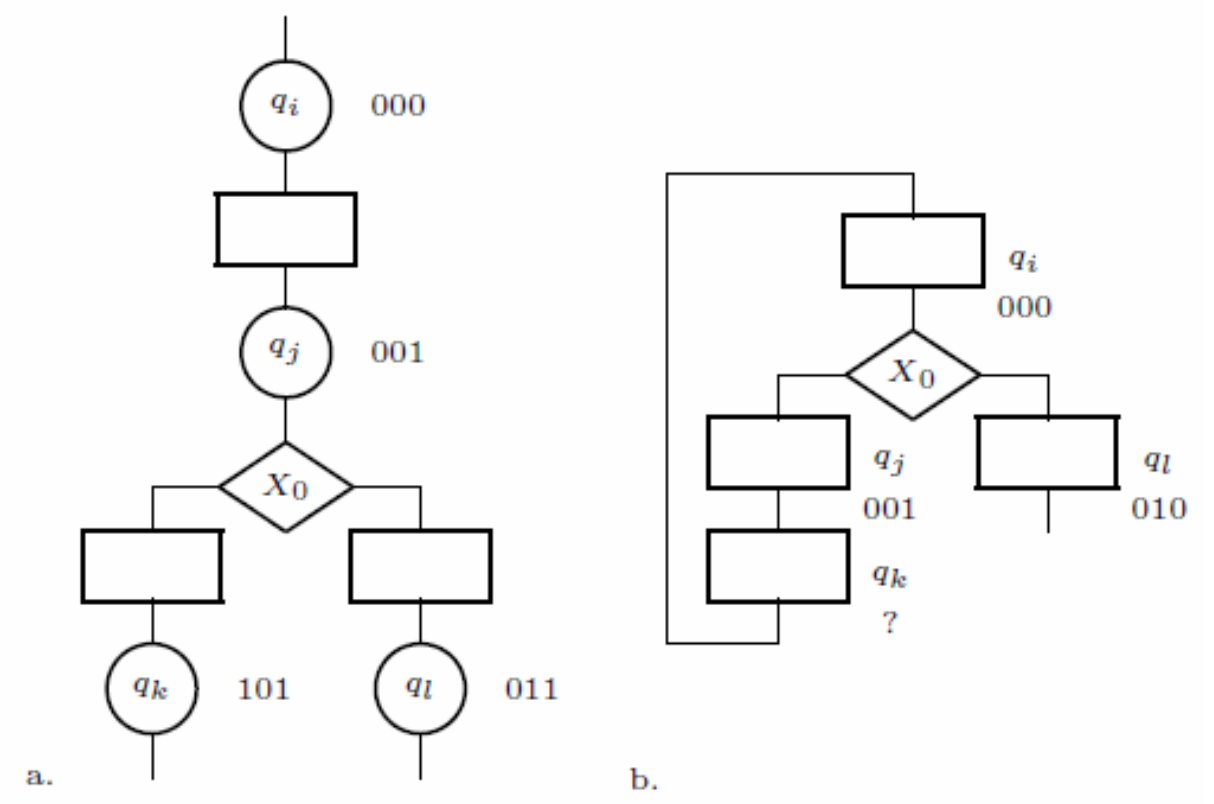
- Implementaţi automatul din exemplul anterior (slide-urile 14 şi 15 din cursul 10), într-una din cele două variante
  - soluţia completă conţine atât diagramele, cât şi circuitul cu porţi logice



Codarea stărilor

- Variaţie minimă: stările succesive au coduri adiacente
- Dependenţă redusă: stările care urmează după o aceeaşi stare au coduri adiacente
- Incrementare: stările succesive au coduri consecutive
- One-hot: o singură valoare de 1

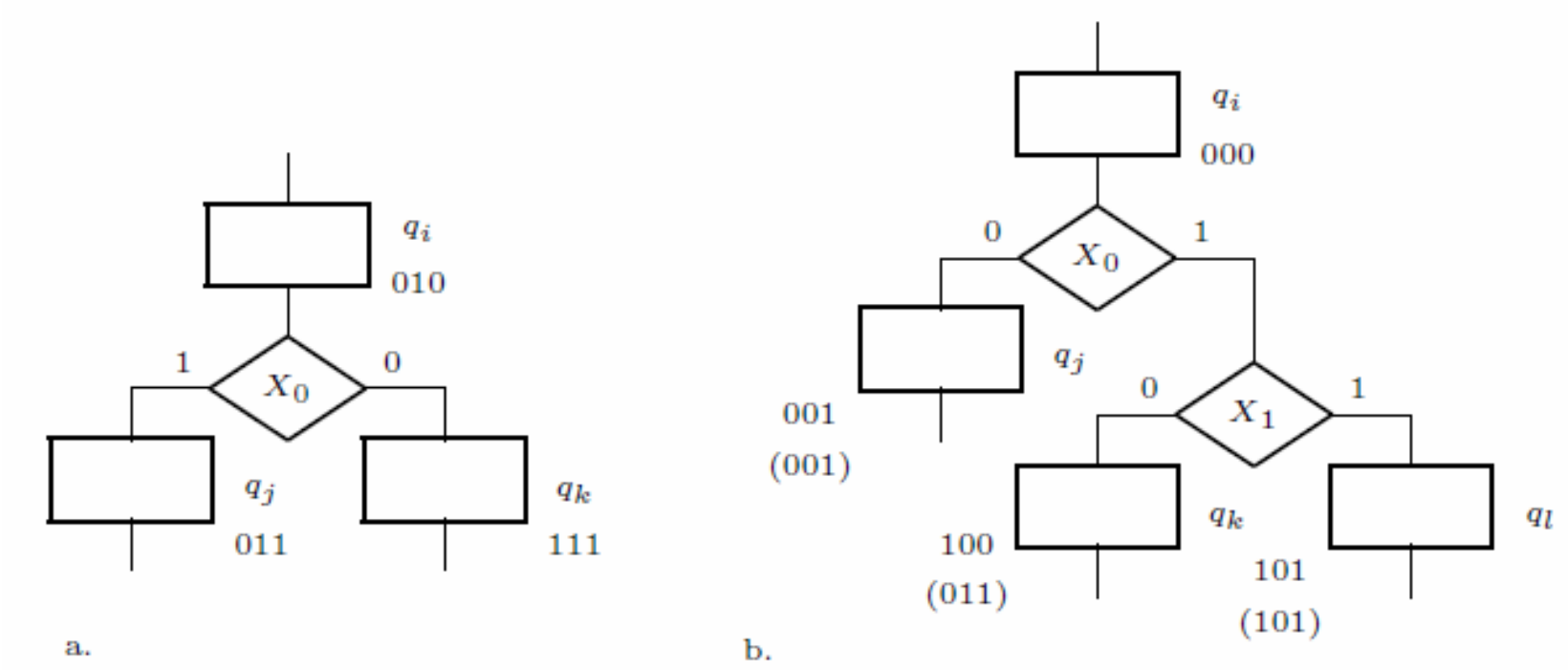
Codarea stărilor cu variaţie minimă



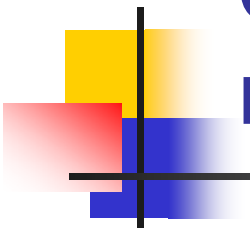
Hazardul la ieşirea automatelor

- comutări nedorite ale ieşirilor, datorate:
  - comutării biţilor de stare
  - propagării semnalelor în circuit
  - hazard dinamic, pentru circuite legate în cascadă
- soluţie:
  - Moore imediat - se codează stările cu variaţie minimă şi se ţine cont la implementarea funcţiilor de ieşire (suprafeţe suplimentare...)
  - se sincronizează ieşirile (automat cu întârziere)

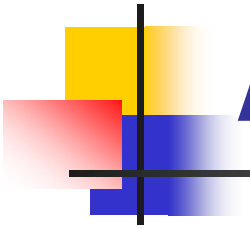
Codarea stărilor cu dependenţă redusă



Codarea stărilor cu dependenţă redusă



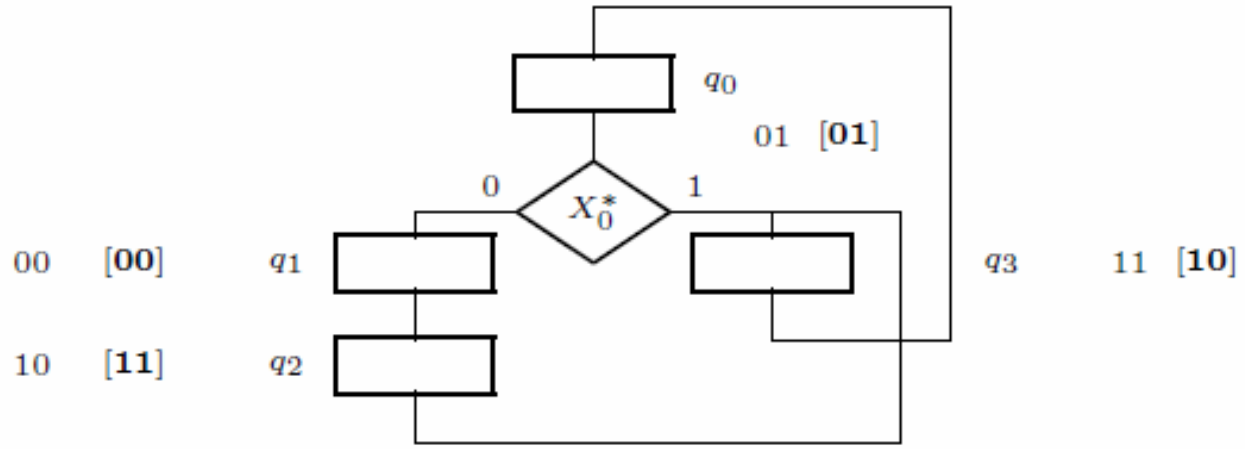
- stările care urmează după o aceeaşi stare au coduri adiacente
- se aplică în funcţie de o anumită variabilă sau mai multe
- nu se poate aplica pentru două variabile care sunt testate în aceeaşi stare



Automate cu intrări asincrone

- variabilele asincrone îşi pot modifica valoarea în intervalul "interzis" din jurul semnalului de ceas
- Codarea stărilor cu dependenţă redusă după variabilele asincrone
- nu se poate aplica dacă două variabile asincrone sunt testate în aceeaşi stare – se introduce o stare suplimentară

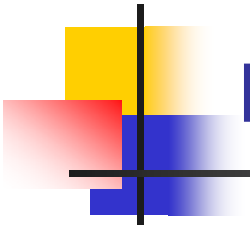
Exemplu



- dacă variabila se modifică în intervalul $t_{su} + t_{+} + t_h$

atunci nu este sigur că registrul de stare comută corect

- dacă diferă un singur bit de stare nu e o problemă
- dacă diferă doi, evoluţia în spaţiul stărilor este greşită

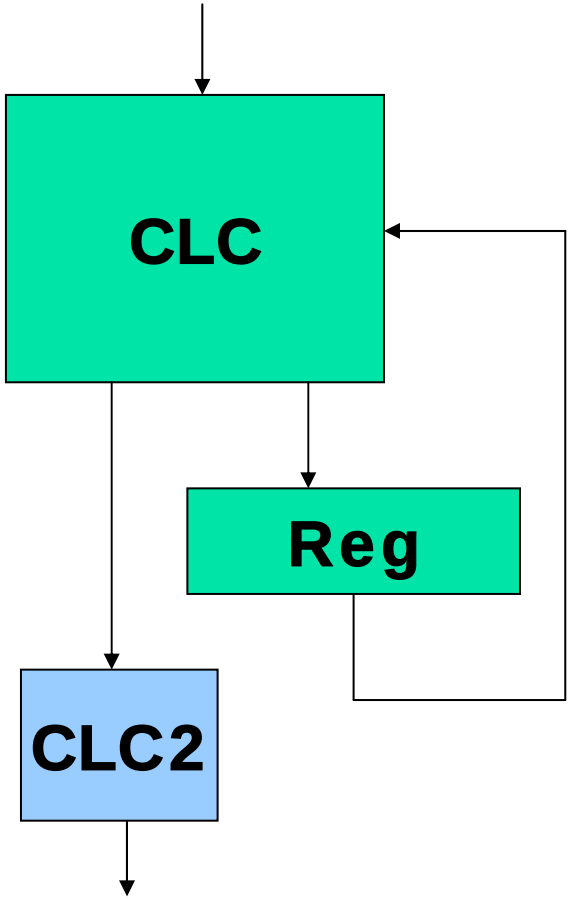


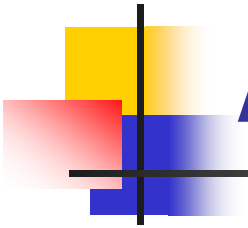
Descrierea automatelor în Verilog

- Variante posibile: automate Mealy, Moore, one-hot, one-cold, Huffman

Variante de automate

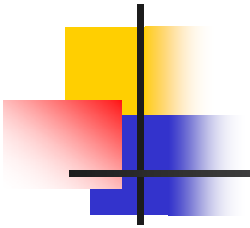
- Elementul central al schemei – registrul de stare (pentru utilizator este "invizibil")
- În funcţie de cum se calculează ieşirea, automatele pot fi:
  - Mealy
  - Moore
- După sincronizarea ieşirii
  - Imediate
  - Cu întârziere





Alte variante în Verilog

- După codurile binare alocate stărilor, one hot (sau one cold): în fiecare stare se modifică un singur bit
  - În fiecare stare, un singur bit de stare e 1 (respectiv 0)
  - Corespondent teoretic – varianta cea mai simplă a automatelor cu spaţiul stărilor structurat
  - De ce? Se simplifică funcţiile din CLC, chiar dacă avem mai mulţi biţi de stare
- Codare Huffman: se poate aplica oricăror circuite secvenţiale
  - Un bloc combinaţional care are o reacţie printr-un şir de registre

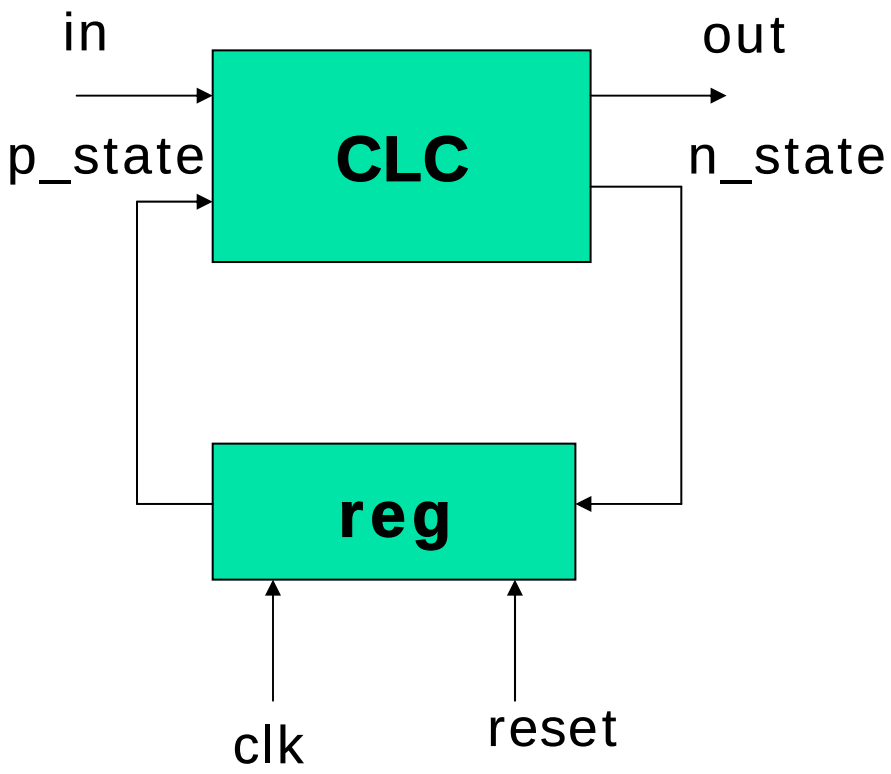


- Definirea stărilor – ca parametri sau parametri locali (sau cu define în afara modulului)

```
// State codes
  parameter q0 = 3'b000, // sau localparam
            q1 = 3'b001,
            q2 = 3'b010,
            q3 = 3'b011,
            q4 = 3'b100;
```

- Cea mai frecventă variantă de descriere Verilog: cu două blocuri always, instrucţiune case şi/sau assign
- http://www.asic-world.com/tidbits/verilog_fsm.html

Partiţionare Huffman pentru un automat Mealy



- Fiecare dintre cele două blocuri este definit printr-o instrucţiune always
-
```
always @ (p_state or in)
      begin: combinational
case (p_state)…
endcase
end
```
-
```
always @ (posedge clk)
      begin: register
      if (rst) p_state=reset;
      else p_state=n_state;
      end
```
- `assign out=…`

Exemplu: automat pentru comanda semafoarelor

```
/*************** Automaton for semaphors
   In the crosing point of two streets, a and b, there is a semaphor.
   The trafic on the street a is indicated by the value 1 of variable
   tr_a and for the trafic on the street b we have the variable tr_b.
   If green = 1, then the cross is accessible for the trafic from the
   street a. If green = 0, then the cross is accessible for the trafic
   from the street b. The clock of the automaton that triggers the
   semaphor has a period of 4 second. (let be 4 time units in our
   simulator). The semaphor stayes 20 seconds on red/green and only 4
   seconds on yelow.
   How behave the automaton?
      - if no trafic on both directions the light will changes
      - if trafic on both directions the light also will change
      - if trafic only on the not allowed directions the ligth will
   change
      - if trafic only on the allowed direction the light will not
   change. */
```

```
module trafic_light_aut(green, yellow, red, tr_a, tr_b, reset, clock);

  input    tr_a, // trafic on the direction a
           tr_b, // trafic on the direction b
           reset, clock;
   output   green, yellow, red; // each output for one collor

  reg      green, yellow, red; // variable used to compute the outputs
  reg[3:0] state_reg,  // the state register;
           next_state; // the variable used to compute the next state

  always @(state_reg or tr_a or tr_b)
    case (state_reg)
      4'b0000: {next_state, green, yellow, red} = {4'b0001, 1'b0, 1'b1, 1'b0};
   ….
   endcase

    always @(posedge clock) if (reset) state_reg = 0;
                            else     state_reg = next_state;
    endmodule
```

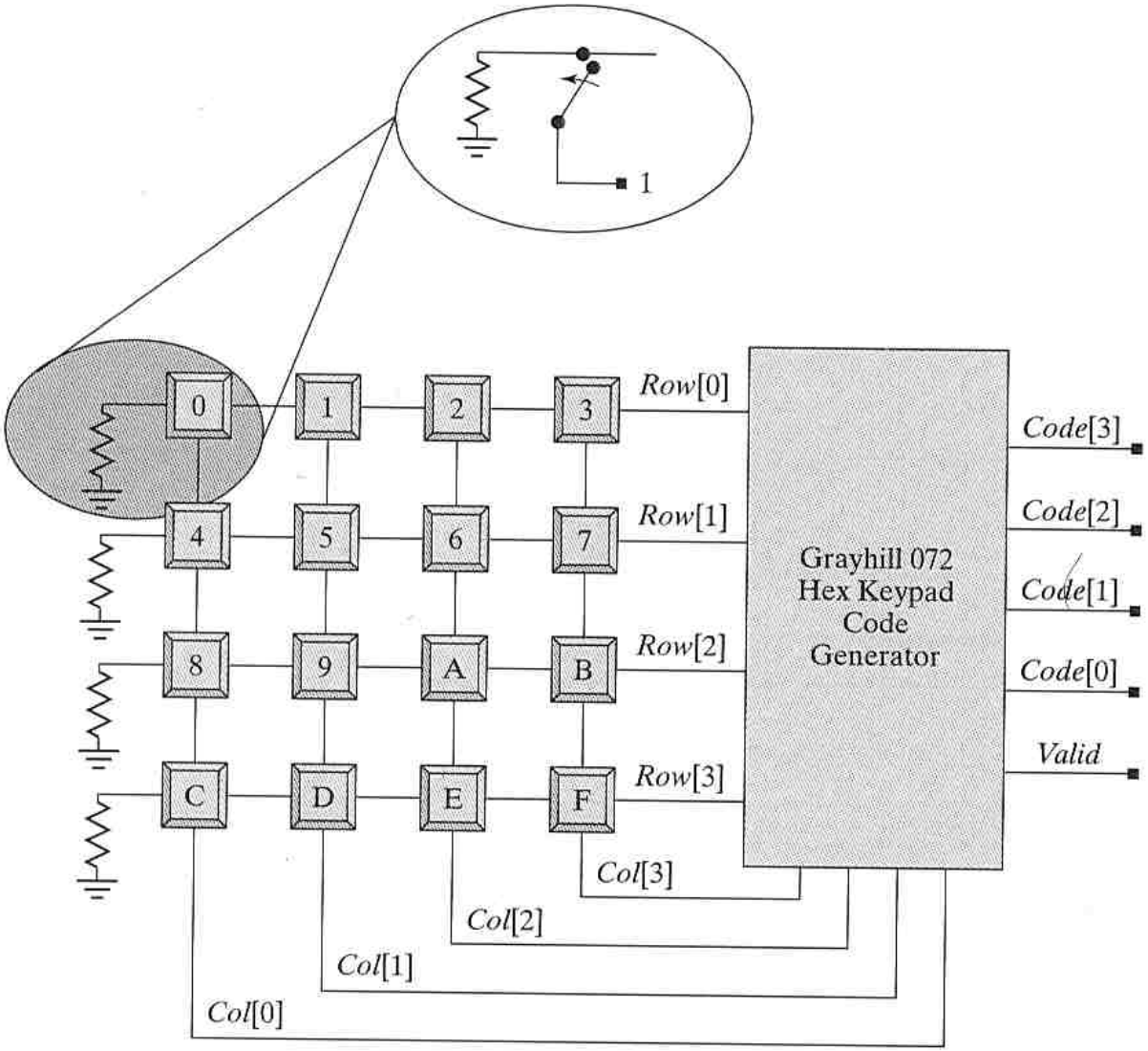
```
4'b0001: {next_state, green, yellow, red} = {4'b0010, 1'b0, 1'b0, 1'b1}; // free
    for cars from the street b
      4'b0010: {next_state, green, yellow, red} = {4'b0011, 1'b0, 1'b0, 1'b1};
      4'b0011: {next_state, green, yellow, red} = {4'b0100, 1'b0, 1'b0, 1'b1};
      4'b0100: {next_state, green, yellow, red} = {4'b0101, 1'b0, 1'b0, 1'b1};
      4'b0101: if (~tr_a && tr_b)
    // no trafic on a and trafic on b: light must not change
          {next_state, green, yellow, red} = {4'b0001, 1'b0, 1'b0, 1'b1}; //  red
          else {next_state, green, yellow, red} = {4'b1010, 1'b0, 1'b0, 1'b1};
                  // trafic will change
      ….
      4'b1111: if (tr_a && ~tr_b)
    // no trafic on b and trafic on a: light must not change
                  {next_state, green, yellow, red} = {4'b1011, 1'b1, 1'b0, 1'b0};
    // the light remains green
            else {next_state, green, yellow, red} = {4'b0000, 1'b1, 1'b0, 1'b0};
    // trafic will change
```

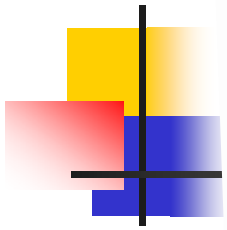
```
module trafic_aut_test;
  reg tr_a, tr_b, reset, clock;
  wire green, yellow, red;
  initial begin clock = 0;
                forever #2 clock = ~clock;
          end
  initial begin      reset = 1;
                     {tr_a, tr_b} = 2'b00;
                  #4  reset = 0;
                  #30 {tr_a, tr_b} = 2'b01;
                  #50 {tr_a, tr_b} = 2'b10;
                  #90 {tr_a, tr_b} = 2'b11;
                  #90 $stop;
          end
  trafic_light_aut  dut(green, yellow, red, tr_a, tr_b, reset, clock);

endmodule
```

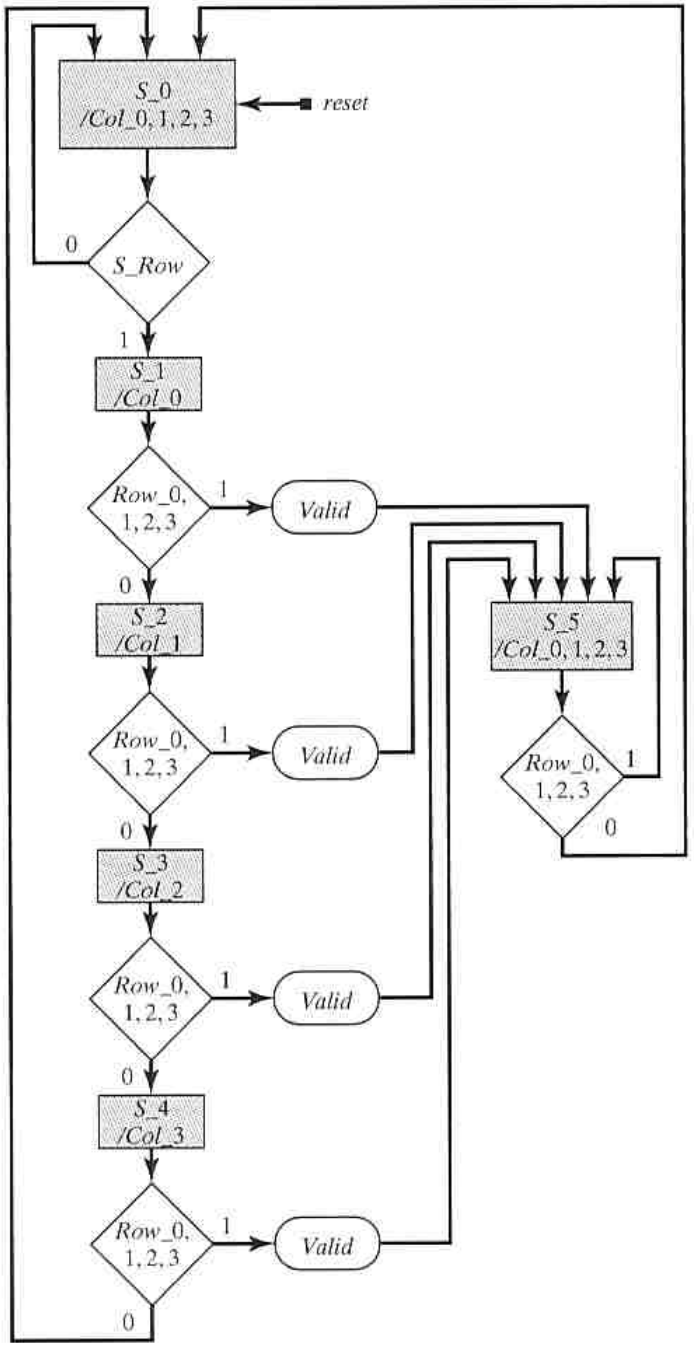
Scanner si codificator pentru o tastatura hexazecimala

- Trebuie sa detecteze daca este apasata vreo tasta,
- detecteaza care anume tasta este apasata,
- genereaza un cod unic pentru fiecare tasta si un semnal de validare.
- Daca o tasta este apasata lung nu se interpreteaza ca mai multe tastari
- Trebuie sa se tina cont de natura asincrona a intrarilor


1
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
Row[0]
Row[1]
Row[2]
Row[3]
Col[3]
Col[2]
Col[1]
Col[0]
Grayhill 072
Hex Keypad
Code
Generator
Code[3]
Code[2]
Code[1]
Code[0]
Valid



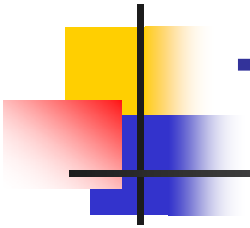
Key	*Row[3:0]*	*Col[3:0]*	Code
0	0001	0001	0000
1	0001	0010	0001
2	0001	0100	0010
3	0001	1000	0011
4	0010	0001	0100
5	0010	0010	0101
6	0010	0100	0110
7	0010	1000	0111
8	0100	0001	1000
9	0100	0010	1001
A	0100	0100	1010
B	0100	1000	1011
C	1000	0001	1100
D	1000	0010	1101
E	1000	0100	1110
F	1000	1000	1111


S_0
/Col_0, 1, 2, 3
reset
S_Row
0
1
S_1
/Col_0
Row_0,
1, 2, 3
1
Valid
0
S_2
/Col_1
Row_0,
1, 2, 3
1
Valid
0
S_3
/Col_2
Row_0,
1, 2, 3
1
Valid
0
S_4
/Col_3
Row_0,
1, 2, 3
1
Valid
0
S_5
/Col_0, 1, 2, 3
Row_0,
1, 2, 3
1
0

```
module Hex_Keypad_Grayhill_072(Code, Col, Valid, Row, S_Row, clock, reset);
    output [3:0] Code;
    output Valid;
    output [3:0] Col;
    input [3:0] Row;
    input S_Row;
    input clock, reset;
    reg [3:0] Col, Code;
    reg [5:0] state, next_state; //one-hot
    localparam S_0=6'b000001, S_1=6'b000010
    ….     //completati restul starilor
    assign Valid = ((state==S_1)||(…..  ))&&Row;

    always @ (Row or Col)                  // circuitul care calculeaza iesirile
          case ({Row, Col})
            8'b0001_0001: Code = 0;
             ….//etc restul codurilor
            default: …//
          endcase

    always @ (posedge clock or posedge reset)
          if (reset) state <=S_0;
          else state <= next_state;
    always @ (state or S_Row or Row) // next-state logic
                ……..
```



Tema 12

- Completaţi codul Verilog pentru calculul stării următoare
- Facultativ: simulaţi funcţionarea acestui automat