Seminar 5 - Logică digitală. Proiectarea şi implementarea automatelor cu stări finite (FSM - Finite State Machine).

Jura Ștefan-Alexandru
Facultatea de Automatică şi Calculatoare, specializarea Calculatoare.

1 Despre automatele finite - sinteză

Un automat cu stări finite (FSM - Finite State Machine) este un model matematic utilizat pentru a descrie comportamentul sistemelor de calcul și control.

Funcționarea unui automat cu stări finite implică mai mulți pași:

Definirea stărilor: Primul pas este de a defini toate stările posibile în care FSM poate fi. Acestea pot fi stări fizice, cum ar fi pornit/oprit, deschis/închis, sau pot fi stări logice, cum ar fi adevărat/fals, sau orice altceva care descrie starea sistemului.

Definirea intrărilor: După ce stările sunt definite, următorul pas este de a defini toate intrările posibile pe care FSM le poate primi. Acestea pot fi semnale de la senzori, date de la alte sisteme sau orice altceva care influențează starea sistemului.

Definirea regulilor de tranziție: Cu stările și intrările definite, următorul pas este de a defini regulile de tranziție între stări. Acest lucru se face de obicei printr-un tabel de tranziție, care arată ce starea următoare se va afla în funcție de starea curentă și intrarea primită.

Definirea ieșirilor: În cele din urmă, trebuie să fie definite toate ieșirile posibile ale sistemului în fiecare stare. Acestea pot fi semnale care controlează acțiunile sistemului sau semnale care raportează starea sistemului către alte sisteme.

Implementarea FSM: După ce au fost definite toate aceste elemente, FSM poate fi implementat folosind orice limbaj de programare sau platformă hardware care acceptă logica de stare finită.

Există 2 tipuri de automate: Automatele Mealy generează date de ieșire în funcție de starea curentă și de intrarea primită și automatele Moore ce generează date de ieșire în funcție numai de starea curentă.

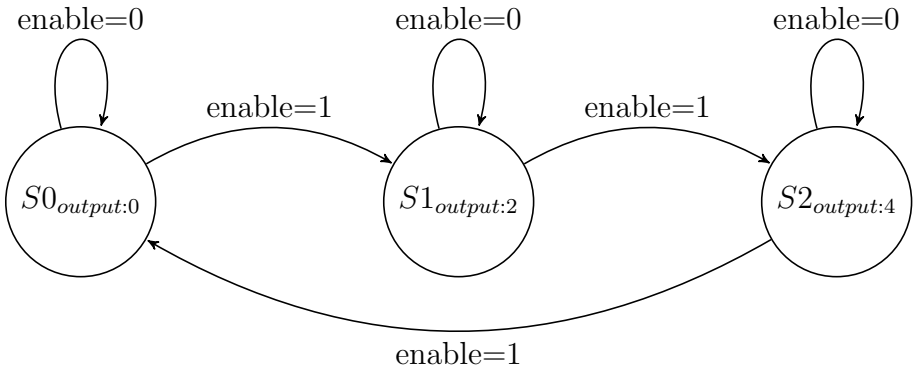
Odată ce FSM este implementat, el va funcționa astfel: la început, FSM va fi într-o stare inițială (stare resetată), apoi va primi intrări și va tranziționa într-o altă stare, în funcție de regulile de tranziție și starea curentă. În timpul fiecărei tranziții de stare, se va genera o ieșire, care poate fi utilizată pentru a controla acțiunile sistemului sau pentru a raporta starea acestuia.

FSM poate fi folosit pentru a modela multe tipuri de sisteme, cum ar fi circuitele electronice, roboții, aplicațiile software și multe altele. De obicei, FSM este utilizat pentru a descrie comportamentul unor sisteme care pot fi descrise prin seturi de stări discrete, cum ar fi procese de producție, circuite logice, sau sisteme de control automat.

2 Model - Implementarea unui automat de tip Moore.

Cerință: Să se implementeze un automat de tip Moore cu 3 stări. La semnalul *enable* = 0, se rămâne în starea curentă, iar la semnalul *enable* = 1 se trece în starea următoarea. Automatul va număra din 2 în 2, începând cu 0.

Soluție: Diagrama de stări a automatului este următoarea:



Se observă că automatul dat este de **tip Moore**, întrucât ieșirea depinde numai de starea circuitului. Pentru implementarea în limbajul Verilog HDL a acestui automat, vom realiza 3 tabele: tabelul de codificare al stărilor, tabelul de tranziții și tabelul de ieșiri, pentru a le putea transpune în cod. Pentru a înțelege implementarea acestui automat, este bine să fie cunoscute diferite noțiuni de bază legate de circuite secvențiale (circuite cu memorie, îndeosebi numărătoare și registre) în special, dar și de circuite combinaționale și algebră booleană. Realizăm pașii menționați la secțiunea anterioară și realizăm cele 3 tabele. Începem cu tabelul de codificare al stărilor, apoi, conform diagramei, realizăm tabelul de tranziții (Q_1Q_0 reprezintă starea curentă, iar D_1D_0 reprezintă starea următoare), iar în final, tabelul ieșirilor. Am ales să codific stările în valoarea lor binară, adică, de exemplu, 2 în binar este 10. Este suficient să folosim 2 biți pentru codificarea stărilor, deoarece cel mai mare număr (2) se scrie pe 2 biți. Dacă aveam și numărul 4, de exemplu, era necesar să utilizăm 3 biți, așa cum vom utiliza la ieșire.

Starea	Q_1	Q_0
S_0	0	0
S_1	0	1
S_2	1	0

Tabela 1: Codificarea stărilor

Q_1	Q_0	enable	D_1	D_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0

Tabela 2: Tabelul de tranziții

Q_1	Q_0	I_2	I_1	I_0
0	0	0	0	0
0	1	0	1	0
1	0	1	0	0

Tabela 3: Tabelul ieșirilor

Codul Verilog aferent automatului modelat este următorul:

```
1
2  `timescale 1ns/100ps
3  module fsm(
4      input wire clk,
5      input wire rst_b,
6      input wire enable,
7      output reg [2:0] out
8  );
9
10
11 `define S0 2'b00
12 `define S1 2'b01
13 `define S2 2'b10
14
15 reg [1:0] st, st_nxt;
16
17
18 always @(*) begin
19     case (st)
20         `S0: begin
21             out=0;
22             if (enable) st_nxt = `S1;
23             else st_nxt = `S0;
24         end
25         `S1: begin
26             out = 2;
27             if (enable) st_nxt = `S2;
28             else st_nxt = `S1;
29         end
30         `S2: begin
31             out = 4;
32             if (enable) st_nxt = `S0;
33             else st_nxt = `S2;
34         end
35     endcase
36 end
37
38
```

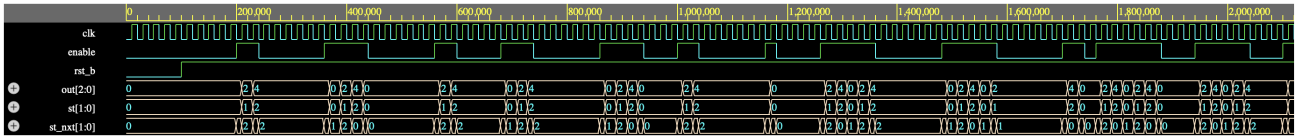
```
always @(posedge clk or negedge rst_b) begin
    if (~rst_b) st <= `S0;
    else st <= st_nxt;
end

endmodule
```

Codul testbench-ului pentru acest FSM (simulat în EDA Playground) este:

```
module fsm_tb;

  reg enable;
  reg clk, rst_b;
  wire [2:0] out;
  integer i;

  parameter PERIOD = 10;

  fsm uut (.clk(clk), .rst_b(rst_b), .enable(enable), .out(out));

  always #PERIOD clk=~clk; //create the cyclic clock signal

  initial begin
    $dumpfile("waveform.vcd");
    $dumpvars(1, clk, rst_b, enable);
    $dumpvars(1, uut.st);
    $dumpvars(1, uut.out);
    $dumpvars(1, uut.st_nxt);
  end

  initial begin
    //initialize variables and activate reset
    rst_b = 0;
        enable =0;
        clk = 0;
        #100;
        //deactivate reset to start normal operation
    rst_b = 1;

        @(posedge clk);
```

```
32      $display("[%0t]Start", $time);
33          @(negedge clk);
34          for (i=0; i<25; i= i+1)
35          begin
36                  repeat ($urandom % 6) @(negedge clk);
37                  enable = ~enable;
38                  $display("[%0t]Drive value %b for input enable...",
39    $time, enable);
40                  @(negedge clk);
41          end
42          $display("[%0t]Done!", $time);
43
44      $finish;
45    end
46  endmodule
```

Diagrama de timp obţinută în urma rulării codului este:



Cu ajutorul diagramei de timp, putem observa funcţionarea automatului. Reset-ul asincron este activ pe 0 logic, fiind independent de clock. La fiecare front crescător al semnalului de clock, unde reset-ul este 1, în funcţie de starea curentă şi de semnalul enable, automatul îşi schimbă starea, aşa cum poate fi observat în figură.

Se observă că am utilizat un registru pentru stocarea biților. În rest, codul este unul destul de trivial, fiind asemănător cu instrucțiunea *switch*(*case*) din limbajul de programare *C*.

Automatul respectiv se poate modela folosind circuite secvenţiale, mai exact flip flop-uri. Se ştie că cele 4 tipuri de flip flop sunt SR (set-reset), JK, D şi T (de la "toggle" - acelaşi lucru, efect opus faţă de flip flop-ul de tip D). Pentru simplitate, vom utiliza un flip flop de tip D. Pentru realizarea design-ului, se realizează un tabel "all in one", care cuprinde starea curentă, input-ul (în cazul nostru, semnalul enable), starea următoare, intrările pentru fiecare flip flop şi ieşirea y. Cum avem intrări codificate pe 2 biţi, vom utiliza 2 flip flop-uri, câte unul pentru fiecare bit. Tabelul este următorul (primele 2 coloane

reprezintă current state-ul, a doua reprezintă inputul, în cazul nostru enable-ul, următoarele 2 starea următoare, următoarele intrările pentru FF-uri, iar coloanele cu y reprezintă ieşirile pentru automat la fiecare stare în funcţie de starea circuitului, fiind un automat de tip Moore):

Reamintim că pentru un flip flop de tip D, ecuaţia de stare este $Q_{next} = D$

Q_1	Q_0	$enable$	Q_1	Q_0	D_1	D_0	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	0	1
0	1	0	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	1	0	0
1	0	1	0	0	0	0	0	0	0

Tabela 4: Tabelul pentru implementarea cu flip flop-uri de tip D.

, iar dacă $D = 0$, atunci $Q_{next} = 0$ şi dacă $D = 1$, atunci $Q_{next} = 1$ Pentru intrările flip flop-ului se realizează hărti Veich-Karnaugh şi se minimizează funcţiile boolene. Intrările flip flop-urilor vor fi funcţii ce depind de starea curentă şi de input-ul "enable", iar ieşirile $y_k\,, k = \overline{0,2}$ sunt funcţii ce depind de starea curentă, întrucăt avem un automat Moore. În urma minimizării, se obţin următoarele expresii boolene:

$$D_1 = \overline{Q_1}Q_0 enable + Q_1\overline{Q_0 enable}$$

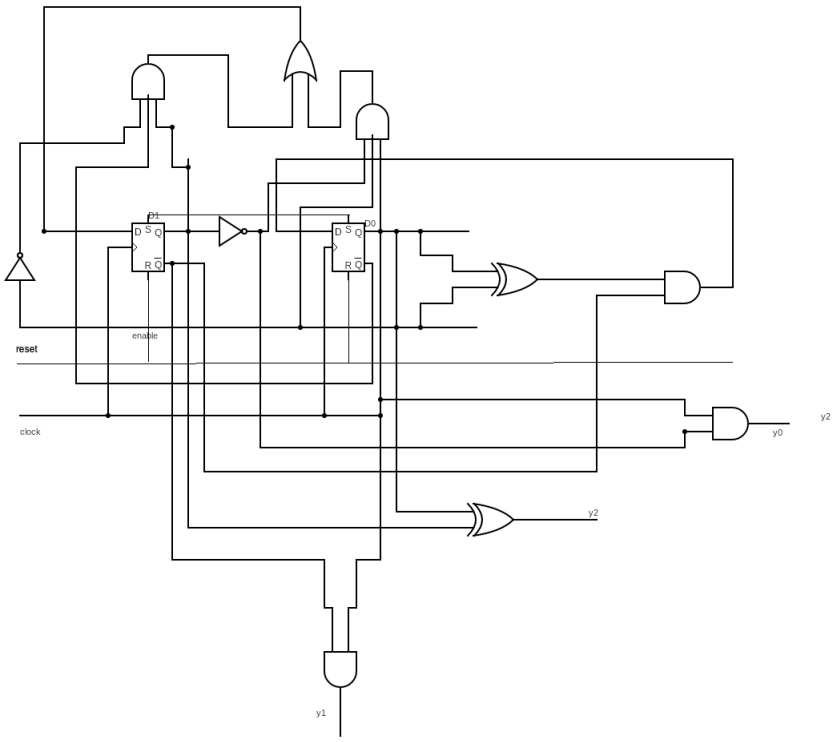
$$D_0 = \overline{Q_1} \cdot (Q_0 \oplus enable)$$

$$y_2 = Q_1 \oplus Q_0$$

$$y_1 = \overline{Q_1}Q_0$$

$$y_0 = \overline{Q_1 Q_0}$$

Diagrama circuitului este



3 Observaţii legate de segmentele combinaţionale şi secvenţiale

Partea de logică combinaţională este automat făcută de către tool-ul de sinteză Verilog. Sunt folosite 2 instrucţiuni de tip "always". Structura "always" este folosită pentru a crea blocuri de cod care vor fi executate de fiecare dată când condiția specificată devine adevărată sau când evenimentele specificate au loc. Este una dintre cele mai importante și folosite structuri în proiectarea circuitelor digitale folosind Verilog. Modelarea Verilog cu "always" se numeşte modelare comportamentală.
În cazul nostru, primul bloc "always" are rolul de a modela un registru ce reţine stările FSM-ului. Cel de-al doilea always are rolul de a "traversa" stările automatului, de fiecare dată actualizându-se ieşirea în funcţie de starea în care se află automatul (se ilustrează, astfel, caracterul Moore al automatului).

Reset-ul ales este asincron (prin intermediul instrucţiunii "posedge sau negedge rst", care face ca circuitul să ajungă într-o stare cunoscută imediat la întâlnirea semnalului de reset, independent de semnalul de clock). Resetul este activ pe 0 logic, aşa cum reiese din instrucţiunea $if(!rst)$. Dacă aveam $if(rst)$, însemna că, la momentul când resetul este 1, în acel moment se trece într-o stare cunoscută. Aşa cum am învăţat, semnalul de clock este un punct de referință pentru momentele în care intrările circuitului sunt evaluate și când ieșirile sunt actualizate, la fiecare semnal de clock/tact al FF-urilor modificându-se starea în funcţie de input-uri (pe frontul crescător al FF-ului, adică pe porţiunea unde se trece din 0 logic în 1 logic). Evident, dacă resetul este activ, atunci se trece în starea cunoscută inițială S_0.

4 Implementare pe placa FPGA Altera DE-10 LITE

Pentru intrări, se folosesc următorii pini:

Table 3-4 Pin Assignment of Slide Switches

Signal Name	FPGA Pin No.	Description	I/O Standard
SW0	PIN_C10	Slide Switch[0]	3.3-V LVTTL
SW1	PIN_C11	Slide Switch[1]	3.3-V LVTTL
SW2	PIN_D12	Slide Switch[2]	3.3-V LVTTL
SW3	PIN_C12	Slide Switch[3]	3.3-V LVTTL
SW4	PIN_A12	Slide Switch[4]	3.3-V LVTTL
SW5	PIN_B12	Slide Switch[5]	3.3-V LVTTL
SW6	PIN_A13	Slide Switch[6]	3.3-V LVTTL
SW7	PIN_A14	Slide Switch[7]	3.3-V LVTTL
SW8	PIN_B14	Slide Switch[8]	3.3-V LVTTL
SW9	PIN_F15	Slide Switch[9]	3.3-V LVTTL

Pentru ieşiri (pe led-uri), se folosesc pinii:

Table 3-5 Pin Assignment of LEDs

Signal Name	FPGA Pin No.	Description	I/O Standard
LEDR0	PIN_A8	LED [0]	3.3-V LVTTL
LEDR1	PIN_A9	LED [1]	3.3-V LVTTL
LEDR2	PIN_A10	LED [2]	3.3-V LVTTL
LEDR3	PIN_B10	LED [3]	3.3-V LVTTL
LEDR4	PIN_D13	LED [4]	3.3-V LVTTL
LEDR5	PIN_C13	LED [5]	3.3-V LVTTL
LEDR6	PIN_E14	LED [6]	3.3-V LVTTL
LEDR7	PIN_D14	LED [7]	3.3-V LVTTL
LEDR8	PIN_A11	LED [8]	3.3-V LVTTL
LEDR9	PIN_B11	LED [9]	3.3-V LVTTL

Pentru a pune pe placă circuitul, se descarcă de pe Campus Virtual dosarul "Altera cmd line script example" şi se modifică fişierul Verilog din cadrul acelui folder. Fişierul "prj.tcl" conţine asocierea cu pinii de pe placă. Acesta este:

```
project_new example1 -overwrite
set_global_assignment -name FAMILY MAX10
set_global_assignment -name DEVICE 10M50DAF484C7G
set_global_assignment -name BDF_FILE example1.bdf
set_global_assignment -name VERILOG_FILE let2sw.v
set_global_assignment -name SDC_FILE example1.sdc
set_global_assignment -name TOP_LEVEL_ENTITY led2sw
set_location_assignment -to clk PIN_AH10
set_location_assignment PIN_C10 -to sw_i ;#SW[0]
set_location_assignment PIN_A8 -to led_o ;#LED[0]
load_package flow
execute_flow -compile
project_close
```

Numele fişierului ("VERILOG_FILE") îl lăsam la fel, în timp ce la "TOP_LEVEL_ENTITY" punem numele modulului pe care l-am creat în interiorul fişierului Verilog. După ce facem modificările necesare (asociem fiecare intrare şi ieşire cu pinii corespunzători etc.), deschidem un terminal Ubuntu în directorul curent şi tastăm următoarele comenzi pentru a putea implementa circuitul pe plaga FPGA:

```
quartus_sh -t prj.tcl
quartus_pgm -l
```

quartus_pgm –auto
quartus_pgm -m JTAG -o ”p;example1.sof@1”

5 Model de problemă care implică utilizarea unui automat.

În cadrul acestui seminar, ne vom concentra asupra proiectării unui automat, dându-ne seama de funcţionalitatea lui din cerinţele specificate în enunţ. În cadrul seminariilor următoare, vom proiecta circuite logice care necesită atât proiectarea unui FSM, dar şi includerea unor alte elemente, precum multiplexoare, decodificatoare etc.

Cerinţă:

O companie specializată în sisteme de securitate dorește să îmbunătățească securitatea accesului în clădirile corporative ale clienților săi printr-un nou sistem de autentificare bazat pe un model de semnale unice. Acest sistem va folosi un dispozitiv de scanare care recunoaște o secvență specifică de semnale introdusă de utilizatori pentru a permite accesul în diferite zone securizate ale clădirii.

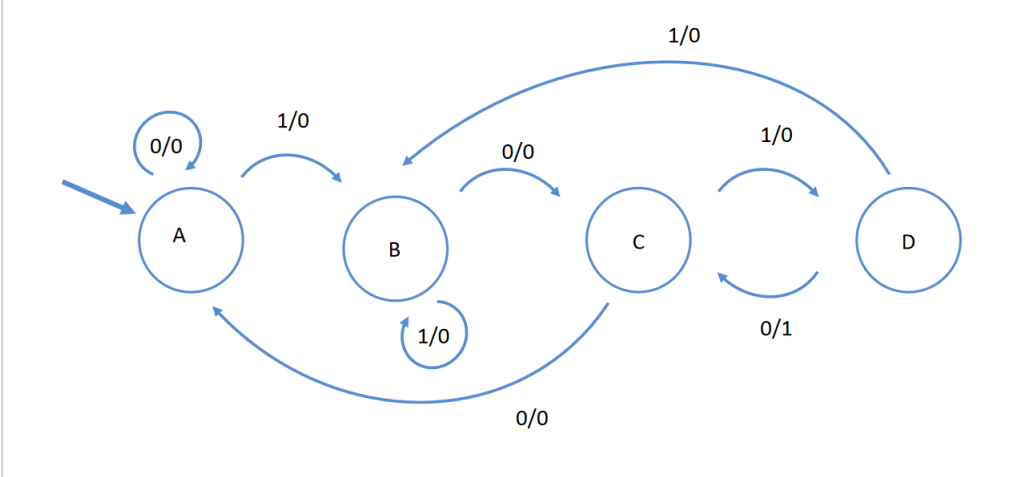
Secvența cheie aleasă de companie este ”1010”, considerată suficient de complexă pentru a preveni accesul neautorizat, dar și ușor de reținut de către angajații autorizați. Dispozitivul de scanare este proiectat să monitorizeze și să înregistreze secvența de semnale introduse, activând mecanismul de deblocare a ușii numai atunci când secvența introdusă corespunde exact cu modelul ”1010”. Pentru orice altă secvență introdusă, accesul va fi refuzat, iar sistemul va înregistra o încercare de acces neautorizat.

Pentru a implementa acest sistem, compania trebuie să dezvolte un automat finit care să fie capabil să recunoască secvența ”1010” în fluxul continuu de semnale introduse de utilizatori. Acest automat va trebui să distingă între secvențele corecte și cele incorecte, generând un semnal de ”1” pentru deblocare în cazul identificării corecte a secvenței și un semnal de ”0” pentru a menține blocarea accesului în caz contrar.

Soluţie:

Pentru rezolvarea acestor probleme, trebuie să ”citim printre rânduri”. Trebuie să ne dăm seama din enunţ că este vorba despre un sequence detector, care detectează secvenţa binară 1010 şi are ca output 1 de fiecare dată

când secvenţa este detectată. De exemplu, dacăm ca input 00110101000, la ieşire vom avea 00000010100. Schema automatului (de tip Mealy) este:



Tabelul ce realizează codificarea stărilor este:

State Table

A = Nothing has been seen	00
B = A '1' has been seen	01
C = A '10' has been seen	10
D = A '101' has been seen	11

Tabelul de tranziţii, alături de ieşiri, este:

Current and Next State Table (along with the Output)

Q1	Q0	Input	Q1+	Q0+	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	0	1	0

D values required

D1	D0
0	0
0	1
1	0
0	1
0	0
1	1
1	0
0	1

Utilizând hărţi Veich-Karnaugh, vom obţine următoarele expresii boolene minimizate:

D1 Generation Table (from Q1/Q0/Input values)

	Q1/Q0			
Input	00	01	11	10
0	0	1	1	0
1	0	0	0	1

D1 = (Q0 & 'X) | (Q1 & 'Q0 & In)

D0 Generation Table (from Q1/Q0/Input values)

	Q1/Q0			
Input	00	01	11	10
0	0	0	0	0
1	1	1	1	1

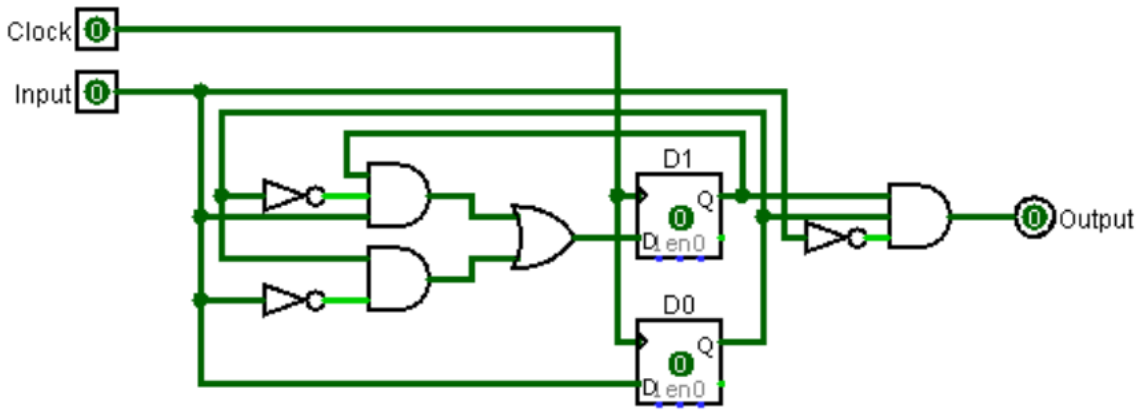
D0 = In

Output Generation Table (from Q1/Q0/Input values)

	Q1/Q0			
Input	00	01	11	10
0	0	0	1	0
1	0	0	0	0

Out = Q1 & Q0 & 'In

Schema circuitului, automatul fiind implementat cu bistabile de tip D, este:



6 Temă.

1) Să se implementeze în Verilog acest automat şi să se realizeze ţestbench-ul şi diagrama de timp.
2) Să se proiecteze un controler pentru un lift. Liftul poate fi la unul dintre cele două etaje: parter sau etajul 1. Există un buton care controlează liftul, și are două valori: Sus (Up) sau Jos (Down). De asemenea, există două lumini în lift care indică etajul curent: roșu pentru parter și verde pentru primul etaj. La fiecare ciclu de tact, controlerul verifică etajul curent și

intrarea curentă, schimbă etajele și luminile în modul evident. Pentru implementare, se vor folosi bistabile de tip J-K. (NU se cere codul Verilog, ci doar implementarea "pe foaie" a circuitului).

Bibliografie

[1] David A. Patterson, John L. Hennessy (2014) *Computer Organisation and Design: The Hardware/Software Interface, Fifth Edition*, Elsevier.

[2] Mircea Vlăduţiu (2012) *Computer Arithmetic: Algorithms and Hardware Implementations*, Springer.

[3] M. Morris Mano, Michael D. Ciletti (2012), *Digital Design with an introduction to Verilog HDL, Fifth Edition*, Colorado Springer.