Lecţia 4:
Proiectarea circuitelor logice

G Stefănescu — Universitatea Bucureşti

Arhitectura sistemelor de calcul, Sem.1
Octombrie 2016—Februarie 2017

După: D. Patterson and J. Hennessy, Computer Organisation and Design

Proiectarea circuitelor logice

Cuprins:

- *Porti, tabele de adevar, ecuatii logice*
- Logica combinationala
- Ceas
- Elemente de memorie
- Masini de stari finite
- Concluzii, diverse, etc.

Circuite digitale

Circuite digitale:

- Calculatoarele moderne sunt *digitale* (nu analogice).
- Ele folosesc sistemul binar pentru că se portiveşte cu modelul abstract din electronică:

  1 (ori *true*) pentru semnal activat şi *0* (ori *false*) pentru semnal dezactivat.

- Proiectarea logică are 2 nivele: proiectarea de *circuite combinaţionale* (fără memorie) şi proiecarea de *circuite secvenţiale* (cu memorie).

Tabele de adevar

Tabele de adevar:

- O *funcţie logică* poate fi definită *semantic* (i.e., prin valori).
- Spre exemplu, fie $f : (A,B,C) \mapsto (D,E,F)$ definită "verbal" astfel:
  - –D este 1 când cel puţin un argument este 1;
  - –E este 1 când exact două argumente sunt 1;
  - –F este 1 când toate argumentele sunt 1.
- In tabelul alăturat este definită *tabela de adevăr* pentru f.
- In stânga avem toate combinaţiile 0/1 pentru A,B,C; în dreapta sunt valorile rezultate pentru D,E,F.

A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Ecuatii logice; algebra Boole

Ecuatii logice:

- O *funcţie logică* poate fi definită şi *sintactic* (i.e., prin relaţii între variabile).
- Folosim următoarele operaţii de bază (numite *negaţie*, *disjuncţie / sumă logică*, şi *conjuncţie / produs logic*):
  - *NOT* (notat ca în $\overline{A}$): $\overline{A}$ este 1 când A este 0, altfel 0.
  - *OR* (notat "+"): $A+B$ este 1 când A ori B este 1, altfel 0.
  - *AND* (notat "·"): $A \cdot B$ este 1 când A şi B sunt 1, altfel 0.
- Putem specifica funcţia de mai sus cu *ecuaţii logice* astfel:

$$D = A + B + C$$
$$E = (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot C)$$
$$F = A \cdot B \cdot C$$

Ecuatii logice; algebra Boole

Algebra Boole: Identităţi utile, definind *algebra Boole*:

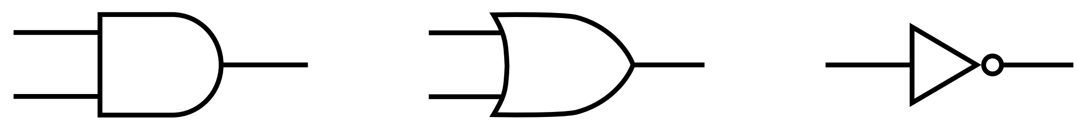
- Legile *identităţilor*: $A+0=A$; $\quad A\cdot 1=A$;
- Legile lui *0/1*: $A+1=1$; $\quad A\cdot 0=0$;
- Legile de *comutativitate*: $A+B=B+A$; $\quad A\cdot B=B\cdot A$;
- Legile de *asociativitate*: $(A+B)+C=A+(B+C)$; $(A\cdot B)\cdot C=A\cdot(B\cdot C)$;
- Legile de *distributivitate*: $A\cdot(B+C)=(A\cdot B)+(A\cdot C)$; $A+(B\cdot C)=(A\cdot B)+(A\cdot C)$;
- Legile de *complementaritate*: $A+\overline{A}=1$; $\quad A\cdot\overline{A}=0$.

Nota: Ecuatiile din pagina anterioara au folosit implicit astfel de reguli, spre exemplu omitand unele paranteze.

Porti

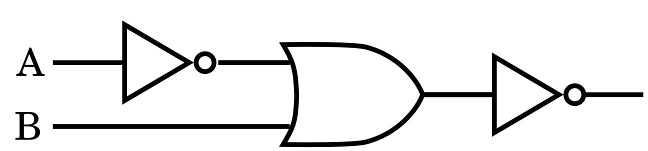
Porti:

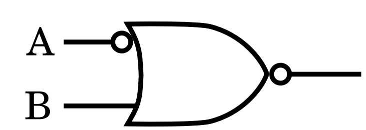
- Blocurile logice se construiesc din *porţi* (engl. *gates*) care implementează operaţiile de bază AND, OR, şi NOT.
- Convenţie: *Desenul standard* pentru AND, OR, şi NOT este



Argumentele vin din stânga, iar rezultatul este în dreapta.

- Un exemplu de implementare pentru $\overline{\overline{A}+B}$ este





In dreapta este un desen echivalent în care desenul *negaţiei* este simplificat.

Universalitate

Universalitate: Avem următorul rezultat teoretic de universalitate:

Teoremă: Toate funcţiile boolene (de tipul $\{0,1\}^n \to \{0,1\}$*) pot fi construite cu operaţiile de bază cu desene ca mai sus.*

O posibila demonstraţie se poate obţine folosind forma normală "pe doua nivele", introdusă mai jos.

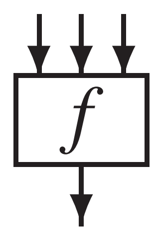
Observatii:

- *Numarul de operatii de baza poate fi micsorat daca se combina NOT cu AND si OR.*
- *Exemple: Operatiile NAND si NOR sunt universale, unde* $NAND(A,B) = \overline{A \cdot B}$ *si* $NOR(A,B) = \overline{A+B}$.

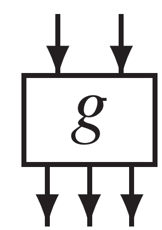
Reprezentare textuala a retelelor

Operatii cu retele (ori circuite):

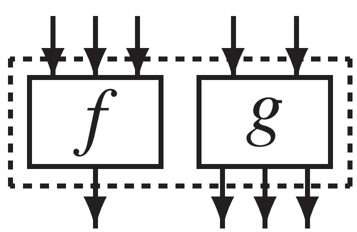
- Operatiile de bază sunt *juxtapunerea* "$\star$", *compunerea* "$\cdot$" şi *feedback-ul* "$\uparrow$".
- Ele au următoarea interpretare:



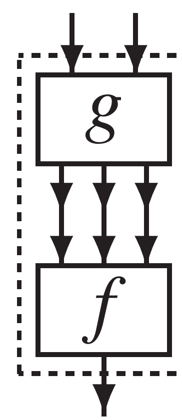
$f : 3 \rightarrow 1$



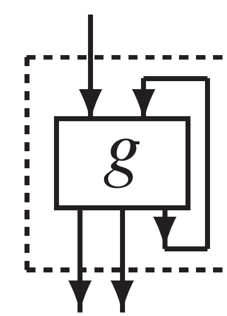
$g : 2 \rightarrow 3$



$f \star g : 5 \rightarrow 4$



$g \cdot f : 2 \rightarrow 1$



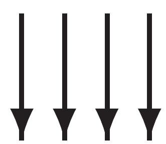
$g \uparrow^1 : 1 \rightarrow 2$

..Reprezentare textuala a retelelor

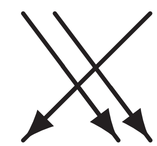
Reprezentarea relatiilor finite:

- Se poate demonstra că orice relaţie finită se obţine cu juxtapunere şi compunere din nişte relaţii elementare: *identităţi* $\mathsf{I}_a : a \to a$, *transpoziţii* ${}^a\mathsf{X}^b : a+b \to b+a$, *ramificări* $\wedge_k^a : a \to ka$, şi *idenitificări* $\vee_a^k : ka \to a$.

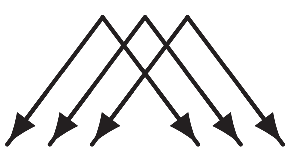
- In desen sunt ilustrate $\mathsf{I}_4, {}^2\mathsf{X}^1, \wedge^3 =_{\text{def}} \wedge_2^3, \perp^3 =_{\text{def}} \wedge_0^3, \vee_2 =_{\text{def}} \vee_2^2, \top_2 =_{\text{def}} \vee_2^0$



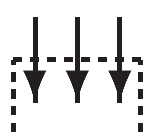
$\mathsf{I}_4 : 4 \to 4$



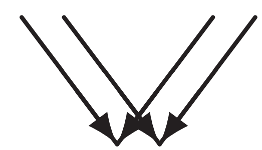
${}^2\mathsf{X}^1 : 3 \to 3$



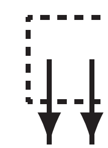
$\wedge^3 : 3 \to 6$



$\perp^3 : 3 \to 0$



$\vee_2 : 4 \to 2$



$\top_2 : 0 \to 2$

..Reprezentare textuala a retelelor

Expresii: *Expresiile de reţele* se obţin astfel:

- ca elemente de plecare se folosesc *variabile* $x : a \to b$ şi *relaţii finite* $f : a \to b$;
- se aplica operaţiile de *juxtapunere*, *compunere*, şi *feedback*.

Exemplu:

- Desenul anterior (pentru $\overline{\overline{A}+B}$) se reprezintă cu expresia $(NOT \star \mathsf{I}_1) \cdot OR \cdot NOT$.

Algebra de retele

- Algebra *BNA* (Basic Network Algebra) este dată de ecuaţiile:

I. Axiome fara feedback

B1	$f \star (g \star h) = (f \star g) \star h$
B2	$\mathsf{I}_0 \star f = f = f \star \mathsf{I}_0$
B3	$f \cdot (g \cdot h) = (f \cdot g) \cdot h$
B4	$\mathsf{I}_a \cdot f = f = f \cdot \mathsf{I}_b$
B5	$(f \star f') \cdot (g \star g') = (f \cdot g) \star (f' \cdot g')$
B6	$\mathsf{I}_a \star \mathsf{I}_b = \mathsf{I}_{a+b}$
B7	${}^a\mathsf{X}^b \cdot {}^b\mathsf{X}^a = \mathsf{I}_{a+b}$
B8	${}^a\mathsf{X}^{b+c} = ({}^a\mathsf{X}^b \star \mathsf{I}_c) \cdot (\mathsf{I}_b \star {}^a\mathsf{X}^c)$
B9	$(f \star g) \cdot {}^c\mathsf{X}^d = {}^a\mathsf{X}^b \cdot (g \star f)$
	pentru $f : a \to c, \quad g : b \to d$

II. Axiome pentru feedback

R1	$f \cdot (g \uparrow^c) \cdot h = ((f \star \mathsf{I}_c) \cdot g \cdot (h \star \mathsf{I}_c)) \uparrow^c$
R2	$f \star g \uparrow^c = (f \star g) \uparrow^c$
R3	$(f \cdot (\mathsf{I}_b \star g)) \uparrow^c = ((\mathsf{I}_a \star g) \cdot f) \uparrow^d$
	pentru $f : a + c \to b + d, \quad g : d \to c$
R4	$f \uparrow^0 = f$
R5	$(f \uparrow^b) \uparrow^a = f \uparrow^{a+b}$
R6	$\mathsf{I}_a \uparrow^a = \mathsf{I}_0$
R7	${}^a\mathsf{X}^a \uparrow^a = \mathsf{I}_a$

- Algebra de reţele *NA* (Network Algebra) se obţine din BNA cu axiome suplimentare pentru $\wedge_k, \vee^k$.

Nota: Noi nu folosim algebra NA in acest curs, ci doar expresiile de retele (ca o varianta textuala a desenelor).

Proiectarea circuitelor logice

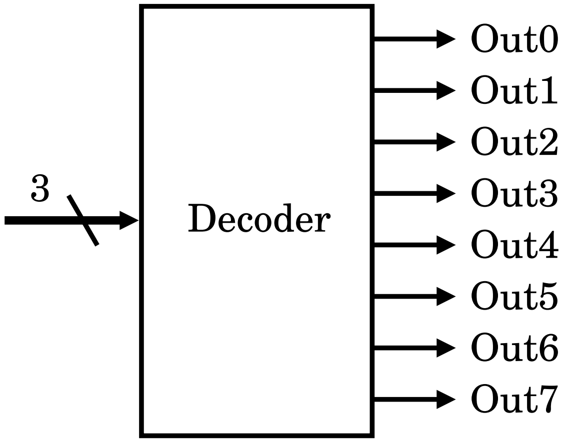
Cuprins:

- Porti, tabele de adevar, ecuatii logice
- *Logica combinationala*
- Ceas
- Elemente de memorie
- Masini de stari finite
- Concluzii, diverse, etc.

Logica combinationala

Exemple de blocuri logice:

- *Decoder:*
  - Tip: $n \to 2^n$;
  - Funcţie: Ieşirea k care are codul binar dat de intrare este 1; restul 0. Exemple: $000 \mapsto Out0$, $010 \mapsto Out2$, $101 \mapsto Out5$, etc.
  - Exemplu: *3-to-8 decoder* (în figură).

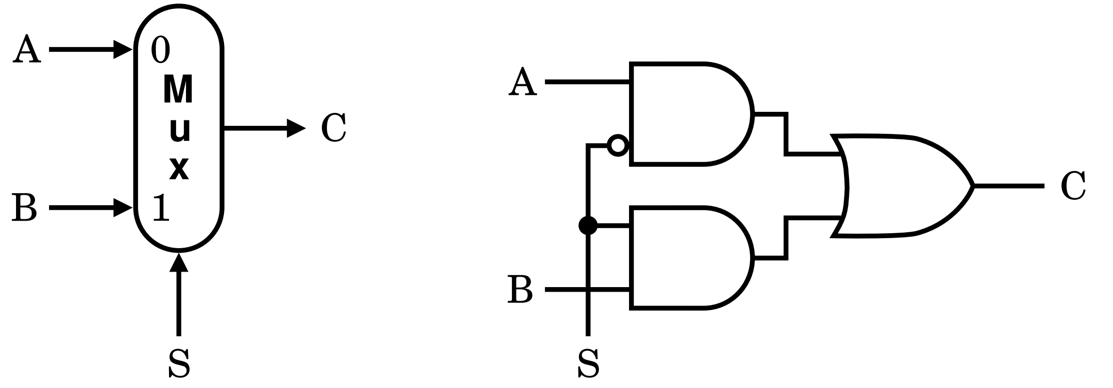


a. A 3-bit decoder

- *Encoder:* Produce funcţia inversă de tip $2^n \to n$ care asociaza intrării k codul ei binar.

Exemple de blocuri logice (cont.)

- *Multiplexor:*



  – Se selectează una din intrările $0,\ldots,n-1$ folosind un selector/control (de lărgime suficient de mare spre a putea codifica toate intrarile).

  – Exemplul conţine un multiplexor cu 2 intrări, i.e.
  $C = (A\cdot\overline{S}) + (B\cdot S)$.

..Logica combinationala

Exemple de blocuri logice (cont.)

– Un multiplexor general cu n intrări va folosi un selector cu $\lceil log_2 n \rceil$ intrări.
El se poate construi astfel:
1. Se foloseşte un *decoder* (de tip $\lceil log_2 n \rceil \to n$) care generează n semnale, câte unul pentru fiecare intrare;
2. Se folosesc n porţi AND care combină aceste semnale cu intrările multiplexorului;
3. Se foloseşte o poartă mare $OR(n)$ (sau mai multe mici) pentru a combina ieşirile porţilor AND.

Formulă: Notăm $\phi(m,n) : m \cdot n \to n \cdot m$ relaţia $(i-1)\cdot n + j \mapsto (j-1)\cdot m + i$ cu $i \in \{1,\ldots,m\}$, $j \in \{1,\ldots,n\}$; formula este

$$(\mathsf{I}_n \star Decoder(n)) \cdot \phi(2,n) \cdot (n\, AND) \cdot OR(n)$$

Forme normale

Forme normale:

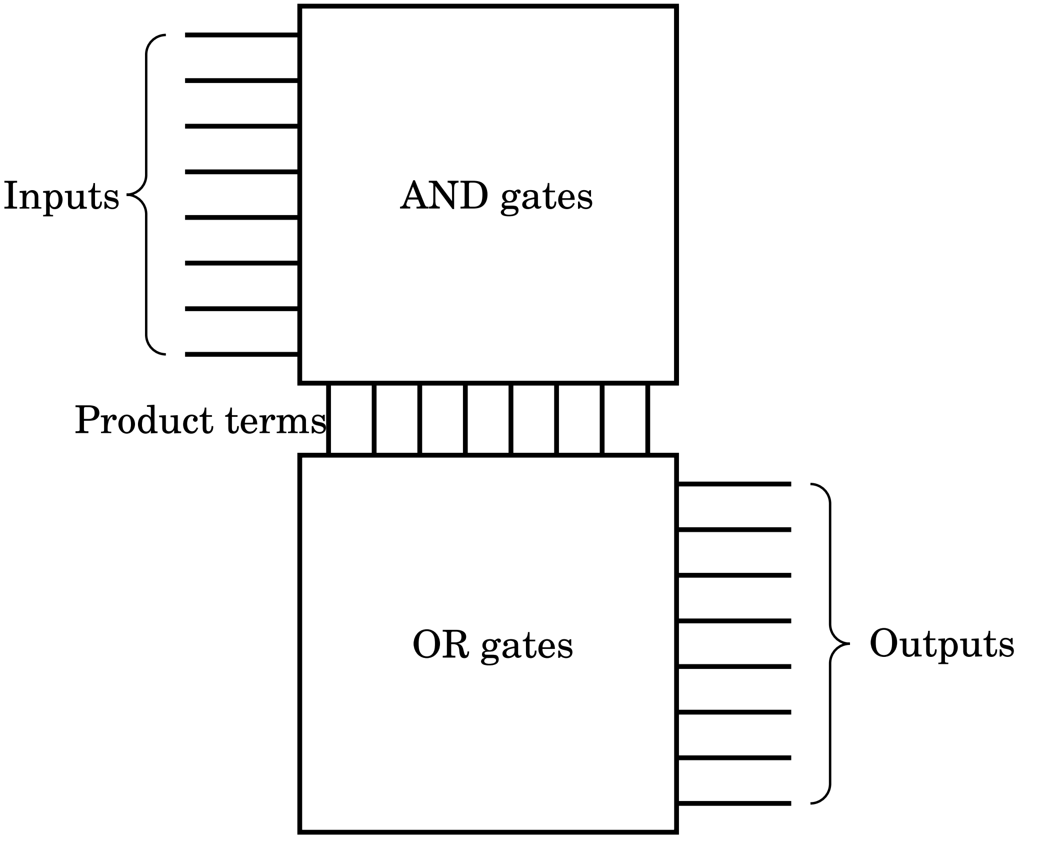
Teoremă: Orice funcţie logică poate fi reprezentată cu termeni peste $0, 1, AND, OR, NOT$ *folosind o* reprezentare pe 2 nivele: *fie ca* (1) sumă de produse, *fie ca* (2) produs de sume.

- Algoritm, pentru (1):
  - Se scrie un produs (monom) pentru fiecare linie din tabela de adevăr unde rezultatul este 1;
  - Dacă argumentul X este 1 se scrie X în produs, altfel $\overline{X}$.
  - Se face suma produselor astfel obţinute.

Nota: Daca suma este vida, rezultatul este zero.

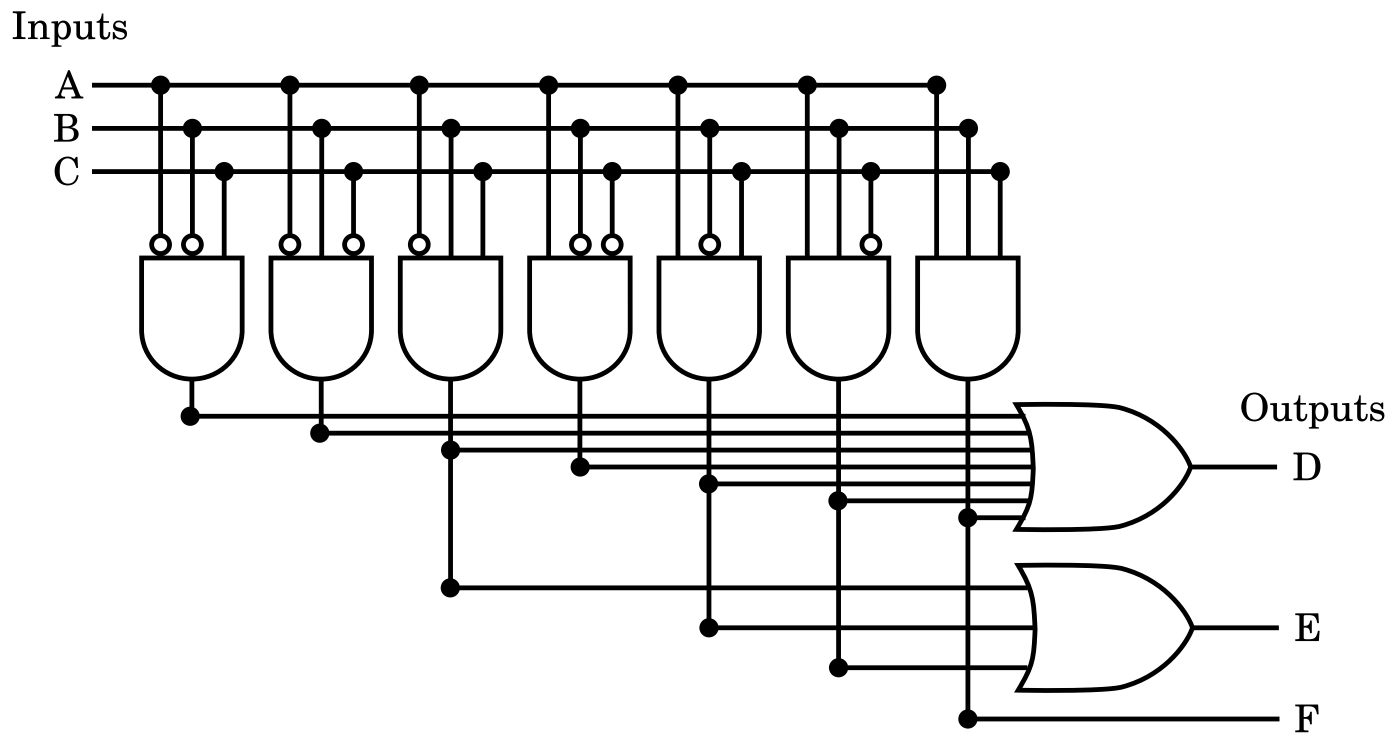
PLA:

- Implementarea bazată pe reprezentarea ca *sumă de produse* se numeşte *PLA - programable logic array* (*matrice logică programabilă*).
- Un exemplu este în figura alăturată.



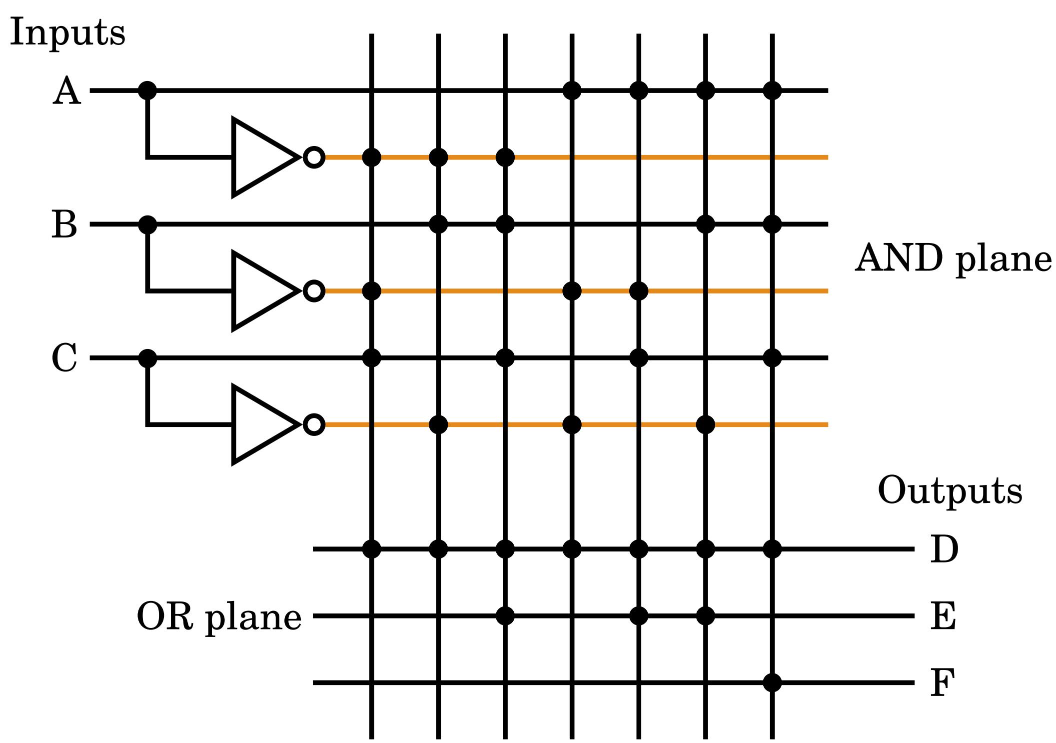
PLA, exemplu:

- Pentru exemplul din fila 4.4, circuitul rezultat cu algoritmul de mai sus este



PLA, exemplu:

- Circuitul poate fi descris şi într-un format simplificat



ROM

ROM:

- *ROM-ul* (*read-only memory*) este un alt exemplu de structură logică. Are un set de *valori* care pot fi *citite*; ele sunt fixate când este creat ROM-ul.
- Există şi ROM-uri *programabile* (PROM-uri), ori PROM-uri care se pot *şterge* (greu).
- Un ROM cu n intrări are 2^n *entităţi adresabile*, fiecare cu lărgimea egală cu numărul de ieşiri.
- ROM-urile pot fi utilizate spre a memora seturi de funcţii logice: pentru fiecare combinaţie de intrări, valoarea returnată este tuplul valorilor funcţilor logice din set.
- Faţă de PLA-uri, ROM-urile au în genere mai multe intrări, conţinând toate combinaţiile de valori.

Optimizări

Optimizări:

- In unele contexte, valoarea funcţiei logice pe o anume intrare este irelevantă (don't care).
- Funcţiile cu astfel de valori se pretează la optimizări.
- Optimizări de mână (pentru exemple mici) se pot face cu *diagrame Karnaugh*.
- Pentru cazurile industriale există pachete performante de optimizare.

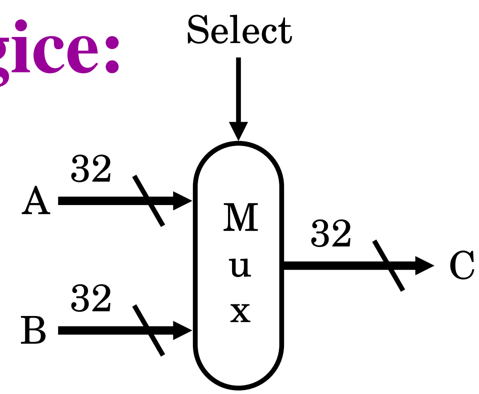
Vectori de elemente logice

Vectori de elemente logice:

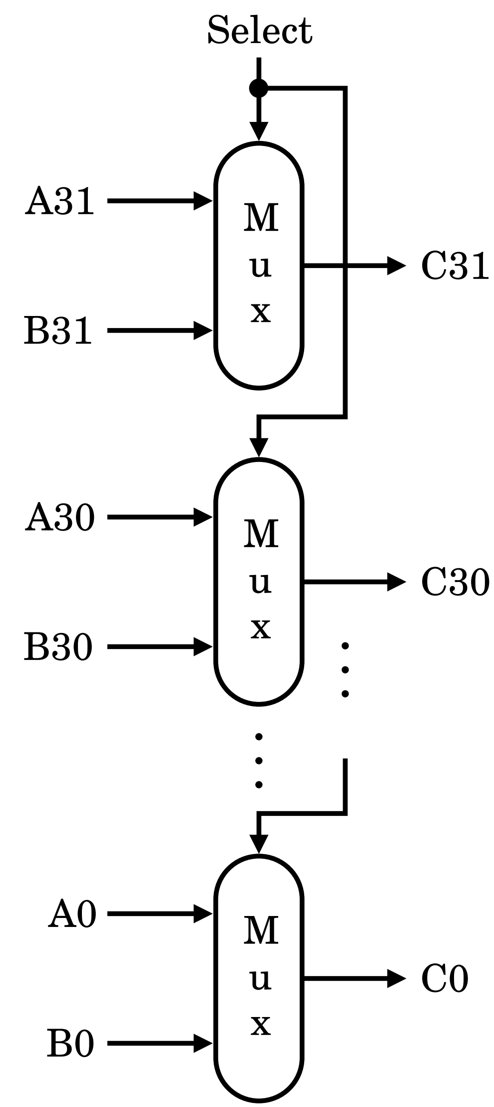
Deseori se folosesc elemente logice similare care operează “în paralel”.

Bus-urile (magistralele) sunt exemple tipice.

In figură (stânga), avem extensii pe cuvinte de 32 biţi, specificate marcând pe arce numărul de repetiţii folosit; în dreapta, desenul în extenso.



a. A 32-bit wide 2-to-1 multiplexor



b. The 32-bit wide multiplex is actually an array of 32 1-bit multiplexors

Proiectarea circuitelor logice

Cuprins:

- Porti, tabele de adevar, ecuatii logice
- Logica combinationala
- *Ceas*
- Elemente de memorie
- Masini de stari finite
- Concluzii, diverse, etc.

Ceas

Ceas: Notaţia din programarea uzuală

```
x = x + 2*x + 1
```

este confuză:

- Simbolul "=" nu este egalitatea matematică, ci o moştenire nefericită! (E.g., nu ne interesează soluţiile ecuaţiei $x = x + 2 * x + 1$, anume $x = -0.5$.)
- O notaţie mai adecvată ar fi ":=" (ori "<="), care reflectă mai exact sensul operaţiei de atribuire: se calculează membrul drept, iar rezultatul se atribuie variabilei din membrul stâng.
- Un mod şi mai fidel de reprezentare ar fi

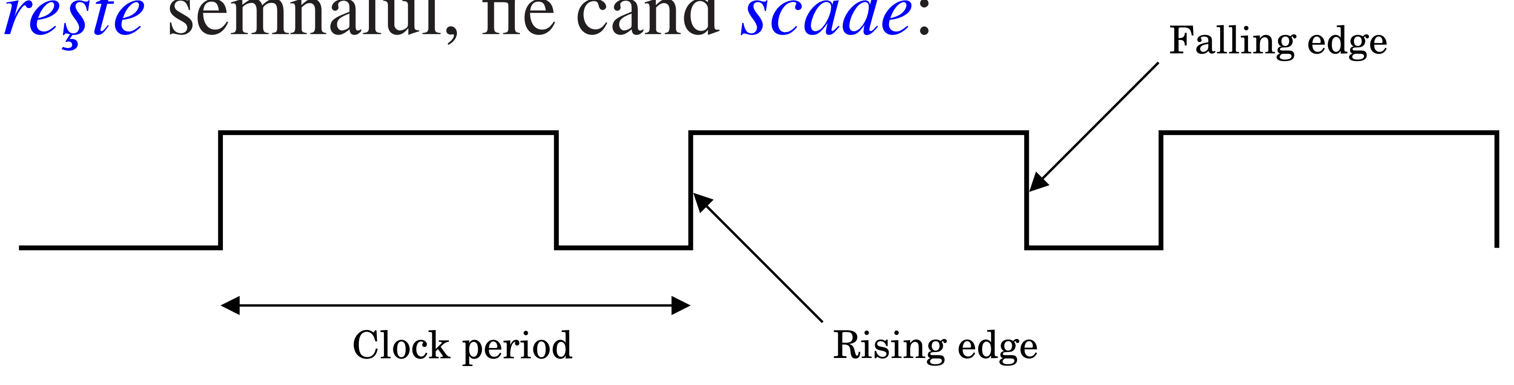
  ```
  x(t+1) = x(t) + 2*x(t) + 1
  ```

  marcând efectiv diferenţa de timp.

..Ceas

Ceas:

- Circuitele combinaţionale pot calcula expresii de tipul `x(t) + 2*x(t) + 1`.
- Este nevoie de *ceas* pentru a modela complet operaţia de atribuire, anume asignarea valorii calculate lui `x`, dar la momentul ulterior de timp.
- Conform tipului de tehnologie, schimbarea stării se face fie când *creşte* semnalul, fie când *scade*:

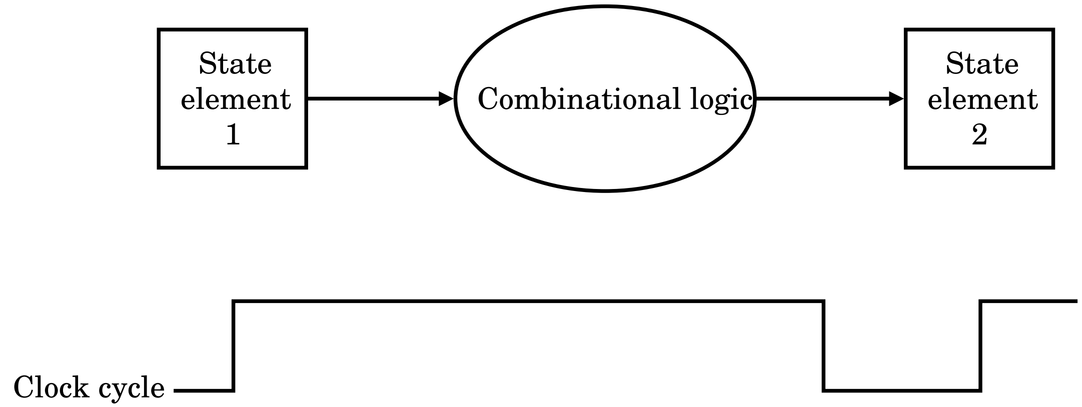


- Circuitele de acest tip, cu ceas şi memorie, se numesc *circuite secvenţiale*.

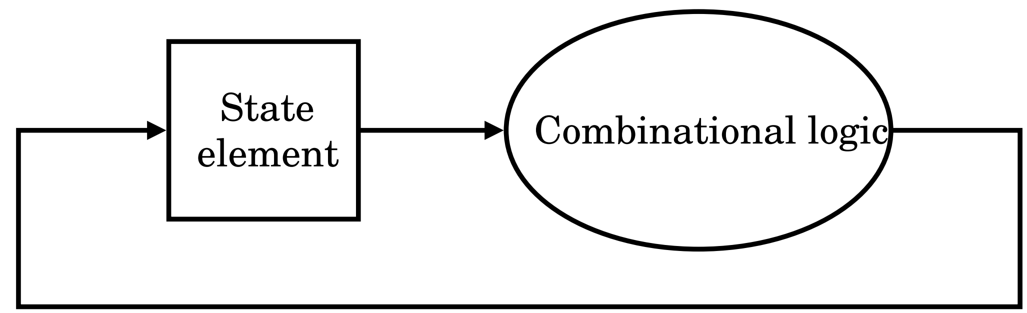
..Ceas

Ceas:

- Constrângerea de a actualiza starea la acelaşi moment de timp conduce la sisteme de calcul *sincrone*.



- Structura de bază a circuitelor secvenţiale este ilustrată în desenul de jos.



Notă: Ciclul de ceas trebuie sa fie suficient de lung pentru a permite completarea calculul combinational; în plus, valorile de intrare trebuie să fie stabile până la actualizarea stării.

Proiectarea circuitelor logice

Cuprins:

- Porti, tabele de adevar, ecuatii logice
- Logica combinationala
- Ceas
- *Elemente de memorie*
- Masini de stari finite
- Concluzii, diverse, etc.

Elemente de memorie

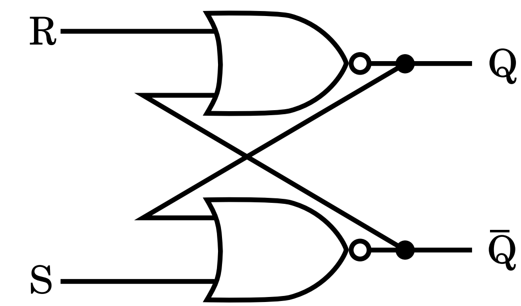
Elemente de memorie:

- De la simplu la complex, elementele de memorie de bază sunt: *latches* (zăvoare), *flip-flops* (bistabili), *fisiere cu regiştri*, *memorii*.
- In blocurile cu memorie ieşirea depinde de intrare, dar *şi de valoarea anterior memorată*.
- Circuitele cu memorie se numesc *secvenţiale*.

Nota: Interesant, ca si in cazul sistemului nervos, starea de "repaus" a memoriei este activa*, anume se consuma energie spre a se conserva. Nu este ca in cazul inregistrarilor uzuale (scriere, integistrare pe CD, etc.) unde conservarea continutului nu consuma resurse.*

S-R latch

S-R latch:



- S-R latch (zăvor cu set-reset) este element de memorie *neblocat* (fără ceas).
- Funcţie: Q devine 1 când este activat S şi 0 când e activat R; altfel păstrează valoarea anterioară. [Se ajunge într-o stare inconsistentă (eroare) dacă R, S se activează simultan.]
- Funcţionare (valoarea anterioară este memorată în perechea complementară $Q, \overline{Q}$):
  - Dacă ieşirea Q este 1 şi intrarile R, S sunt 0, atunci perechea $Q, \overline{Q}$ îşi întreţine valorile prin conexiunile încrucişate. Analog când Q este 0.
  - Dacă S este activat (devine 1), $\overline{Q}$ devine 0 forţând Q să devină 1. Dacă R este activat, Q devine direct 0.

Latch-uri si flip-flops-uri

Latches si fl ip-fl ops:

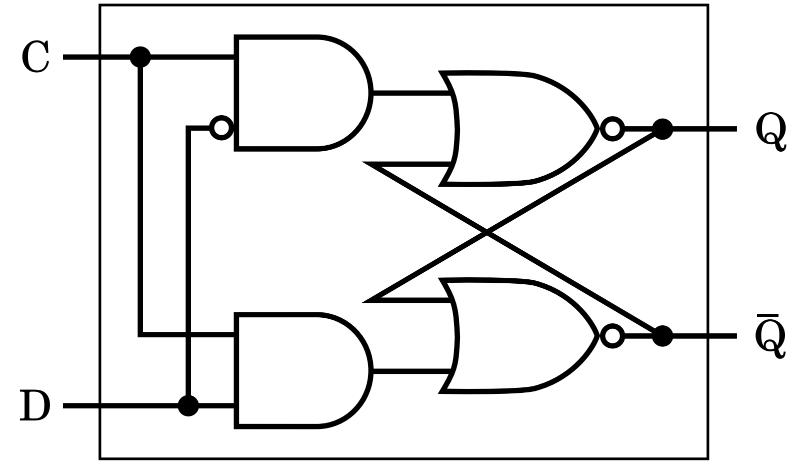
- Spre deosebire de S-R latch-uri, elementele de acum *au intrare de ceas*: schimbarea stării depinde de declanşarea ceasului.

- La latch-uri cu ceas, schimbarea se face *când se schimbă intrarea şi este activat ceasul*. La flip-flops-uri, schimbarea stării se face *pe frontul semnalului de ceas*. Deoarece folosim această ultimă tehnologie de ceas, avem nevoie doar de flip-flops-uri.

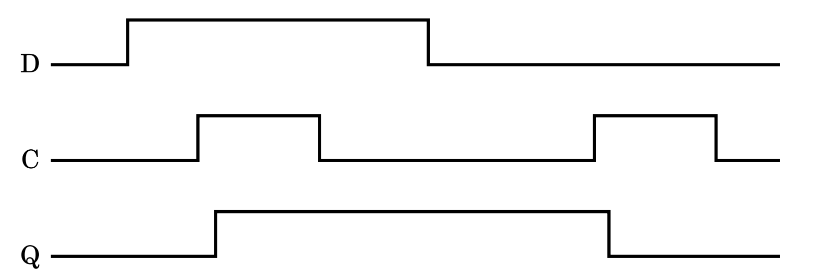
- Deseori flip-flops-urile se crează din latch-uri.

- Sunt multe tipuri de latch-uri si flip-flops-uri; noi folosim *D-latch-uri* si *D-flip-flops*.

D-Latches:



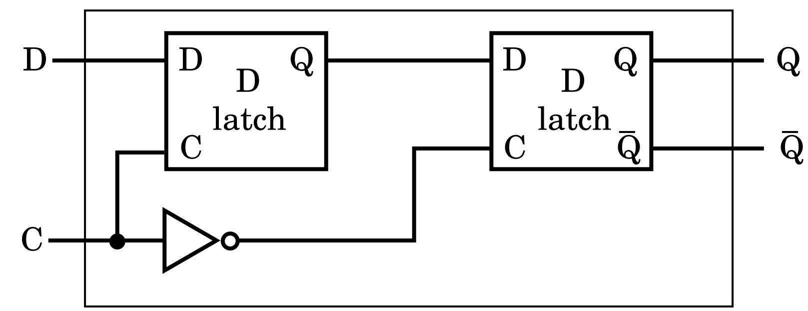
- Un *D-latch* memorează valoarea unui semnal de intrare în memoria internă.
- Are 2 intrări D (pentru date) şi C (pentru ceas) şi 2 ieşiri, anume o pereche complementară $Q, \overline{Q}$.

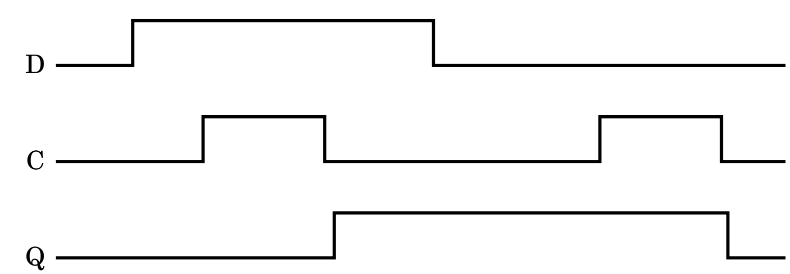


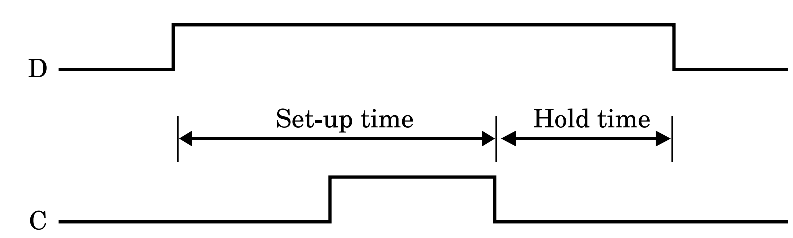
- Când C este 1 (activat), D-latch-ul este *deschis* şi ieşirea Q depinde de D.
- Când C este 0 (neactivat), D-latch-ul este *închis* şi ieşirea Q este valoarea memorată anterior.

D-flip-flops:

- Un *D-flip-flops* este format din 2 D-latch-uri, conectate ca în figură.
- La activarea semnalului de ceas *C* primul D-latch este deschis permiţând trecerea datei.
- La încheierea perioadei de ceas, al doilea D-latch se deschide permiţând scrierea intrării *D* pe ieşirea finală *Q*.
- Astfel modificarea memoriei se face pe frontul schimbării ceasului.

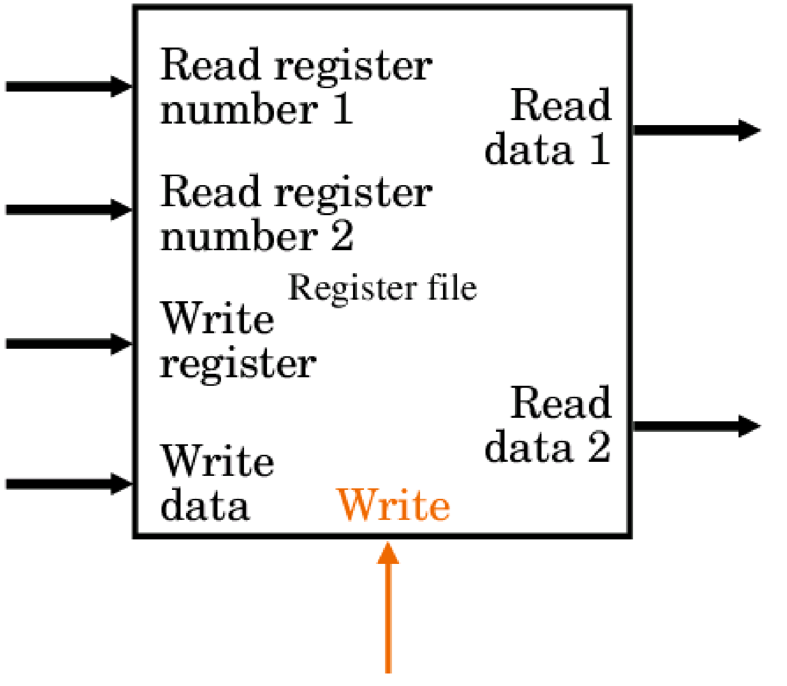






Fisiere cu registri

Fisiere cu registri:



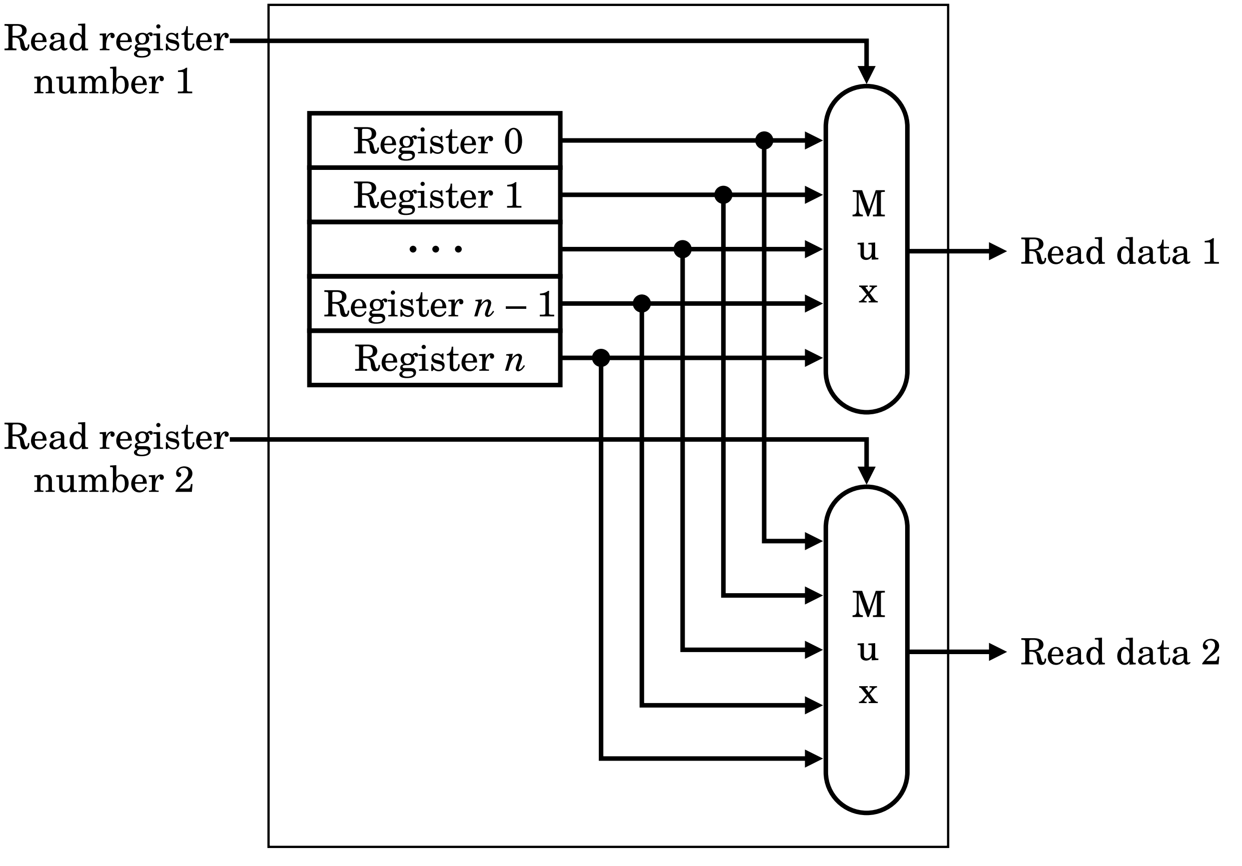
- Un set de regiştri care pot fi citiţi/scrişi, accesul facându-se prin *numărul* lor.
- Se foloseşte un *decoder* pentru fiecare port citire/scriere şi un *vector de D flip-flops-uri*. [In figură există 2 porţi de citire, una de scriere, si controlul de scriere.]
- In figura următoare avem o implementare pentru un fişier de regiştri care are 2 porţi de citire şi n regiştri de 32 biţi.

..Fisiere cu registri

Fisiere cu registri (cont.)

Citire:

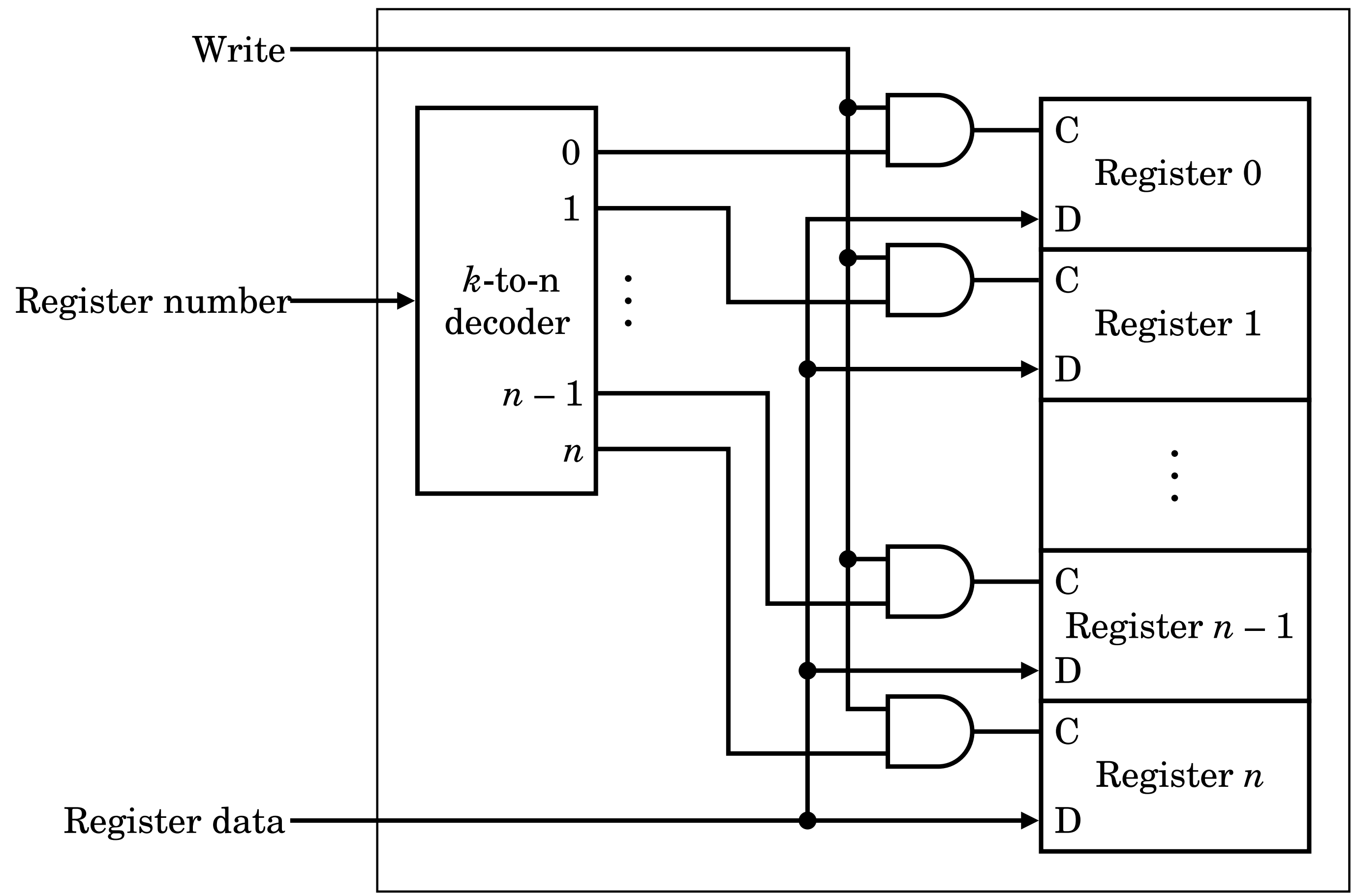
- In figura avem o implementare pentru un fişier de regiştri care are 2 porţi de citire şi *n* regiştri de 32 biţi.
- Se foloseşte un *multiplexor* pentru fiecare port de citire.



Fisiere cu registri (cont.)

Scriere:

- In figura avem o implementare pentru un fişier de regiştri care are o poartă de scriere şi *n* regiştri de 32 biţi.
- Se foloseşte un *decoder* k-to-n, unde $k = \lceil log_2 n \rceil$ (pentru fiecare port de scriere).



Memorii

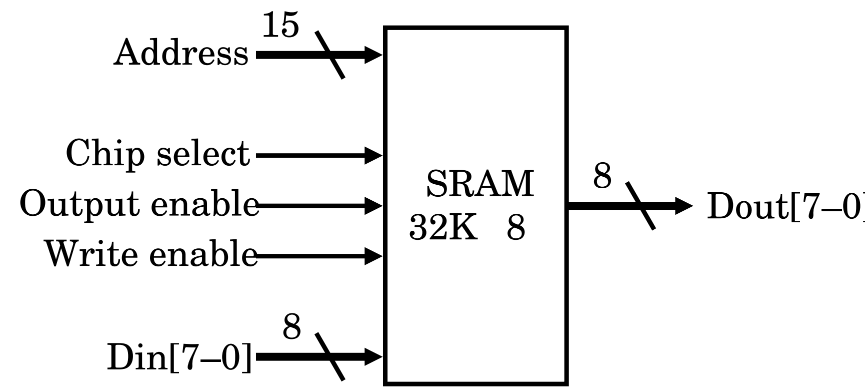
Memorii: Memoriile mari se constuiesc folosind *SRAM*-uri (Static Random Access Memories) ori *DRAM*-uri (Dynamic Random Access Memories).

SRAM:

- SRAM-urile sunt vectori de memorii, bazate pe latch-uri / flip-flops-uri.
- Uzual, ele au un unic port, pe care poate scrie ori citi date.
- Specificarea se face cu o formulă $2^m\text{K} \times n$ SRAM, anume sunt m linii de adresare şi date de n biţi.
- Exemple: O memorie $32\text{K} \times 8$ SRAM are 15 linii de adresare ($2^{15} = 32\text{K}$), datele accesate având 8-biţi; total 256K. O memorie egală, dar de tipul $256\text{K} \times 1$ SRAM, are 18 linii de adresare şi date de 1 bit.

..Memorii

SRAM (cont.)



- Figura indică intrările şi ieşirile la o memorie 32K $\times$ 8 SRAM.
- La citire/scriere semnalul `Chip select` trebuie să fie activat.
- Pentru citire, şi semnalul `Output enable` trebuie activat spre a indica dacă datele selectate se trimit la pini (necesar când memoriile se conectează la un bus).
- Pentru scriere, pe lângă adresă şi date, trebuie ca şi semnalul de control `Write enable` să fie activat.

..Memorii

SRAM (cont.)

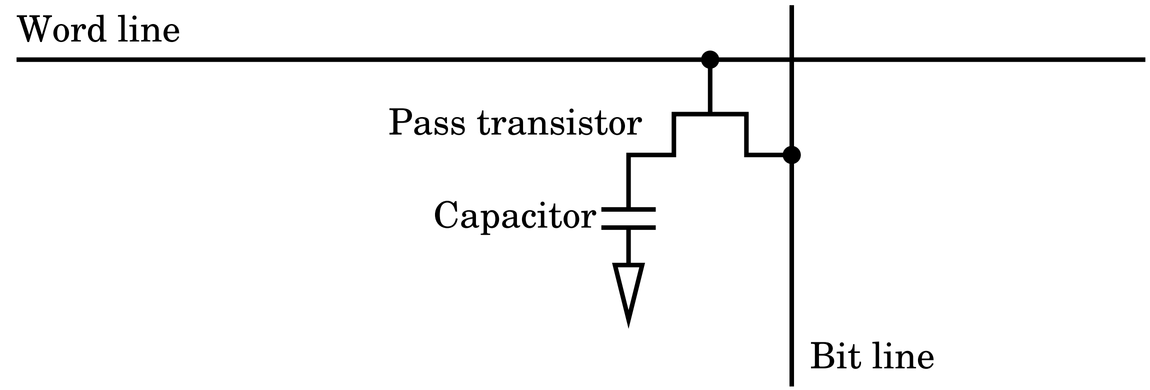
- Viteze tipice (1997):
  - Citire: 5-25ns;
  - Scriere: neclar (depinde de set-up time, hold time, lăţime puls `Write enable`).
- Capacitate (1997): până la 4M.
- Mai recent, s-au dezvoltat versiuni *sincrone* care permit accesul la un *grup* de celule aflate la adrese apropiate.
- Exemple: *SSRAM* (Synchronous SRAM), *SDRAM* (Synchronous DRAM).

Memorii DRAM

DRAM:

- DRAM-urile memorează datele folosind *condensatori*. [Reamintim că SRAM-urile folosesc perechi de porţi inversate.]
- Se foloseşte un unic tranzistor pentru fiecare bit (condensator), deci DRAM-urile pot fi mult mai dense. [SRAM-urile folosesc 4-6 tranzistori pe bit.]
- Incărcarea condensatorilor nu rezistă mult; ei trebuiesc *reîmprospătaţi*, anume citim conţinutul şi scriem din nou.
- Nu se reîmprospătează bit-cu-bit (ar ţine memoria ocupată), ci linii întregi. Tipic 1-2% din timp este pentru reîmprospătare, restul 98-99% pentru citit/scris. (De aceea memoria se numeşte *dinamică*).

..Memorii DRAM



DRAM (cont.)

Cum citeşte/scrie un DRAM?

Scris:

- Tranzistorul ataşat funcţionează ca un comutator: Când linia de acces de scriere este activată, comutatorul conectează condensatorul cu linia bitului, încărcând ori descărcând condensatorul după cum bitul este 1 ori 0.

Citit:

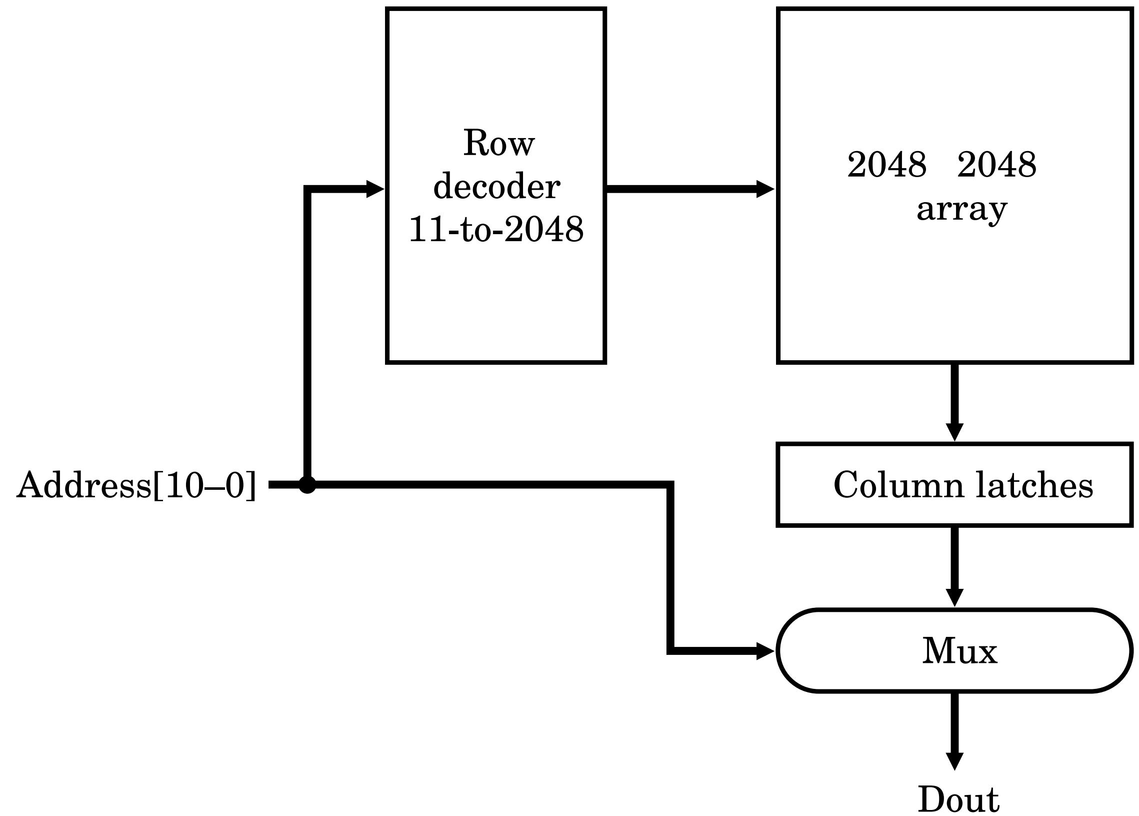
- La citire, linia bitului se încarcă la *jumătate* de voltaj (între max şi min) spre a detecta variaţii minime. La activarea liniei de acces la citire se citeşte condensatorul, mutând uşor voltajul bitului în direcţia respectvă.

..Memorii DRAM

DRAM (cont.)

Exemplu: 4M × 1 DRAM

- Se foloseşte o matrice 2048 × 2048.
- Adresarea se face în două faze: *pe linii*, apoi *pe coloane*.



- După identificarea liniei (cu un decodor), elementele liniei se încarcă într-un vector de latch-uri (column latches).
- Al doilea semnal de adresare identifică coloana şi, cu un multiplexor, accesează elementul cerut.

Proiectarea circuitelor logice

Cuprins:

- Porti, tabele de adevar, ecuatii logice
- Logica combinationala
- Ceas
- Elemente de memorie
- *Masini de stari finite*
- Concluzii, diverse, etc.

Circuitele secventiale

Circuitele secventiale:

- Combinaţia de *circuite combinaţionale* şi *memorie* conduce la *circuitele secvenţiale*.
- Dacă secvenţele de acţiuni utilizate sunt finite, un model abstract util pentru circuitele secvenţiale este dat de *maşinile de stări finite*.
- Dacă secvenţele de acţiuni utilizate sunt infinite (ca la sistemele reactive, e.g., sistemele de operate care funcţionează non-stop), atunci se pot folosi *stream-uri* (şuvoaie) şi *reţele de fluxul datelor sincrone*.
- Aceste modelele abstracte (ca şi logica, anterior), permit dezvoltarea unei teorii tractabile.

Masini de stari finite

Masini de stari finite:

- O *maşină de stări finită* este un tuplu $(S, V, \delta_s, \delta_o, s_0)$, unde

  –S are o mulţime finită de *stări*;

  –V este o mulţime finită de *acţiuni* (V este vocabularul);

  –$\delta_s : S \times V \rightarrow S$ este funcţia care dă *starea următoare*;

  –δ_o defineşte *funcţia de ieşire*;

  –s_0 este *starea iniţială*;

- In *maşinile Moore* $\delta_o : S \times V \rightarrow O$, iar în *maşinile Mealy* $\delta_o : S \rightarrow O$, unde O este alfabetul de ieşire.

- Dată o secvenţă de acţiuni $a_1 \ldots a_k$, maşina trece printr-o secvenţă de stări $s_1, \ldots, s_k$, unde $s_i = \delta_s(s_{i-1}, a_i)$; pe scurt,

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$$

..Masini de stari finite

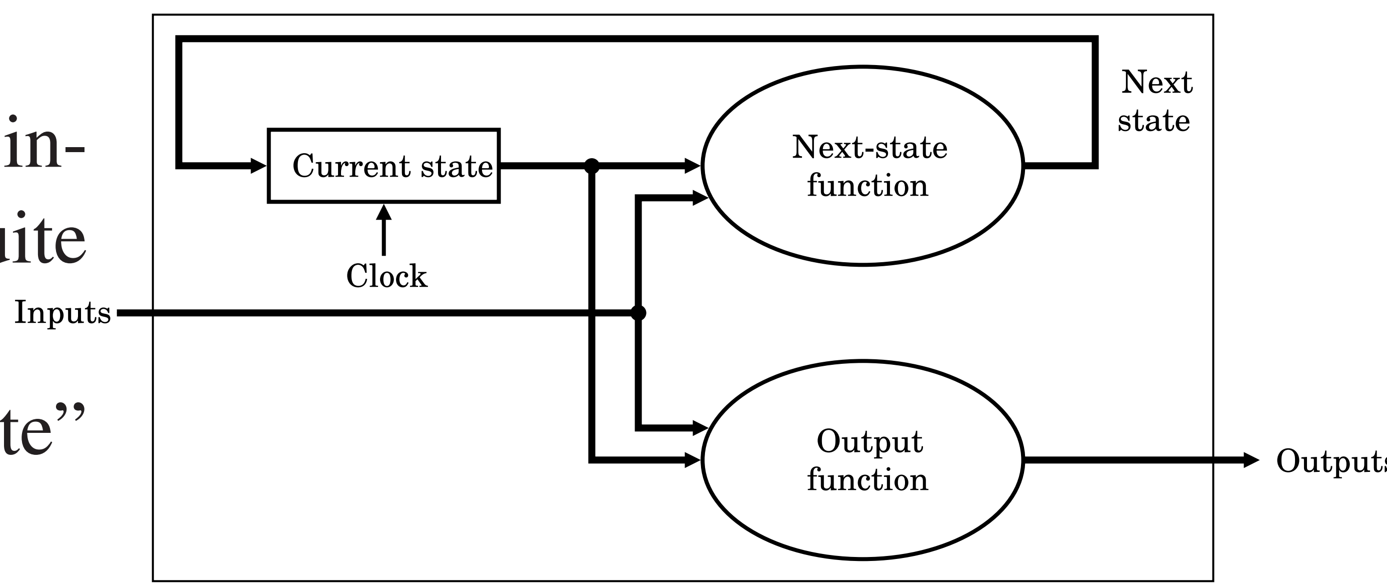
Masini de stari finite (cont.)

- Aplicând funcţia de ieşire δ_o obţinem o secvenţă de ieşiri corespunzătoare intrărilor.

Teoremă: Maşinile Moore şi maşinile Mealy sunt echivalente.

- Avantaje relative: Maşinile Moore pot fi mai rapide, iar cele Mealy, mai mici (cu mai puţine stări).

- Implementare:
  Are memorie internă şi 2 circuite combinaţionale pentru "next-state" şi "output".



..Masini de stari finite

Exemplu: Culorile semaforului într-o intersecţie:

- Restricţii: folosim verde şi roşu; o culoare durează min 30s.
- Intrări: `NScar` - vine maşină în direcţia NS;
  `EWcar` - vine maşină în direcţia EW;
- Ieşiri: `NSlite` - este 1 dnd culoarea NS este verde;
  `EWlite` - este 1 dnd culoarea EW este verde;
- Stări: `NSgreen` - culoarea verde este în direcţia NS;
  `EWgreen` - culoarea verde este în direcţia NS;
- Funcţia de ieşire este

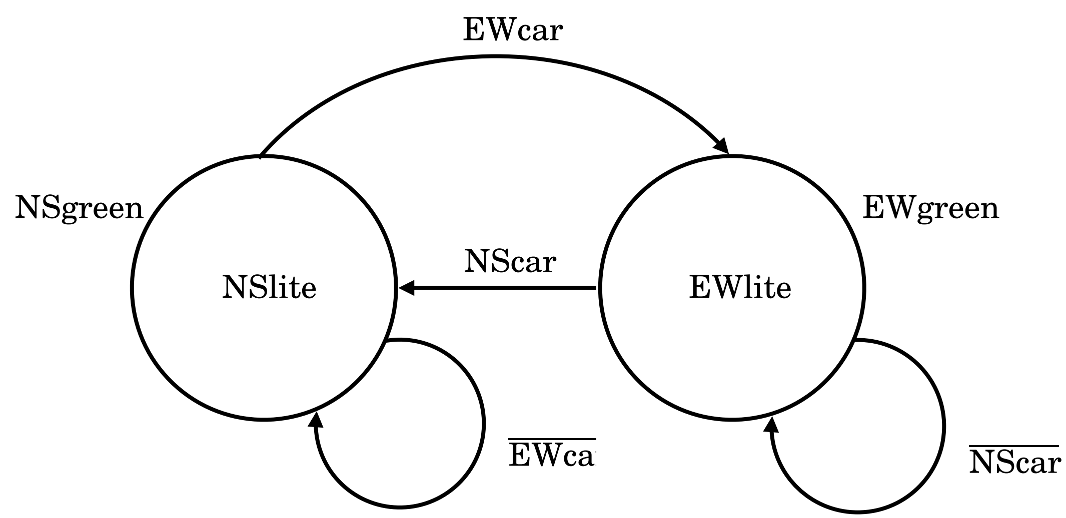
`Stare`	$Out1$: `NSlite`	$Out2$: `EWlite`
`NSgreen`	1	0
`EWgreen`	0	1

..Masini de stari finite

- Funcţia de tranziţie (next-state) este

Current state	$in1$: NScar	$in2$: EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

- Maşina (simplificată!) este descrisă de diagrama din figură.



Proiectarea circuitelor logice

Cuprins:

- Porti, tabele de adevar, ecuatii logice
- Logica combinationala
- Ceas
- Elemente de memorie
- Masini de stari finite
- *Concluzii, diverse, etc.*

Concluzii, diverse, etc.

A se insera...