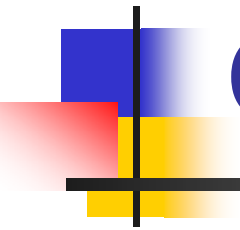
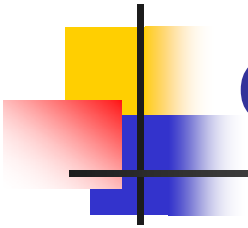


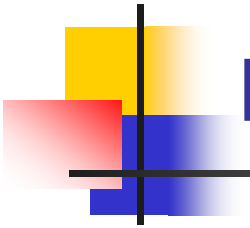
Circuite integrate digitale

Curs 3



Curs 3

- algebră logică
- funcţii logice
  - reprezentări
  - minimizare
- circuite combinaţionale cu porţi logice
- descrierea funcţiilor logice şi a circuitelor combinaţionale în Verilog
- **Aplicaţii**: operaţii aritmetice şi descrierea lor logică; funcţii logice exprimate în limbaj natural



Reprezentările funcţiilor logice

- "limbaj natural"
- expresii logice
- tabel de adevăr
- diagrame logice (VK de la Veitch şi Karnaugh)

Exemplu: sumator complet pe 1 bit

sum = a′b′c + a′bc′ + ab′c′ + abc=

= a′(b′c + bc′) + a(b′c′ + bc)=

= a′(b ⊕ c) + a(b ⊕ c)′=

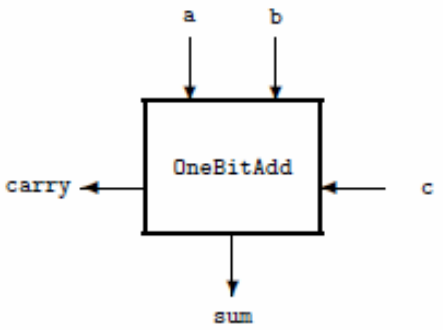
= a ⊕ (b ⊕ c)=

= a ⊕ b ⊕ c

carry = a′bc + ab′c + abc′ + abc=

= (a′b+ab′)c+ab(c′+c)=

= (a ⊕ b)c + ab



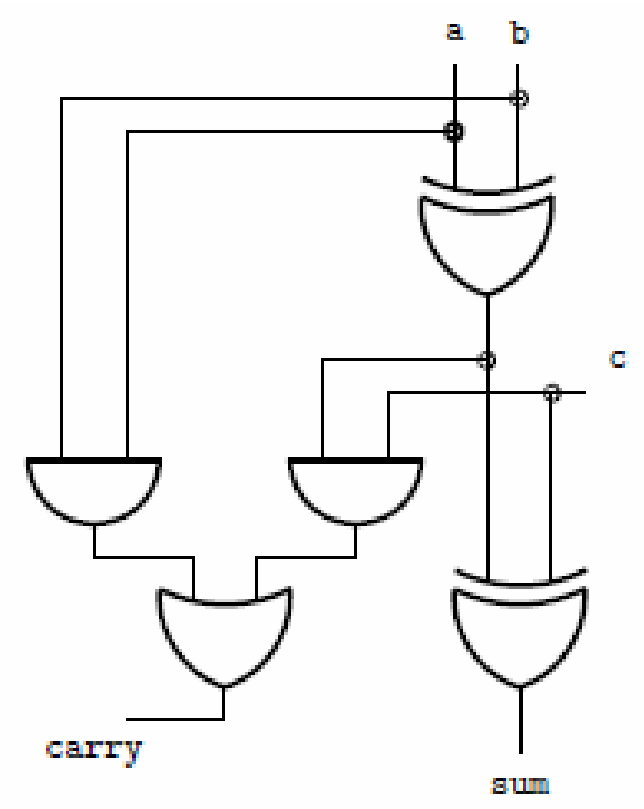
a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sumator complet pe 1 bit: schemă logică

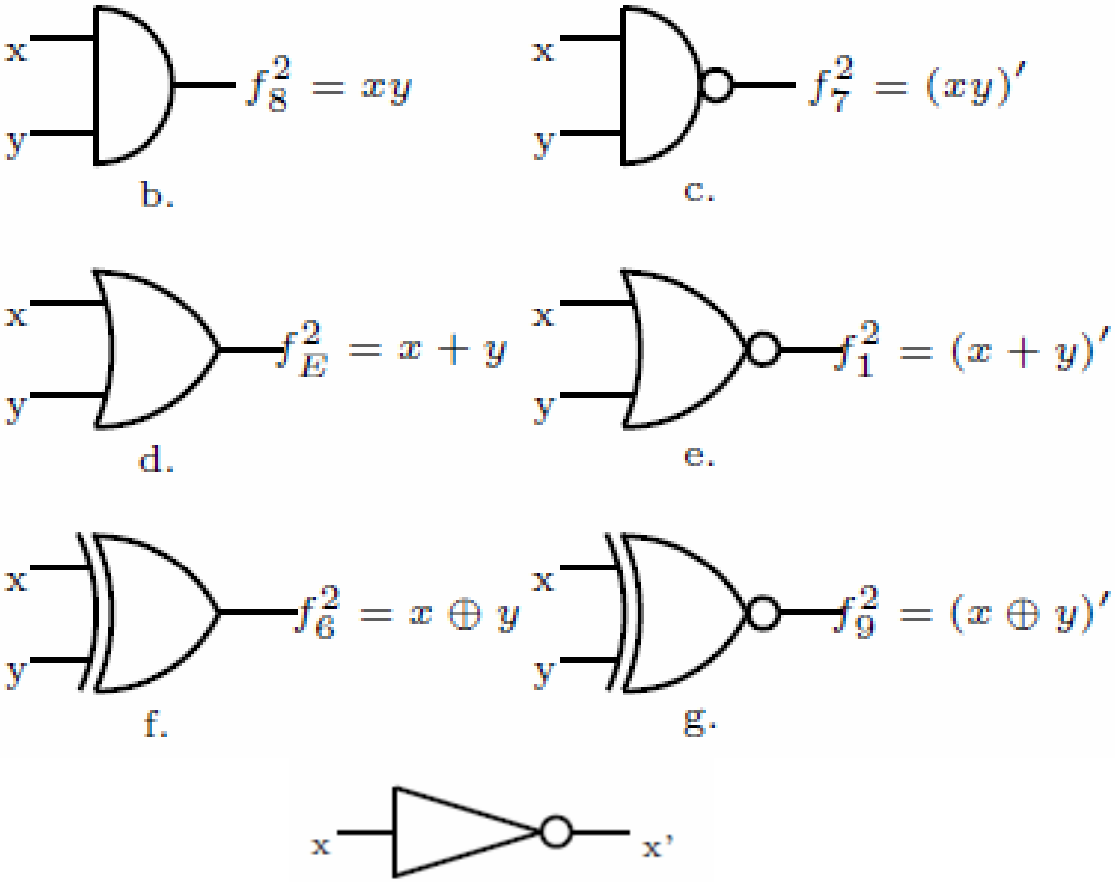
Expresii logice:

sum = a ⊕ b ⊕ c = (a ⊕ b) ⊕ c

carry = a b + c (a ⊕ b)



Simbolurile porţilor logice



Sumator complet pe 1 bit: expresii logice în Verilog

sum = a ⊕ b ⊕ c = (a ⊕ b) ⊕ c

carry = a b + c (a ⊕ b)

```
assign sum = a ^ b ^ c;
assign carry = a & b | c & (a ^ b);
```

Sumator complet pe 1 bit: descriere în Verilog cu operaţii aritmetice

```
module adder #(parameter n = 4)// defines a n-bit adder
        (output [n-1:0] sum,   // the n-bit result
         output         carry, // carry output
         input          c,     // carry input
         input  [n-1:0] a, b); // the two n-bit numbers

    assign {carry, sum} = a + b + c;
endmodule
```

Simbolurile funcţiilor logice

funcţie logică	simbol	operator Verilog
ŞI	· (se poate omite)	&
SAU	+	|
XOR	$\oplus$	^
NOT	’	~

NOR, NAND şi NXOR nu au un simbol distinct, scriem de ex. (a+b)’

Proiectarea unui circuit logic

- **funcţie exprimată în limbaj natural**

... cu ajutorul tabelului sau al raţionamentelor logice deducem

- **expresia logică a funcţiei**

... cu ajutorul algebrei logice sau al diagramelor urmărim să ajungem la

- **expresia minimală a funcţiei** (cea mai simplă)

- **circuitul logic cu porţi** "transcrie" această expresie

Precizări privind expresia minimală (pe baza exemplului anterior)

În cazul funcţiei sum nu s-a făcut o simplificare a funcţiei, ci doar o rescriere, folosind un alt simbol

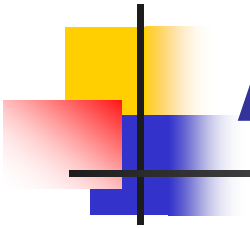
sum = a′b′c + a′bc′ + ab′c′ + abc= a′(b′c + bc′) + a(b′c′ + bc)=

= a′(b ⊕ c) + a(b ⊕ c)′= a ⊕ (b ⊕ c)= a ⊕ b ⊕ c

Atunci când avem mai multe funcţii logice, facem minimizare pe ansamblu. Expresia lui carry se poate simplifica mai mult!

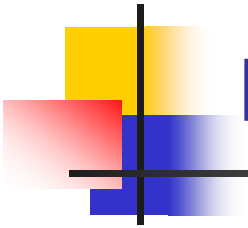
carry = a′bc + ab′c + abc′ + abc=(a′b+ab′)c+ab (c′+c)=

= (a ⊕ b)c + ab



Algebra logică

- ne ajută să operăm cu expresii logice
- ne interesează mai ales:
  - echivalenţa expresiilor logice
  - aducerea la forma cea mai simplă (minimizare)



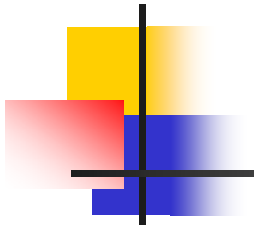
Expresii logice

- Definiţie: expresii care conţin combinaţii ale unor variabile logice (cu ajutorul funcţiilor logice definite mai sus)
- Orice funcţie logică se poate defini cu ajutorul
  - unei combinaţii de funcţii: inversare, şi, sau
  - funcţiei nand
- Două expresii logice sunt echivalente atunci când au aceeaşi valoare pentru orice combinaţie a variabilelor
  - tabel
  - algebră

Algebra logică – teoreme şi pricipii

- principiul identităţii a=a
- dubla negaţie (a')' = a

- funcţiile şi, sau, xor sunt comutative şi asociative
  - a + b = b + a
  - a + (b +c) = (a + b) + c
- funcţiile şi negat (nand), sau negat (nor), nxor sunt comutative, dar **nu sunt asociative**
  - **atenţie la implementarea cu porţi!**



- $a+a=a$
- $aa=a$
- $a+a'=1$
- $aa'=0$

- $a \oplus a = ?$
- $a \oplus a' = ?$

- Element neutru
  - $a + 0 = a$
  - $a \cdot 1 = a$
  - $a \oplus 0 = a$

  - $a + 1 = 1$
  - $a \cdot 0 = 0$
  - $a \oplus 1 = a'$

Algebra logică – teoreme şi pricipii

- reducere
  - ab + ab’= a
- absorbţie
  - a + ab = a
- semiabsorbţie
  - a + a’b = a + b

- teoremele lui de Morgan
  - (a + b)’ = a’b’
  - (ab)’ = a’ + b’

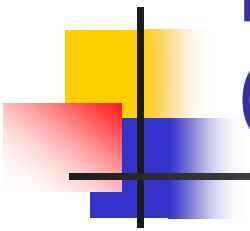
Minimizarea funcţiilor logice

- aducerea la forma cea mai simplă
  - cu algebră logică – reducere, absorbţie, semiabsorbţie
  - cu un algoritm care foloseşte diagrame logice
- circuitele cu porţi logice traduc expresia!

carry = a'bc + ab'c + abc' + abc= a'bc + abc + ab'c + abc+ abc' + abc=bc(a+a') + ac (b'+b) + ab(c'+c)=bc+ac+ab

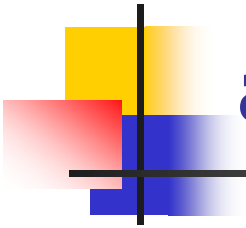
Desenaţi schemele cu porţi logice pentru diferite variante!

Descrierea circuitelor logice în Verilog (pentru funcţii relativ simple!)



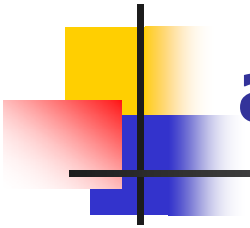
- schemă cu porţi (rar folosită)
- expresii logice – instrucţiunea **assign**
- alte modalităţii de descriere: **if, case...** care ne permit să descriem comportamental

- descrierea în Verilog ne permite să sărim peste etapa de descriere cu funcţii logice



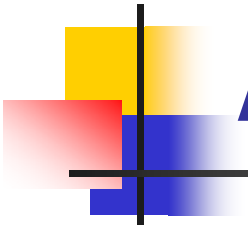
assign

- **Atribuirea continuă,** assign, se calculează din nou la fiecare pas de simulare.
- Se foloseşte pentru funcţii logice, expresii aritmetice etc.
- Atribuirea continuă are rolul de a descrie funcţionarea circuitelor combinaţionale, în care ieşirea urmează modificările intrărilor.
- Toate instrucţiunile assign se execută în paralel, aşa cum toate componentele unui circuit funcţionează în acelaşi timp.



assign ... ?

- Instrucţiunea de **atribuire condiţionată** are forma
  - assign w = x ? a : b;
- Atribuirea condiţionată depinde de valoarea lui x:
  - dacă x = 1, w =a,
  - iar pentru x = 0, w = b
- Se pot include mai multe condiţii una în alta



Atribuire condiţionată

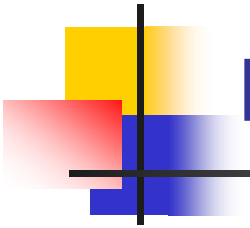
- ca semnificaţie, este “un fel de” if
- exemplu: comparatorul pe 1 bit
  - assign f = (a==b) 1: 0;
  - if (a == b) f = 1;
    else f = 0;
- atenţie! instrucţiunile if şi case nu se folosesc niciodată de sine stătător
- variabilele atribuite trebuie să fie definite de tip **reg**

Instrucţiunea **always**

- proces care se execută atunci când se modifică cel puţin un semnal din lista de senzitivităţi (lista semnalelor care apare în paranteză, după simbolul @)).

```
always @ (a or b or c) // aici nu se pune ";"
y = a & b & c; // atentie! y este de tipul reg
```

- Mai simplu, putem scrie always @ (*),
  unde notaţia (*) desemneaza orice modificare.
- Atunci când se simulează funcţionarea unui circuit, toate blocurile always se execută în paralel.

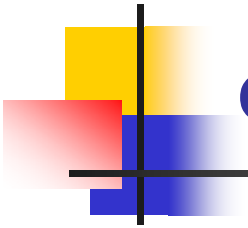


Evenimente

- eveniment = schimbarea valorii unui semnal

- *event-driven simulation*

@ (at)
always / forever
$monitor



case

Instrucţiunea case în Verilog are următorul format:

```
case (expresie_case)
   valoare1: instrucțiune1;
...
   default: instrucțiune; /* cazul implicit,
                pentru valorile care nu au fost
                                precizate*/
endcase
```

ca şi instrucţiunea if, case **se foloseşte numai într-un bloc always; variabilele care sunt atribuite trebuie să fie declarate de tip reg.**

definim întotdeauna cazul implicit, din cauza logicii cu 4 stări!

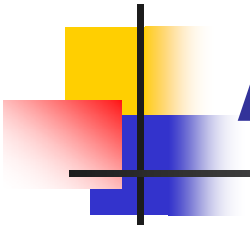
Logică cu 4 valori

- 0
- 1
- x (valoare necunoscută)
- z (impedanţă înaltă)
- alte simboluri folosite în tabele Verilog:
  - b (binar, adică 0 sau 1)
  - ? (0, 1 sau x)

Toate tipurile de semnale şi variabile sunt definite implicit x sau z

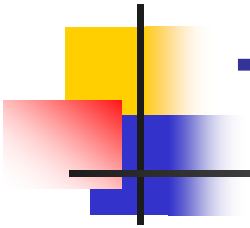
Tabele (logică cu 4 stări)

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x



Aplicaţii

- exemplu de circuit definit în limbaj natural (expresie)
- **sum** = a'b'c + a'bc' + ab'c' + abc=
  = a'(b'c + bc') + a(b'c' + bc)=
  = a'(b ⊕ c) + a(b ⊕ c)'=
  = a ⊕ (b ⊕ c)=
  = a ⊕ b ⊕ c
- exemplu de circuit definit în limbaj natural (tabel)
- operaţia de înmulţire



Tema 3

- Deduceţi funcţiile realizate de un comparator pentru două numere binare de câte doi biţi, cu ajutorul tabelului de adevăr. Circuitul calculează 3 funcţii distincte, pentru cele 3 cazuri (<, =, >).
  - **indicaţie:** circuitul are 4 variabile de intrare, care sunt biţii celor două numere comparate.