

Sisteme VLSI reconfigurabile

Curs 02

Introducere în Verilog – partea II

Operatorii definiți în limbajul Verilog

1. Operatori aritmetici

+	adunare sau operator de semn
-	scădere sau operator de semn
*	înmulțire
/	împărțire (rezultatul este rotunjit la un număr întreg)
%	modulo (furnizează restul împărțirii)

Exemplu de tratare a valorilor semnalelor de tip integer, respectiv reg, în calculele aritmetice

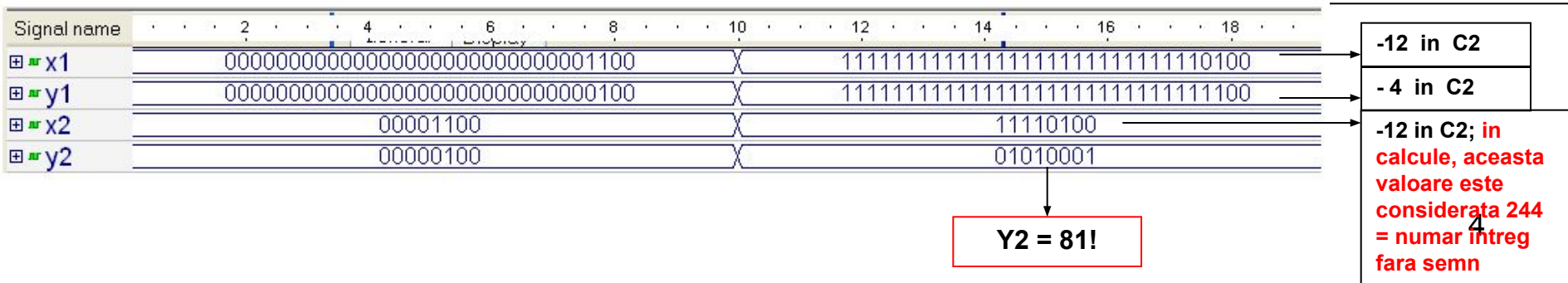
```

`timescale 1ns/100ps // seteaza unitatea de masura pt. timp, //
    respectiv rezolutia simularii
module c02ex01;

integer y1, x1; // se declara 2 variabile de tip integer
initial
begin // begin-end defineste un segment de cod
    x1 = 8'd12; // se initializeaza variabila x1=12
    y1 = x1/3; // lui y1 i se atribuie expresia x1/3
    # 10 x1 = -8'd12; // dupa 10ns, x1=-12 (-12 se va reprezenta in C2)
    y1 = x1/3; // y1 isi actualizeaza valoarea; rezultatul este corect!
end

reg [7:0] y2, x2; // se declara 2 variabile de tip reg pe 8 biti
initial
begin
    x2=8'd12;
    y2=x2/3;
    # 10 x2=-8'd12; // ! x3 = -12 (-12 se va reprezenta in C2)
    y2=x2/3; // rezultatul este incorect deoarece in expresii,
end // valoarea unui semnal de tip reg este interpretata
endmodule // ca numar intreg fara semn

```



Utilizarea semnalelor de tip `integer`, respectiv a valorii semnalelor de tip `reg` cu operatori

- valorile variabilelor de tip `integer` sunt interpretate ca fiind **valori cu semn**; în limbajul Verilog, reprezentarea binară a valorilor cu semn este în cod **complement față de 2**.
- **in cazul operatorului de divizare, valorile semnalelor de tip `reg` sunt interpretate ca valori fără semn**; dacă unui semnal de tip `reg` i se atribuie o valoare negativă, atunci aceasta este **reprezentată** binar în complement față de 2, iar în expresii în care este utilizat operatorul de divizare, valoarea respectiva este **interpretată** ca fiind valoarea unui întreg fără semn

2. Operatori relaționali

>	mai mare
>=	mai mare sau egal
<	mai mic
<=	mai mic sau egal

Utilizarea unei valori de negative pentru un semnal de tip `reg` generează erori (aceeași interpretare ca și în cazul utilizării operatorilor aritmetici)

3. Operatori de egalitate

===	egalitate precisă: compara si valorile x respectiv z
!==	inegalitate precisă: compara si valorile x respectiv z
==	egalitate
!=	inegalitate

exemplu

Operand 1	Operand 2	===	!==	==	!=
1xx1	1xx1	1	0	x	x

nu se utilizează în sinteză

rezultatul generat de aplicarea operatorilor relaționali sau de egalitate este boolean: **0** pentru **fals**, **1** pentru **adevarat**, respectiv **x** pentru ambiguu, daca in continutul operanzilor care se compara apar valori de **x** sau **z** (cu excepția operatorilor `===` și `!==` care compară și valorile **x**).

daca este necesar, operanzii sunt extinși la aceeași dimensiune prin completarea pozițiilor MSB, funcție de tipul operanzilor (întregi cu semn vs. întregi fără semn).

Exemplu de utilizare a operatorilor relaționali

```
`timescale 1ns/100ps
module c02ex02;
integer x1,y1,z1;
initial
begin
    x1 = 4'd5;           // x1 = +5
    y1 = 4'd10;         // y1 = +10
    z1 = x1 > y1;       // z1 = 0 = fals (rezultatul testului 5 > 10 ?)
    # 10 y1 = -4'd10;   // dupa 10ns: y1 = -10
    z1 = x1 > y1;       // z1 = 1 = adevarat (rezultatul testului 5 > -10 ?)
end

reg [7:0] x2,y2;
reg z2;
initial
begin
    x2 = 4'd5;           // x2 = +5
    y2 = 4'd10;         // y2 = +10
    z2 = x2 > y2;       // z2 = 0 = fals (rezultatul testului 5 > 10 ?)
    # 10 y2 = -4'd10;   // dupa 10ns: y2 = -10 => -10 este reprezentat in C2
    z2 = x2 > y2;       // z2 = 0 = fals (rezultatul testului 5 > 246 ?) ;
end
endmodule
```

rezultat eronat

Signal name	2	4	6	8	10	12	14	16
x1					5			
y1			10		X			-10
z1			0		X			1
x2					5			
y2			10		X			246
z2				0				

valoarea -10 a fost reprezentata intern in C2 11110110, dar a fost utilizata in expresie ca numar intreg fara semn (11110110 = 246) datorita faptului ca a fost atribuita unui semnal reg

Utilizarea semnalelor de tip `integer`, respectiv a valorii semnalelor de tip `reg` cu operatori

- valorile variabilelor de tip `integer` sunt interpretate ca fiind **valori cu semn**; în limbajul Verilog, reprezentarea binară a valorilor cu semn este în cod **complement față de 2**.
- **in cazul operatorilor relationali, valorile semnalelor de tip `reg` sunt interpretate ca valori fără semn**; dacă unui semnal de tip `reg` i se atribuie o valoare negativă, atunci aceasta este **reprezentată** binar în complement față de 2, iar în expresii in care sunt utilizati operatori relationali, valoarea respectiva este **interpretată** ca fiind valoarea unui întreg fără semn.

4. Operatori logici

&&	ȘI logic
	SAU logic
!	negare logică

compară logic 2 expresii = operanzi de dimensiuni oarecare

- **operanzii utilizați sunt reduși la 1 singur bit,** după cum urmează:
- operanzii care conțin valorile **x** sau **z** se reduc la valoarea **x**, iar rezultatul este **x**;
- **operanzii care conțin numai biți de 0 ȘI 1 se reduc la bitul 1 (adevărât)**
- **operanzii care conțin numai biți de 0 se reduc la bitul 0 (fals)**

rezultatul este generat pe un singur bit:

- 0 pentru fals
- 1 pentru adevărat
- **x** pentru necunoscut

5. Operatori logici pe biți

&	ȘI logic pe biți
	SAU logic pe biți
~	negare logică pe biți
^	XOR pe biti (SAU EXCLUSIV)
~^ sau ^~	XNOR pe biti (SAU EXCLUSIV NEGAT)

combină logic biții a 2 operanzi de dimensiuni oarecare care sunt aduși la aceeași dimensiune (dimensiunea de reprezentare cea mai mare dintre cei 2 operanzi)

operatorii care conțin valorile **x** sau **z** dau rezultatul **x** în poziția bitului corespunzător

rezultatul este precizat pe biți
dimensiunea rezultatului este aceeași cu dimensiunea operanzilor

6. Operatorul ternar

target = (conditie)? expresie1: expresie0;

se executa daca: conditie = adev. conditie = fals 9

Rezultatele generate de utilizarea operatorilor logici, respectiv a operatorilor pe biți

```

`timescale 1ns/100ps
module c02ex03 (
output reg zAnd, zOr, zNot,
output reg [3:0] zAndB, zOrB, zNotB,
output reg z);

```

```

reg [3:0] x,y;

```

```

initial

```

```

begin

```

```

x = 4'b1001;

```

```

y = 4'b1111;

```

```

# 5 y = 4'b0000;

```

```

# 5 y = 4'b0001;

```

```

end

```

```

always @(x,y) // bloc de cod care se repeta la fiecare
// tranzitie a semnalelor x si y

```

```

begin

```

```

zAnd = x && y; // operatorii logici genereaza rezultate booleene

```

```

zOr = x || y;

```

```

zNot = !x;

```

```

zAndB = x & y; // operatorii logici pe biti genereaza rezultate

```

```

zOrB = x | y; // de dimensiunea operanzilor

```

```

zNotB = ~x;

```

```

z = ((x & (~y)))? !(x | y) : (x ^ y)&& y;

```

```

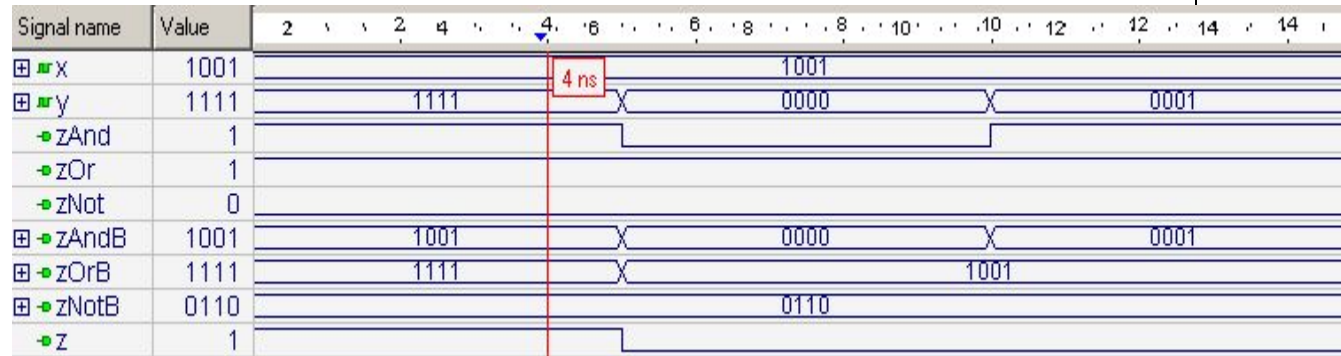
end

```

```

endmodule

```



7. Operatori de reducere

&	reducere prin ȘI logic	<code>& 4'b1010 // rezultat 0</code>
~&	reducere prin ȘI NU logic	<code>~& 4'b1010 // rezultat 1</code>
 	reducere prin SAU logic	<code> 4'b1010 // rezultat 1</code>
~ 	reducere prin SAU NU logic	<code>~ 4'b1010 // rezultat 0</code>
^	reducere prin XOR logic	<code>^ 4'b1010 // rezultat 0</code>
~^ sau ^~	reducere prin XNOR logic	<code>~^ 4'b1010 // rezultat 1</code>

reduc un operand la o reprezentare pe 1 bit, aplicînd o funcție logică fiecărei perechi de biți, în direcția MSB spre LSB ⇒ se reduc biții MSB cu MSB-1, apoi rezultatul reducerii acestora se reduce cu MSB-2, etc. până la bitul LSB.

operatorii care conțin valorile **x** sau **z** se reduc la bitul **x** în poziția bitului corespunzător; rezultatul este precizat pe biti.

8. Operatori de concatenare

`{semnal_MSB, ..., semnal_LSB}` // se generează bus-ul de date [semnal_MSB, ... semnal_LSB]

`{ dimensiune {semnal} }` // se generează bus de date [semnal, ..., semnal] de
// dimensiunea specificată în câmpul **dimensiune**

9. Operatori de deplasare – nu se folosesc în sinteza

<code>>> nr_pozitii</code>	deplasare biti spre dreapta cu numărul de poziții precizat
<code><< nr_pozitii</code>	deplasare biti spre stînga cu numărul de poziții precizat

V. Descrierea comportamentală a sistemelor digitale

□ comportamentul sistemului este descris în interiorul unor procese, definite de blocurile **always** respectiv **initial**:

□ blocul **always**:

- este echivalentul unei bucle infinite; blocul se reactivează în mod continuu;
- după activarea blocului, instrucțiunile din interiorul unui bloc **always** se execută secvențial pînă la ultima, după care blocul se reactivează, instrucțiunile din conținutul său se execută din nou secvențial, etc.
- blocul **always** este utilizat numai în combinații cu anumite **elemente de control temporal**

□ blocul **initial**:

- se activează o singură dată, la începutul simulării (la timpul **0** de simulare), sau la timpul de simulare programat prin intermediul unui element de control temporal (de exemplu, întârziere);
- după activare, instrucțiunile din interiorul său se execută secvențial;
- după executarea tuturor instrucțiunilor, blocul se dezactivează definitiv;
- în general, este utilizat pentru inițializarea valorilor semnalelor;
- **în sinteza logică, blocul **initial** nu este utilizat, pentru descrieri comportamentale!**
- **blocul **initial** este utilizat pentru inițializarea semnalelor în testbench-uri**

□ în cadrul aceluiași modul, mai multe blocuri **always** (și/sau **initial**) se execută concurrent (în paralel); blocurile **always** și **initial** reprezintă **processe concurente**;

□ în interiorul blocurilor **always** și **initial**, semnalele “țintă” (cele plasate în stînga instrucțiunilor de atribuire) pot fi numai de tipul **register** (**reg** sau **integer**).

Blocurile `always`, respectiv `initial`

```
always
  begin
    specificații secvențiale;
  end
```

se activează în mod continuu;
instrucțiunile din interiorul blocului se execută secvențial;
utilizarea unui bloc `always` în acest format blochează simularea într-o buclă infinită, generată de blocul `always` respectiv;

evitarea buclei infinite generate de blocul `always` este realizată prin introducerea în acesta a unor anumite elemente de control temporal

```
always element_control_temporal
  begin
    specificații secvențiale;
  end
```

```
initial
  begin
    specificații secvențiale;
  end
```

se activează o singură dată;
instrucțiunile din interiorul blocului se execută secvențial;

```
`timescale 1ns/10ps
module c02ex04;
  reg clk;
  initial
    clk = 0;
  always
    #10 clk <= ~clk; // buclă
                    // infinita

  // iesirea din buclă (oprirea simulării)
  // se realizează numai pe baza comenzii
  // $finish (nu se folosește în sinteză)

  initial
    #100 $finish;
endmodule
```

Conceptul de timp și evenimente în simularea HDL

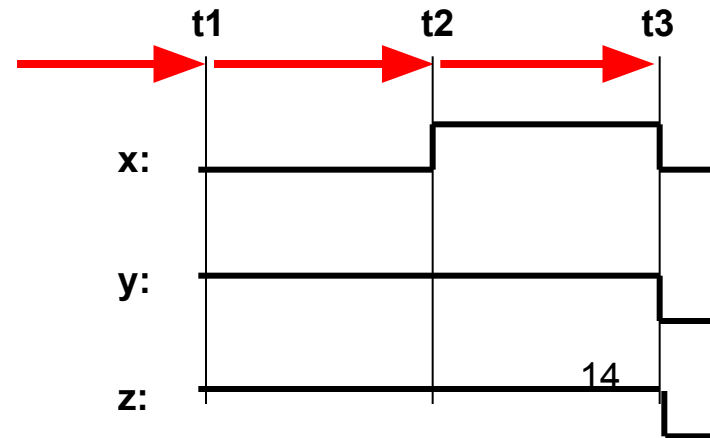
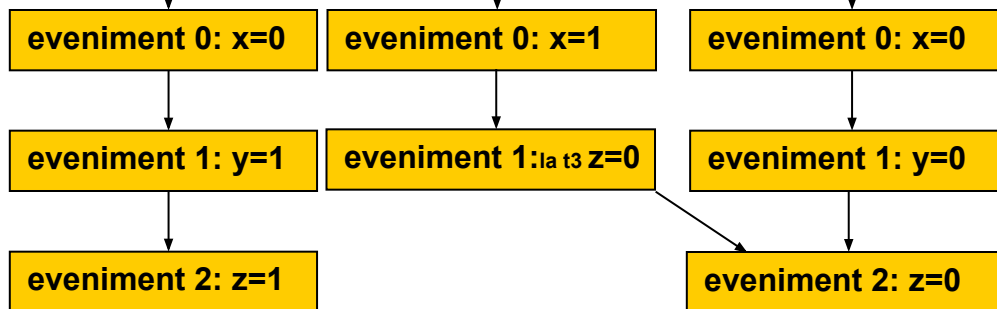
- **timpul de simulare** este modelat prin intermediul unei **variabile globale**;
- la fiecare valoare a timpului de simulare pot fi programate pentru a fi executate, unul sau mai multe **evenimente**, care alcătuiesc o **lista de evenimente** asociată timpului de simulare curent;
- un eveniment este compus dintr-un **segment de cod HDL** care este rulat de către simulator fără a fi întrerupt;
- simulatorul HDL **execută secvențial evenimentele** din lista de evenimente asociată unui anumit timp de simulare, după execuție, evenimentul respectiv este eliminat din listă;
- în momentul în care în lista de evenimente asociate timpului curent de simulare nu mai sunt programate evenimente, simulatorul trece la următorul timp de simulare (următoarea valoare a variabilei globale);
- pe măsură ce sunt executate anumite evenimente, noi evenimente pot fi generate, care sunt planificate la timpuri de simulare viitori.

timp

simulare: t1

t2

t3



Elemente de control temporal în procesele care modelează comportamentul sistemelor electronice

Elementul de control temporal **suspendă execuția procesului** pentru o durată de timp specificată, sau până la apariția unui eveniment de timp specificat;

Forme de control temporal:

**expresie**

suspendă execuția procesului pentru o durată de timp egală cu **valoarea** generată de **expresie**;
utilizată în instrucțiuni de atribuire, pentru modelarea întârzierilor;

@ **eveniment**

suspendă execuția procesului până la momentul în care evenimentul specificat în câmpul **eveniment** are loc;
modelează evenimentele sensibile la tranziții de semnale sau pe frontul semnalelor; utilizat în sintetiză.

ambele forme de control temporal elimină evenimentele din lista de evenimente asociată timpului curent de simulare și le reprogamează pentru timpi de simulare viitori

wait
(expresie_bool)

suspendă execuția procesului până la momentul în care expresia din câmpul **expresie_bool** devine **adevărată**;
modelează **evenimentele sensibile pe palier**

Activarea proceselor controlate de forma de control temporal # expresie

```
`timescale 1ns/100ps
module c02ex05;
  reg r;
```

```
initial #70 $stop;
```

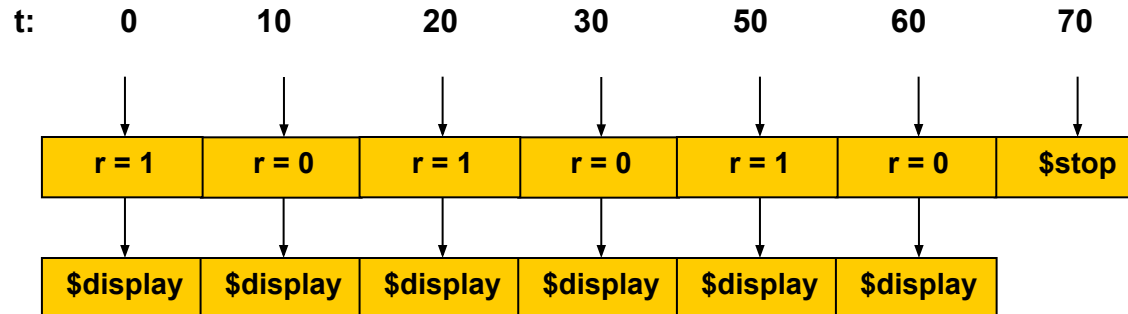
```
initial
begin
  #10 r = 0;
  #20 r = 0;
  #30 r = 0;
end
```

```
initial
begin
  r = 1;
  #20 r = 1;
  #30 r = 1;
end
```

```
always @ r
$display("la momentul %d, valoarea lui r este %d", $time, r);
```

```
endmodule
```

```
▸ # Selected Top-Level: c01ex03real (c01ex03real)
▸ run 90 ns
▸ # KERNEL: la momentul 0, valoarea lui r este 1
▸ # KERNEL: la momentul 10, valoarea lui r este 0
▸ # KERNEL: la momentul 20, valoarea lui r este 1
▸ # KERNEL: la momentul 30, valoarea lui r este 0
▸ # KERNEL: la momentul 50, valoarea lui r este 1
▸ # KERNEL: la momentul 60, valoarea lui r este 0
▸ # RUNTIME: RUNTIME_0070 c01ex03real.v (5): $stop called.
▸ # KERNEL: Time: 70 ns, Iteration: 0, Instance: /c01ex03real, Process: @INITIAL#5_0@.
▸ # KERNEL: stopped at delta: 0 at time 70 ns.
```



Forma de control temporal @ eveniment

Tipuri distincte:

@ variabila

@ (var1, var 2, ...)

@ (var1 or var 2 or ...)

suspendă execuția procesului pînă la momentul în care are loc o **tranziție** a variabilei/variabilelor specificate după operatorul @

@ posedge variabila

suspendă execuția procesului pînă la momentul în care are loc o **tranziție pozitivă** (din 0, x, sau z în 1) a variabilei specificate

@ negedge variabila

suspendă execuția procesului pînă la momentul în care are loc o **tranziție negativă** (din 1, x, sau z în 0) a variabilei specificate

@ variabila_event

suspendă execuția procesului pînă la momentul în care are loc evenimentul specificat ca variabilă **event**; evenimentul declarat ca variabila **event** poate fi declanșat în oricare proces cu operatorul **->**; nu se utilizează în modelele sintetizabile

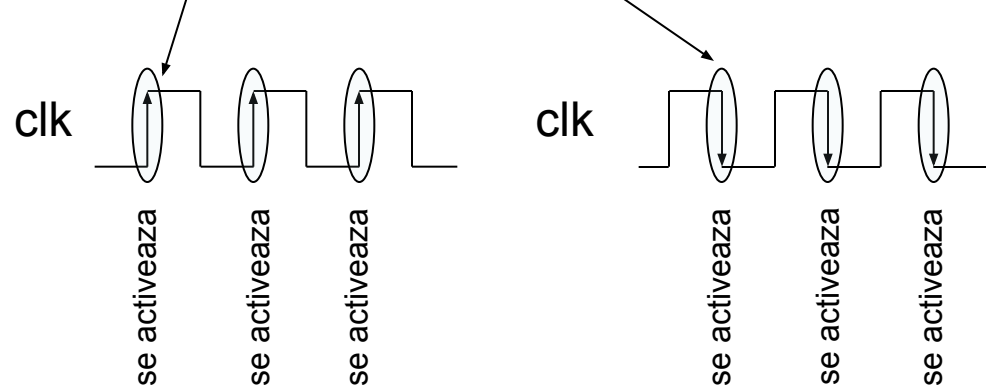
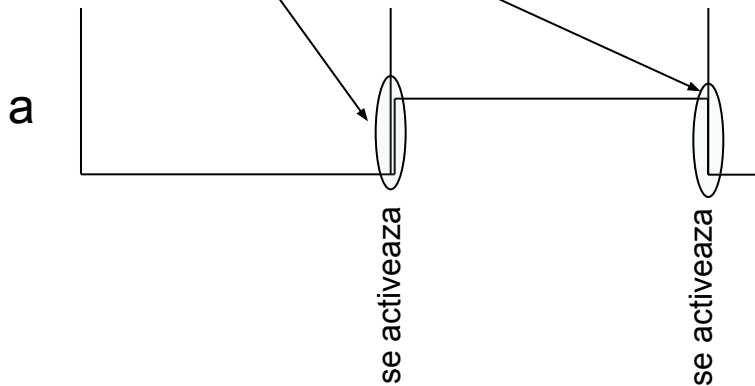
Activarea proceselor controlate de forma de control temporal @ variabila

```
always @(a, b) // procesul se activează
// dacă a sau b își
// modifică valoarea
....

always @(a) // procesul se activează
// dacă semnalul a își
// modifica valoarea
....
```

```
always @(posedge clk) // procesul se
// activează la frontul // crescator al
// semnalului clk
....

always @(negedge clk) // procesul se
// activează la frontul // descrescator al
// semnalului clk
....
```



Activarea proceselor controlate de forma de control temporal @ variabila_eventiment

```
module c02ex06;
```

```
run 90 ns
# KERNEL: evenimentul C
# KERNEL: evenimentul A
# KERNEL: evenimentul B
# KERNEL: stopped at time: 90 ns
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
```

```
// se declara 2 variabile eveniment
event e1, e2;
```

```
// proces suspendat pina la declansarea evenimentului notat e1
initial @ e1
begin
    $display ("evenimentul A"); // comanda de afisare text
    -> e2; // declansare eveniment e2
end
```

```
// proces suspendat pina la declansarea evenimentului notat e2
initial @ e2
    $display ("evenimentul B");
```

```
initial
begin
    $display ("evenimentul C");
    -> e1; // declansare eveniment e1
end
```

```
endmodule
```

Activarea proceselor controlate de forma de control temporal `wait (expresie)`

- utilizarea formei de control `wait` permite modelarea evenimentelor sensibile pe palier;
- procesul controlat de forma de control `wait` este activat dacă semnalul generat de câmpul `expresie` este egal cu 1 (adevărat), sau 0 (fals):

□ Exemple:

```
always
  wait (x)
  ...
```

```
always
  wait (~x)
  ...
```

- utilizarea neadecvată a formei de control `wait` poate genera bucle infinite, din care simulatorul nu mai poate ieși:

```
always
begin
  wait (ready)
  y = x;
end
```

- blocul `always` se activează în momentul în care semnalul `ready` devine 1 și rămâne activ atât timp cât `ready` rămâne în 1; atât timp cât `always` este activ, instrucțiunile din interiorul său se execută permanent;
- dacă valoarea semnalului `ready` este controlată de un alt proces (alt bloc `always` sau `initial`), atunci activarea blocului `always` prezentat alături (care conține controlul `wait`) generează o buclă din care simulatorul nu mai poate ieși,
- bucla infinită este generată deoarece, odată activat blocul `always` prezentat, acesta nu mai permite transferul fluxului de execuție al codului spre procesul (blocul `always`) care controlează semnalul `ready` și din acest motiv condiția `ready = 1` se menține permanent, ceea ce menține blocul `always` prezentat în stare activă la infinit;
- pentru evitarea buclor infinite, controlul `wait` este întodeauna utilizat împreună cu încă o formă de control temporal, care permite ieșirea din bucla infinită.

Tipuri de instrucțiuni de atribuire utilizate în blocurile `always` și `initial`

Formatul instrucțiunilor de atribuire:

```
semnalTinta    operator_atribuire    expresie;
```

Execuția instrucțiunilor de atribuire se realizează în **2 etape**:

1. **evaluare expresie**: se evaluează valoarea expresiei generate de către câmpul `expresie`
2. **actualizare valoare semnalTinta**: se atribuie valoarea generată în câmpul `expresie` semnalului precizat în câmpul `semnalTinta`

Observație: dacă în instrucțiunea de atribuire se utilizează elemente de control temporal, atunci cele două etape se pot desfășura la timpi de simulare diferiți!

Tipuri de instrucțiuni de atribuire utilizate în blocurile `always` și `initial`

□ instrucțiuni de atribuire **blocante**:

- utilizează operatorul `=`

- mod de execuție:

- cele 2 etape specifice execuției instrucțiunii de atribuire (evaluare expresie, actualizare semnal) nu pot fi decuplate în timp ⇒ **fluxul de execuție al instrucțiunii blocante nu poate fi cedat altor procese**;

- atât timp cât execuția etapei 1 (evaluare expresie) nu s-a terminat, fluxul de execuție al codului rămâne **blocat** la nivelul execuției instrucțiunii de atribuire;

- fluxul de execuție al codului este cedat numai după terminarea execuției instrucțiunii de atribuire, adică după finalizarea etapei 2 (actualizare semnal);

- utilizată pentru modelarea unui flux de atribuiri **secvențiale**;

```
z = x & y;    // mai întâi se evaluează expresia x & y, apoi valoarea astfel generată
              // este furnizată semnalului z; fluxul de execuție al codului nu este cedat
              // următoarei instrucțiuni, până când ambele etape nu sunt executate
w = u | v;
```

Tipuri de instrucțiuni de atribuire utilizate în blocurile `always` și `initial`

□ instrucțiuni de atribuire **neblocante**:

□ utilizează operatorul `<=`

□ mod de execuție:

□ între cele 2 etape specifice execuției instrucțiunii de atribuire (evaluare expresie, actualizare semnal), **fluxul de execuție al instrucțiunii curente este cedat** pentru executarea următoarelor instrucțiuni;

□ după execuția etapei 1 (evaluare expresie) fluxul de execuție al codului Verilog este cedat următoarei instrucțiuni ⇒ **nu rămâne blocat la nivelul instrucțiunii curente**,

□ etapa 2 (actualizare semnal) se execută abia la suspendarea blocului **`always`**, sau **`initial`**, în care este plasată instrucțiunea curentă;

□ se spune că după execuția etapei 1 s-a planificat o valoare pentru semnalul țintă (valoarea generată de către expresie), iar după execuția etapei 2 (la suspendarea blocului **`always`**, respectiv **`initial`**), valoarea astfel planificată a devenit curentă;

□ utilizată pentru modelarea unui flux de atribuiri **concurrente**;

```
always @....
z <= x & y; // se evaluează expresia x & y, iar valoarea astfel generată este
           // planificată a fi furnizată lui z
w <= u | v; // se evaluează expresia u | v, iar valoarea astfel generată este
           // planificată a fi furnizată lui w
           // valorile planificate pentru z și w devin valori curente abia la
           // suspendarea blocului always
```

Modelarea întârzierilor

- utilizează operatorul `# expresie`
- unitatea de timp a valorii generate de `expresie` este stabilită de directiva ``timescale unitate_timp / rezolutie`
- operatorul `#` poate fi utilizat în dreapta, sau în stânga instrucțiunii de atribuire

A. Modul de execuție a instrucțiunii de atribuire, pentru cazul în care operatorul de întârziere `#` este utilizat în **stânga** instrucțiunii de atribuire:

```
# expresieTimp      semnalTinta      operator_atribuire      expresie;  
( #10 y = x; )
```

- execuția instrucțiunii este suspendată până la momentul de timp egal cu valoarea generată de `expresieTimp`;
- ambele etape specifice execuției instrucțiunii de atribuire (evaluare `expresie`, actualizare `semnal`) se execută la timpul de simulare specificat de valoarea generată de `expresie` (`t=10`)

B. Modul de execuție a instrucțiunii de atribuire, pentru cazul în care operatorul de întârziere `#` este utilizat în **dreapta** instrucțiunii de atribuire:

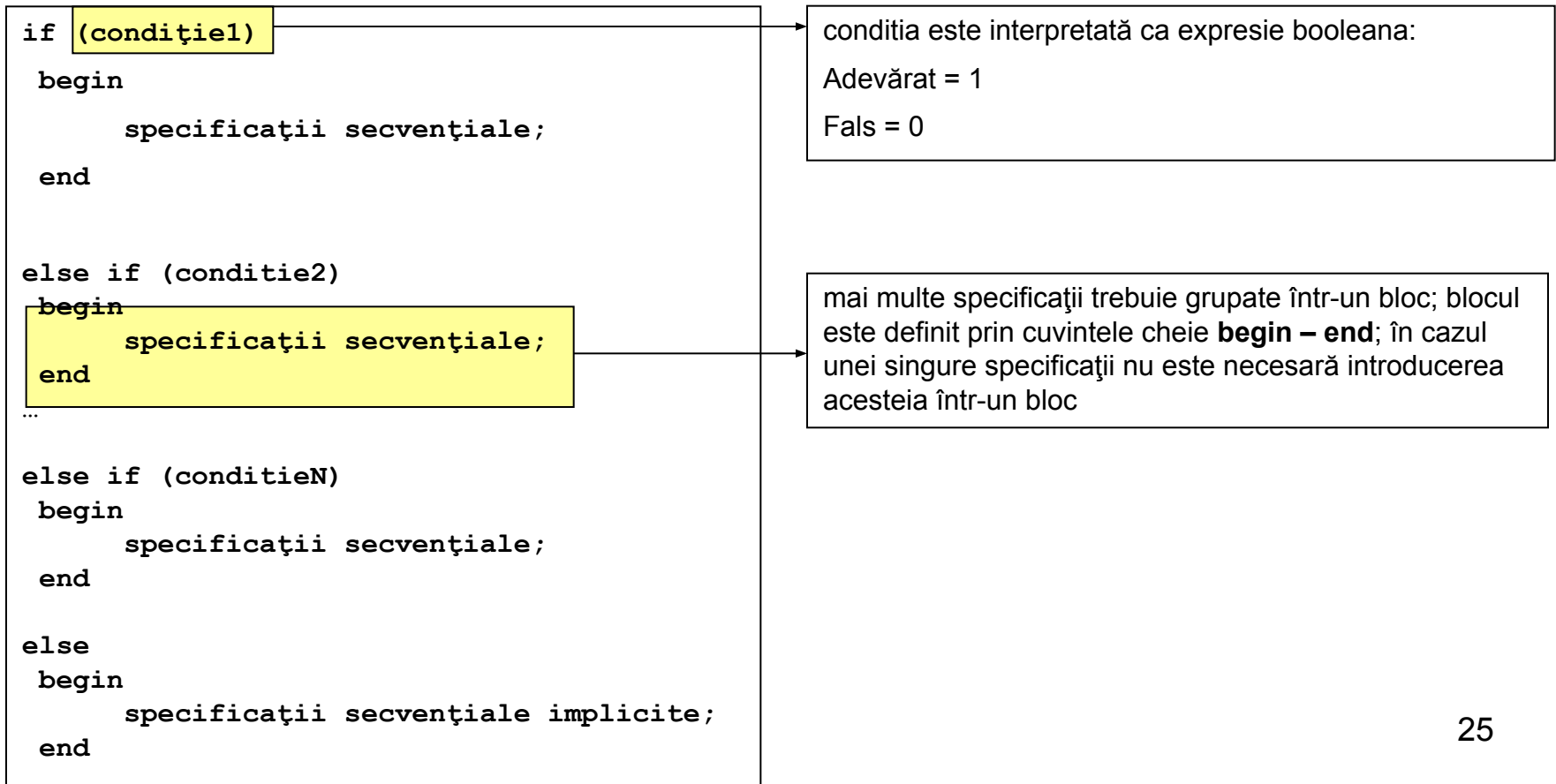
```
semnalTinta      operator_atribuire      # expresieTimp      expresie;  
( y = #10 x; )
```

- etapa 1, evaluare `expresie` se execută la timpul curent de simulare (`t=0`);
- etapa 2, actualizare `semnal` se execută abia la timpul de simulare specificat de valoarea generată de `expresieTimp` (`t=10`);

Întârzierile nu se utilizează în modelele sintetizabile

Specificația condițională `if`

- utilizată pentru implementarea ramificațiilor controlate de anumite **condiții booleene**
- în sinteza logică este specifică descrierii circuitelor care funcționează pe baza unei **logici prioritare** (codificatoarele prioritare, semnale cu priorități diferite: reset, activare circuit, diferite funcții)
- în circuitele care includ logică prioritară, mai multe condiții pot fi simultan adevărate, la un moment dat; (condițiile se referă la expresii compuse din semnalele de intrare), dar numai cea care are prioritate maximă este luată în considerare



Exemple pentru specificația condițională `if`

```
if (sel)           // testeaza sel =1
    dout = di1;   // daca e adevarat atunci executa asta
else              // altfel ( sel ∈ {0 ,x, z} )
    dout = di0;   // executa asta
```

```
if (!en)          // testeaza en = 0
    dout = di1;   // daca e adevarat atunci executa asta
else              // altfel ( en ∈ {1 ,x, z} )
    dout = 1'b0; // fa asta
```

```
if (ctl == 2'b0)  // testeaza ctl = 0
begin            // inceput bloc de cod
    do1 = di1 + di0; // daca ctl = 0 atunci face asta
    do0 = di1 - di0; // si asta
end              // sfirsit bloc de cod

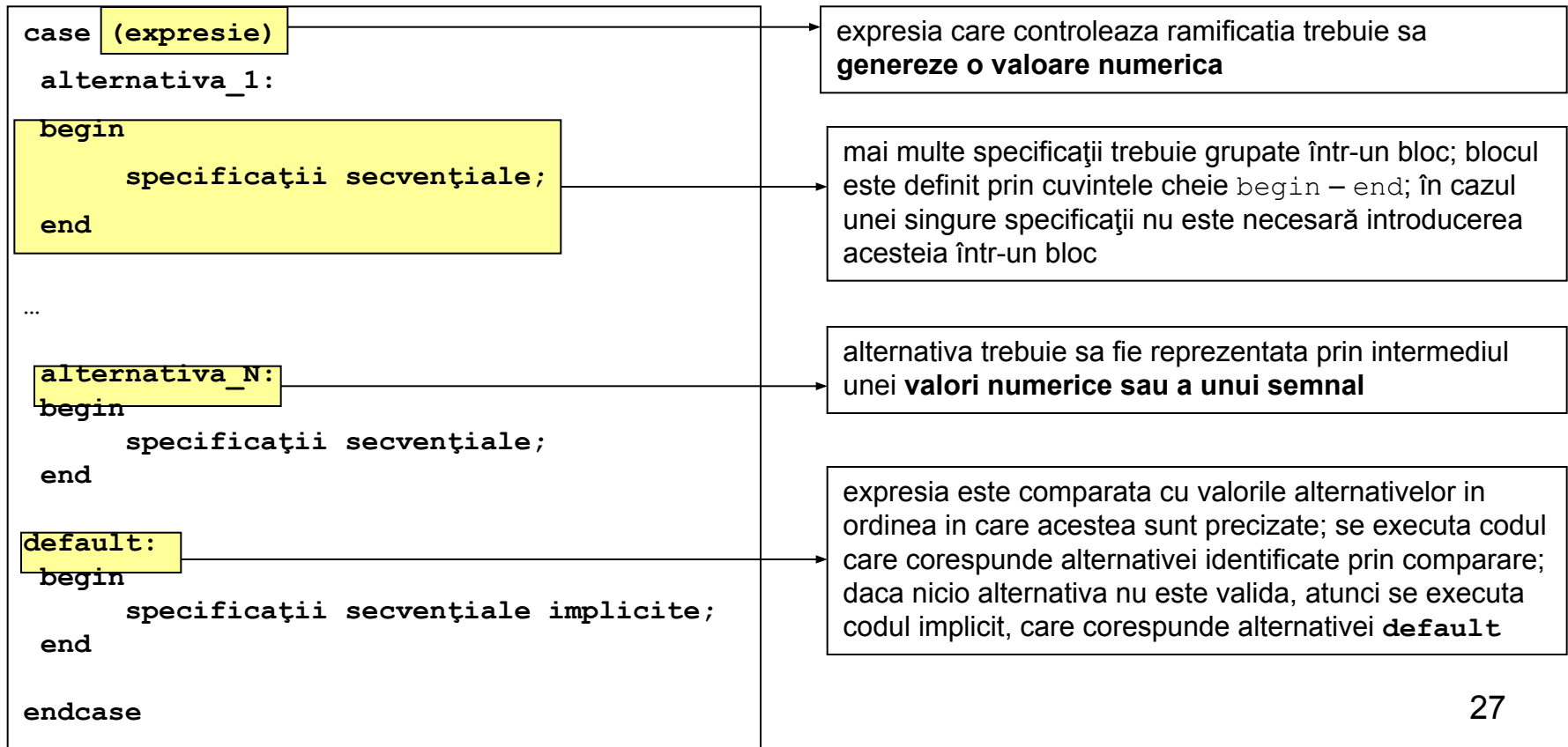
else if (ctl == 2'b01) // testeaza ctl = 1
begin            // inceput bloc de cod
    do1 = di1 & di0; // daca ctl = 1 atunci face asta
    do0 = di1 | di0; // si asta
end              // sfirsit bloc de cod

else if (ctl == 2'b10) // testeaza ctl = 2
begin            // inceput bloc de cod
    do1 = di1 * di1; // daca ctl = 2 atunci face asta
    do0 = di0 * di0; // si asta
end              // sfirsit bloc de cod

else             // altfel
begin            // inceput bloc de cod
    do1 = di1;     // daca ctl ≠0, 1, 2 atunci face asta
    do0 = di0;     // si asta
end              // sfirsit bloc de cod
```

Specificația de selecție case

- utilizată pentru implementarea ramificațiilor controlate de anumite **valori numerice**
- în sinteza logică, se recomandă utilizarea acestei instrucțiuni **pentru circuitele în care expresiile între semnalele de intrare au valori mutual exclusive;**
- valori mutual exclusive = numai o singură valoare poate fi validă la un moment dat



Specificația de selecție casez respectiv casex

- **casez**: tratează toate valorile **z** care apar în alternative sau în expresie ca valori “nu contează”
- **casex**: tratează toate valorile **x** sau **z** care apar în alternative sau în expresie ca valori “nu contează”

```
casex (sel)                // se testează valoarea semnalului sel
  4'b 1xxx: do = di3;      // daca bitul sel[3] = 1 (iar ceilalti biti ai semnalului sel nu conteaza) atunci executa asta
  4'b 01xx: do = di2;      // daca bitul sel[2] = 1 (iar ceilalti biti ai semnalului sel nu conteaza) atunci executa asta
  4'b 001x: do = di1;      // daca bitul sel[2] = 1 (iar ceilalti biti ai semnalului sel nu conteaza) atunci executa asta
  4'b 0001: do = di0;      // daca bitul sel[0] = 1 (iar ceilalti biti ai semnalului sel nu conteaza) atunci executa asta
  default: do = 1'b0;      // altfel executa asta
endcase
```

Bucla for

- se utilizează pentru implementarea secvențelor repetitive
- se utilizează în descrierea structurală, pentru descrierea structurilor repetitive
- în sinteză, bucla **for** nu se utilizează pentru descrieri comportamentale (algoritmi)

```
for (initializare_index; conditie; iteratie)
    specificatii
```

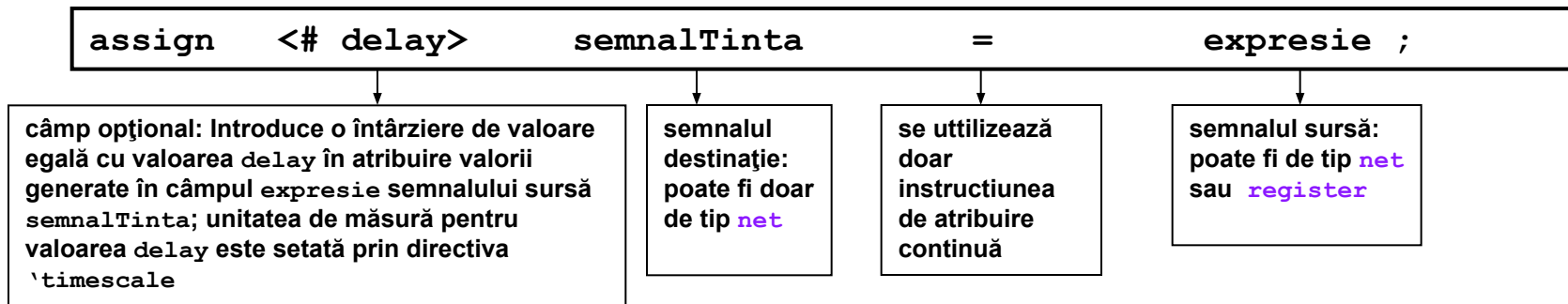
Cazul semnalelor multisursă

Constituie o sursă de erori în modelarea sistemelor digitale!

```
module c02ex 07 (  
    input x,y,  
    output reg z);  
  
    always@(x) z = x; // deoarece blocurile always se executa  
    always@(y) z = y; // concurent, semnalul z primeste  
                    // simultan valori din 2 surse diferite  
  
endmodule
```

VI. Descrierea “dataflow” a sistemelor digitale

- utilizată pentru descrierea ecuațiilor care modelează sistemul = ieșirea este exprimată direct, prin intermediul unei funcții, care are ca argumente intrările;
- funcția este implementată prin intermediul unei expresii în care sunt utilizați operatori logici, aritmetici, pe biți, de reducere, condiționali sau de concatenare;
- descrierea **dataflow** este realizată pe baza instrucțiunii de **atribuire continuă assign** = (*continuous assignment*)



- instrucțiunile de atribuire continuă **NU** se introduc în blocuri **always** sau **initial**;
- într-un modul, toate instrucțiunile de atribuire continuă se execută în mod concurrent;
- instrucțiunea de atribuire se execută în mod continuu (atribuirea valorilor nu se poate planifica în funcție de un eveniment);
- semnalul țintă poate fi numai de tip `net`.
- operanzii implicați în `expresie` pot fi de tip `net`, `register`, constante sau apeluri de funcții.

Exemple:

```
assign      dout = di1 + di0; // atribuie lui dout valoarea di1 + di0
assign # 5  dout = di1 or di0; // dupa o intirziere de 5 unitati de timp, atribuie lui dout valoarea logică di1 sau
// di0 unitatea de masura pentru unitatea de timp se stabileste prin directiva // `timescale
assign      dout = sel? di1:di0; // atribuie lui dout valoarea di1 daca sel = 1, respectiv di0 daca sel = 0
```

VII Descrierea structurală

Elemente utilizate:

a. Componente:

instanțieri de module

primitive Verilog (porți logice)
sau definite de utilizator

b. Conexiuni între componente:

semnale de tip **wire**

Instanțierea modulelor:

```
numeModul numeComponenta (lista conexiuni interfață)
```

Specificarea conexiunilor interfeței:

a. prin nume (recomandată):

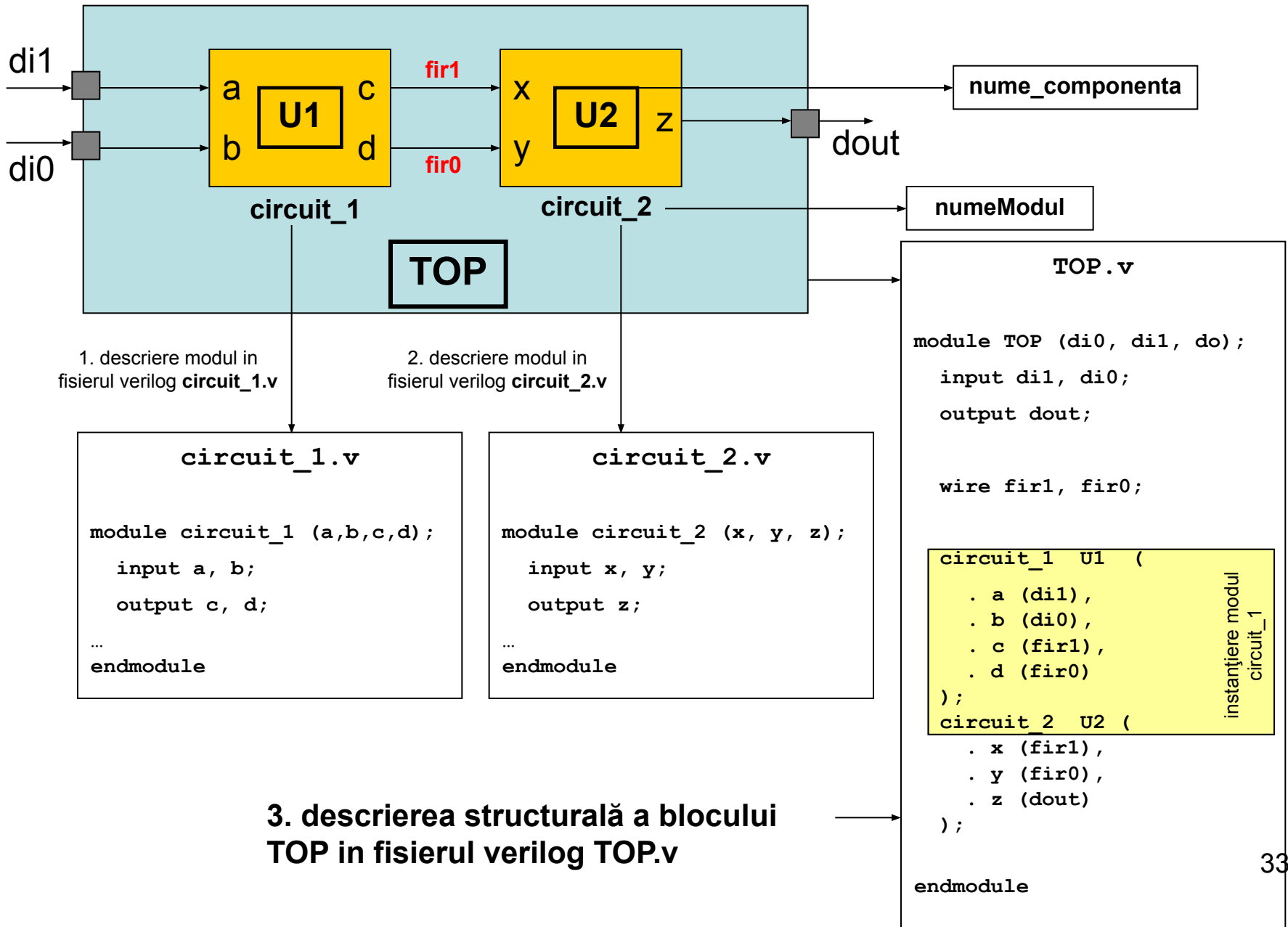
```
numeModul numeComponenta (  
    . semnal_interfață (semnal_extern) ,  
    .....  
    . semnal_interfață (semnal_extern) );
```

b. prin poziție:

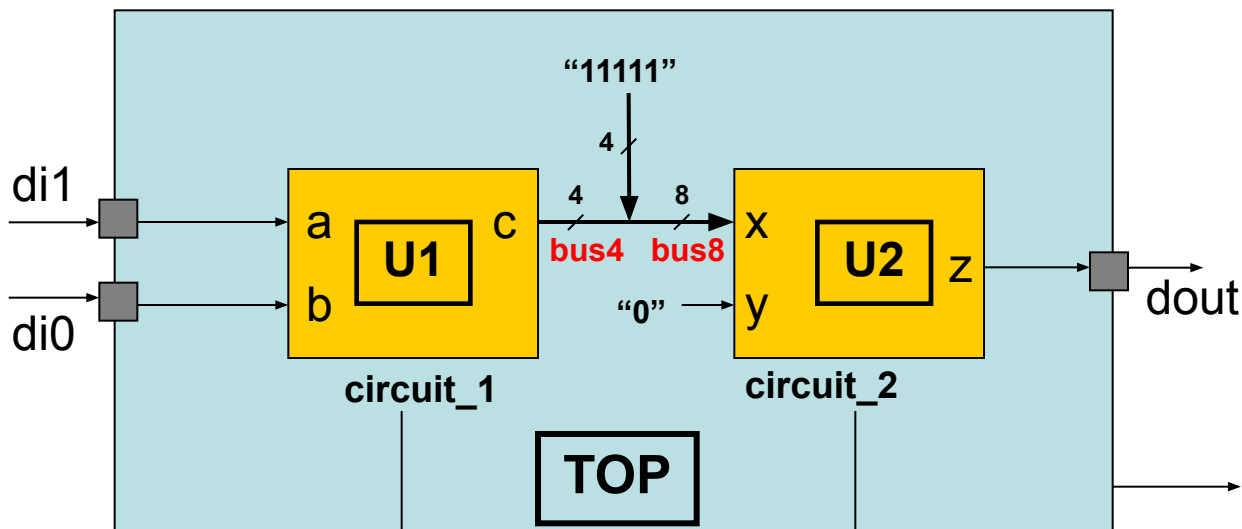
```
numeModul numeComponenta (lista semnale externe) );
```

**Semnalele din interfață care nu trebuie conectate, nu se mai specifică în lista de conexiuni;
Semnalelor din interfață li se pot atribui și valori binare.**

Exemplul 1 de descriere structurală



Exemplul 2 de descriere structurală



1. descriere modul in
fisierul verilog **circuit_1.v**

```

circuit_1.v

module circuit_1 (
    input a, b,
    output[3:0] c
);
...
endmodule
    
```

2. descriere modul in
fisierul verilog **circuit_2.v**

```

circuit_2.v

module circuit_2 (
    input[7:0] x,
    input y,
    output z
);
...
endmodule
    
```

3. descrierea structurală a blocului
TOP in fisierul verilog **TOP.v**

```

TOP.v

module TOP (
    input di1, di0,
    output dout
);

wire[3:0] bus4;
wire[7:0] bus8;

circuit_1 U1 (
    . a (di1),
    . b (di0),
    . c (bus4)
);

assign bus8 = {4'b1111, bus4};

circuit_2 U2 (
    . x (bus8),
    . y (1'b0),
    . z (dout)
);

endmodule
    
```

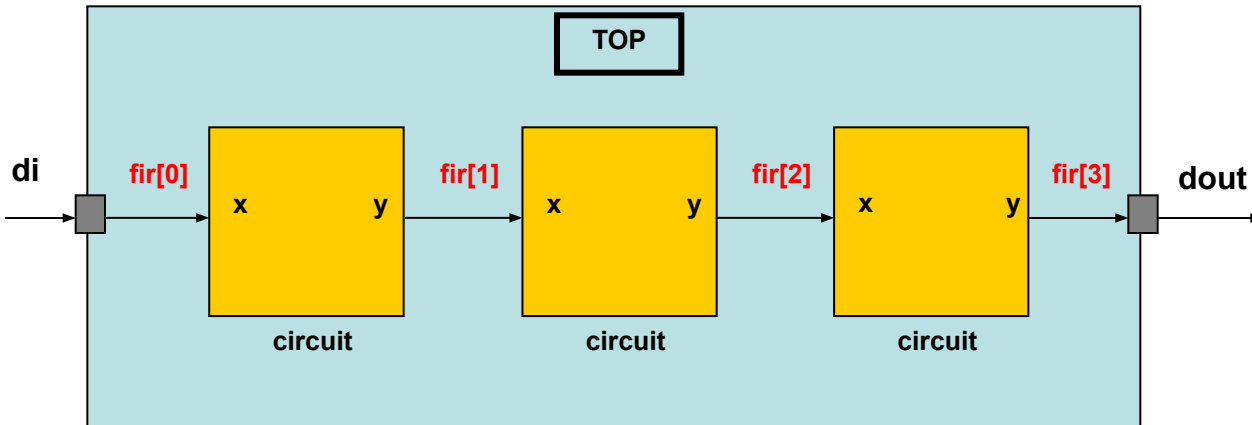
Descrierea structurilor regulate prin bucla for

```
generate
  genvar index;

  for (initializare_index; conditie; iteratie)

    begin: nume_structura_regulata
      instantiere modul
    end

endgenerate
```



```
wire [3:0] fir;

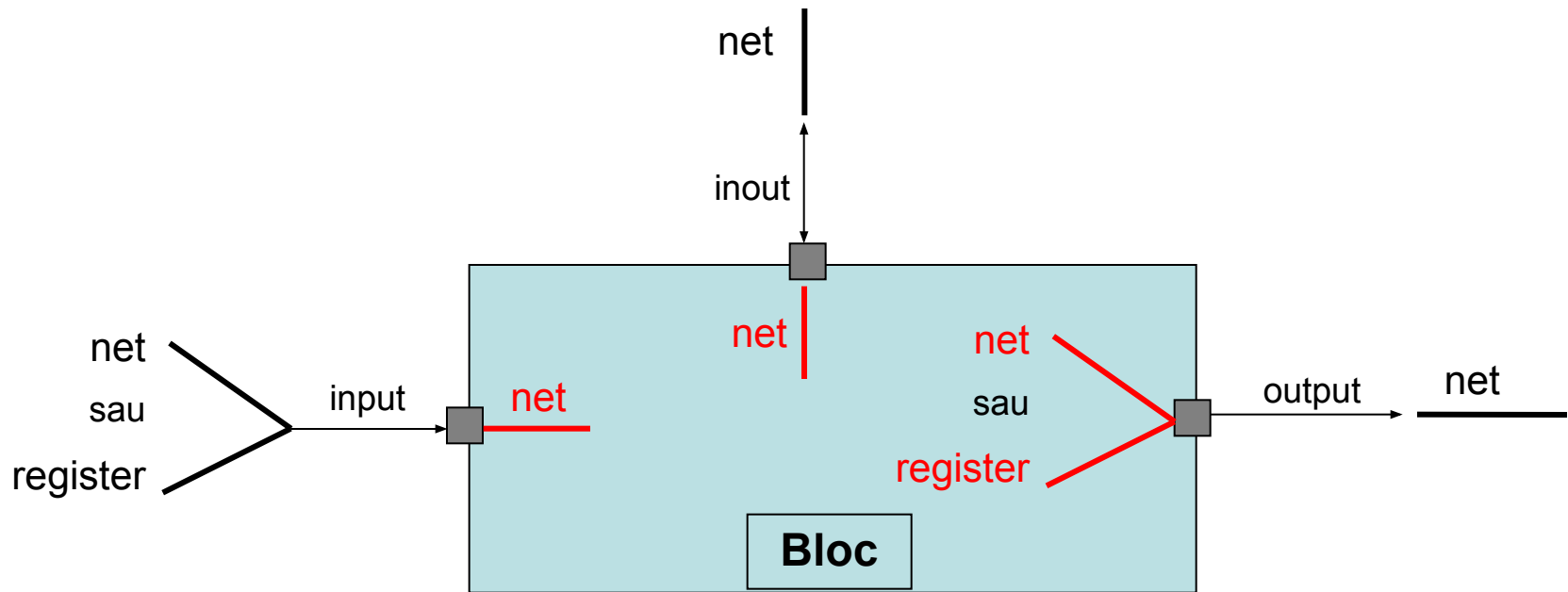
generate
  genvar i;

  for (i=0; i<=2; i=i+1)
    begin: struct_regulata
      circuit U (
        .x (fir[i]),
        .y (fir[i+1])
      );
    end

endgenerate

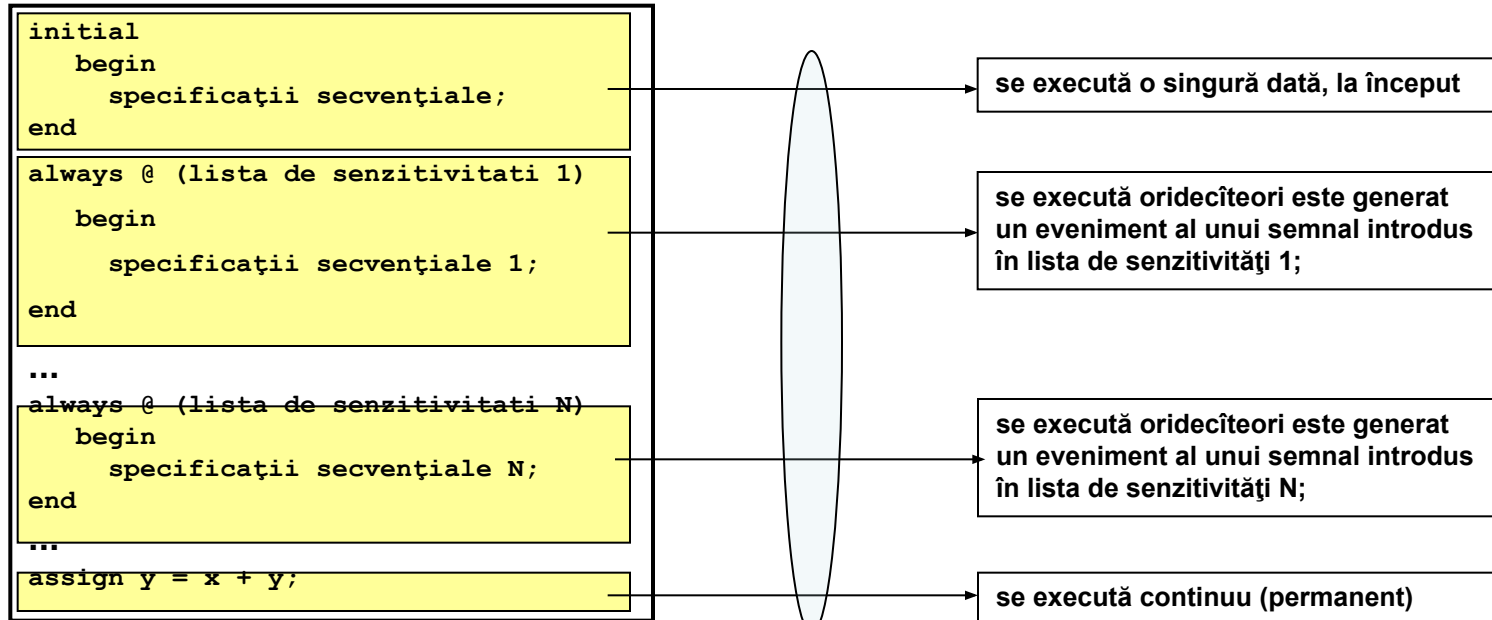
assign fir[0] = di;
assign dout = fir[3];
```

Tipurile de conexiuni permise în descrierile structurale pentru semnalele din interfața moduluiui



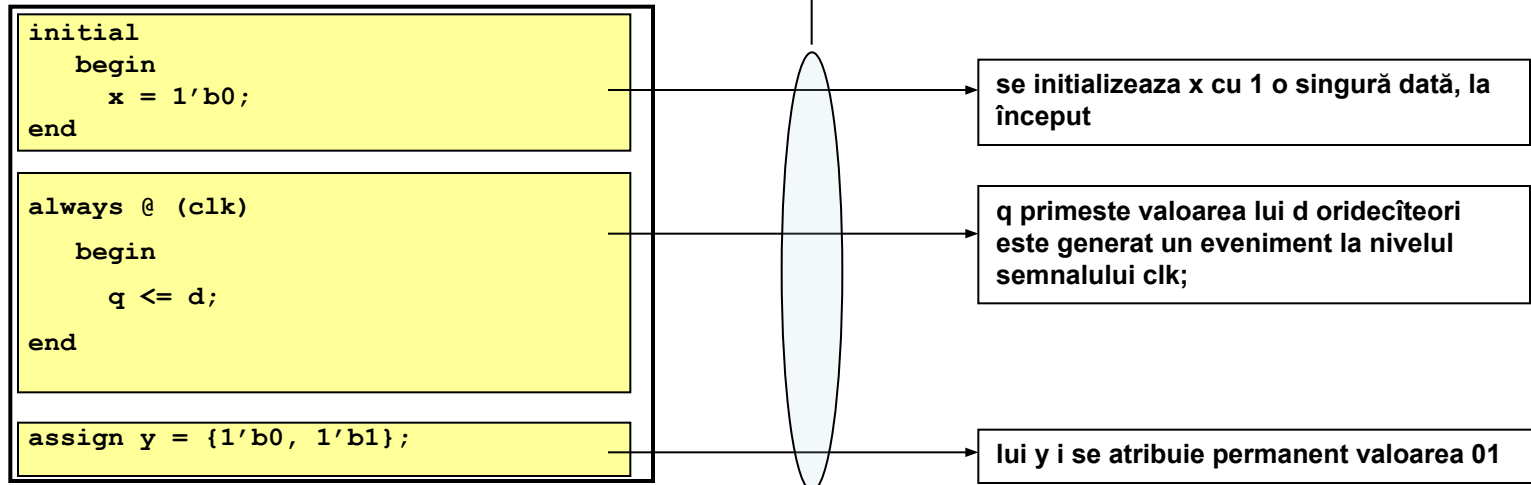
directie semnal interfata:	input	inout	output		
tip conexiune permisa din/spre exterior:		net sau register	net	net	
tip conexiune permisa spre/din interior:	net	net	net sau register		

Execuția concurentă a blocurilor



se execută concurrent

exemplu:



Execuția proceselor în timpul simulării

