

LIMBAJUL VHDL - 3



PROCESE

Definiții

- **unitatea de bază** pentru descrierea de **tip comportamental** (funcțional)
- **procesul** = o **serie de operații secvențiale** care în **timpul simulării** constituie **o singură acțiune**
- **procesul** = obiectul fundamental manipulat de simulator → orice descriere VHDL = un set de procese caracterizate de:
 - semnalele la care sunt sensibile (active)
 - operațiile secvențiale executate de fiecare

PROCESE

Sintaxa

```
{etichetă:} {postponed} process {listă_de_sensibilitate}  
...  
... Zona de declarații locale procesului  
...  
begin  
...  
... Instrucțiuni secvențiale  
...  
end {postponed} process {etichetă};
```

Sintaxa

- În partea declarativă:
 - este **interzisă declararea semnalelor**
 - se pot declara variabile
 - se pot declara subprograme interne

PROCESE

Execuția

- un proces există nedefinit - este global
- durata de viață a unui proces este cea a simulării
- **timpul de execuție** al unui proces este **zero**
- procesele se execută în **paralel** , în mod **concurrent**

PROCESE

Execuția

- un sistem real își execută secvența de activități specifice, pentru care a fost construit, în buclă infinită
 - orice instrucțiune concurentă poate fi transcrisă în termenii unui proces (procesul echivalent)
 - un proces nu se termină niciodată - el execută în buclă lista de instrucțiuni secvențiale - este ciclic
-

Suspendarea și reactivarea

- funcționarea dispozitivelor electronice:
 - operează în buclă infinită
 - execută operațiile specifice
 - își **suspendă** funcționarea
 - așteaptă îndeplinirea unor **condiții de reactivare**
 - **reiau** operațiile
- un proces se execută până se întâlnește o instrucțiune **wait**

Instrucțiunea wait

- **scop**: emularea funcționării reale
- **suspendă** procesul când operațiile secvențiale prevăzute au fost efectuate
- **reactivează** procesul când sunt **îndeplinite condițiile** specificate → mai multe tipuri de instrucțiuni **wait**, pentru a asigura varietatea de condiții reale

Instrucțiunea wait

- **3 tipuri** de instrucțiuni
 - **wait for** expresie de tip Time - se așteaptă trecerea unui interval de timp
 - **wait until** condiție de tip Boolean - se așteaptă până condiția devine True în urma unei modificări
 - **wait on** listă de sensibilitate - se așteaptă până un semnal din listă își modifică valoarea

Instrucțiunea wait

■ localizarea

- **wait** poate apărea oriunde în proces
- într-un proces pot exista mai multe instrucțiuni **wait**
- lista de sensibilitate poate apărea după **process** și e echivalentă cu “**wait on** listă de sensibilitate” aflată la sfârșitul procesului

Instrucțiunea wait

■ restricții

- semnalele din lista de sensibilitate - să fie statice
- **wait on** nu poate fi utilizată în proces când există listă de sensibilitate
- **wait** nu poate fi utilizată în procedurile apelate de proces

PROCESE

Procese pasive

- procesele sunt pasive dacă nici un semnal nu apare în membrul stâng al unei instrucțiuni de asignare
 - execuția proceselor pasive nu antrenează execuția altor procese
 - în entități pot fi folosite instrucțiuni concurente care au procese echivalente pasive
 - exemplu: instrucțiunea concurentă **assert** are proces pasiv echivalent
-

PROCESE

Procese amânate

- procesele amânate se activează în momentul **ultimei întârzieri delta** a unui ciclu de simulare
 - se folosește **postponed**
 - instrucțiunile concurente care se pot transforma în procese amânate:
 - instrucțiunea concurentă **assert**
 - apelul concurent de procedură
 - asignarea concurentă de semnal
-

Procese amânate

- restricții

- nu pot conține asignări de semnale cu întârziere nulă
- nu se poate folosi valoarea atributelor predefinite asupra semnalelor

PROCESE

Semnale în procese

- **restricții** de utilizare a semnalelor în procese:
 - în procese nu se pot declara semnale
 - orice asignare a unei valori unui semnal are efect doar când procesul se suspendă - până atunci se păstrează valorile anterioare
 - ultima asignare a unei valori unui semnal este luată în considerare la suspendarea procesului
 - dacă semnalul este pe lista de sensibilitate a procesului, modificarea lui reactivează procesul
-

PROCESE

Variabile în procese

- variabila permite stocarea temporară a datelor
 - se poate defini în cadrul procesului - cuvânt cheie **variable**
 - utilizare la descriere de algoritmi în procese
-

Variabile în procese

- asignarea de valori:
 - cu simbolul $:=$
 - instantanee
 - de câte ori este necesar
- poate avea orice tip sau subtip posibil, constrâns sau neconstrâns
- valoarea inițială - expresie statică, de același tip cu tipul variabilei

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea assert

- supraveghează o condiție și dacă este falsă emite un mesaj
- sintaxa:

assert condiție{**report** mesaj}{**severity** nivel_de_severitate_al_erorii};

- mesajul implicit: Assertion Violation
- nivelul de severitate al erorii este de tipul Severity_Level (Note, Warning, Error, Failure), cu Error valoare implicită

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea report

- permite afișarea unui mesaj

- sintaxa:

```
{etichetă:} report mesaj report  
    {nivel_de_severitate_al_mesajului};
```

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea de asignare a variabilelor

- valoarea este preluată imediat
- asignarea se poate face la declarare

Instrucțiunea de apel de procedură

- apelarea trebuie să indice numele și în paranteză lista parametrilor de apel

INSTRUCȚIUNI SECVENȚIALE

Structura condițională

- este structurată
- permite **executarea condiționată** a unor secvențe de instrucțiuni
- când condiția booleană este True se execută ramura **if**, pentru False se execută ramura **else**
- ramura **elsif** permite înlănțuirea condițiilor

INSTRUCȚIUNI SECVENȚIALE

Structura condițională

■ sintaxa:

if A>B then

-- Secvența_de_instrucțiuni_1

elsif A=B then

-- Secvența_de_instrucțiuni_2

else

-- Secvența_de_instrucțiuni_3

end if;

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea case

- permite **selectarea**, în funcție de valoarea unei expresii, a unei secvențe de instrucțiuni dintre mai multe alternative
- expresia și valorile trebuie să fie de același tip discret (enumerat)
- ordinea ramurilor nu contează
- ramura **others** trebuie să fie ultima și este obligatorie dacă nu sunt specificate toate valorile posibile ale expresiei

100

■ sintaxa:

when Valoare_1 =>... -- Secv_instrucțiuni_1

```
when Valoare_5 to Valoare_6 =>...
-- Secv_instructiuni_3
```

```
...
when others =>...           -- Secv_instrucțiuni_n
```

```
end case;
```

INSTRUCȚIUNI SECVENȚIALE

Structura de buclă

- permite **repetarea** secvenței de instrucțiuni din cadrul ei
- fiecare trecere se numește **iterație**
- dacă schema de iterație nu este precizată → număr infinit de iterații
- dacă schema este precizată → arată numărul de repetări

INSTRUCȚIUNI SECVENȚIALE

Structura de buclă

- sintaxa generală:

```
{etichetă:} {schemă de iterație} loop  
    -- Secvența_de_instrucțiuni  
end loop {etichetă};
```


INSTRUCȚIUNI SECVENȚIALE

Structura de buclă

- prima schemă de iterație:
 - atât timp cât condiția este adevărată se repetă instrucțiunile din secvență
 - testarea condiției - la începutul fiecărei iterații

while condiție **loop**

-- Secvența_de_instrucțiuni

end loop;

INSTRUCȚIUNI SECVENȚIALE

Structura de buclă

- a doua schemă de iterație:
 - secvența de instrucțiuni se repetă de un număr de ori cunoscut doar la execuție
 - variabila de buclă:
 - contorizează numărul de cicluri efectuate prin parcurgerea unui tip enumerat
 - nu trebuie declarată
 - este cunoscută numai în interiorul buclei
 - nu i se poate atribui nici o altă valoare → nu poate fi modificată

INSTRUCȚIUNI SECVENȚIALE

Structura de buclă

- a doua schemă de iterație:
 - nu există posibilitatea specificării unui pas
 - secvența nu se execută pentru interval vid sau negativ

```
for Indice in 1 to 100 loop
```

```
-- Secvența de instrucțiuni
```

```
end loop;
```


INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea next

- permite oprirea iterației în curs de desfășurare a unei bucle
- execuția continuă cu iterația următoare (dacă există)
- poate fi:
 - imperativă: **next** {eticheta_buclei};
 - condițională (întrerupere când condiția este adevărată): **next** {eticheta_buclei} **when** condiție;
- fără etichetă se referă la bucla cea mai de jos

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea exit

- permite ieșirea din buclă
- întrerupe toate iterațiile restante ale buclei
- execuția continuă cu instrucțiunea de după **end loop**
- poate fi:
 - imperativă: **exit** {eticheta_buclei};
 - condițională (ieșire când condiția este adevărată):
exit {eticheta_buclei} **when** condiție;
- fără etichetă se referă la bucla cea mai de jos

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea return

- este rezervată subprogramelor
- la execuția ei se suspendă subprogramul și controlul revine apelantului
- poate fi folosită pentru a întrerupe o procedură și a reveni în programul apelant → nu trebuie să i se asocieze o valoare: **return;**

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea return

- orice funcție se termină dinamic prin **return** urmată de valoarea returnată: **return** Valoare;
- valoarea returnată trebuie să aibă tipul declarat în specificația funcției
- o funcție poate avea mai multe instrucțiuni **return**, pentru că pot exista mai multe ramuri de decizie

INSTRUCȚIUNI SECVENȚIALE

Instrucțiunea nulă

- se trece la executarea instrucțiunii următoare
 - sintaxa: **null;**
 - practic se folosește la instrucțiuni de selecție (case) când toate ramurile trebuie luate în considerare
 - nu este necesară la compilarea unui proces sau a unui corp de procedură vid
-

SUBPROGRAME

Generalități

- permit scrierea unor algoritmi reutilizabili
- valorile parametrilor, diferite la apel, duc la efecte diferite
- 2 tipuri de subprograme:
 - proceduri: **procedure**
 - funcții: **function**
- apelul unei proceduri este o instrucțiune
- apelul unei funcții apare ca o valoare, în membrul drept al instrucțiunilor de asignare

SUBPROGRAME

Declarația de subprogram

- subprogramele au 2 părți:
 - declarația (specificația)
 - corpul
 - declarația indică:
 - genul subprogramului (procedură sau funcție)
 - numele
 - lista parametrilor formali (fiecare cu mod și tip)
 - pt. funcție și tipul valorii returnate
-

SUBPROGRAME

Declarația de subprogram

- sintaxa:

- procedură

procedure nume_procedură (lista_parametrilor_formali);

- funcție

{**pure/impure**} **function** nume_funcție (lista_parametrilor_formali)

return tipul_rezultatului;

- lista parametrilor formali

{clasă_obiect} nume_parametru_1{, nume_parametru_2}:

{mod_transmitere}

type valoare_implicită;

SUBPROGRAME

Modul de transmitere a parametrilor

- intrare - **in**
 - implicit
 - parametrii pot fi citiți, nu pot fi modificați
 - ieșire - **out**
 - numai pentru proceduri
 - parametrii nu se pot citi
 - combinat intrare / ieșire - **inout**
 - numai pentru proceduri
 - permite orice citire și scriere
-

SUBPROGRAME

Corpul subprogramului

- conține algoritmul implementat
- nu permite declararea semnalelor
- sintaxa:

antet_sub-program is

{partea declarativă}

begin

{partea rezervată instrucțiunilor}

end {nume_sub-program};

SUBPROGRAME

Apelul

- poate fi secvențial sau concurent
- 2 moduri de indicare a parametrilor actuali ai subprogramelor:
 - prin poziție
 - prin denumire (cu =>)

SUBPROGRAME

Supraîncărcarea

- 2 subprograme cu același nume, dar profiluri diferite
- profilul: numărul, ordinea și tipul parametrilor formali și tipul rezultatului pentru funcție

ARHITECTURI CONCURENTE

Generalități

- VHDL descrie sistemele ca mulțimi de **subsisteme** funcționale care **operează** în mod **concurrent**
 - fiecare subsistem specificat prin proces separat → nivel de detaliere dat de necesități
 - toate **procesele** din interiorul unei arhitecturi **se execută concurrent**
 - în VHDL → **combinare** între operații concurente și secvențiale
-

ARHITECTURI CONCURENTE

Generalități

- transferul de informații între procese se face prin semnale
- procesele se activează indiferent dacă modificarea valorii semnalelor la care sunt active este produsă de mediul extern sau de către alt proces

ARHITECTURI CONCURENTE

Procese elementare

- procesele care conțin o singură instrucțiune
→ instrucțiuni singulare de asignare concurentă de semnal
 - instrucțiunile singulare de asignare concurentă de semnal:
 - apar în arhitecturi și se execută concurent cu alte procese
 - sunt sensibile la modificarea oricărui semnal care apare în membrul drept
 - asignarea se poate întârzia cu **after**
-

ARHITECTURI CONCURENTE

Valori de semnale - pilotul (driver)

- semnalele primesc valori noi în momentul suspendării proceselor prin instrucțiunea **wait**
- stocarea informațiilor referitoare la evenimentele de pe semnal → prin **pilot** (driver)
- compilatorul creează un pilot pentru fiecare semnal care primește o valoare într-un proces
- există un singur pilot / semnal / proces

ARHITECTURI CONCURENTE

Valori de semnale - pilotul (driver)

- toate operațiile sunt efectuate asupra pilotului
 - pilotul copiat în semnal când procesul se suspendă
 - semnalul cunoaște valorile trecute, prezente și viitoare → pilotul are asignată o formă de undă
 - forma de undă formată din tranzacții
 - tranzacția: pereche valoare semnal + valoare timp (Time)
-

ARHITECTURI CONCURRENTE

Semnale cu mai mulți piloți

- semnale care au informații provenite din **mai multe surse**
 - exemplu: magistrala internă a procesorului primește informații de la procesor, memoria internă, hard discuri, dispozitive de intrare / ieșire
- în unele sisteme situația trebuie obligatoriu evitată, în altele este utilizată (exemplu: pentru ȘI și SAU cablat)
- necesită stabilirea unei metode de determinare a valorii rezultate pentru semnal

ARHITECTURI CONCURRENTE

Rezolvarea semnalelor multi-sursă

- simulatorul nu poate ști dacă un semnal va fi activat din mai multe surse
- simulatorul trebuie să fie pregătit pentru a realiza “rezolvarea” (mixarea) semnalelor
- regulile de mixare se specifică într-un tabel care reprezintă o **funcție de rezoluție** → conține toate valorile posibile pentru semnal

ARHITECTURI CONCURENTE

Valori pentru semnale

- și semnalele uni-sursă pot avea mai mult de 2 valori (specifice reprezentării binare)
- în VHDL există tipul de date Std_Ulogic, nerezolvat, care cuprinde:
 - 'X' - valoare indiferentă
 - 'Z' - înaltă impedanță - la buffere three-state
 - 'U' - valoare necunoscută
 - 'L' - valoare '0' slabă
 - 'H' - valoare '1' slabă
 - 'W' - valoare necunoscută slabă
 - '-' - valoare fără importanță

ARHITECTURI CONCURENTE

Funcția de rezoluție

- semnalele rezolvate - au funcție de rezoluție
 - funcția de rezoluție poate să apară la declarații de subtipuri de date și la declarații de semnal
 - la tipurile compuse funcția de rezoluție pentru tipul compus maschează funcțiile de rezoluție pentru elementele tipului compus
 - în simulare funcția de rezoluție este **utilizată automat**, nu este controlată de proiectant
-

ARHITECTURI CONCURENTE

Funcția de rezoluție

- tipul de date rezolvat este Std_Logic din pachetul Std_Logic_1164
- există și versiunea rezolvată pentru Std_Logic_Vector

INSTRUCȚIUNI CONCURENTE

Paralelism

- la instrucțiuni concurente ordinea de execuție este oarecare
- se aplică un paralelism real, dar dispare simularea în timp real
- concurența din VHDL = **paralelism real în timp virtual** (de simulare)

INSTRUCȚIUNI CONCURENTE

Instrucțiunea block

- are 3 funcții principale:
 - încapsularea declarațiilor - reunire de instrucțiuni concurente care au acces la declarații locale
 - utilizarea instrucțiunilor gardate - permite scrierea de instrucțiuni de asignare condiționate
 - suport pentru ierarhizare - se pot scrie proiecte ierarhizate
 - blocul este unitatea de bază echivalentă a structurării în VHDL
-

INSTRUCȚIUNI CONCURENTE

Instrucțiunea block

■ sintaxa:

```
etichetă: block {(condiție_de_gardă)}  
    {antet_generice_și_porturi}  
    -- Declarații locale  
begin  
    -- Instrucțiuni concurente  
end block {etichetă};
```

INSTRUCȚIUNI CONCURENTE

Instrucțiunea block

- condiția de gardă = expresie booleană
 - antetul opțional indică importarea din mediul exterior:
 - valori pentru parametri generici
 - semnale pentru porturi
 - partea declarativă:
 - vizibilă numai local
 - nu permite declarații de variabile locale
-

INSTRUCȚIUNI CONCURENTE

Instrucțiuni gardate

- înlocuiesc descrierile în care se repetă aceeași condiție de multe ori
- garda unui bloc factorizează condițiile
- garda este o expresie booleană
- semnalul Guard - poate fi declarat explicit și folosit la asignare gardată

INSTRUCȚIUNI CONCURENTE

Apel concurent de procedură

- aceeași sintaxă ca la apelul secvențial
- are în plus în procesul echivalent instrucțiuni **wait**
- parametrii pot fi doar constante sau semnale
- util la aplicații de gestionare a stării interne a automatelor finite

INSTRUCȚIUNI CONCURENTE

Instrucțiunea assert

- aceeași sintaxă ca la apelul secvențial
- poate să apară în entități sau arhitecturi
- monitorizarea condiției date este permanentă

INSTRUCȚIUNI CONCURENTE

Instrucțiunea de asignare de semnal

- are 2 forme: condițională și selectivă
- sintaxa formei condiționale:

```
{etichetă:} nume_sau_agregat <= {guarded}  
  formă_de_undă_1 when condiție_booleană_1 else  
  formă_de_undă_2 when condiție_booleană_2 else  
  ...  
  formă_de_undă_n;
```


INSTRUCȚIUNI CONCURENTE

Instrucțiunea de asignare de semnal

- forma condițională echivalentă (procesul echivalent):

```
if condiție_booleană_1 then
    nume_sau_agregat <= {transport} formă_de_undă_1;
elsif condiție_booleană_2 then
    nume_sau_agregat <= {transport} formă_de_undă_2;
...
else
    nume_sau_agregat <= {transport} formă_de_undă_n;
end if;
```

INSTRUCȚIUNI CONCURENTE

Instrucțiunea de asignare de semnal

- sintaxa formei selective:

{etichetă:} **with** expresie **select**

nume_sau_agregat <= {**guarded**}{**transport**}

formă_de_undă_1 **when** alegere_1,

formă_de_undă_2 **when** alegere_2,

...

formă_de_undă_n **when** alegere_n;

INSTRUCȚIUNI CONCURENTE

Instrucțiunea de asignare de semnal

- forma selectivă echivalentă (procesul echivalent):

```
case expresie is  
when alegere_1 =>  
    nume_sau_agregat <= {transport} formă_de_undă_1;  
when alegere_2 =>  
    nume_sau_agregat <= {transport} formă_de_undă_2;  
    ...  
when alegere_n =>  
    nume_sau_agregat <= {transport} formă_de_undă_n;  
end case;
```


INSTRUCȚIUNI CONCURENTE

Instrucțiunea de instanțiere a unei componente

- ia o **copie** a unui model (componentă) declarat anterior și o personalizează pentru o necesitate particulară
- sintaxa:

etichetă: numele_componentei_model

{corespondența parametrilor generici}

{corespondența porturi efective / porturi locale modelului};

INSTRUCȚIUNI CONCURENTE

Bloc sau componentă

- nu există o diferență fundamentală între un bloc și o instanță de componentă
- ambele pot fi folosite la descrierea ierarhizată
- **componenta** are avantajul că este reutilizabilă

INSTRUCȚIUNI CONCURENTE

Instrucțiunea generate

- permite elaborarea condițională sau iterativă a liniilor de cod sursă VHDL
 - forma condițională - instrucțiunile concurente vor exista la elaborare după îndeplinirea condiției booleene
 - forma iterativă - creează ansambluri de instrucțiuni în număr egal cu numărul de elemente din intervalul discret
-

INSTRUCȚIUNI CONCURENTE

Instrucțiunea generate

- sintaxa:

- forma condițională

etichetă: **if** condiție_booleană **generate**

... Secvență de instrucțiuni concurente

end generate {etichetă};

- forma iterativă

etichetă: **for** nume_parametru_de_generare **in**
interval_discret **generate**

... Secvență de instrucțiuni concurente

end generate {etichetă};

MODULE DE SIMULARE

Scop

- orice proces de proiectare presupune și **etapa de verificare**
 - în VHDL - mai multe metode de verificare
 - modul de simulare (test bench) = mediu în care un proiect = **UST** (unitate supusă testării) este verificat prin aplicarea unor semnale numite **stimuli** și observarea răspunsurilor generate
-

MODULE DE SIMULARE

Elemente

- soclu (socket) - în el se plasează sistemul testat
 - generator de stimuli - subsistem care aplică stimuli proiectului testat
 - stimuli generați intern
 - stimuli preluați de la o sursă externă de semnale
 - instrumente de monitorizare a răspunsurilor la stimuli, generate de sistemul supus testării
-

MODULE DE SIMULARE

Elemente în VHDL

- modulul de simulare este o specificație VHDL, care este simulată de simulatorul VHDL integrat în mediul de dezvoltare VHDL
 - modulul de simulare este alcătuit din:
 - instanțierea unității supuse testării (UST)
 - procese sensibile la stimuli aplicați unității supuse testării (UST)
 - specificație hibridă: structurală + comportamentală
-

MODULE DE SIMULARE

Elemente în VHDL

- stimulii pot fi specificați în arhitectura modulului de simulare sau pot fi citați din fișier extern
- reacțiile unității testate pot fi observate prin:
 - forme de undă generate de simulatorul VHDL
 - fișierele de raport cu mesaje generate de simulator
 - mesajele generate de simulator la consolă
 - scrierea în fișiere folosind operațiile de intrare / ieșire în mod text disponibile în pachetul Textio

MODULE DE SIMULARE

Structură

- specificație cu entitate și arhitectură
 - entitatea modulului de simulare:
 - nu are porturi și parametri generici pentru că modulul de simulare nu este un dispozitiv real
 - nu poate lipsi pentru că arhitecturile nu pot fi specificate fără entități asociate
 - în arhitectură se realizează instanțierea UST
= o specificație de tip structural - relația dintre modulul de simulare și UST
 - poate fi instanțiere directă
 - instanțierea unei componente
-

MODULE DE SIMULARE

Structură

- stimuli - element esențial:
 - set de semnale declarate intern în arhitectura modulului de simulare
 - asigurați porturilor UST prin **port map** la instanțierea ei
 - definiți ca forme de undă:
 - prin instrucțiuni concurente de asignare de valori la semnale
 - în cadrul unuia sau a mai multor procese comportamentale, cu instrucțiuni **wait for** și la sfârșit cu o instrucțiune **wait** vidă pentru suspendarea procesului

MODULE DE SIMULARE

Structură

- arhitectura modulului de simulare fiind un domeniu concurrent nu contează ordinea în care este instanțiată UST și definiți stimulii

MODULE DE SIMULARE

Utilizare

- la sisteme complexe simularea poate fi costisitoare
 - verificare mai simplă cu module de simulare
 - **se simulează modulul de simulare**, nu unitatea supusă testării (ea este doar o instanță de componentă)
 - nu există limitări la dimensiunea modulului de simulare
-

Afişare şi raportare rezultate

- verificarea trebuie să şi afişeze sau să raporteze rezultatele
 - modalităţi de afişare şi raportare:
 - afişarea listei valorilor semnalelor care se modifică în timp (echivalentă cu afişarea formelor de undă)
 - scrierea rezultatelor simulării într-un fişier (log file)
 - folosirea instrucţiunii **assert**
-

MODULE DE SIMULARE

Instrucțiunea assert

- utilizată pentru raportarea răspunsurilor eronate generate de unitatea supusă testării
 - moduri posibile de utilizare:
 - se aplică o instrucțiune **assert** de fiecare dată când se așteaptă o nouă valoare a unui semnal de ieșire al UST
 - valoarea prognozată se specifică drept condiție
 - se folosesc mesaje de eroare precise și detaliate (CE nu funcționează și CÂND a avut loc evenimentul)
-

PACHETE STANDARD ȘI PREDEFINITE

Pachete standard

- Standard
 - Textio
 - definite în manualul de referință VHDL
 - nu pot fi modificate de proiectanți
-

PACHETE STANDARD ȘI PREDEFINITE

Pachete predefinite

- Std_Logic_1164 - normă IEEE
 - IEEE Numeric_Std
 - IEEE Numeric_Bit
 - Std_Logic_Arith al firmei Synopsys
 - Std_Logic_Unsigned al firmei Synopsys
-