

# **LUCRAREA NR. 1**

## **INTRODUCERE ÎN VHDL**

### **1. Scopul lucrării**

În această lucrare se prezintă elementele fundamentale legate de limbajul de descriere hardware VHDL: scurt istoric, definiții, avantaje și dezavantaje, principalele sale caracteristici, însoțite de exemple elementare de circuite, în principalele stiluri de descriere existente în VHDL.

### **2. Considerații teoretice**

#### **2.1 VHDL – scurt istoric**

VHDL nu este un limbaj de programare, ci un limbaj de descriere a sistemelor electronice hardware pornind de la structura lor modulară și de la interconexiunile dintre acestea. El a fost definit și integrat în rândul instrumentelor de **CAD** (Computer-Aided Design) din domeniul electronicii, pentru a introduce o metodologie riguroasă de proiectare în ciclul de dezvoltare al sistemelor hardware.

VHDL a devenit un limbaj industrial standardizat, utilizat pentru descrierea hardware de la nivelul abstract până la nivelul concret. VHDL a fost rapid asimilat ca un mediu universal de comunicație în proiectare. Toți producătorii de stații de lucru și de software **CAE** (Computer-Aided Engineering) își standardizează produsele pentru a avea intrări și ieșiri standard VHDL. Aceste produse includ software pentru simulare, sinteză și trasare de cablaj imprimat.

Limbajul provine din programul **VHSIC** (Very High Speed Integrated Circuit) inițiat de Departamentul Apărării din Statele Unite ale Americii în 1980. În fața importante creșteri a complexității sistemelor electronice și mai ales a costurilor de întreținere rezultante, s-a făcut simțită nevoia apariției unui limbaj modern și standardizat.

Necesitatea unei descrieri lipsite de ambiguitate a sistemelor hardware a apărut în mod pregnant la Departamentul Apărării (DOD) al Statelor Unite la începutul anilor 1980. Efortul de standardizare a fost

eșalonat între anii 1983 și 1987 sub egida DOD. Efortul prestat de numeroase societăți americane, grupate în jurul lui INTERMETRICS, IBM și TEXAS INSTRUMENTS, a dus la apariția normei IEEE 1076B, aprobată în 10 decembrie 1987. De atunci, VHDL este supervizat de grupul american VASG (VHDL Analysis and Standardisation Group), care face la rândul lui parte din IEEE.

## 2.2 Ce este un limbaj de descriere hardware?

VHDL este un *limbaj de descriere hardware*: el permite descrierea funcționării componentelor unui sistem hardware, precum și a relațiilor dintre aceste componente și a legăturilor lor cu exteriorul.

Există o strânsă similitudine între proiectarea hardware și proiectarea software. Tot ceea ce ține de metoda de specificare, de organizarea software-ului, de algoritmică, poate fi transpus direct la nivel hardware. Acest lucru este posibil deoarece VHDL conține toate elementele de descriere algoritmică proprii limbajelor de programare. De aceea este adeseori considerat drept un limbaj informatic sau chiar drept un limbaj de informaticieni. Este deci preferabil ca programatorul începător în VHDL să posede cunoștințe minime într-un limbaj de programare structurată de nivel înalt, pe lângă cunoștințe de proiectare a sistemelor numerice.

Spre deosebire de un limbaj de programare, un limbaj de descriere hardware nu vizează o „executare”, chiar dacă are un simulator asociat. Un limbaj de descriere hardware poate servi la:

- *demonstrații formale* („oare circuitul face într-adevăr transformata Fourier?”);
- *sinteză* (în acest caz, limbajul are rolul de a furniza intrări unui instrument „inteligent” pentru realizarea rapidă a prototipului hardware);
- *elaborarea de specificații*;
- *elaborarea de documentații*.

Un limbaj de descriere hardware propune adeseori mai multe niveluri de descriere. O descriere VHDL poate fi:

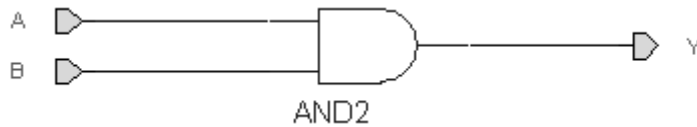
- *structurală*;
- *comportamentală*;
- de tip „*flux de date*”;
- *o combinație a acestor trei tipuri de bază*.

Nivelul cel mai ușor de înțeles este cel *structural*. O descriere pur structurală constă în a descrie modelul prin structura sa, adică printr-un

ansamblu de elemente interconectate. De aici se pot deduce imediat două proprietăți esențiale:

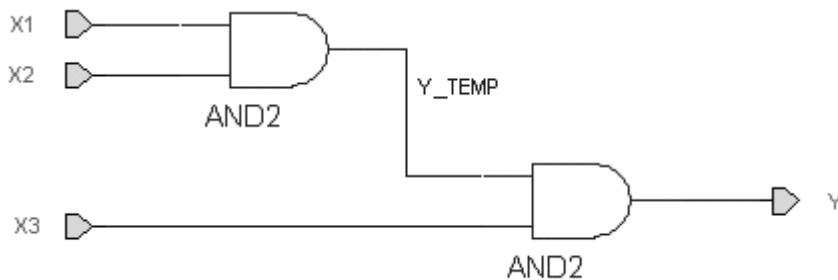
- o asemenea descriere nu ia în calcul timpul;
- o asemenea descriere nu poate constitui o „frunză” (element terminal în ierarhia de descriere) în momentul simulării. O frunză va fi în mod necesar descrisă în stil comportamental sau „flux de date”.

În cele ce urmează oferim un prim exemplu de descriere structurală:



**Figura 1.1** Poarta ȘI

```
-- specificația unei porți ȘI
entity AND_B is
  port (A,B: in bit;
        Y: out bit);
end AND_B;
architecture ARHITECTURA_1 of AND_B is
begin
  Y <= A and B;
end ARHITECTURA_1;
```



**Figura 1.2** Circuit cu două porți ȘI - descriere structurală

```
-- Specificație structurală: circuit conținând două porți ȘI
entity PORȚI_ȘI is
  port (X1, X2, X3: in bit;
```

```

        Y: out bit);
end PORȚI_ȘI;
architecture ARHITECTURA_STRUCTURALĂ of PORȚI_ȘI is
component AND_B
    port (A, B: in bit;
          Y: out bit);
end component AND_B;
signal Y_TEMP: bit;
begin
    U1: AND_B port map (X1, X2, Y_TEMP);
    U2: AND_B port map (Y_TEMP, X3, Y);
end ARHITECTURA_STRUCTURALĂ;

```

Descrierea de nivel *comportamental* urmărește să descrie *funcționarea (comportamentul)* unui model fără a se preocupa de o eventuală împărțire în blocuri, mai apropiată de nivelul implementării fizice. Descrierea capătă adeseori forma unui algoritm de genul celor utilizați în limbajele de programare clasice. Acest nivel de descriere, deși este prezent în „frunzele” ierarhiei, se poate adresa și proiectanților care doresc să modeleze un sistem la un nivel înalt de abstractizare. Aici poate interveni și timpul. A modela sistemul în stil comportamental înseamnă a fi preocupat de funcționalitatea modelului descris, ceea ce face ca adeseori să se folosească termenul de „descriere funcțională” în loc de „descriere comportamentală”.

```

-- specificație comportamentală: circuit conținând 2 porți ȘI
entity PORȚI_ȘI is
    port (X1, X2, X3: in bit;
          Y: out bit);
end PORȚI_ȘI;

architecture ARHITECTURA_COMPORTAMENTALĂ of PORȚI_ȘI is
    signal Y_TEMP: bit;
begin
    PORȚI: process (X1, X2, X3, Y_TEMP)
    begin
        Y_TEMP <= X1 and X2;
        Y <= Y_TEMP and X3;
    end process PORȚI;
end ARHITECTURA_COMPORTAMENTALĂ;

```

O descriere de tip „*flux de date*”, care nu este decât o formă prescurtată a unei descrieri comportamentale, exprimă fluxurile datelor care

ies din model în funcție de intrările primite, fără a se preocupa de structura acestuia.

```
-- specificație „flux de date”: circuit conținând 2 porți ȘI
entity PORȚI_ȘI is
port (X1, X2, X3: in bit;
      Y: out bit);
end PORȚI_ȘI;

architecture ARHITECTURA_FLUX_DE_DATE of PORȚI_ȘI is
signal Y_TEMP: bit;
begin
    Y_TEMP <= X1 and X2;
    Y <= Y_TEMP and X3;
end ARHITECTURA_FLUX_DE_DATE;
```

Obiectivul unui limbaj de programare îl constituie *descrierea unei execuții* a unui program, în vreme ce un limbaj precum VHDL are ca obiectiv *descrierea structurii hardware* a unui sistem. Partea de hardware are o structură fixă a cărei *stare logică* evoluează în decursul timpului. Această evoluție face parte din descrierea în VHDL a sistemului. Se poate spune că un program VHDL are o *structură fixă* și o *execuție evolutivă*.

*Deosebirea esențială* constă în faptul că *programul este secvențial*, în vreme ce *descrierea hardware este concurentă*. Aceasta este o problemă de interpretare a codului: este posibil să se dea o descriere hardware cu ajutorul unui limbaj de programare modificând semantica limbajului.

O *deosebire majoră* între un limbaj de programare și un limbaj de descriere pur structural este *separarea elementelor lor constructive de bază*: *sub-programul* - în cazul limbajului algoritmic, și *componenta* - în cazul limbajului de descriere.

*Sub-programul* este apelat la un anumit moment dat, își îndeplinește misiunea, apoi este oarecum „uitat”.

*Componenta* există în sine, căci o descriere hardware la nivel structural este „în afara timpului”. Componentele unui sistem există în mod static și funcționează în mod *concurent*. De fapt, timpul nu intervine decât în partea „flux de date” (sau comportamentală) a unui asemenea limbaj, care permite scrierea unei instrucțiuni de genul: „lui A i se atribuie valoarea ‘0’, apoi, după 10 ns, valoarea ‘1’, apoi după încă 5 ns valoarea ‘0’ ” etc.

O altă deosebire dintre cele două categorii de limbaje este dată de abstracțiunile purtătoare de valori. Într-un limbaj de programare, o variabilă are o durată de viață limitată de cea a sub-programului care o conține, și

poartă succesiv (sau poate purta) mai multe valori. Un semnal în VHDL există de-a lungul întregii simulări (dacă se efectuează simularea) și păstrează într-o agendă *sui-generis* lista valorilor care au fost generate de interacțiunea sa cu celelalte semnale, sau care i-au fost transmise de către sub-descrierile funcționale sau „flux de date”. De asemenea, el poartă în mod implicit lista valorilor pe care le-a luat în trecutul său (așa-numitul său „istoric” - *history*).

La modul general, o atribuire de variabilă (notată cu „:=” în VHDL) are un efect limitat în timp.

```
A := B; -- instrucțiune de asignare de variabilă
```

înseamnă că variabila A ia valoarea B atunci când se execută instrucțiunea, în vreme ce atribuirea de semnal (notată cu „<=” în VHDL)

```
S <= E; -- instrucțiune de asignare de semnal
```

semnifică un scurt-circuit permanent între semnalele S și E (dacă atribuirea este concurentă).

### 2.3 Avantaje și dezavantaje ale VHDL

Principalul inconvenient al VHDL este complexitatea sa. Ea nu trebuie deloc neglijată îndeosebi datorită faptului că limbajul se adresează proiectanților de sisteme electronice care nu au în mod necesar cunoștințe foarte temeinice de programare.

În schimb, VHDL este un standard IEEE recunoscut de către toți producătorii de instrumente CAD. Perenitatea garantată de normă le permite producătorilor să investească într-un instrument care nu este doar o modă efemeră. Aceste investiții se reflectă în multitudinea de instrumente și de biblioteci de modele care apar pe piață. Disponibilitatea unor modele comportamentale și structurale ale circuitelor existente permite efectuarea simulării componentelor în contextul lor operațional. Invers, un proiectant ar fi putut ezita să integreze în sistemul său o componentă al cărei model VHDL nu este disponibil, această componentă interzicându-i simularea ansamblului.

Deocamdată, nu se pune problema apariției vreunui alt limbaj de descriere hardware care să înlocuiască VHDL. În plus, DOD din S.U.A

susține în continuare standardul VHDL și solicită, de exemplu, descrierea în VHDL a fiecărui circuit integrat comandat.

Din punct de vedere tehnic, VHDL este un limbaj modern, puternic și general:

- *Lizibilitatea* VHDL este excelentă, grație unei structurări riguroase a expresiilor utilizate;
- *Modularitatea* VHDL este remarcabilă, grație organizării sub formă de componente și a parametrilor generici - care permit parametrizarea dimensiunilor componentelor;
- *Securitatea în exploatare* este foarte mare, grație declarării legăturilor externe separat de descrierea internă a unei componente și datorită tipizării legăturilor externe și a modurilor asociate acestora (intrare, ieșire, intrare / ieșire);
- *Fiabilitatea* descrierilor VHDL este deosebită.

Aceste caracteristici decurg direct din conceptele moderne (specifice limbajelor de programare clasice) aplicate în VHDL: *unitățile de compilare separate, tipizarea puternică sau existența parametrilor generici.*

VHDL mai oferă (printre altele):

- *o bună tratare a concurenței;*
- *o definiție foarte solidă a noțiunii de timp*, care permite o descriere precisă și sintetică a evoluției temporale a sistemului descris, ceea ce constituie însăși baza descrierii unui sistem hardware, grație unei semantici specifice introduse în limbaj;
- *posibilitatea combinării stilurilor de descriere: structurală, comportamentală și „flux de date”;*
- *posibilitatea definirii unor funcții de rezoluție (de gestiune a conflictelor) evaluate.*

*Ciclul de proiectare* al unui sistem hardware se împarte, la modul tradițional, în trei faze:

- faza de proiectare propriu-zisă (un fel de analiză funcțională a problemei);
- faza de realizare (codificare);
- faza de depanare.

Cu cât o eroare este depistată mai rapid în ciclul de proiectare, cu atât mai ieftină este corectarea sa. Așadar, toate limbajele moderne (printre care și VHDL) se străduiesc, prin intermediul verificărilor efectuate, să detecteze cât mai multe erori încă de la compilare. Prin urmare, utilizarea VHDL are o influență foarte puternică asupra ciclului de proiectare al produsului.

Un alt aspect remarcabil al VHDL este dat de *proiectarea modulară și ierarhizată* pe care o induce acest limbaj. Ea le permite responsabililor să schimbe specificațiile sau realizarea unei componente a produsului foarte târziu în cadrul ciclului de proiectare fără a genera o adevărată catastrofă (cum era cazul până la apariția acestui gen de limbaj).

În sumar, avantajele VHDL sunt:

1. *Limbaj complet* - acoperă diferitele etape ale proiectării sistemelor numerice: specificare, modelare, simulare, sinteză. Nu mai este necesară învățarea câte unui limbaj diferit pentru fiecare dintre aceste etape;
2. *Limbaj independent* - fiind un standard IEEE, VHDL nu aparține nimănui care să acapareze evoluția și utilizarea sa. Independența sa față de arhitecturi, sisteme gazdă (PC-uri sau stații de lucru) și tehnologie este totală;
3. *Limbaj flexibil* - VHDL îi permite utilizatorului să-și efectueze descrierile plasându-se la diferite nivele de abstractizare: comportamental, „flux de date” sau structural;
4. *Limbaj modern* - VHDL are o sintaxă lizibilă, este puternic tipizat și controlat, minimizează riscul de erori și propagarea lor insidioasă, favorizează calitatea. VHDL permite proiectarea modulară, arhivarea și re-utilizarea codului;
5. *Limbaj standard* - este o garanție a compatibilității, a portabilității, a stabilității și a perenității;
6. *Limbaj deschis* - un limbaj standardizat acum 10 ani n-ar avea nici o șansă de a rezista dacă nu ar evolua. Dar evoluțiile sunt controlate, devenind și ele standard. De altfel, VHDL suportă mecanisme de funcții și proceduri care le permit proiectanților să extindă funcționalitățile limbajului, respectând principiul portabilității și o serie de reguli foarte stricte.

## 2.4 Caracteristici generale

### 2.4.1 Organizarea în biblioteci



VHDL este un limbaj modular. Stilul de lucru constă în scrierea unor unități mici, ierarhizate. Anumite părți ale acestor descrieri pot fi compilate separat. Ele sunt suficiente pentru a fi înțelese: acestea sunt *unitățile de proiectare*.

Întrucât complexitatea sistemelor de modelat este adeseori considerabilă, munca de echipă se dovedește din ce în ce mai necesară. Munca de echipă necesită însă o anumită disciplină, sau metodologie, care este oferită - parțial - de către VHDL.

De fiecare dată când o unitate de proiectare VHDL este considerată corectă de către compilatorul VHDL, ea este automat plasată într-o așa-numită *bibliotecă de lucru* generată de mediul VHDL. Proiectantul lucrează în biblioteca sa proprie, făcând referință, dacă este nevoie, la biblioteca comună a proiectului sau la alte biblioteci aparținând colegilor săi, care conțin utilitare sau modele generale. Acestea sunt, pentru el, *biblioteci de resurse*.

Toate verificările de coerență relative la biblioteci sunt asumate de către compilatorul VHDL: este rolul său de „bibliotecar”. Aceste biblioteci nu conțin decât unități de proiectare. Un fișier conținând cod sursă VHDL, odată analizat și compilat, nu mai există pentru proiectant; numai unitățile de proiectare rezultate sunt plasate în biblioteci. *În VHDL, compilăm fișiere și utilizăm (referim) unități de proiectare.*

#### 2.4.2 Încapsularea în unități de proiectare

O descriere hardware este constituită dintr-un ansamblu de modele și de algoritmi utilizați pentru acele modele.

Un model poate fi văzut ca având două părți:

- *partea externă*, care arată conexiunile sale cu lumea exterioară;
- *partea internă*, care descrie realizarea sa sub formă de interconexiuni ale altor modele mai simple sau pur și simplu sub formă de algoritmi.

Un model VHDL este o pereche *entitate / arhitectură*, partea sau vederea sa externă fiind numită *entitate*, iar vederea sa internă - *arhitectură*.

Fiecare dintre aceste două părți este o unitate de proiectare și aceste unități alcătuiesc bibliotecile VHDL. O bibliotecă poate conține, de exemplu, o sută de unități de proiectare. Separarea entității de arhitectură apare pentru a permite modificarea ușoară a funcționării interne fără a mai fi necesară redefinirea viziunii externe a modelului, lucru permis de compilarea separată.

În plus, aceeași vedere externă poate avea mai multe vederi interne asociate, acestea din urmă depinzând adeseori de gradul de finețe dorit pentru descrierea funcționării modelului. Totuși, pentru o anumită simulare dată, va fi necesară alegerea unei vederi interne precise asociate modelului.

Algoritmii utilizați frecvent pot fi separați, obținându-se astfel *sub-programe* care sunt adeseori grupate în *pachete*. Aceste pachete (*packages*) sunt supuse aceleiași împărțiri ca și modelele: vederea lor externă, adică perspectiva tuturor posibilităților pe care le oferă (pe care le „exportă”), este decuplată de vederea internă care conține algoritmică ce realizează toate aceste funcționalități. În terminologie VHDL, vederea externă se numește *specificația pachetului*, pe când vederea internă se numește *corpul pachetului*. Pachetele nu se limitează doar la a exporta sub-programe, ci pot oferi și alte obiecte, cum ar fi semnale globale, tipuri și componente.

Separarea interfeței sub-programului de corpul său permite modificarea unui algoritm fără a se modifica și „imaginea” (vederea) pe care o are lumea exterioară (celelalte modele și pachete) asupra pachetului corespunzător.

Pentru a nu bloca relația instanță / model, modul de asociere al componentelor și modelelor va forma o unitate de proiectare de sine stătătoare, care va fi și ea stocată într-o bibliotecă. Această unitate va efectua corespondența dintre instanță și model (cuplul entitate / arhitectură al cărui „reprezentant” este). Ea este o *configurație* VHDL, ultima categorie de unitate de proiectare care poate fi găsită într-o bibliotecă VHDL.

#### 2.4.3 Separarea în domeniul concurent și domeniul secvențial

Un sistem hardware este în mod natural concurent. Nu trebuie însă trasă concluzia pripită că descrierile secvențiale ar fi inutile: de exemplu, un protocol de citire din memorie se exprimă cel mai firesc în mod secvențial: poziționăm semnalul X, apoi semnalul Y, așteptăm un interval de timp  $\Delta t$  etc.

În VHDL, domeniile concurent și secvențial vor coabita. Proiectantul va avea posibilitatea de a exprima fiecare parte a descrierii sale în domeniul care i se pare cel mai adecvat. Fiecare dintre aceste domenii are setul său de instrucțiuni.

#### 2.4.4 Clasificarea după tipuri

Toate limbajele de programare moderne utilizează tipizarea obiectelor manipulate și oferă posibilitatea creării de noi tipuri. Astfel se obține o creștere a puterii verificărilor și o limitare a propagării erorilor.

Fiecare obiect va fi clasificat într-un tip menit să definească valorile pe care le poate lua și cele care-i sunt interzise, operațiile permise și cele interzise.

Orice obiect are în mod obligatoriu un tip pe care nu și-l schimbă niciodată. Există patru familii de tipuri:

- tipurile scalare (întregi, flotanți, tipuri fizice și tipuri enumerate);
- tipurile compuse (tablouri și articole);
- tipurile acces sau poantorii (*pointers*);
- tipurile fișier.

#### 2.4.5 Clasele de obiecte

În VHDL există trei clase de obiecte: *constantele*, *variabilele* și *semnalele*.

*Constantele* au o valoare fixă definită o dată pentru totdeauna, cel mai târziu după o fază de inițializare numită *elaborare*.

*Variabilele* au o valoare modificabilă prin atribuire. Constantele și variabilele sunt obiecte ce pot fi întâlnite și în limbajele de programare.

*Semnalele* sunt specifice limbajelor de descriere hardware. Ele modelează informația care tranzitează prin fire, magistrale sau - la modul general - între componentele hardware.

#### 2.4.6 Reguli de scriere în VHDL

*Comentariile* încep cu două liniuțe „--” și se continuă până la sfârșitul liniei. Dacă e necesară continuarea comentariului pe linia următoare, este din nou necesar să apară cele două liniuțe:

```
A <= B and C ;      -- în VHDL simbolul <= efectuează o
                      -- atribuire: A ia valoarea operației (B
                      -- and C)
```

VHDL nu face deosebirea dintre literele mari și mici: **SALUT** este identic cu **salut** sau **SAluT**.

VHDL manipulează o mare diversitate de obiecte: semnale, pachete, procese etc. desemnate prin numele lor. Regulile de denumire sunt același pentru toate aceste obiecte:

- Numele lor este alcătuit dintr-o serie de caractere alfanumerice (cele 26 de litere ale alfabetului), numerice (cele 10 cifre zecimale) sau din caracterul „\_”. Sunt excluse caracterele ASCII speciale;
- Primul caracter trebuie să fie o literă;
- Caracterul „\_” nu are voie să se afle la sfârșitul unui nume, nici să apară de două ori consecutiv;
- Numele nu are voie să fie un cuvânt VHDL rezervat;
- Lungimea oricărui nume nu poate depăși o linie. Această toleranță permite atribuirea de nume explicite, preferate de obicei în locul abrevierilor. Caracterul „\_” este adeseori utilizat ca separator pentru numele „compuse”.

Există o serie de cuvinte cheie care nu pot fi utilizate ca identificatori:

abs	access	after	alias
all	and	architecture	array
assert	attribute	begin	block
body	buffer	bus	case
component	configuration	constant	disconnect
downto	else	elsif	end
entity	exit	file	for
function	generate	generic	group
guarded	if	impure	in
inertial	inout	is	label
library	linkage	literal	loop
map	mod	nand	new
next	nor	not	null
of	on	open	or
others	out	package	port
postponed	procedure	process	protected
pure	range	record	register
reject	rem	report	return
rol	ror	select	severity
signal	shared	sla	sll
sra	srl	subtype	then
to	transport	type	unaffected
units	until	use	variable
wait	when	while	with
xnor	xor		

*Literalii* sunt valori explicite atribuite diferitelor obiecte: constante, variabile, attribute etc. Notăția lor diferă în funcție de tipul obiectelor cărora li se aplică:

- **numere întregi zecimale**: este cazul cel mai simplu, folosindu-se notația zecimală obișnuită, fără simboluri suplimentare. De exemplu, **22** sau **1971**. Există posibilitatea includerii caracterului „\_” pentru ameliorarea lizibilității, fără nici o altă consecință: **1\_999\_234** este identic cu **1999234**.

- **caractere**: se scriu între apostrofuri simple: 'A', '%', 'f', ' ' (spațiu). Caracterele acceptate sunt cele din setul ASCII. Atenție, literele mari și cele mici sunt diferite atunci când este vorba despre valori literale: 'D' nu este identic cu 'd'.

- **șiruri de caractere**: se scriu între ghilimele. Iată câteva exemple: "salut", "Salut" (aceste două valori literale sunt diferite), "1001", "E\*". Un șir de caractere se poate obține prin concatenarea mai multor șiruri cu ajutorul operatorului &: "**Totul " & "este " & "minunat"**" este echivalent cu "**Totul este minunat**". Această posibilitate permite îndeosebi utilizarea mai multor linii pentru definirea unui șir, de exemplu:

**"Totul" &**

**"este" &**

**"minunat";**

Operația de concatenare în sine nu adaugă nici un spațiu - va trebui deci urmărită cu atenție prezența spațiilor în șirurile care urmează să fie concatenate.

- **biți**: se utilizează notația caracterelor. Biții pot avea ca valori numai '0' și '1'. Tipul STD\_LOGIC utilizează în plus valorile 'U', 'X', 'H', 'L', 'W', 'Z' și '-'.  
 - **vectori de biți**: utilizează notația șirurilor de caractere, constituite din simbolurile 0 și 1: de exemplu, "10011010". Sistemul de numerație implicit este cel binar, dar este posibilă utilizarea notației octale sau hexazecimale prin prefixarea șirului: O"127", X"01AC".

*Spațiile* nu sunt semnificative, cu excepția apariției lor în valorile literale și în cazurile de separare a identificatorilor. Ele nu sunt necesare între identificatori și simboluri. Următoarele două linii sunt corecte și echivalente:

<pre>A &lt;= B and C;</pre>
<pre>A&lt;= B and C ;</pre>

O instrucțiune se poate întinde pe mai multe linii, atâta timp cât sfârșitul de linie nu este plasat în mijlocul unui identificator, al unui simbol de operator sau al unui literal.

```
A <= B when C = '0' else D;
```

se poate scrie și:

```
A <= B when C = '0'  
      else D;
```

*Sfârșitul unei instrucțiuni este semnalat de prezența caracterului „;”.*

#### 2.4.7 Structura unei descrieri în VHDL

Orice descriere VHDL se descompune în mod ierarhic, la fel cum un program se descompune în sub-programe. Se pornește, la nivelul rădăcinii, cu sistemul complet văzut ca o cutie neagră, cu intrările și ieșirile sale. Această cutie neagră este descompusă în componente interconectate, care sunt la rândul lor văzute ca niște cutii negre. Se ajunge la un arbore care poate fi definit în stil ascendent (*bottom-up*) sau descendent (*top-down*), asemănător descompunerii ierarhice a unui program în funcții.

Frunzele acestei ierarhii corespund componentelor elementare descrise în manieră algoritmică secvențială. Modul de descriere a acestor componente elementare depinde de nivelul de abstractizare al descrierii (specificare, proiectare, realizare). Criteriile după care se decide oprirea descompunerii ierarhice sunt legate de complexitatea componentelor terminale și de conectivitatea acestor componente între ele. Oprirea descompunerii ierarhice corespunde obținerii unui nivel de complexitate suficient de redus, care să permită trecerea la o descriere algoritmică a funcționalității componentelor de la nivelul cel mai de jos, care poate fi ea însăși descompusă în mod ierarhic în manieră software, sub formă de apeluri imbricate de sub-programe. Conținutul unei componente descrie ieșirile în funcție de intrările și de stările sale interne.

Limita descompunerii ierarhice poate depinde de numărul de intrări și de ieșiri ale componentelor terminale, precum și de dimensiunea și deci de complexitatea codului fiecărei componente. Acest criteriu se reduce așadar la izolarea corectă a funcționalităților, astfel încât acestea să fie pe cât posibil independente unele față de celelalte.

Uneori este dificil să „spargem” ierarhia fără a fi început descrierea internă a unei componente. În cursul acestei descrieri se poate opera o nouă etapă de descompunere, atunci când o parte a componentei a fost deja scrisă.

Un alt criteriu de descompunere locală îl poate constitui separarea între partea de control și cea de prelucrare a datelor, numită și cale de date (*data path*).

Felul în care se va face în cele din urmă descompunerea depinde la modul esențial de experiența proiectantului și de familiarizarea sa cu problema de rezolvat.

### **3. Desfășurarea lucrării**

- 3.1 Se vor efectua operațiile necesare pentru descrierea și încărcarea pe FPGA a porților logice fundamentale ȘI, SAU, NU, SAU-NU, ȘI-NU, SAU-EXCLUSIV, COINCIDENȚĂ.
- 3.2 Se vor efectua operațiile necesare pentru simularea porților logice fundamentale de la punctul anterior conform pașilor din Anexa nr.1.