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, definiţii, avantaje şi dezavantaje, principalele sale caracteristici, însoţite de exemple elementare de circuite, în principalele stiluri de descriere existente în VHDL.

2. Consideraţii 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 şi de la interconexiunile dintre acestea. El a fost definit şi integrat în rândul instrumentelor de **CAD** (**C**omputer-**A**ided **D**esign) 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 comunicaţie în proiectare. Toţi producătorii de staţii de lucru şi de software **CAE** (**C**omputer-**A**ided **E**ngineering) îşi standardizează produsele pentru a avea intrări şi ieşiri standard VHDL. Aceste produse includ software pentru simulare, sinteză şi trasare de cablaj imprimat.

Limbajul provine din programul **VHSIC** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) iniţiat de Departamentul Apărării din Statele Unite ale Americii în 1980. În faţa importantei creşteri a complexităţii sistemelor electronice şi mai ales a costurilor de întreţinere rezultante, s-a făcut simţită nevoia apariţiei unui limbaj modern şi 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 eşalonat între anii 1983 şi 1987 sub egida DOD. Efortul prestat de numeroase societăţi americane, grupate în jurul lui INTERMETRICS, IBM şi TEXAS INSTRUMENTS, a dus la apariţia normei IEEE 1076B, aprobată în 10 decembrie 1987. Îmbunătăţirea standardului VHDL este supervizată de IEEE. Ultima variantă a fost publicată în decembrie 2019.

2.2 Ce este un limbaj de descriere hardware?

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

Există o strânsă similitudine între proiectarea hardware şi proiectarea software. Tot ceea ce ţine de metoda de specificare, de organizarea software-ului, de algoritmică, poate fi transpus direct la nivel hardware. Acest lucru este posibil deoarece VHDL conţine 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 cunoştinţe minime într-un limbaj de programare structurată de nivel înalt, pe lângă cunoştinţe 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:

- *demonstraţii 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 specificaţii*;
- *elaborarea de documentaţii*.

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 combinaţie a acestor trei tipuri de bază*.

Nivelul cel mai uşor de înţeles 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 proprietăţi esenţiale:

- 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:

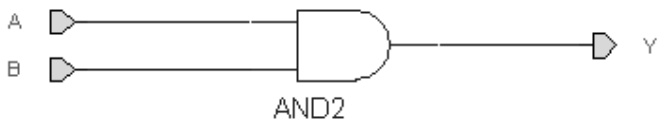

Figura 1.1 *Poarta ŞI*

```
-- specificaţia unei porţi ŞI
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;
```

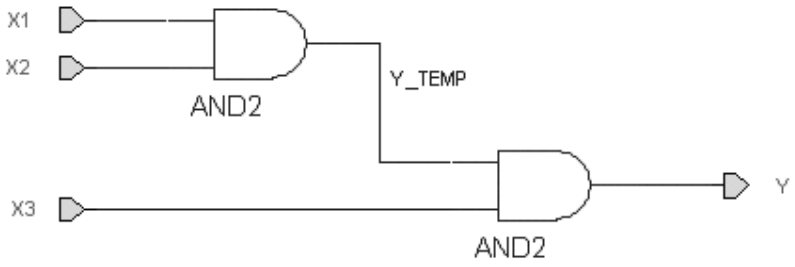

Figura 1.2 *Circuit cu două porţi ŞI - descriere structurală*

```
-- Specificaţie structurală: circuit conţinând două porţi ŞI
entity PORŢI_ŞI is
      port (X1, X2, X3: in bit;
            Y: out bit);
end PORŢI_ŞI;
architecture ARHITECTURA_STRUCTURALĂ of PORŢI_ŞI 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* urmăreşte să descrie *funcţionarea* (*comportamentul*) unui model fără a se preocupa de o eventuală împărţire în blocuri, mai apropiată de nivelul implementării fizice. Descrierea capătă adeseori forma unui algoritm de genul celor utilizaţi în limbajele de programare clasice. Acest nivel de descriere, deşi este prezent în „frunzele" ierarhiei, se poate adresa şi proiectanţilor care doresc să modeleze un sistem la un nivel înalt de abstractizare. Aici poate interveni şi timpul. A modela sistemul în stil comportamental înseamnă a fi preocupat de funcţionalitatea modelului descris, ceea ce face ca adeseori să se folosească termenul de „descriere funcţională" în loc de „descriere comportamentală".

```
-- specificaţie comportamentală: circuit conţinând 2 porţi ŞI
entity PORŢI_ŞI is
      port (X1, X2, X3: in bit;
            Y: out bit);
end PORŢI_ŞI;

architecture ARHITECTURA_COMPORTAMENTALĂ of PORŢI_ŞI is
signal Y_TEMP: bit;
begin
PORŢI: process (X1, X2, X3, Y_TEMP)
      begin
            Y_TEMP <= X1 and X2;
            Y <= Y_TEMP and X3;
      end process PORŢI;
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 funcţie de intrările primite, fără a se preocupa de structura acestuia.

```
-- specificaţie „flux de date”: circuit conţinând 2 porţi ŞI
entity PORŢI_ŞI is
port (X1, X2, X3: in bit;
      Y: out bit);
end PORŢI_ŞI;

architecture ARHITECTURA_FLUX_DE_DATE of PORŢI_ŞI 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 execuţii* 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 evoluţie face parte din descrierea în VHDL a sistemului. Se poate spune că un program VHDL are *o structură fixă* şi *o execuţie evolutivă*.

Deosebirea esenţială constă în faptul că *programul este secvenţial*, 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 şi un limbaj de descriere pur structural este *separarea elementelor lor constructive de bază*: *sub-programul* - în cazul limbajului algoritmic, şi *componenta* - în cazul limbajului de descriere.

Sub-programul este apelat la un anumit moment dat, îşi îndeplineşte 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 şi funcţionează în mod *concurent*. De fapt, timpul nu intervine decât în partea „flux de date” (sau comportamentală) a unui asemenea limbaj, care permite scrierea unei instrucţiuni 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 abstracţiunile purtătoare de valori. Într-un limbaj de programare, o variabilă are o durată de viaţă limitată de cea a sub-programului care o conţine, şi poartă succesiv (sau poate purta) mai multe valori. Un semnal în VHDL există de-a lungul întregii simulări (dacă se efectuează simularea) şi păstrează într-o agendă *sui-generis* lista valorilor care au fost generate de interacţiunea sa cu celelalte semnale, sau care i-au fost transmise de către sub-descrierile funcţionale sau „flux de date". De asemenea, el poartă în mod implicit lista valorilor pe care le-a luat în trecutul său (aşa-numitul său „istoric" - *history*).

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

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

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

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

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

2.3 Avantaje şi dezavantaje ale VHDL

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

În schimb, VHDL este un standard IEEE recunoscut de către toţi producătorii de instrumente CAD. Perenitatea garantată de normă le permite producătorilor să investească într-un instrument care nu este doar o modă efemeră. Aceste investiţii se reflectă în multitudinea de instrumente şi de biblioteci de modele care apar pe piaţă. Disponibilitatea unor modele comportamentale şi structurale ale circuitelor existente permite efectuarea simulării componentelor în contextul lor operaţional. 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 apariţiei vreunui alt limbaj de descriere hardware care să înlocuiască VHDL. În plus, DOD din S.U.A susţine în continuare standardul VHDL şi solicită, de exemplu, descrierea în VHDL a fiecărui circuit integrat comandat.

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

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

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

VHDL mai oferă (printre altele):

- *o bună tratare a concurenţei*;
- *o definiţie foarte solidă a noţiunii de timp*, care permite o descriere precisă şi sintetică a evoluţiei temporale a sistemului descris, ceea ce constituie însăşi baza descrierii unui sistem hardware, graţie unei semantici specifice introduse în limbaj;
- *posibilitatea combinării stilurilor de descriere: structurală, comportamentală şi „flux de date"*;
- *posibilitatea definirii unor funcţii de rezoluţie (de gestiune a conflictelor) evoluate*.

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

- faza de proiectare propriu-zisă (un fel de analiză funcţională 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. Aşadar, toate limbajele moderne (printre care şi VHDL) se străduiesc, prin intermediul verificărilor efectuate, să detecteze cât mai multe erori încă de la compilare. Prin urmare, utilizarea VHDL are o influenţă foarte puternică asupra ciclului de proiectare al produsului.

Un alt aspect remarcabil al VHDL este dat de *proiectarea modulară şi ierarhizată* pe care o induce acest limbaj. Ea le permite responsabililor să schimbe specificaţiile sau realizarea unei componente a produsului foarte târziu în cadrul ciclului de proiectare fără a genera o adevărată catastrofă (cum era cazul până la apariţia 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 învăţarea câte unui limbaj diferit pentru fiecare dintre aceste etape;
2. *Limbaj independent* - fiind un standard IEEE, VHDL nu aparţine nimănui care să acapareze evoluţia şi utilizarea sa. Independenţa sa faţă de arhitecturi, sisteme gazdă (PC-uri sau staţii de lucru) şi tehnologie este totală;
3. *Limbaj flexibil* - VHDL îi permite utilizatorului să-şi 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 şi controlat, minimizează riscul de erori şi propagarea lor insidioasă, favorizează calitatea. VHDL permite proiectarea modulară, arhivarea şi re-utilizarea codului;
5. *Limbaj standard* - este o garanţie a compatibilităţii, a portabilităţii, a stabilităţii şi a perenităţii;
6. *Limbaj deschis* - un limbaj standardizat acum 10 ani n-ar avea nici o şansă de a rezista dacă nu ar evolua. Dar evoluţiile sunt controlate, devenind şi ele standard. De altfel, VHDL suportă mecanisme de funcţii şi proceduri care le permit proiectanţilor să extindă funcţionalităţile limbajului, respectând principiul portabilităţii şi 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 unităţi mici, ierarhizate. Anumite părţi ale acestor descrieri pot fi compilate separat. Ele sunt suficiente pentru a fi înţelese: acestea sunt *unităţile de proiectare*.

Întrucât complexitatea sistemelor de modelat este adeseori considerabilă, munca de echipă se dovedeşte din ce în ce mai necesară. Munca de echipă necesită însă o anumită disciplină, sau metodologie, care este oferită - parţial - 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 aşa-numită *bibliotecă de lucru* generată de mediul VHDL. Proiectantul lucrează în biblioteca sa proprie, făcând referinţă, dacă este nevoie, la biblioteca comună a proiectului sau la alte biblioteci aparţinând colegilor săi, care conţin utilitare sau modele generale. Acestea sunt, pentru el, *biblioteci de resurse*.

Toate verificările de coerenţă relative la biblioteci sunt asumate de către compilatorul VHDL: este rolul său de „bibliotecar”. Aceste biblioteci nu conţin decât unităţi de proiectare. Un fişier conţinând cod sursă VHDL, odată analizat şi compilat, nu mai există pentru proiectant; numai unităţile de proiectare rezultate sunt plasate în biblioteci. *În VHDL, compilăm fişiere şi utilizăm (referim) unităţi de proiectare.*

2.4.2 Încapsularea în unităţi de proiectare

O descriere hardware este constituită dintr-un ansamblu de modele şi de algoritmi utilizaţi pentru acele modele.

Un model poate fi văzut ca având două părţi:

- *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 şi 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 părţi este o unitate de proiectare şi aceste unităţi alcătuiesc bibliotecile VHDL. O bibliotecă poate conţine, de exemplu, o sută de unităţi de proiectare. Separarea entităţii de arhitectură

apare pentru a permite modificarea uşoară a funcţionării interne fără a mai fi necesară redefinirea viziunii externe a modelului, lucru permis de compilarea separată.

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

Algoritmii utilizaţi frecvent pot fi separaţi, obţinându-se astfel *sub-programe* care sunt adeseori grupate în *pachete*. Aceste pachete (*packages*) sunt supuse aceleiaşi împărţiri ca şi modelele: vederea lor externă, adică perspectiva tuturor posibilităţilor pe care le oferă (pe care le „exportă"), este decuplată de vederea internă care conţine algoritmica ce realizează toate aceste funcţionalităţi. În terminologie VHDL, vederea externă se numeşte *specificaţia pachetului*, pe când vederea internă se numeşte *corpul pachetului*. Pachetele nu se limitează doar la a exporta sub-programe, ci pot oferi şi alte obiecte, cum ar fi semnale globale, tipuri şi componente.

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

Pentru a nu bloca relaţia instanţă / model, modul de asociere al componentelor şi modelelor va forma o unitate de proiectare de sine stătătoare, care va fi şi ea stocată într-o bibliotecă. Această unitate va efectua corespondenţa dintre instanţă şi model (cuplul entitate / arhitectură al cărui „reprezentant" este). Ea este o *configuraţie* VHDL, ultima categorie de unitate de proiectare care poate fi găsită într-o bibliotecă VHDL.

2.4.3 Separarea în domeniul concurent şi domeniul secvenţial

Un sistem hardware este în mod natural concurent. Nu trebuie însă trasă concluzia pripită că descrierile secvenţiale ar fi inutile: de exemplu, un protocol de citire din memorie se exprimă cel mai firesc în mod secvenţial: poziţionăm semnalul X, apoi semnalul Y, aşteptăm un interval de timp Δt etc.

În VHDL, domeniile concurent şi secvenţial 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 instrucţiuni.

2.4.4 Clasificarea după tipuri

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

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

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

- tipurile scalare (întregi, flotanţi, tipuri fizice şi tipuri enumerate);
- tipurile compuse (tablouri şi articole);
- tipurile acces sau poantorii (*pointers*);
- tipurile fişier.

2.4.5 Clasele de obiecte

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

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

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

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

2.4.6 Reguli de scriere în VHDL

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

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

VHDL nu face deosebirea dintre literele mari şi 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 acelaşi 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 sfârşitul unui nume, nici să apară de două ori consecutiv;
- Numele nu are voie să fie un cuvânt VHDL rezervat;
- Lungimea oricărui nume nu poate depăşi o linie. Această toleranţă 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, atribute etc. Notaţia lor diferă în funcţie de tipul obiectelor cărora li se aplică:

- **numere întregi zecimale**: este cazul cel mai simplu, folosindu-se notaţia zecimală obişnuită, fără simboluri suplimentare. De exemplu, **22** sau **1971**. Există posibilitatea includerii caracterului „_” pentru ameliorarea lizibilităţii, fără nici o altă consecinţă: **1_999_234** este identic cu **1999234**.

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

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

```
"Totul" &
"este" &
"minunat";
```

Operaţia de concatenare în sine nu adaugă nici un spaţiu - va trebui deci urmărită cu atenţie prezenţa spaţiilor în şirurile care urmează să fie concatenate.

- **biţi**: se utilizează notaţia caracterelor. Biţii pot avea ca valori numai '0' şi '1'. Tipul STD_LOGIC utilizează în plus valorile 'U', 'X', 'H', 'L', 'W', 'Z' şi '-'.

- **vectori de biţi**: utilizează notaţia şirurilor de caractere, constituite din simbolurile 0 şi 1: de exemplu, "10011010". Sistemul de numeraţie implicit este cel binar, dar este posibilă utilizarea notaţiei octale sau hexazecimale prin prefixarea şirului: O"127", X"01AC".

Spaţiile nu sunt semnificative, cu excepţia apariţiei lor în valorile literale şi în cazurile de separare a identificatorilor. Ele nu sunt necesare între identificatori şi simboluri. Următoarele două linii sunt corecte şi echivalente:

```
A <= B and C;
A<= B and   C     ;
```

O instrucţiune se poate întinde pe mai multe linii, atâta timp cât sfârşitul 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 şi:

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

Sfârşitul unei instrucţiuni este semnalat de prezenţa 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 porneşte, la nivelul rădăcinii, cu sistemul complet văzut ca o cutie neagră, cu intrările şi ieşirile sale. Această cutie neagră este descompusă în componente interconectate, care sunt la rândul lor văzute ca nişte 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 funcţii.

Frunzele acestei ierarhii corespund componentelor elementare descrise în manieră algoritmică secvenţială. 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 şi de conectivitatea acestor componente între ele. Oprirea descompunerii ierarhice corespunde obţinerii unui nivel de complexitate suficient de redus, care să permită trecerea la o descriere algoritmică a funcţionalităţii componentelor de la nivelul cel mai de jos, care poate fi ea însăşi descompusă în mod ierarhic în manieră software, sub formă de apeluri imbricate de sub-programe. Conţinutul unei componente descrie ieşirile în funcţie de intrările şi de stările sale interne.

Limita descompunerii ierarhice poate depinde de numărul de intrări şi de ieşiri ale componentelor terminale, precum şi de dimensiunea şi deci de complexitatea codului fiecărei componente. Acest criteriu se reduce aşadar

la izolarea corectă a funcţionalităţilor, astfel încât acestea să fie pe cât posibil independente unele faţă 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 şi cea de prelucrare a datelor, numită şi cale de date (*data path*).

Felul în care se va face în cele din urmă descompunerea depinde la modul esenţial de experienţa proiectantului şi de familiarizarea sa cu problema de rezolvat.

3. Desfăşurarea lucrării

3.1 Se va lansa în execuţie mediul de dezvoltare Active-HDL. Se consultă documentaţia *on-line* referitoare la utilizarea editoarelor, a simulatorului şi a compilatorului de fişiere VHDL.

3.2 Se vor efectua operaţiile necesare pentru descrierea şi compilarea porţilor logice fundamentale ŞI, SAU, NU, SAU-NU, ŞI-NU, SAU-EXCLUSIV, COINCIDENŢĂ.

3.3 Se vor efectua operaţiile necesare pentru simularea porţilor logice fundamentale de la punctul anterior.