LUCRAREA NR. 12
PACHETE STANDARD ŞI PREDEFINITE

1. Scopul lucrării

Lucrarea prezintă rolul pachetelor predefinite şi standardizate în cadrul proiectării sistemelor hardware în VHDL. Se studiază pachetele STANDARD, TEXTIO şi STD_LOGIC_1164, după care sunt trecute în revistă pachetele aritmetice şi logice ale principalelor firme producătoare de instrumente integrate de proiectare cu VHDL.

2. Consideraţii teoretice

2.1 Generalităţi

Pentru a se evita complicarea inutilă a limbajului VHDL şi pentru a i se păstra supleţea, există un anumit număr de pachete de importanţă majoră care au fost normalizate. Acestea sunt pachetele *standard*.

Pachetele standard nu pot fi modificate de către proiectant. Două dintre aceste pachete (STANDARD şi TEXTIO) sunt definite în manualul de referinţă al VHDL.

Pachetul STD_LOGIC_1164 nu face parte din mediul standard al VHDL definit în manualul de referinţă. El constituie o normă IEEE de sine stătătoare; a fost normalizat pentru a facilita portabilitatea fişierelor care conţin cod sursă VHDL sintetizabil. Acesta este primul pachet care urmăreşte acoperirea lipsurilor existente la nivelul semanticii VHDL pentru sinteză. Acţiunea sa este completată de alte pachete, împreună cu care alcătuieşte un mediu normalizat pentru sinteză: NUMERIC_STD şi NUMERIC_BIT (care conţin definirea unor funcţii aritmetice pe tipul BIT finit) sau pachetele VITAL (prin intermediul cărora devine posibilă luarea în considerare a unor valori temporale precise). Cel mai utilizat pachet numeric este, la ora actuală, pachetul STD_LOGIC_ARITH realizat de firma SYNOPSYS, acest pachet devenind un standard *de facto*, întrucât este

propus şi vândut de către actualul lider al pieţei mondiale în materie de instrumente de dezvoltare bazate pe VHDL.

2.2 Pachetul STANDARD

Pachetul STANDARD reuneşte declaraţii de tipuri, de sub-tipuri şi de funcţii extrem de utile (chiar indispensabile) pentru proiectare. De exemplu, acest pachet conţine definiţia tipului BIT.

La începutul fiecărei unităţi de concepţie există o clauză **use** implicită (pe care proiectantul nu trebuie neapărat s-o scrie). Aşadar, acest pachet este accesibil de oriunde (din orice unitate de proiectare) în VHDL.

Alcătuirea sa este următoarea:

- definiţia tipului BOOLEAN;
- definiţia tipului BIT;
- definiţia tipului CHARACTER;
- definiţia tipului SEVERITY_LEVEL (pentru instrucţiunile **assert**);
- definiţia tipului INTEGER (capetele intervalului de definiţie depind de implementare);
- definiţia tipului REAL (capetele intervalului de definiţie depind de implementare);
- definiţia tipului fizic TIME (timpul de simulare);
- specificaţia funcţiei NOW care returnează timpul curent al simulării (de tip TIME);
- definiţia sub-tipului întreg NATURAL (cu valori cuprinse între 0 şi limita superioară a intervalului de definiţie a tipului INTEGER);
- definiţia sub-tipului întreg POSITIVE (cu valori cuprinse între 1 şi limita superioară a intervalului de definiţie a tipului INTEGER);
- definiţia tipului STRING ca vector neconstrâns de caractere (de tipul CHARACTER);
- definiţia tipului BIT_VECTOR ca vector neconstrâns de biţi.

Specificaţia pachetului STANDARD este dată în continuare:

```
package STANDARD is
-- Tipuri enumerate predefinite
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
type CHARACTER is
       (NUL,  SOH,   STX,   ETX,   EOT,   ENQ,   ACK,   BEL,
        BS,   HT,    LF,    VT,    FF,    CR,    SO,    SI,
        DLE,  DC1,   DC2,   DC3,   DC4,   NAK,   SYN,   ETB,
        CAN,  EM,    SUB,   ESC,   FSP,   GSP,   RSP,   USP,
        ' ',  '!',   '"',   '#',   '$',   '%',   '&',   ''',
        '(',  ')',   '*',   '+',   ',',   '-',   '.',   '/',
        '0',  '1',   '2',   '3',   '4',   '5',   '6',   '7',
        '8',  '9',   ':',   ';',   '<',   '=',   '>',   '?',
        '@',  'A',   'B',   'C',   'D',   'E',   'F',   'G',
        'H',  'I',   'J',   'K',   'L',   'M',   'N',   'O',
        'P',  'Q',   'R',   'S',   'T',   'U',   'V',   'W',
        'X',  'Y',   'Z',   '[',   '\',   ']',   '^',   '_',
        '`',  'a',   'b',   'c',   'd',   'e',   'f',   'g',
        'h',  'i',   'j',   'k',   'l',   'm',   'n',   'o',
        'p',  'q',   'r',   's',   't',   'u',   'v',   'w',
        'x',  'y',   'z',   '{',   '|',   '}',   '~',   'DEL');
type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
-- Tipuri numerice predefinite
type INTEGER is range -2_147_483_648 to -2_147_483_647;
type REAL is range -16#0.7FFFFF8#E+32 to 16#0.7FFFFF8#E+32;
-- Tipul TIME predefinit
type TIME is range -9_223_372_036_854_775_808
       to 9_223_372_036_854_775_807; -- pentru implementarea
                                     -- pe 64 de biţi
      units fs;                 -- femtosecundă
            ps = 1000 fs;       -- picosecundă
            ns = 1000 ps;       -- nanosecundă
            us = 1000 ns;       -- microsecundă
            ms = 1000 us;       -- milisecundă
            sec = 1000 ms;      -- secundă
            min = 60 sec;       -- minut
            hr = 60 min; -- oră
      end units;
-- Funcţia care returnează timpul curent al simulării
function NOW return TIME;
-- Sub-tipuri numerice predefinite
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
-- Tipuri tablou predefinite
type STRING is array (POSITIVE range <>) of CHARACTER;
type BIT_VECTOR is array (NATURAL range <>) of BIT;
end STANDARD;
```

2.3 Pachetul TEXTIO

2.3.1 Prezentare generală

Acest pachet este, din păcate, unul din punctele slabe ale VHDL. Rolul său este de a furniza primitivele de intrare / ieşire ASCII ale VHDL. Aşadar, el joacă un rol foarte important şi este utilizat atât pentru interacţiunea utilizator-consolă (de exemplu, introducerea de parametri) cât şi pentru scrierea şi citirea fişierelor (de exemplu, citirea conţinutului unui modul de memorie ROM).

Felul în care este scris aminteşte de intrările şi ieşirile din limbajul PASCAL; însă aspectul cel mai grav îl constituie faptul că *specificaţia pachetului TEXTIO nu respectă regulile generale VHDL*. Orice tentativă de a compila acest pachet este sortită eşecului (de exemplu, în funcţia ENDLINE se găseşte un parametru de tip acces, ceea ce este interzis în VHDL, după cum am arătat în cadrul lucrării nr. 10 dedicată sub-programelor).

Constructorii de instrumente VHDL au încercat să remedieze această lipsă de omogenitate, realizând fiecare propriul pachet TEXTIO. A rezultat o lipsă evidentă de portabilitate.

În cazul acestui pachet, clauza **use** nu mai este implicită (la fel ca în cazul pachetului STANDARD), deci pachetul TEXTIO va trebui referit în mod explicit dacă se intenţionează realizarea de operaţii de intrare / ieşire.

Alcătuirea acestui pachet este următoarea:

- Definiţia tipului LINE. Acest tip va reprezenta linia de fişier manipulată. El este definit ca un acces la tipul STRING, adică la un vector neconstrâns de caractere;
- Definiţia tipului TEXT. Este tipul fişierelor de şiruri de caractere care vor fi manipulate de către acest pachet;
- Definiţia tipului SIDE. Mai puţin important, acest tip enumerat permite scrierea aliniată la stânga sau la dreapta;
- Definiţia tipului WIDTH. Este un tip de date utilizat pentru lungimea şirurilor de caractere; este de fapt o redenumire a tipului NATURAL;
- Definiţia fişierelor de text standard. În funcţie de implementare, aici se vor găsi corespondenţele dintre fişierele INPUT şi OUTPUT şi respectiv intrarea de la tastatură şi ieşirea la consolă

a sistemului. Operaţiile de intrare / ieşire efectuate asupra fişierelor INPUT şi OUTPUT constituie aşa-numitul „dialog consolă";

- Definiţia primitivelor de citire. Primitiva READLINE efectuează citirea unei linii întregi dintr-un fişier dat. Această linie este returnată ca parametru de ieşire al acestei proceduri. Pe această linie vor opera procedurile READ;
- Cele 16 proceduri de citire (READ) primesc ca intrare linia dată, extrag din ea (dacă este posibil) valoarea de tipul cerut şi returnează restul liniei, valoarea citită şi, în cazul a 8 dintre ele, o variabilă booleană care arată dacă operaţia a fost efectuată cu succes;
- Cele 8 tipuri de date care pot constitui obiectul unei operaţii de citire sunt: BIT, BIT_VECTOR, BOOLEAN, CHARACTER, INTEGER, REAL, STRING şi TIME;
- Definiţia primitivelor de scriere. Procedura WRITELINE efectuează scrierea într-un fişier a unei linii date ca parametru de intrare. Pentru utilizarea acestei primitive, trebuie ca în prealabil această linie să fie construită prin primitive de citire sau prin mai multe apeluri de proceduri WRITE. Există 8 proceduri WRITE, fiecare dintre ele efectuând scrierea într-o linie a unuia dintre cele 8 tipuri menţionate mai sus. Această linie este transmisă în mod **inout**; prin urmare, mai multe proceduri WRITE pot completa o aceeaşi linie înainte de scrierea sa cu ajutorul procedurii WRITELINE;
- Definiţia primitivelor de sfârşit de linie şi de sfârşit de fişier. O funcţie ENDLINE primeşte ca intrare o linie şi returnează valoarea booleană TRUE dacă s-a ajuns la sfârşitul liniei. În anumite implementări, această funcţie este transformată în procedură pentru a i se permite să se încadreze în sintaxa şi semantica generală VHDL (parametrul LINE al acestei funcţii este de tip acces, ceea ce este interzis în VHDL);
- Funcţia ENDFILE este definită pe toate tipurile fişier şi deci şi pe tipul TEXT. Ea returnează valoarea booleană TRUE dacă s-a ajuns la sfârşitul fişierului.

În VHDL'87 nu existau primitive de creare, de deschidere sau de închidere a fişierelor. Deschiderea fişierului (sau crearea lui, dacă încă nu

exista) era efectuată la declararea tipului fişier. Închiderile se efectuau implicit ori de câte ori era necesar: la fiecare ieşire din blocul în care variabila era vizibilă, fişierul era închis. O dată cu apariţia VHDL'93, aceste constrângeri au fost anulate: se presupune că utilizatorul este suficient de matur pentru a-şi gestiona singur acest nou tip de comunicare. Remarcăm deci apariţia unei noi modalităţi de comunicare între procese: *prin fişiere*, nu doar prin semnale.

Observaţie

Rolul acestor constrângeri în VHDL'87 era acela de a evita comunicaţia între procese prin intermediul fişierelor. Asemenea manipulări ar fi generat un cod VHDL dependent de implementare, deci ne-portabil. În VHDL'87 era imposibil să scriem într-un fişier din interiorul unui proces, să-l închidem şi apoi să-l redeschidem din interiorul altui proces.

Începând cu apariţia VHDL'93, aceste lucruri devin posibile şi prin urmare au fost introduse 2 noi primitive: FILE_OPEN şi FILE_CLOSE, care sunt declarate în mod automat cu tipul TEXT. Alte îmbunătăţiri aduse de VHDL'93 sunt: trecerea parametrilor procedurilor READLINE şi WRITELINE în mod **inout** şi declararea explicită a funcţiei ENDFILE ca funcţie impură.

2.3.2 Specificaţia pachetului TEXTIO

```
package TEXTIO is
-- Câteva definiţii de tipuri utile
type LINE is access STRING;
type TEXT is file of STRING;
type SIDE is (RIGHT, LEFT);
subtype WIDTH is NATURAL;

-- Definiţia fişierelor de text standard („dialog consolă”)
file INPUT: TEXT is in "STD_INPUT";
file OUTPUT: TEXT is out "STD_OUTPUT";

-- Primitive de citire a tipurilor standard
procedure READLINE (variable F: in TEXT; L: out LINE);

procedure READ(L: inout LINE; VALUE: out BIT; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out BIT);

procedure READ(L: inout LINE; VALUE: out BIT_VECTOR; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out BIT_VECTOR);

procedure READ(L: inout LINE; VALUE: out BOOLEAN; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out BOOLEAN);
```

```
procedure READ(L: inout LINE; VALUE: out CHARACTER; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out CHARACTER);

procedure READ(L: inout LINE; VALUE: out INTEGER; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out INTEGER);

procedure READ(L: inout LINE; VALUE: out REAL; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out REAL);

procedure READ(L: inout LINE; VALUE: out STRING; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out STRING);

procedure READ(L: inout LINE; VALUE: out TIME; GOOD: out BOOLEAN);
procedure READ(L: inout LINE; VALUE: out TIME);

-- Primitive de scriere a tipurilor standard
procedure WRITELINE (F: out TEXT; L: in LINE);
procedure WRITE(L: inout LINE; VALUE: in BIT; JUSTIFIED: in SIDE :=
           RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in BIT_VECTOR; JUSTIFIED: in
           SIDE := RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in BOOLEAN; JUSTIFIED: in
           SIDE := RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in CHARACTER; JUSTIFIED: in
           SIDE := RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in INTEGER; JUSTIFIED: in
           SIDE := RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in REAL; JUSTIFIED: in SIDE :=
           RIGHT; FIELD: in WIDTH := 0; DIGITS: in NATURAL := 0);
procedure WRITE(L: inout LINE; VALUE: in STRING; JUSTIFIED: in SIDE :=
           RIGHT; FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in TIME; JUSTIFIED: in SIDE :=
           RIGHT; FIELD: in WIDTH := 0; UNIT: in TIME := NS);

-- Primitive de sfârşit de linie şi de sfârşit de fişier
function ENDLINE(L: in LINE) return BOOLEAN;
function ENDFILE(F: in TEXT) return BOOLEAN; --Opţională (nu apare în
                                              --toate implementările)
end TEXTIO;
```

2.3.3 Exemple de utilizare

Prima problemă cu care se confruntă proiectantul este, de regulă, următoarea: linia de cod de mai jos este sistematic respinsă de compilator, deoarece notaţia "" desemnează atât valori, cât şi tipuri BIT_VECTOR şi şiruri de caractere.

```
WRITE (linie, "şir de caractere");
```

Această ambiguitate nu poate fi rezolvată decât de către compilator; prin urmare, proiectantul este cel care trebuie să utilizeze expresii calificate, ca în exemplul de mai jos:

```
WRITE (linie, STRING'("şir de caractere"));
```

Iată în continuare un mic exemplu de pachet care utilizează pachetul TEXTIO. Singura procedură din componenţa sa efectuează achiziţia de la consolă a două valori: una de tip TIME şi alta de tip BIT. Acestea sunt returnate apelantului, însă numai după ce se trasează achiziţia lor prin scrierea într-un fişier. Pentru simplificare, nu se va face nici o verificare la achiziţie: presupunem că operatorul va furniza întotdeauna valoarea adecvată la momentul potrivit.

```
package EXEMPLU is
procedure ACHIZIŢIE-TRAS(DATĂ: out TIME; VALOARE: out BIT);
end EXEMPLU;

use STD.TEXTIO.all; --Referire la pachetul TEXTIO

package body EXEMPLU is
file FIŞIER: TEXT is out "TRACE.VHDL"; -- Fişierul de trasare
procedure ACHIZIŢIE-TRAS(DATĂ: out TIME; VALOARE: out BIT) is
--Variabile de lucru
variable LINIE: LINE;
variable DATA: TIME;
variable VALOAREA: BIT;
begin
--Achiziţia unei perechi dată / valoare
READLINE(INPUT, LINIE);
READ(LINIE, DATA);
READ(LINIE, VALOAREA);
--Resetare la valoarea null a variabilei de lucru de tip LINE
LINIE:= null;
--Scriere în fişierul de trasare
WRITE(LINIE, STRING'("Perechea dată / valoare introdusă: "));
WRITE(LINIE, DATA);
WRITE(LINIE, STRING'(" "));
WRITE(LINIE, VALOAREA);
WRITELINE(FIŞIER, LINIE);
--Actualizarea parametrilor de retur
DATĂ:= DATA;
VALOARE:= VALOAREA;
end ACHIZIŢIE-TRAS;
end EXEMPLU;
```

La citire, linia este constituită din două elemente: o valoare de tip TIME şi o valoare de tip BIT. Utilizatorul va introduce deci, de exemplu:

```
10 ns  1
```

şi apoi această linie, citită de procedura READLINE, va fi descompusă (prin intermediul celor două proceduri READ) într-o dată (10 ns) şi o valoare de bit ('1').

La scriere, linia este alcătuită din patru elemente. Cu ocazia celor patru apeluri ale procedurii WRITE, se scrie succesiv: un şir de caractere (`"Perechea dată / valoare introdusă: "`), o dată de tip TIME (aici, `10 ns`), un şir de caractere care conţine două spaţii (" ") şi o valoare de tip BIT (aici, '1'). O dată construită, această linie este scrisă în fişierul FIŞIER cunoscut în sistemul de operare sub numele de TRACE.VHDL, cu ajutorul procedurii WRITELINE.

Observaţii

Două apeluri succesive ale procedurii ACHIZIŢIE_TRAS provoacă scrierea celei de-a doua trasări în continuarea celei dintâi, în fişierul FIŞIER. Dacă declaraţia acestui fişier s-ar fi efectuat în partea declarativă a procedurii (şi nu, ca în exemplul de mai sus, în zona declarativă a pachetului), atunci fişierul ar fi fost redeschis la fiecare apel şi nu s-ar fi păstrat decât ultima trasare!

Pentru a scrie o procedură de achiziţie mai realistă, ar fi preferabil să utilizăm procedurile READ care returnează o valoare booleană de control (ultimul parametru). Testarea acestui parametru ar permite detectarea erorilor (introducerea unei valori de alt tip decât cel aşteptat) şi tratarea lor (de exemplu, prin declanşarea unei noi achiziţii).

2.4 Pachetul STD_LOGIC_1164

2.4.1 Interpretarea tipului logic: semantica pentru simulare

Acest pachet defineşte logica cu nouă stări; el trebuie să facă parte din biblioteca IEEE. Aşadar, toate unităţile de compilare care fac referire la acest pachet trebuie să înceapă cu următoarele clauze **library** şi **use**:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Ansamblul pachetului este bazat pe definiţia tipului enumerat numit STD_ULOGIC (sau STD_LOGIC, atunci când este rezolvat). Cele nouă stări au fost deja prezentate în cadrul lucrării nr. 8 referitoare la semnale multi-sursă; ele sunt următoarele:

```
type STD_ULOGIC is (     'U',  --Neiniţializat
                         'X',  --Necunoscut tare
                         '0',  --0 tare
                         '1',  --1 tare
                         'Z',  --Înaltă impedanţă
                         'W',  --Necunoscut slab
                         'L',  --0 slab
                         'H',  --1 slab
                         '-',  --Fără importanţă
                    );
```

Valoarea 'U' fiind prima din tipul enumerat STD_ULOGIC, ea va fi valoarea luată implicit pentru toate obiectele declarate de acest tip ('U' vine de la „*Uninitialized*"). Semantica sa este simplă: dacă această valoare apare în timpul unei simulări, înseamnă că obiectul respectiv nu a fost iniţializat sau cu alte cuvinte nu i s-a asignat nici o valoare.

Semantica valorii 'X' este asemănătoare: ea arată că este imposibil să se definească o valoare '0' sau '1' sau 'Z' (de exemplu ca urmare a unui conflict între mai multe valori asignate unui semnal multi-sursă). Singura informaţie disponibilă este că obiectul a fost iniţializat.

Valorile '0', '1' şi 'X' dispun de valori asociate numite "slabe": respectiv, 'L', 'H' şi 'W'. Scopul acestei duplicări este acela de a lua în considerare niveluri electrice mai fine decât cele necesare reprezentării lui '0' şi '1' logic. De exemplu, o magistrală cu o rezistenţă „*pull-up*"[1] va avea, în starea de repaus, valoarea 'H'. Singura semantică oficială este de fapt cea rezumată în tabelul de mai jos (care este de fapt, în acelaşi timp, şi funcţia

[1] Se spune despre un semnal care are o rezistenţă legată la borna de alimentare de potenţial pozitiv. În consecinţă, în absenţa oricărei comenzi (de exemplu, în cazul unei intrări neconectate), semnalul ia valoarea logică '1'.

de rezoluţie a semnalelor de acest tip STD_LOGIC, deja prezentată în cadrul lucrării nr. 8 referitoare la domeniul concurent în VHDL).

```
------------------------------------------------------------
| U    X     0     1     Z     W     L     H     -       |   |
------------------------------------------------------------
('U', 'U',  'U',  'U',  'U',  'U',  'U',  'U',  'U'),-- | U |
('U', 'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X'),-- | X |
('U', 'X',  '0',  'X',  '0',  '0',  '0',  '0',  'X'),-- | 0 |
('U', 'X',  'X',  '1',  '1',  '1',  '1',  '1',  'X'),-- | 1 |
('U', 'X',  '0',  '1',  'Z',  'W',  'L',  'H',  'X'),-- | Z |
('U', 'X',  '0',  '1',  'W',  'W',  'W',  'W',  'X'),-- | W |
('U', 'X',  '0',  '1',  'L',  'W',  'L',  'W',  'X'),-- | L |
('U', 'X',  '0',  '1',  'H',  'W',  'W',  'H',  'X'),-- | H |
('U', 'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X'),-- | - |
```

Observaţii

'X', 'W' şi 'U' se numesc *valori meta-logice* şi natura lor este diferită de cea a celorlalte semnale. Proiectantul trebuie să manifeste prudenţă în cazul utilizării lor. De exemplu, rezultatul unei simple comparaţii poate fi eronat: expresia 'U' < '0' este întotdeauna adevărată (conform semanticii de simulare), datorită definiţiei tipului enumerat, însă această expresie nu are un sens real.

Valoarea '-' („fără importanţă") nu are nici o semnificaţie pentru simulare şi ea nu trebuie să apară niciodată în urma rezultatelor unei simulări. Utilizarea sa este rezervată sintezei: ea le oferă instrumentelor de sinteză posibilitatea de a codifica această valoare în '0' sau '1', în funcţie de optimizările urmărite.

2.4.2 Descrierea pachetului STD_LOGIC_1164

Acest pachet exportă tipul enumerat cu nouă stări, precum şi vectorii şi sub-tipurile derivate, însoţite de o funcţie de rezoluţie. Operatorii logici sunt supraîncărcaţi şi se exportă de asemenea şi funcţii de conversie de tip. Declaraţia acestui pachet este următoarea:

```
package STD_LOGIC_1164 is

--Declaraţia tipului enumerat, fundamentul logicii cu nouă stări
type STD_ULOGIC is (  'U',    --Neiniţializat
                      'X',    --Necunoscut tare
                      '0',    --0 tare
                      '1',    --1 tare
                      'Z',    --Înaltă impedanţă
                      'W',    --Necunoscut slab
                      'L',    --0 slab
                      'H',    --1 slab
                      '-',    --Fără importanţă
                   );

--Tablou neconstrâns de STD_ULOGIC utilizat în funcţia de rezoluţie
type STD_ULOGIC_VECTOR is array (NATURAL range <>) of STD_ULOGIC;

-- Funcţia de rezoluţie: comportamentul acestei funcţii nu face parte
-- din specificaţia pachetului, ea este însă normalizată
function RESOLVED(S: STD_ULOGIC_VECTOR) return STD_ULOGIC;

-- Tipul rezolvat STD_LOGIC este cel care apare cel mai adesea în
-- definiţiile modelelor industriale. Din raţiuni de simplitate, se
-- practică o utilizare abuzivă a tipului rezolvat chiar şi pentru
-- un semnal uni-sursă.
subtype STD_LOGIC is RESOLVED STD_ULOGIC;

-- Tablou neconstrâns de STD_LOGIC pentru declaraţii de tablouri
-- de semnale rezolvate
type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC;

--Alte definiţii de sub-tipuri
subtype X01 is RESOLVED STD_ULOGIC range 'X' to '1';  --'X','0','1'
subtype X01Z is RESOLVED STD_ULOGIC range 'X' to 'Z'; --'X','0','1','Z'
subtype UX01 is RESOLVED STD_ULOGIC range 'U' to '1'; --'U', 'X', '0', '1'
subtype UX01Z is RESOLVED STD_ULOGIC range 'U' to 'Z';--'U','X','0','1','Z'

--Supraîncărcarea operatorilor
function "and" (L: STD_ULOGIC; R: STD_ULOGIC) return UX01;
function "nand"(L: STD_ULOGIC; R: STD_ULOGIC) return UX01;
function "or"  (L: STD_ULOGIC; R: STD_ULOGIC) return UX01;
function "nor" (L: STD_ULOGIC; R: STD_ULOGIC) return UX01;
function 'xor" (L: STD_ULOGIC; R: STD_ULOGIC) return UX01;
function 'not" (L: STD_ULOGIC) return UX01;

--Supraîncărcarea operatorilor pe vectori
function "and"  (L, R: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "and"  (L, R: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function "nand" (L, R: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "nand" (L, R: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function "or"   (L, R: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "or"   (L, R: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function "nor"  (L, R: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "nor"  (L, R: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function "xor"  (L, R: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "xor"  (L, R: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function "not"  (L: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
```

```
function "not"  (L: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;

--Funcţii de conversie
function TO_BIT (S: STD_ULOGIC; XMAP: BIT:='0') return BIT;
function TO_BITVECTOR (S: STD_LOGIC_VECTOR; XMAP: BIT:='0') return
                       BIT_VECTOR;
function TO_BITVECTOR (S: STD_ULOGIC_VECTOR; XMAP: BIT:='0') return
                       BIT_VECTOR;
function TO_STDULOGIC (B: BIT) return STD_ULOGIC;
function TO_STDLOGICVECTOR (B: BIT_VECTOR) return STD_LOGIC_VECTOR;
function TO_STDLOGICVECTOR (S: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
function TO_STDULOGICVECTOR (B: BIT_VECTOR) return STD_ULOGIC_VECTOR;
function TO_STDULOGICVECTOR (S: STD_LOGIC_VECTOR) return STD_ULOGIC_VECTOR;

--Transformarea codificării şi conversii de tip
function TO_X01 (S: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
function TO_X01 (S: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01 (S: STD_ULOGIC) return X01;
function TO_X01 (B: BIT_VECTOR) return STD_LOGIC_VECTOR;
function TO_X01 (B: BIT_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01 (B: BIT) return X01;
function TO_X01Z (S: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
function TO_X01Z (S: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01Z (S: STD_ULOGIC) return X01Z;
function TO_X01Z (B: BIT_VECTOR) return STD_LOGIC_VECTOR;
function TO_X01Z (B: BIT_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01Z (B: BIT) return X01Z;
function TO_UX01 (S: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
function TO_UX01 (S: STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_UX01 (S: STD_ULOGIC) return UX01;
function TO_UX01 (B: BIT_VECTOR) return STD_LOGIC_VECTOR;
function TO_UX01 (B: BIT_VECTOR) return STD_ULOGIC_VECTOR;
function TO_UX01 (B: BIT) return UX01;

--Detectarea fronturilor
function RISING_EDGE (signal S: STD_ULOGIC) return BOOLEAN;
function FALLING_EDGE (signal S: STD_ULOGIC) return BOOLEAN;

--Funcţie de testare a obiectelor conţinând o valoare necunoscută 'X'
function IS_X (S: STD_ULOGIC_VECTOR) return BOOLEAN;
function IS_X (S: STD_LOGIC_VECTOR) return BOOLEAN;
function IS_X (S: STD_ULOGIC) return BOOLEAN;
end STD_LOGIC_1164;
```

2.4.3 Interpretarea tipului logic: semantica pentru sinteză

Scopul grupului de normalizare însărcinat cu definirea acestui pachet era de a defini o semantică precisă pentru fiecare valoare a tipului enumerat STD_ULOGIC şi de a propune o interpretare unică şi simplă în vederea utilizării sale.

Valorile ‘0’ şi ‘1’ trebuie interpretate de către instrumentele de sinteză, respectiv, „conectat la masă” şi „conectat la alimentare”. Nu există semantică definită pentru valorile ‘L’ şi ‘H’: instrumentele de sinteză sunt

libere să le interpreteze cum doresc. De exemplu, este posibil ca ‘L’ să fie interpretat drept ‘0’ şi ca ‘H’ să fie interpretat ca ‘1’.

Observaţie

Luarea în considerare a valorilor implicite sau a valorilor iniţiale constituie o problemă foarte serioasă care apare la sinteză. Unii proiectanţi le utilizează în modelele lor ca nişte valori care apar ca urmare a unui semnal de *reset*. Cu toate acestea, pentru aceste valori, nu există nici o interpretare standard unanim recunoscută. Proiectanţii trebuie să fie conştienţi că un model scris astfel *nu este portabil* de la un instrument de sinteză la altul. Dacă funcţionalitatea modelului depinde de valorile iniţiale, iniţializarea acestora trebuie să fie făcută în mod explicit ca urmare a apariţiei unui eveniment.

Pentru celelalte valori ale tipului STD_ULOGIC, ‘Z’, ‘U’, ‘W’ şi ‘X’ (valoarea ‘-‘ va fi studiată separat), semnificaţiile respective depind de utilizarea acestor valori. Se pot distinge patru cazuri:

1. Valoarea este utilizată ca valoare implicită sau ca valoare iniţială. În acest caz nu este definită nici o semantică (a se vedea observaţia de mai sus);
2. Valoarea este asignată unei variabile sau unui semnal. În acest caz, valorile ‘U’, ‘W’ şi ‘X’ vor fi considerate „fără importanţă”. Valoarea „fără importanţă” (‘-‘) poate fi înlocuită la sinteză fie cu un ‘0’ fie cu un ‘1’. Valoarea ‘Z’ va solicita instrumentului de sinteză generarea unui amplificator cu trei stări (*tri-state*). Ieşirea acestui amplificator va constitui ţinta instrucţiunii de asignare;
3. Valoarea este utilizată într-o comparaţie implicită prin intermediul instrucţiunii **case**. Instrucţiunile care apar atunci sunt ignorate (aici se vede foarte clar diferenţa de semantică dintre sinteză şi simulare);
4. Valoarea este utilizată de către un operator de comparare ca expresie a unei condiţii în cadrul unei instrucţiuni **if**. Expresiile A = B şi A /= B returnează FALSE, respectiv TRUE dacă unul dintre operanzi utilizează una dintre valorile: ‘U’, ‘W’ sau ‘X’. Comparaţiile realizate cu ajutorul operatorilor „>”, „>=”, „<”, „<=” returnează, toate, FALSE.

Valoarea „fără importanţă” este foarte utilă şi este foarte utilizată în cadrul sintezei. Ea permite două funcţionalităţi:

1. Atunci când valoarea se află în membrul drept al unei instrucţiuni de asignare, este posibilă optimizarea.

```
S <= '-';
V := '-';
ARRAY_S <= "----";
```

   Instrumentul de sinteză este liber să înlocuiască fiecare valoare ‘-‘ fie cu un ‘1’, fie cu un ‘0’. Scopul acestei flexibilităţi este de a oferi posibilitatea de a optimiza hardware-ul generat.
2. Când valoarea apare în contextul unei comparaţii, ea este luată drept valoare generică (*wild card*). Acest lucru poate apărea implicit în cadrul unei instrucţiuni **case** sau explicit în cadrul unei instrucţiuni **if** sau în cadrul unei expresii care întrebuinţează un operator de comparare. Valoarea generică permite construirea de filtre. De exemplu, “1101” şi “1--1” se potrivesc (match) deoarece comparaţia este filtrată (comparaţia nu se realizează efectiv decât asupra primului şi ultimului bit). Valorile “1101” şi “10-1” nu se potrivesc din cauza celui de-al doilea bit.

Este uşor de intuit cât de importantă este această posibilitate de construcţie a filtrelor pentru optimizarea resurselor hardware generate în urma procesului de sinteză. Din păcate, ea introduce o importantă diferenţă de semantică între sinteză şi simulare, astfel încât rezultatele obţinute la simularea efectuată înainte şi după sinteză riscă să difere considerabil.

În urma scrierii expresiei “1101” = “1--1”, evaluarea sa va returna valoarea FALSE la simulare, dar în urma evaluării aceleiaşi expresii de către un instrument de sinteză se va obţine valoarea TRUE! Firește, astfel se va obţine un comportament diferit la simularea pre şi post sinteză.

Pentru evitarea unei asemenea situaţii, se recomandă utilizarea unei funcţii specifice numite STD_MATCH, care a fost definită în cadrul unui pachet normalizat IEEE 1076.3 NUMERIC_STD. Principiul acestei funcţii constă în aplicarea unei semantici de valoare generică valorii ‘-‘. În consecinţă, comparaţiile în care va apărea această valoare vor avea semantici coerente la simulare şi la sinteză.

În exemplul de mai jos, rezultatul obţinut în urma sintezei va fi acelaşi şi pentru linia 1 şi pentru linia 2, însă numai linia 2 va fi coerentă la simulare atât înainte cât şi după sinteză:

```
signal A: STD_ULOGIC_VECTOR (3 downto 0);
...
if A /= "10--" then      -- Linia 1
...
if STD_MATCH(A, "10--") then  -- Linia 2
...
```

Se recomandă insistent utilizarea funcţiei STD_MATCH. În cadrul pachetului normalizat NUMERIC_STD au fost definite mai multe supraîncărcări ale acestei funcţii (STD_ULOGIC_VECTOR, STD_ULOGIC, STD_LOGIC, STD_LOGIC_VECTOR, UNSIGNED şi SIGNED), iar valoarea returnată este de tip BOOLEAN.

2.5 Pachete aritmetice

Un pachet aritmetic are întotdeauna următoarea structură:
- definiţia celor două tipuri numerice SIGNED şi UNSIGNED;
- declararea operaţiilor aritmetice, a comparaţiilor, a deplasărilor (*shift*) şi a rotaţiilor (*rotate*);
- declararea unui ansamblu de funcţii de conversie care permit trecerea de la domeniul vectorilor de biţi spre întregi, şi invers.

2.5.1 Declaraţii de tipuri

În cazul pachetului NUMERIC_BIT sunt făcute următoarele declaraţii:

```
type UNSIGNED is array (NATURAL range <>) of BIT;
type SIGNED   is array (NATURAL range <>) of BIT;
```

În cazul pachetului NUMERIC_STD sunt făcute următoarele declaraţii:

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED   is array (NATURAL range <>) of STD_LOGIC;
```

Observaţii

Cele două tipuri SIGNED şi UNSIGNED sunt definite ca tipuri noi şi NU ca sub-tipuri ale lui BIT_VECTOR sau STD_LOGIC_VECTOR. Graţie acestei precauţii, obiectelor de tipul SIGNED sau UNSIGNED nu li se vor putea asigna valori decât cu ajutorul procedurilor sau funcţiilor puse special în acest scop la dispoziţia proiectanţilor. În caz contrar, ar fi fost posibilă amestecarea obiectelor de tip SIGNED sau UNSIGNED, fără a avea un control formal asupra lor.

Se observă, în cazul pachetului NUMERIC_STD, că a fost preferat tipul STD_LOGIC în locul lui STD_ULOGIC, datorită faptului că primul este mai general, fiind rezolvat, şi astfel poate fi aplicat fără constrângeri în orice context. Ideal ar fi fost să se ofere ambele posibilităţi, dar această abordare ar fi impus definirea de multe alte definiţii de tipuri şi de funcţii asociate, ceea ce ar fi complicat foarte mult contextul general de lucru.

2.5.2 Declaraţii de operatori

Toţi operatorii aritmetici şi de comparare definiţi pe întregi sunt supraîncărcaţi pe tipurile SIGNED şi UNSIGNED. Astfel, sunt redefiniţi operatorii „+”, „-“, „*”, „/”, „**rem**”, „**mod**”, „>”, „>=”, „<”, „<=”. Pentru toţi aceşti operatori binari, supraîncărcarea este completă, oferind utilizatorului un maximum de supleţe şi de uşurinţă în exploatare. Spre exemplu, prezentăm supraîncărcarea operatorului binar de adunare:

```
function "+"(L, R: UNSIGNED)              return UNSIGNED;
function "+"(L, R: SIGNED)                return SIGNED;
function "+"(L: UNSIGNED; R: NATURAL)     return UNSIGNED;
function "+"(L: NATURAL; R: UNSIGNED)     return UNSIGNED;
function "+"(L: INTEGER; R: SIGNED)       return SIGNED;
function "+"(L: SIGNED; R: INTEGER)       return SIGNED;
```

Dimensiunea rezultatului este întotdeauna specificată în mod foarte precis. Astfel, în cazul adunării (şi la fel stau lucrurile şi pentru scădere), dimensiunea rezultatului este întotdeauna dimensiunea celui mai mare dintre cei doi vectori primiţi ca parametri sau dimensiunea singurului vector primit ca parametru (în cazul în care celălalt operand este de tipul INTEGER sau NATURAL).

În cazul înmulţirii, dimensiunea rezultatului trebuie să fie egală cu suma dimensiunilor celor doi operanzi atunci când aceştia sunt vectori. Atunci când numai un operand este vector (celălalt fiind un întreg), dimensiunea rezultatului este egală cu dublul dimensiunii operandului vector (se presupune deci că operandul întreg este mai întâi tradus într-un vector de dimensiunea celuilalt operand de natură vectorială).

În ceea ce priveşte împărţirile, dimensiunea rezultatului este egală cu dimensiunea primului operand (deîmpărţitul) atunci când acesta este un vector. În cazul contrar (deîmpărţitul este de tip INTEGER sau NATURAL), dimensiunea rezultatului este luată implicit egală cu dimensiunea împărţitorului (cel de-al doilea operand). În acest ultim caz, conversia deîmpărţitului se face ţinând cont de dimensiunea împărţitorului, la nevoie efectuându-se o trunchiere.

Operaţiile *modulo* (**mod**) şi „*restul împărţirii întregi*” (**rem**) sunt absolut analoage împărţirii în privinţa calculării şi dimensiunii rezultatului (numărul calculat este întotdeauna mai mic decât împărţitorul). Rezultă de aici că dimensiunea rezultatului este egală cu dimensiunea celui de-al doilea operand (împărţitorul) atunci când acesta este un vector. În caz contrar (dacă împărţitorul este de tip INTEGER sau NATURAL), atunci dimensiunea rezultatului este luată implicit drept dimensiunea deîmpărţitului (primul operand). În această situaţie, conversia împărţitorului se efectuează ţinând cont de dimensiunea deîmpărţitului, la nevoie efectuându-se o trunchiere.

Comparaţiile sunt efectuate cu operanzi de dimensiuni eventual diferite (dacă operanzii respectivi sunt vectori).

Deplasările şi rotaţiile sunt definite simultan ca funcţii şi ca operatori. Motivul este evident: operatorii **sll**, **srl**, **rol** şi **ror** nu au apărut decât o dată cu versiunea VHDL'93; deci, pentru păstrarea compatibilităţii cu vechea versiune VHDL'87, au fost declarate şi funcţiile SHIFT_LEFT, SHIFT_RIGHT, ROTATE_LEFT şi ROTATE_RIGHT. Dimensiunea rezultatului trebuie să fie identică cu dimensiunea vectorului asupra căruia se efectuează deplasarea sau rotaţia.

2.5.3 Declaraţiile funcţiilor de conversie

Funcţiile de conversie definite în pachetele aritmetice sunt de două feluri:

1. *Funcţii care permit trecerea de la o reprezentare vectorială la o altă reprezentare*, neschimbându-se decât dimensiunea rezultatului faţă de

dimensiunea argumentului. Pe scurt, aceste funcţii efectuează fie *o extensie a numărului* (dimensiunea rezultatului este mai mare decât dimensiunea argumentului de intrare), fie *o trunchiere* (dimensiunea rezultatului este mai mică decât dimensiunea argumentului de intrare). Modul de traducere al extensiei diferă, în funcţie de argument. Dacă argumentul este de tip SIGNED, atunci bitul de semn (bitul cel mai semnificativ al vectorului argument) este duplicat de atâtea ori cât este necesar pentru a se ajunge la dimensiunea rezultatului dorit. Dacă argumentul este de tip UNSIGNED, atunci se va insera un nou bit de valoare '0' la stânga vectorului argument, de câte ori este necesar, până se ajunge la dimensiunea rezultatului dorit. Există două asemenea funcţii, una pentru transformările elementelor de tip SIGNED, cealaltă pentru transformările elementelor de tip UNSIGNED. Specificaţiile lor sunt:

```
function RESIZE (ARG: SIGNED; NEW_SIZE: NATURAL) return SIGNED;
function RESIZE (ARG: UNSIGNED; NEW_SIZE: NATURAL) return UNSIGNED;
```

2. *Funcţii care permit trecerea de la o reprezentare sub formă de vectori (SIGNED sau UNSIGNED) la o reprezentare sub formă de întregi (INTEGER sau NATURAL)* şi invers. Funcţiile respective sunt:

```
function TO_INTEGER(ARG: UNSIGNED) return NATURAL;
function TO_INTEGER(ARG: SIGNED) return INTEGER;
function TO_UNSIGNED(ARG, SIZE: NATURAL) return UNSIGNED;
function TO_SIGNED(ARG: INTEGER; SIZE: NATURAL) return SIGNED;
```

2.5.4 Declaraţii diverse

Toate funcţiile logice au fost supraîncărcate pe tipurile SIGNED şi UNSIGNED.

Există o funcţie care permite forţarea la valoarea '0' sau '1' a oricărui element al unui vector de tip SIGNED şi UNSIGNED. Această funcţie se numeşte TO_01. Parametrul XMAP, având implicit valoarea '+', permite determinarea valorii finale a elementelor unui vector ale cărui valori nu sunt '0', '1', 'L' sau 'H'.

Mai multe detalii referitoare la pachetele standard şi nestandardizate se găsesc în Anexă.

3. Desfăşurarea lucrării

3.1 Se va testa funcţia NOW din cadrul pachetului STANDARD.

3.2 Se va testa funcţia de rezoluţie a tipului STD_LOGIC prin crearea unui semnal multi-sursă şi luarea în consideraţie a tuturor cazurilor.

3.3 Se vor utiliza funcţiile de conversie predefinite în cadrul pachetelor aritmetice pentru implementarea unui numărător bidirecţional pe 8 biţi.