Limbajul VHDL

În prezent se utilizează mai mule limbaje de descriere hardware HDL (Hardware Description Language): VHDL, Verilog, ABEL (Advanced Booleean Expression Language), AHDL.

Avantajele utilizării limbajelor de descriere hardware:

- Portabilitatea proiectelor pentru diferite tehnologii în care sunt realizate cipurile, în care urmează să se facă implementarea;
- Maniera de descriere a proiectului sub formă de cod îi conferă acestuia claritate mai bună decât dacă ar fi fost descris sub formă de schemă;
- Timp de proiectare redus;
- Opțiuni de optimizare a proiectului, cum ar fi cele de arie sau/şi viteză.
- Folosirea diferitelor construcții specifice HDL cum ar fi: pachetele (package) şi bibliotecile (library), permit reutilizarea lor în alte proiecte.

Limbajul VHDL a fost dezvoltat la mijlocul anilor ’80 de către departamentul de apărare al Statelor Unite în colaborare cu IEEE (**I**nstitute of **E**lectrical and **E**lectronical **E**nginner). Limbajul a fost standardizat pentru prima dată de *IEEE* în 1987 (*VHDL-87*) şi a fost extins în 1993 (*VHDL-93*).

Abrevierea VHDL derivă din ***VHSIC H****ardware* ***D****escription Language* (limbajul de descriere hard VHSIC), abrevierea *VHSIC* însemnând la rândul ei ***V****ery* ***H****igh* ***S****peed* ***I****ntegrated* ***C****ircuit.*

Caracteristicile de bază ale limbajului sunt:

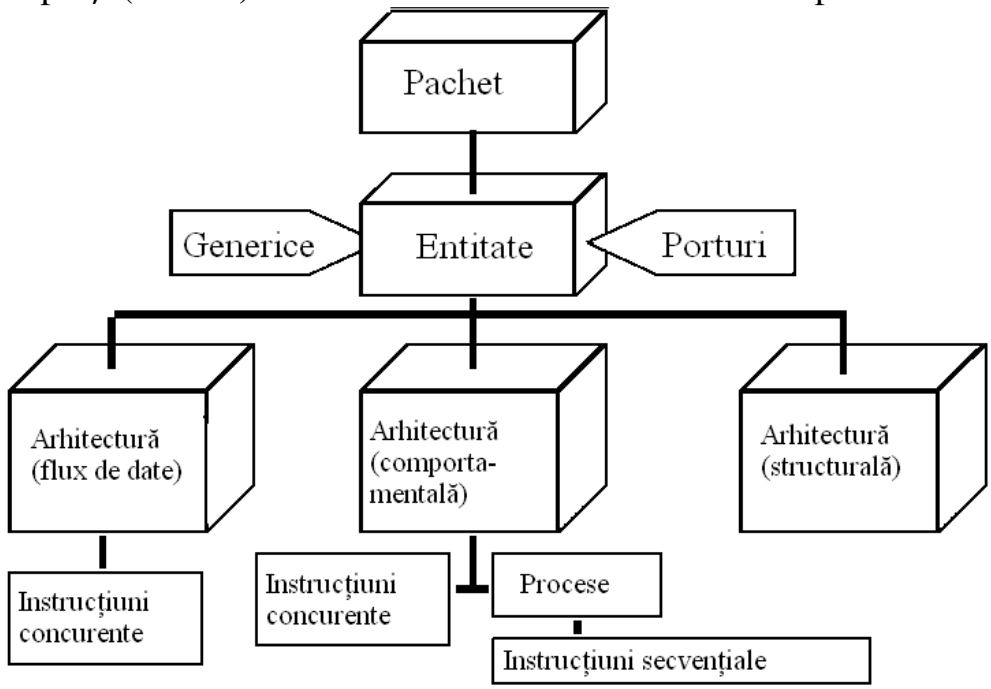
- Descrierea funcționării componentelor electronice de la nivelul de poartă logică până la procesoare și sisteme de calcul.
- Descrierea ierarhică a sistemelor digitale. Modelele VHDL realizate pot fi utilizate ca blocuri în descriere unor circuite complexe;
- Descrierea structurală/comportamentală a sistemelor în scopul sintezei automate;
- Descrierea evenimentelor concurente, specifice funcționării reale a circuitelor digitale;
- Verificarea funcțională (prin simulare) şi temporală a sistemelor prin intermediul unor programe speciale, numite ***test bench***. Ele conțin descrierea stimulilor și a rezultatelor ce ar trebui obținute prin aplicarea acestora, în scopul depistării automate a unor erori funcționale.

Particularitățile limbajelor de descriere hardware și , în particular al limbajului VHDL, sunt următoarele:

- Există unele deosebiri între un program VHDL pentru sinteza circuitului logic și un program VHDL pentru simularea lui;

- Este posibil ca un program VHDL, perfect, din punct de vedere sintactic, să fie neimplementabil datorită descrierii eronate a unor fenomene fizice din circuit;
- În anumite situații este necesară intervenția proiectantului pentru a înlătura anumite anomalii din soluția generată automat.

Principalele părți (blocuri) care alcătuiesc un model VHDL complet sunt:



Arhitectura poate fi de trei tipuri:

1. Structurală
2. Comportamentală
3. Flux de data

La nivel ***structural*** sistemul este descris ca o colecţie de porţi şi conexiuni între ele. Această descriere se apropie de realizarea fizică a sistemului.

Modelarea structurală impune o proiectare ierarhizată în care se pot defini componente folosite de mai multe ori. Aceasta reduce semnificativ complexitatea proiectelor mari.

La nivel ***comportamental (funcţional)*** sistemul este descris prin modul cum se comportă şi nu prin componentele sale. Această descriere foloseşte atât instrucţiuni secvenţiale care se execută în ordinea specificată, cât şi instrucţiuni concurente care se execută în paralel.

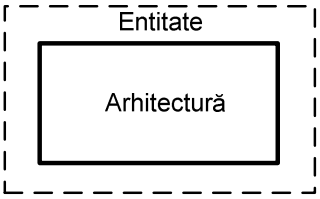
Reprezentarea de tip **Data Flow** descrie modul cum circulă datele prin sistem la nivel de transfer de date între registre (RTL). Această descriere foloseşte instrucţiuni concurente, care se execută în paralel.

Structura unui cod VHDL

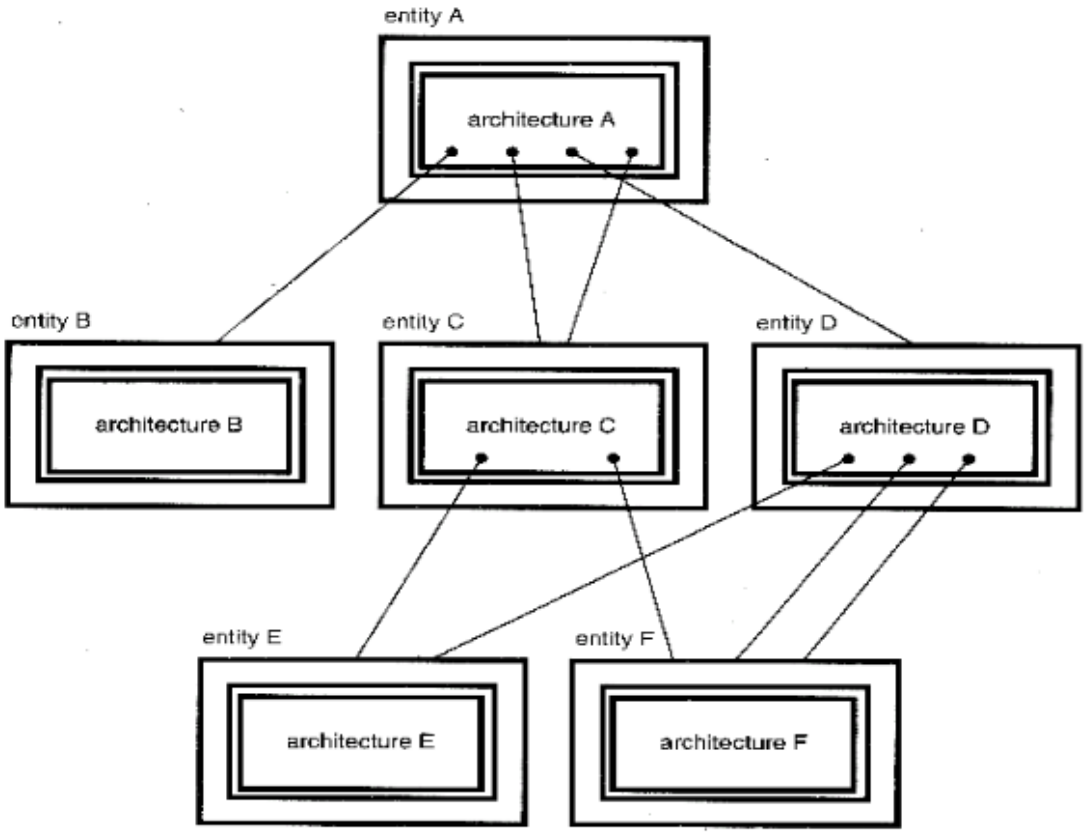
VHDL a fost proiectat ca şi program structurat, împrumutându-se unele idei de la limbajele de programare soft Pascal şi Ada.

O idee de bază este acea de a definii o interfaţă a modulului hard în timp ce detaliile interne sunt ascunse.

Astfel o entitate (*entity*) *VHDL* este o simplă declarare a intrărilor şi ieşirilor modulului în timp ce arhitectura (*architecture*) *VHDL* este o descriere structurală sau comportamentală detailată a funcţionării modulului.

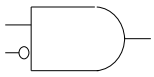


Acest concept formează bazele proiectării ierarhice a sistemelor, şi anume arhitectura entităţii de la nivelul cel mai superior poate conţine (*instantiate*) alte entităţi ale căror arhitecturi sunt "invizibile" de la nivelele superioare. O arhitectură de la nivel superior poate folosii entităţi de la nivelul inferior de mai multe ori, iar mai multe arhitecturi de la nivel superior pot folosii aceeaşi entitate de la nivel inferior fiecare la rândul ei.



Exemplu de program VHDL:

Inhibiţie (BUT/NOT)



```
entity BUT_NOT  is   -- poarta SI cu o intrare inversata
        Port (X,Y:   in   BIT;
              Z:     out   BIT) ;
End BUT_NOT;

Architecture BUT_NOT_arch  of  BUT_NOT  is
        Begin
              Z <='1' when X='1'  and  Y='0' else  '0';
End BUT_NOT_arch;
```

Aici, entity, port, end, is, in, out, architecture, begin, when, else – sunt *cuvinte rezervate* sau *cuvinte cheie*. Acestea nu pot fi folosite de utilizator ca nume de semnale sau identificatori.

Identificatorii sunt definiţi de utilizator. În exemplul de mai sus aceşti identificatori sunt: BUT_NOT, X, Y, Z şi BIT.

BIT este un identificator preexistent şi poate fi redefinit.

Identificatorii pot conţine numai simboluri alfanumerice (A-Z, a-z, 0-9) şi caracterul „underscore” (_). caracterul „underscore” nu poate apărea dublat sau în ultima poziţie. Cuvintele cheie şi identificatorii pot fi scrişi atât cu majuscule cât şi cu litere mici, nu sunt “*case sensitive*”.

Cuvintele cheie definite în VHDL:

abs	**acces**	**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**	**proctected**
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**		

Sintaxa pentru declararea entităţii:

entity nume_entitate **is**
port (nume-semnale : ***mod*** tip-semnale;
nume-semnale : ***mod*** tip-semnale;
...
nume-semnale : ***mod*** tip-semnale) ;
end nume-entitate;

mod = unul din următoarele patru cuvinte rezervate, specificând direcţia semnalului:
- ***in*** – pentru semnal de intrare în entitate;

- ***out*** – pentru semnal de ieşire din entitate, valoarea semnalului nu poate fi "citită" înăuntrul entităţii, ci numai de alte entităţi care-l folosesc;
- ***buffer*** – defineşte un semnal de ieşire din entitate, iar valoare lui poate fi citită şi în interiorul arhitecturii din entitate;
- ***inout*** – defineşte un semnal de intrare/ieşire din entitate, acesta se foloseşte frecvent pentru a descrie pini three-state.

Sintaxa pentru declararea arhitecturii:

arhitecture nume_arhitectură **of** nume_entitate **is**
Zona de declaraţii (tipuri, semnale, constante,
funcţii, proceduri, componente)
begin
instrucţiuni_concurente
end nume_arhitectură;

Numele entităţii (*entity-name*) trebuie să fie acelaşi cu cel folosit la definirea entităţii. Numele arhitecturii (*architecture-name*) este un identificator definit de utilizator, dacă se doreşte poate fi la fel cu cel al entităţii sau diferit.

Reprezentări numerice

Reprezentarea numerică obişnuită este reprezentarea zecimală.
VHDL permite reprezentarea de tip întreg şi real.
Tipul întreg : 13, 25, 45E6
Tipul real : 1.2 , 3.14E-2

Pentru reprezentarea unui număr în altă bază se foloseşte notarea:
baza#număr#
Exemple: 2#101101# reprezentare binară
8#356# reprezentare octală
16#1C# reprezentare hexazecimală
Pentru a face citirea numerelor mai uşoară se admit caractere underscore: 2#1010_1100_1110#

Caractere, şiruri de caractere şi şiruri de biţi

Pentru a putea folosi caractere se folosesc ghilimele simple:

‘a’, ‘A’, ‘.’

Şirurile de caractere se plasează între ghilimele duble:

”caracter”

Un şir de biţi reprezintă o secvenţş de valori 0 şi 1. Şiruri de biţi pot fi reprezentate în sistemele binar, octal şi hexazecimal (mai compact):

Exemple:

Binar B”1001_1011”

Octal O”324”

Hexazecimal X”3D4A”

Obiecte de tip date: semnale, variabile şi constante

Un obiect de tip dată este creat de o declaraţie a obiectului şi are asociată o valoare şi un tip.

Constante

O constantă poate avea o singură valoare de un tip specificat şi nu se poate modifica pe parcursul modelării.

Sintaxa declarării constantei:

constant *nume-constantă* ***:*** *nume-tip* ***: =*** *valoare;*

Constantele se declară la începutul unei arhitecturi şi se pot folosi apoi oriunde în arhitectură. Constantele declarate într-un proces se pot folosi doar în procesul respectiv.

Exemple:

constant X: integer := 24;
constant BUS_SIZE: integer :=32; -- reprezintă lăţimea componentei
constant Y: integer : = BUS_SIZE-1; -- numărul de biţi ai lui Y
constant Z: character := ‘Z’; -- sinonim cu valoarea de înaltă impedanţă
constant T: time := 2 ns;

Constantele pot fi definite în pachet, entitate, arhitectură și proces.

Variabile

O variabilă poate avea o singură valoare la un moment dat, dar ea poate fi şi actualizată folosind o instrucţiune de actualizare. Variabila se actualizează fără nici o întârziere, imediat ce instrucţiunea este executată. Variabilele se declară doar în interiorul proceselor.

Sintaxa declarării variabilelor:

variable *nume-variabile : tip-variabile [:= valori iniţiale];* --[…] opţional

Exemple:

variable CT : bit :=0 ;
variable VAR : boolean := FALSE;
variable SUM : integer **range** 0 **to** 256 := 16 ; -- 16 este valoarea iniţială
variable X_BIT : bit_vector (7 **downto** 0) ; -- defineşte un vector de 8 biţi
Pentru actualizarea variabilei se poate folosi o instrucţiune de asignare:
nume_variabilă := expresie;

Semnale

Semnalele reprezintă porturile de intrare/ieşire şi conexiunile interne ale unui circuit. Semnalele se definesc în cadrul entităţii.

```
entity nume_entitate is
    port ( nume-semnale : mode  tip-semnale;
           nume-semnale : mode  tip-semnale;
           ...
           nume-semnale : mode  tip-semnale) ;
end nume-entitate;
```

De asemenea se pot definii şi semnale interne arhitecturii, dar ele vor acţiona doar local.

Sintaxa pentru definirea unui semnal intern:
signal *nume_semnal: tip_semnal [:=valori iniţiale]*; -- nu este specificat modul

Exemple:

signal SUMA, CARRY : std_logic;
signal CLK : bit ;
signal DATA_BUS : bit_vector (0 to 7) ;
signal VAL : integer range 0 to 100 ;

Semnalele sunt actualizate când se execută o instrucţiune de asignare, cu o anumită întârziere:

SUMA <= (A xor B) **after** 2 ns ;

În particular se poate astfel specifica şi o formă de undă:

signal unda : std_logic ;
unda <= ‘0’, ‘1’ **after** 5 ns, ‘0’ **after** 10 ns, ‘1’ **after** 20 ns ;

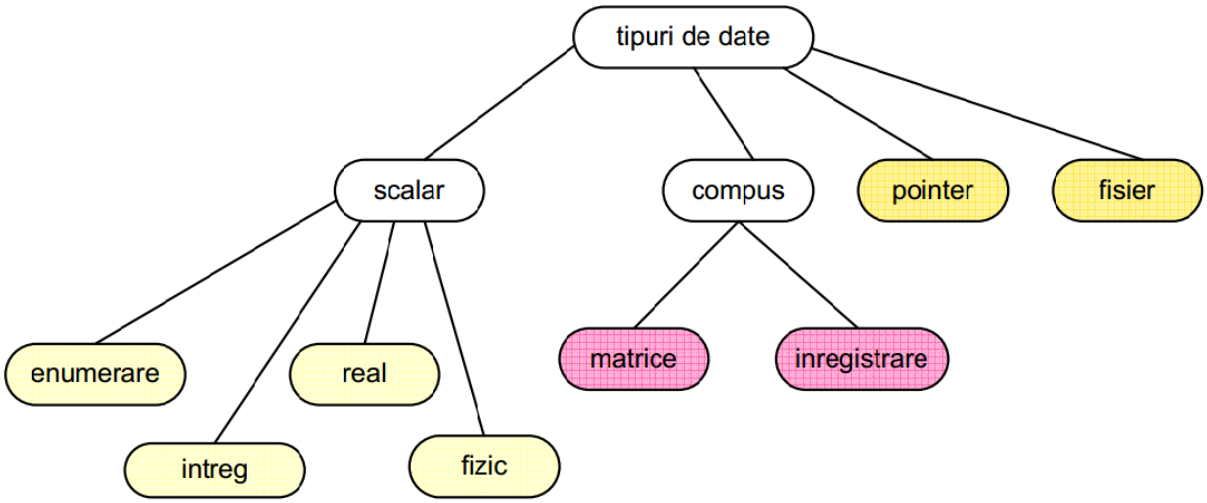
Tipuri de date

In VHDL există doua feluri de tipuri: tipuri SCALARE si tipuri COMPUSE.

Tipurile scalare includ numere, cantitati fizice, enumerari si tipuri predefinite.

Tipurile compuse sint matrice şi înregistrări.

În VHDL sânt definite şi tipurile 'access' (pointeri) si 'file' (fisiere).



Toate semnalele, variabilele şi constantele dintr-un program *VHDL* trebuie să aibă asociat un tip. În cadrul tipului se specifică un set de valori pe care le poate lua obiectul (semnal, variabilă …) şi de asemenea există şi un set de operatori (+, AND etc) asociaţi tipului respectiv.

Tipurile de date predefinite din limbajul VHDL sunt descrise în pachetul (eng. package) ***standard*** din biblioteca ***std*** **.**

Tipurile predefinite din limbajul VHDL

Tip	Domeniul de valori	Exemple
Bit	‘0’, ‘1’	signal B: bit :=1;

Bit_vector	un șir de elemente de tip bit	signal BUS: bit_vector (7 downto 0);
Boolean	TRUE, FALSE	variable X: boolean :=true;
Character	orice caracter permis în VHDL	variable Z: character :='a' ;
Integer	Domeniul include numerele întregi de la -(2^{31}-1) până la +(2^{31}-1).	constant A: integer :=12;
Natural	De la 0 pâna la valoarea maximă admisă	variable C: natural :=4;
Positive	De la 1 pâna la valoarea maximă admisă	variable D: positive :=10;
String	Un șir de elemente de tip caracter	variable list: string :="abcde&*%";

Tipul *character* (caracter) include toate caracterele din setul de caractere ISO* (*International Organization of Standardization*) exprimate pe 8 biţi, primele 128 fiind caractere ASCII.

Tipul fizic:

Tipul fizic este un tip numeric de reprezentare a unor cantități fizice (lungime, timp, volți). Declarația de tip include specificarea unei unități de măsură de bază și eventual un număr de unități de măsură secundare, care reprezintă multiplii ai unității de bază.

Exemplu:

type length **is range** 0 **to** 1E9
units
um;
mm = 1000 um;
cm = 10 mm;
m = 1000 mm;
end units;

Există tipul predefinit 'time', folosit in simulări VHDL pentru specificarea întârzierilor.

type **time** is range **interval_maxim_din_implementare**
units
fs;
ps = 1000 fs;
ns = 1000 ps;
us = 1000 ns;

ms = 1000 us;
sec = 1000 ms;
min = 60 sec;
hr = 60 min;
end units;

Înregistrări: Înregistrările în VHDL sânt colecţii de elemente, care pot avea tipuri diferite.

Exemplu:

type instruction **is**
record
op_code : processor_op;
address mode : mode;
operand1,operand2 : integer range 0 to 15;
end record;

Cele mai folosite tipuri în programele *VHDL* sunt aşa numitele tipuri definite de utilizator (*user defined types*), unul dintre acestea este ***tipul enumerare*** (*enumerated type*) definit printr-o înşiruire de valori.

Exemple de declarare a tipului enumerare:

Type nume-tip **is** (listă de valori);

Lista de valori reprezintă enumerarea tuturor elementelor posibile pentru respectivul tip, separate prin virgulă. Valorile enumerate pot fi atât caractere (un caracter este cuprins între ghilimele simple) cât şi identificatori definiţi de utilizator.

Exemple.

type oct_digit **is** (`0`, `1`, `2`,`3`,`4`,`5`,`6`,`7`) ;
type PC_OPER **is** (load, store, add, sub) ;
type MY_WORD **is range** 31 **downto** 0 ;
type cmos_level **is range** 0.0 to 3.3 ;

Exemple de obiecte care folosesc aceste tipuri:

variable ALU_OP : PC_OPER ;
signal SIG : oct_digit ;

Dacă nu se iniţializează semnalul, iniţializarea implicită este valoarea extremă in stânga domeniului.

Limbajul *VHDL* permite definirea de subtip (*subtype*) corespunzător unui tip. Valorile unui subtip trebuie să se afle într-un subdomeniu de valori ale tipului de bază, continuu faţă de valorile tipului de bază.

Exista o serie de tipuri de enumerări predefinite:

type severity-level is (note,warning,error,failure);
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, ` `, `!`, … `z`, `{`, `|`, `}` , `~`, DEL);

Subtype nume-subtip **is** nume-tip valoare-iniţială **to** valoare-finală; -- ordine crescătoare.

Subtype nume-subtip **is** nume-tip valoare-iniţială **downto** valoare-finală; -- ordine descrescătoare.

Exemple:
Subtype twoval_logic **is** std_logic **range** `0` **to** `1` ;
Subtype negint **is** integer **range** `-2147483647` **to** `-1` ;
Subtype data_word **is** my_word **range** 7 **downto** 0 ;

VHDL include două subtipuri integer predefinite:
Natural : subtype natural is integer range 0 to *cel mai mare întreg*;
Positive: subtype positive is integer range 1 to **cel mai mare întreg**;

Un tip foarte important este ***std_logic*****,** care este un tip standard definit de utilizator şi este parte a package-ului standard IEEE 1164. Acest pachet include tipul de date std_ulogic care are 9 valori logice utile în simularea unui semnal logic dintr-un circuit real.

`U` - neinitializat
`X` - puternic necunoscut
`0` - puternic 0 -- poate fi sintetizat
`1` - puternic 1 -- poate fi sintetizat
`Z` - impendanta inalta -- poate fi sintetizat
`W` - slab necunoscut
`L` - slab 0

`H` - slab 1
`-` indiferent.

O altă categorie foarte importantă de tip definit de utilizator este tipul matrice (***array type***). Acest tip defineşte o matrice ca fiind un set de elemente de acelaşi tip în care fiecare element este selectat printr-un indice de matrice (*array index*)

Versiuni de sintaxă folosite în declararea unei matrici:

Type nume-tip **is array** (valoare-initiala **to** valoare-finala) **of** tip-element;
Type nume-tip **is array** (valoare-initiala **downto** valoare-finala) **of** tip-element;

Exemple:

Type luni_count **is array** (1 **to** 12) **of** integer;
Type byte **is array** (7 **downto** 0) **of** std_logic;

Constant WORD_LEN: integer :=32
type word **is array** (WORD_LEN-1 **downto** 0) **of** word;
type matrice3x2 **is array** (1 to 3, 1 to 2) **of** natural ; -- matrice bidimensională

Se consideră implicit că elementele unei matrici sunt ordonate de la stânga la dreapta, astfel cel mai din stânga element al matricilor luni_count, byte, word este 1, 7, 31.

Uneori este util să nu se specifice domeniul.
type nume **is array** (tip **range <>) of** tip_elemente ;

Accesarea unui singur element din matrice:

W(5) , unde W este semnal aferent matricii word.

O matrice literală se defineşte (*array literals*) prin înşiruirea între paranteze a valorilor elementelor. Elementele variabilei B de tipul byte pot primi toate valoarea *1* logic scriind o expresie de forma:

B := (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’);

Limbajul *VHDL* permite de asemenea notaţii mai scurte, de exemplu pentru a atribui valoarea *1* logic tuturor biţilor variabilei *W* de tip *word*, în afară de LSB din fiecare se va scrie expresia:

W := (0=>'0', 8=>'0', 16=>'0', 24=>'0', others=>'1');

Expresiile anterioare pot fi rescrise folosind şirurile de caractere, după cum urmează:

B := "11111111";

W:= "11111110111111101111111011111110";

Este posibil de asemenea să se facă referire la un subset de valori (*slice*) dintr-o matrice, specificând începutul şi sfârşitul indicilor elementelor din subset, exemple: M(6 to 9), B(3 downto 0), W(15 downto 8) etc.

Cel mai important tip de matrice des întâlnit în programele *VHDL* este cel aparţinând standardului logic definit de utilizator IEEE 1164 (*std_logic_vector*), definiţia acestui standard este:

type STD_LOGIC_VECTOR **is array** (natural range < >) **of** STD_LOGIC

Datorită faptului că *VHDL*-ul este un limbaj puternic tipizat, adesea apare necesitatea convertiri unui semnal sau variabile dintr-un anumit tip în altul. Packageul IEEE 1164 conţine câteva astfel de funcţii de conversie, de exemplu din *BIT* în *STD_LOGIC* sau invers. O conversie foarte utilizată şi care nu este definită, din cauză că proiecte diferite pot avea nevoie de o interpretare diferită a numerelor (ex. numere cu semn sau fără semn), este conversia din *STD_LOGIC_VECTOR* în *integer.*

Conversii suportate de package-ul std_logic_1164	
Conversia	**Funcția**
std_ulogic -> bit	to_bit(*expresie*)
std_logic_vector -> bit_vector	to_bitvector(*expresie*)
std_ulogic_vector -> bit_vector	to_bitvector(*expresie*)
bit -> std_ulogic	To_StdULogic(*expresie*)
bit_vector -> std_logic_vector	To_StdLogicVector(*expresie*)
bit_vector -> std_ulogic_vector	To_StdUlogicVector(*expresie*)
std_ulogic -> std_logic_vector	To_StdLogicVector(*expresie*)
std_logic -> std_ulogic_vector	To_StdUlogicVector(*expresie*)

Atribute

În VHDL există două categorii de atribute:

1. Predefinite ca parte a standardului 1076;
2. Introduse de utilizator.

Atributele predefinite sunt întotdeauna aplicate ca un prefix numelui unui semnal, variabile sau tip.

Atributele sunt folosite pentru e returna diferite tipuri de informaţie.

Exemple de atribute predefinite:

nume_semnal`event - întoarce valoarea booleană True dacă semnalul a avut o tranziţie şi False dacă nu;

nume_semnal`active - întoarce valoarea booleană True dacă a existat o atribuire a valorii semnalului şi False dacă nu.

Operatori VHDL

Limbajul VHDL suportă diferite clase de operatori care operează pe semnale, constante şi variabile.

Clasa						
1. Operatori logici	and	or	nand	nor	xor	xnor
2. Operatori relaţionali	=	/=	<	<=	>	>=
3. Operatori de deplasare	sll	srl	Sla	sra	rol	ror
4. Operatori aditivi	+	–	&			
5. Operatori unari	+	-				
6. Operatori multiplicativi	*	/	mod	rem		
7. Alţi operatori	**	abs	Not			

Prioritatea operatorilor este maximă pentru operatorii din clasa 7. Dacă nu sunt folosite paranteze, ei se execută primii. Operatorii din aceeaşi clasă au aceeaşi prioritate şi se execută de la dreapta la stînga într-o expresie.

Operatorii logici sunt definiţi pentru tipurile bit, boolean, std.logic şi pentru vectorii de aceste tipuri.

Operatorii nand şi nor nu sunt asociativi. De exemplu expresia: **X nand Y nand Z** va da o eroare de sintaxă. Corect se va scri: **(X nand Y) nand Z.**

Operatorii relaţionali testează relaţia dintre două tipuri scalare şi oferă ca rezultat o ieşire booleană TRUE sau FALSE. De menţionat ca simbolul operatorului „mai mic sau egal” este acelaşi cu cel al operatorului de atribuire pentru semnale şi variabile.

Operatorii de deplasare execută o deplasare sau rotire la nivel de bit într-un vector de elemente de tip bit (std.logic) sau boolean.

Operatorii aditivi sunt folosiţi pentru operaţii aritmetice pe orice tip numeric de operanzi. Operatorul de concatenare este folosit pentru a concatena doi vectori, de exemplu prin concatenarea ‘0’ & ’1’ & ‘1Z’ se obţine “011Z”. Pentru a folosi aceşti operatori trebuie specificate pachetele std_logic_unsigned sau std_logic_arith pe lângă pachetul std_logic_1164.

Operatorii unari se folosesc pentru a specifica semnul unui tip numeric.

Operatorii de multiplicare se folosesc pentru a executa funcţii matematice pe tipurile numerice (întreg sau virgulă mobilă).

mod – împărţire modulo;

rem - împărţire modulo cu rest;

****** - ridicare la putere (pentru tipurile numerice);

abs - valoare absolută (pentru tipurile numerice);

not – negare (pentru tipurile bit şi boolean);

Conversiile dintre std_logic_vector şi tipurile numerice:

Data type of a	To data type	Conversion function/type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

Sa presupunem ca sunt declarate următoarele semnale:

s i g n a l sl, s 2 , s3: std_logic_vector (3 downto *0) ;*

s i g n a l ul, u2, u3: unsigned (3 downto 0) ;

Operaţiile:

u l <= s l ;

s2 <= u 3 ;

nu sunt corecte, deoarece au loc atribuiri între diferite tipuri.

Corect se va scrie:

u l <= unsigned(s1);

s2 <= std_logic_vector(u3);

Operaţii aritmetice:

u4 <= u2 + ul; -- corect

s5 <= s2 + sl; -- incorect, std_logic_vector nu susţine operatori aritmetici

s5 <= std_logic_vector(unsigned(s2) + unsigned(s1)); -- ok

Exemple cu operatorul ***concatenare***

signal a1: std_logic;

signal a4: std_logic_vector (3 downto 0);

signal b8, c8: std_logic_vector (7 downto 0);

...

b8 <= a4 & a 4 ;

c8 <= a1 & a1 & a4 & ” 0 0 ” ;

Operatorul & poate fi utilizat şi pentru a deplasa datele. Cu toate că operatorii de deplasare sunt definiţi în limbajul VHDL, ei uneori nu pot fi sintetizaţi automat de compilator.

```
signal a : std_logic_vector (7 downto 0);
signal rot, shl, sha : std_logic_vector (7 downto 0);
...
rot <= a (2 downto 0) & a (7 downto 3); -- deplasare ciclica cu 3 biti la dreapta
shl <= “000”& a(7 downto 3);  -- deplasare logica cu 3 biti la dreapta
sha <= a ( 7 ) & a ( 7 ) & a ( 7 ) & a ( 7 downto 3 ) ; -- deplasare aritmetica cu 3
-- biti la dreapta
```

Biblioteci şi package-uri

O bibliotecă (*library*) *VHDL* este locul unde compilatorul depune toate informaţiile despre un anumit proiect şi anume toate fişierele intermediare folosite în analiza, simularea şi sinteza proiectului.

Pentru un proiect *VHDL* dat compilatorul creează şi foloseşte în mod automat o bibliotecă numită *work*.

Pentru a utiliza biblioteca standard care conţine elemente (funcţii, tipuri, etc) predefinite se foloseşte directiva *library* la începutul codului *VHDL*, ex.

library ieee;

Astfel se obţine accesul la toate entităţile şi arhitecturile aflate în biblioteca respectivă.

Pentru a accesa tipurile definite se folosesc pachetele (*package*-urile).

Un *package* este un fişier care conţine obiecte definite folosite în mod curent. Tipurile de obiecte care se află într-un *package* sunt: semnale, definiri de tipuri, constante, funcţii, proceduri şi declarări de componente.

Un proiect poate folosi un *package* dacă în codul *VHDL* se include directiva *use,* de exemplu pentru a apela toate componentele definite în package-ul standard IEEE 1164 scriem următoarea secvenţă de cod:

use ieee.std_logic_1164.all;,

unde *ieee* este numele unei librării care a fost dat împreună cu directiva *library.* Sintaxa *std_logic_1164* este numele unui fişier care conţine componentele definite, iar sufixul *all* îi spune compilatorului să le folosească pe toate.

În locul respectivului sufix se poate scrie numai numele unei anumite componente care să fie luată în considerare de compilator.

Exemple de pachete:

std_logic_1164 - defineşte tipurile de date standard;

std_logic_arith - conţine funcţii aritmetice, de conversie şi comparaţie pentru tipurile de date signed, unsigned, integer, std_logic şi std_logic_vector;

std_logic_unsigned;

std_logic_misc - defineşte tipuri suplimentare, subtipuri, constante şi funcţii pentru pachetul std_logic_1164.

Definirea package-urilor nu se limitează doar la cele standard aflate în biblioteci, utilizatorul poate el însuşi să-şi definească package-uri.

Sintaxa de declarare a unui pachet este următoarea:

```
package  numele_pachetului is
        declaraţiile pachetului
end package numele_pachetului ;
package body numele_pachetului is
        declaraţiile corpului pachetului
end package body numele_pachetului;
```

De exemplu, în proiectarea structurală, funcţiile de bază pentru componentele AND2, OR2, NAND2, NOR2, XOR2 etc., sunt necesare ca să fie definite înainte să fie utilizată una din ele. Acest lucru poate fi făcut într-unul dintre pachete, de ex. **base_func** pentru fiecare dintre aceste componente, după cum urmează :

```
-- declaraţia pachetului
library ieee ;
use ieee.std_logic_1164.all ;
package  basic_func is

-- declaraţia pentru AND2
        component  AND2
                generic (DELAY: timp  :=5ns);
                port (in1, in2: in std_logic; out1:  out  std_logic);
        end component;

--declaraţia pentru OR2
        component   OR2
                generic (DELAY: timp  :=5ns);
                port (in1, in2: in std_logic; out1:  out  std_logic);
        end component;
```

```
end package  basic_func ;

-- declaraţiile corpului pachetului
library ieee ;
use ieee.std_logic_1164.all;
package body basic_func  is

        -- poartă AND cu două intrări
        entity AND2   is
                generic (DELAY:  timp);
                port (in1, in2:  in std_logic;  out1:  out  std_logic);
        end AND2;
        arhitecture  model_conc  of AND2  is
                begin
                        out1 <= in1  and  in2  after  DELAY;
        end model_conc;

        -- poartă OR  cu două intrări
        entity OR2   is
                generic (DELAY:  timp);
                port (in1, in2:  in std_logic; out1:  out  std_logic);
        end  OR2;
        arhitecture  model_conc2  of AND2  is
                begin
                        out1 <= in1  or  in2  after  DELAY;
        end model_conc2;

end package body basic_func;
```

În program a fost inclusă o întârziere (delay) de 5 ns. Totuşi, ar trebui de notat că specificaţiile întârzierii sunt ignorate de către Foundation synthesis tool. A trebuit să utilizăm tipul predefinit std_logic care este declarat în pachetul std_logic_1164. S-au inclus în acest pachet **library** şi clauza **use**. Acest pachet are nevoie să fie compilat şi plasat într-o librărie. O să numim aceasta ***library my_func***. Pentru a utiliza componentele acestui pachet, o dată ce a fost declarat, se utilizează *library* şi clauza *use* :

```
library  ieee, my_func;
use  ieee.std_logic_1164.all, my_func.basic_func.all;
```

Funcţii şi proceduri

La fel ca şi în alte limbaje de nivel înalt şi în *VHDL* o funcţie acceptă un anumit număr de argumente şi returnează un rezultat. Atât argumentele cât şi rezultatul returnat de o funcţie sunt de un tip predeterminat.

Sintaxa *VHDL* prin care se defineşte o funcţie:

function nume-funcţie (
nume-semnal : tip-semnale;
...
nume-semnal : tip-semnale;
) **return** tip-întoarcere **is**
declaraţii de tip
declaraţii de constante
declaraţii de variabile
definiţii de funcţii
definiţii de proceduri
begin
instrucţiune secvenţială
...
Instrucţiune secvenţială
end nume funcţie;

După ce se dă un nume funcţiei (*function -name*) se poate definii o listă de parametri formali care vor fi folosiţi în structura funcţiei, toţi aceşti parametrii trebuie să fie de acelaşi tip. În cadrul unei funcţii se pot definii local tipuri, constante, variabile şi de asemenea funcţii şi proceduri imbricate (*nested*). Între cuvintele cheie *begin* şi *end* este cuprinsă o serie de instrucţiuni secvenţiale care sunt executate ori de câte ori este apelată funcţia . În cadrul funcţiei cuvântul cheie *return* indică momentul din care funcţia returnează o valoare care trebuie să fie de tipul celei definite la declararea funcţiei.

Exemplu:

```
entity BUT_NOT  is   -- poarta SI cu o intrare inversata
        Port (X1,X2:   in   BIT;
              Z:       out   BIT) ;
End BUT_NOT;
Architecture BUT_NOT_archf  of  BUT_NOT  is
```

```
        function ButNot (A, B: bit)  return bit is
          begin
              if B = `0` then return A;
              else return `0
              end if;
          end ButNot;
    Begin
        Z <= ButNot (X1,X2);
    End BUT_NOT_archf;
```

În limbajul *VHDL* mai este definită noţiunea de procedură (*procedure*) care este similară cu funcţia doar că nu returnează o valoare. La fel cum o funcţie poate fi apelată în locul unei expresii şi o procedură poate fi apelată în locul unei declaraţii. Dacă argumentele unei proceduri sunt declarate de tipul *out* sau *inout*, va exista totuşi o valoare returnată.