LIMBAJUL VHDL - 2

UNITĂŢI DE PROIECTARE

Unităţi de proiectare primare

- entitate (interfaţa sistemului)
- specificaţie de pachet (vedere externă a posibilităţilor puse la dispoziţie)
- configuraţie (asociere componentă - model)

UNITĂŢI DE PROIECTARE

Unităţi de proiectare secundare

- arhitectură (descrierea sistemului)
- corp de pachet (descrierea internă a funcţionalităţilor)

UNITĂŢI DE PROIECTARE

Entitate

entity nume_entitate **is**

{**generic** (listă de parametri generici);}

{**port** (listă de porturi);}

{**begin**

listă de instrucţiuni concurente}

end {nume_entitate};

UNITĂŢI DE PROIECTARE

Entitate

- **numele entităţii** - **unic** în biblioteca respectivă
- **parametri generici** - pentru a **reutiliza** entităţile
- **port** - informaţii pentru **semnale de interfaţă** (nume, mod, tip, valori iniţiale)
  - **mod** - cuvinte rezervate - specifică **direcţia** semnalelor
  - mod: **in**, **out**, **inout**, **buffer**, **linkage**

UNITĂŢI DE PROIECTARE

Exemple de entităţi

- Poartă logică de tip ŞI-NU cu 2 intrări

```
entity nand2 is
        port (a, b: in bit; y: out bit);
end nand2;
```

- Poartă logică de tip SAU – numărul intrărilor specificat prin parametru generic

```
entity or is
        generic (input_no: natural := 2);
        port (input: in bit_vector (1 to input_no); y: out bit);
end or;
```

UNITĂŢI DE PROIECTARE

Exemple de entităţi

- Multiplexor 2:1

```
entity mux_2_1 is
        port (i0, i1, s: in bit; y: out bit);
end mux_2_1;
```

- Bistabil D sincron, cu intrări asincrone prezente

```
entity d_ff is
        port (d, clk, r, s: in std_logic; q, qn: out std_logic);
end d_ff;
```

UNITĂŢI DE PROIECTARE

Arhitectură

architecture nume_arhitectură **of** nume_entitate **is**

... Zona de declaraţii (tipuri, semnale, constante, funcţii, proceduri, componente)

begin

... **Instrucţiuni concurente**

end {nume_arhitectură};

UNITĂŢI DE PROIECTARE

Arhitectură

- tipuri de descriere:
  - **structurală** = interconectare de alte “cutii” negre
  - **comportamentală** = funcţională
  - **flux de date** = descriere algoritmică
  - **hibridă** = combinaţii între primele 3
- la o entitate - mai multe arhitecturi posibile
- **Observaţie** - **entitatea şi arhitectura trebuie să se găsească în aceeaşi bibliotecă**

UNITĂŢI DE PROIECTARE

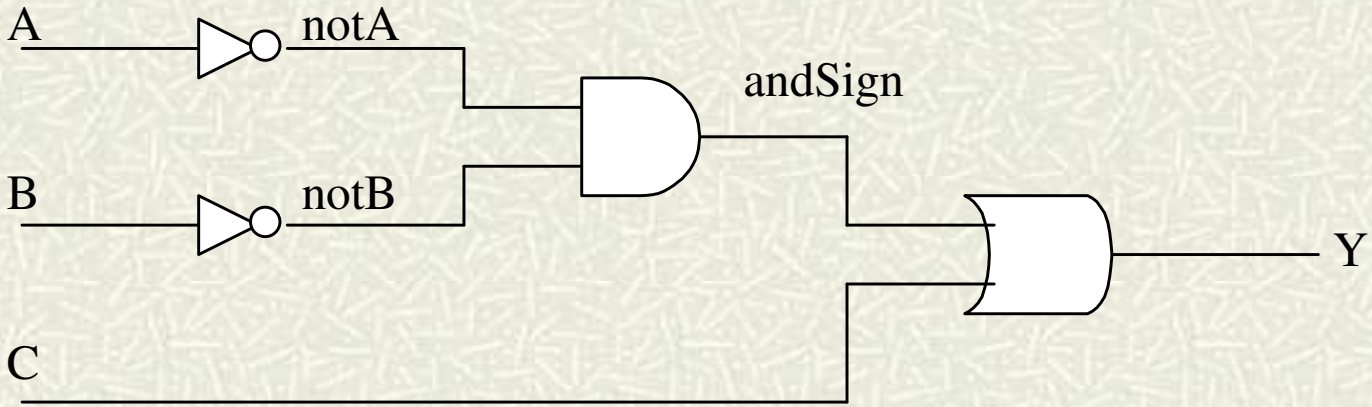
Arhitectură

- **nume_entitate** trebuie să corespundă cu numele dat entităţii
- arhitectura face parte din **domeniul concurent** $\Rightarrow$ **nu** se admit **declaraţii de variabile**
- funcţionalitatea - descrisă de instrucţiuni concurente care se **execută asincron**

UNITĂȚI DE PROIECTARE

Exemple de arhitecturi

- Descrieri pentru circuitul din figură:




```
library IEEE;
use IEEE.std_logic_1164.all;
use work.primitive.all;
   entity LogicF is
        port (A, B, C: in std_logic; Y: out std_logic);
   end LogicF;
```

UNITĂŢI DE PROIECTARE

Exemple de arhitecturi

- Arhitectură **structurală**

```
architecture structural of LogicF is
    signal notA, notB, andSign: std_logic;
begin
    inv1: inverter port map (i => A, o => notA);
    inv2: inverter port map (i => B, o => notB);
    si1: and2 port map (i1 => notA, i2 => notB, y => andSign);
    sau1: or2 port map (i1 => andSign, i2 => C, y => Y);
end structural;
```

UNITĂŢI DE PROIECTARE

Exemple de arhitecturi

- Arhitectură **comportamentală**

```
architecture behavioral of LogicF is
begin
    fcn: process (A,B,C)
        begin
            if (A = '0' and B = '0') then Y <= '1';
            elsif C = '1' then Y <= '1';
            else Y <= '0';
            end if;
        end process;
end behavioral;
```

UNITĂŢI DE PROIECTARE

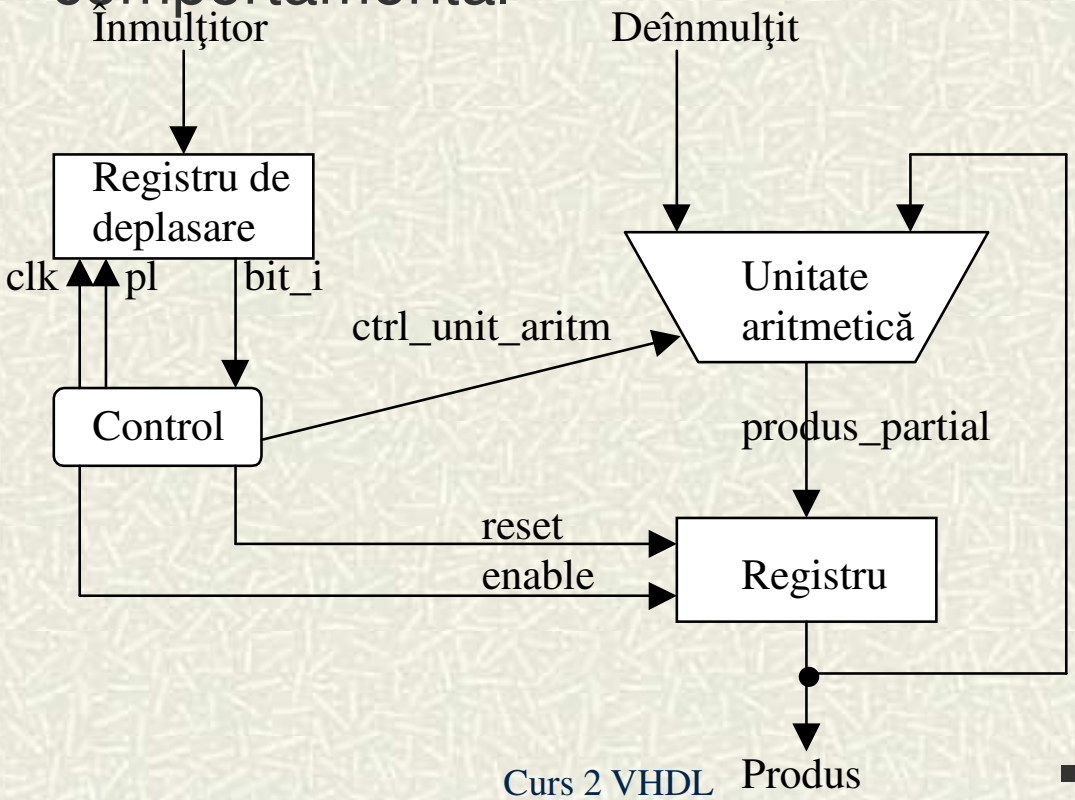
Exemple de arhitecturi

- Arhitectură **flux de date**

```
architecture dataflow of LogicF is
begin
        Y <= '1' when (A = '0' and B = '0') or (C = '1') else '0';
end dataflow;
```

Exemple de arhitecturi

- Arhitectură **mixtă (hibridă)** pentru înmulţire
  - calea de date - structural
  - controlul - comportamental



UNITĂŢI DE PROIECTARE

Exemple de arhitecturi

- Entitatea pentru înmulţire

```
entity inmultire is
        port (clk, reset: in bit; deinmultit, inmultitor: in integer;
        produs: out integer);
end entity inmultire;
```

Exemple de arhitecturi

- Arhitectura mixtă pentru înmulţire

```
architecture mixta of inmultire is
    signal produs_partial, produs_total: integer;
    signal ctrl_unit_aritm, enable, bit_i, pl: bit;
begin
    unit_aritmetica: entity work.unitate_aritmetica(behavior)
        port map (intrare => deinmultit, intrare_pi => produs_total, suma =>
        produs_partial, control_adunare => ctrl_unit_aritm);
    rezultat: entity work.registru(behavior)
        port map (d => produs_partial, q => produs_total, en => enable, reset
        => reset);
    deplasare: entity work.registru_deplasare(behavior)
        port map (d => inmultitor, q => bit_i, load => pl, clk => clk);
    produs <= produs_total;
```

Exemple de arhitecturi

- Arhitectura mixtă pentru înmulţire (continuare)

```
control: process is
        -- declaraţii de variabile pentru secţiunea de control
        begin
        -- instrucţiuni secvenţiale pentru asignarea de valori semnalelor de
        -- control
        wait on clk, reset;
        end process control;
end architecture mixta;
```

UNITĂŢI DE PROIECTARE

Specificaţie de pachet

package nume_pachet **is**

definiţii;

. . . -- conţinutul pachetului;

declaraţii;

end nume_pachet;

UNITĂŢI DE PROIECTARE

Specificaţie de pachet

- ce exportă pachetul prin specificaţie:
  - obiecte (semnale, constante, fişiere, variabile partajate)
  - tipuri şi subtipuri
  - subprograme (funcţii şi proceduri)
  - declaraţii de componente şi alias-uri
  - specificaţii sau declaraţii de atribute
  - specificaţii de conectare
  - clauze **use**
- **utilizarea pachetului**:

use biblioteca**.**nume_pachet**.all**;

UNITĂŢI DE PROIECTARE

Exemplu de specificaţie de pachet

- tipuri şi subtipuri, obiecte, subprograme

```
package tip is
    subtype byte is std_logic_vector (7 downto 0);
        -- creează un subtip pt. un vector de 8 biţi
    constant clear : byte := (others => '0');
        -- constantă iniţializată la 0
    procedure reg8 (reset, clk : in std_logic; data_in : in byte;
        Qout : out byte);
        -- registru de memorare
end tip;
```

UNITĂŢI DE PROIECTARE

Corp de pachet

package body nume_pachet **is**

{declaraţii interne}

… -- subprograme;

end nume_pachet;

UNITĂŢI DE PROIECTARE

Corp de pachet

- este **unic**
- conţine **algoritmii** (**strict secvenţiali**) pentru subprograme
- face parte din **domeniul secvenţial** - **nu** se pot declara semnale
- declaraţii interne utilizate local - nu sunt vizibile nici măcar din propria specificaţie
  - tipuri şi subtipuri
  - constante, fişiere, alias-uri
  - subprograme (declaraţie + corp)

UNITĂŢI DE PROIECTARE

Exemplu de corp de pachet

- subprogramul reg8 din specificaţia pachetului tip

```
package body tip is
   procedure reg8 (reset, clk : in std_logic; data_in : in byte;
                   Qout : out byte) is
      begin
                   if reset = '1' then Qout <= clear;
                   elsif clk = '1' and clk'event then Qout <= data_in;
                   end if;
   end reg8;
end tip;
```

UNITĂŢI DE PROIECTARE

Configuraţie

configuration nume_configuraţie **of** nume_entitate **is**

{Zona declarativă (numai clauza **use** şi specificarea atributelor)}

{Zona rezervată configuraţiei}

end {nume_configuraţie};

for eticheta_instanţei_componentei: nume_componentă

use entity nume_entitate

(numele_arhitecturii){parametrii generici}

{corespondenţa porturi formale / porturi actuale};

end for;

UNITĂŢI DE PROIECTARE

Configuraţie

- descrie **corespondenţa** dintre componente (declarate în arhitecturi structurale sau hibride) şi arhitecturi precizate pentru entitate
- **configurare imediată** - pt. componenta folosită:
  - se specifică modelul al cărei instanţă este
  - se face corespondenţa între porturile formale şi cele actuale

UNITĂŢI DE PROIECTARE

Configuraţie

- **configurare amânată** - se face în interiorul configuraţiei
  - permite schimbarea descrierii unei componente
  - configurarea componentelor este separată de restul descrierii (entitate + arhitectură)
- poate fi ierarhizată

UNITĂŢI DE PROIECTARE

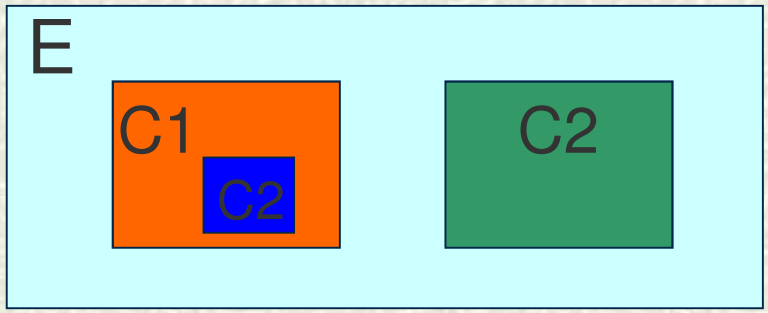
Configuraţie

- VHDL permite **configurarea incrementală** pt. descrieri
  - separat şi în locuri diferite pentru parametri generici
  - în locuri diferite pentru porturile neconectate
- **utilizare biblioteci** - cel mai simplu în momentul în care e nevoie de ele în cadrul modelului

UNITĂŢI DE PROIECTARE

Exemplu

- avem un sistem cu entitatea E şi cu arhitectura A
  - descrierea în arhitectura A este structurală şi există 2 componente C1 şi C2
  - componenta C1 are 2 arhitecturi, A1 şi A2 cu descriere structurală şi utilizează şi componenta C2
  - componenta C2 are 2 arhitecturi, A3 şi A4



UNITĂŢI DE PROIECTARE

Exemplu

```
entity C1 is
        -- generic …
        port (…);
end C1;
architecture A1 of C1 is
-- definiţii de tip
-- componente
-- constante, semnale
begin
        et 1: C2 port map …
        …
end A1;

architecture A2 of C1 is
-- definiţii de tip
-- componente
-- constante, semnale
begin
        et 1: C2 port map …
        …
end A2;
```

```
entity C2 is
                -- generic …
                port (…);
end C2;
architecture A3 of C2 is
-- definiţii de tip
-- componente
-- constante, semnale
begin
…
…
end A3;

architecture A4 of C2 is
-- definiţii de tip
-- componente
-- constante, semnale
begin
…
…
end A4;
```

```
entity E is
        -- generic …
        port (…);
end E;

architecture A of E is
-- definiţii de tip
-- componente
-- constante, semnale
begin
        eticheta 1: C1 port map …
        eticheta 2: C2 port map …
end A;
```

```
configuration CO1 of C1 is
for A2
                for et1: C2
                                use entity work.C2(A4);
                end for;
end for;
end CO1;

configuration CO of E is
for A
                for eticheta 1: C1
                                use entity work.CO1;
                end for;
                for eticheta 2: C2
                                use entity work.C2(A3);
                end for;
end for;
end CO;
```

SEMNALE

Comunicaţie

- procesul comunicării - implică **transmiterea informaţiei**: sursă - destinaţie
- **semnal** - purtător de informaţie
- în general - semnal = fenomen fizic care se poate modifica în timp şi/sau în spaţiu, iar modificările se pot specifica prin instrucţiuni formale
- semnale: electrice, mecanice, acustice, optice …

SEMNALE

Clasificarea semnalelor (şi în VHDL)

- **externe** - purtătoare de informaţie între dispozitive
  - reprezintă interfaţa
  - în VHDL se declară **numai în entitate**
- **interne** - purtătoare de informaţie în interiorul dispozitivelor
  - nu sunt vizibile
  - în VHDL se declară **numai în arhitecturi**

SEMNALE

Semnale electrice

- rol esenţial în orice dispozitiv electronic
- permit **analizarea relaţiilor temporale**
- în VHDL semnalele conţin şi informaţii prezente şi informaţii viitoare (istoria - history)
- linii de semnal:
  - singulare (de ex.: Clock) - o sigură valoare binară
  - multiple (magistrale) - combinaţie de valori binare (vectori de biţi)

SEMNALE

Semnale în VHDL

- în VHDL semnalul = corespunde reprezentării hardware a conceptului de purtător de informaţie
- **reprezentarea** = structură de date simplă sau complexă, funcţie de tipul datelor purtate de semnal
- **declaraţiile** de semnal - **în domeniul concurent** (entitate şi arhitectură)

SEMNALE

Semnale în VHDL

- acces la valori trecute, prezente şi viitoare - prin **pilot (driver)** de semnal
- se memorează evenimentele care indică o **schimbare de valoare** la un moment de timp bine definit
- pot fi **modificate numai valorile** (evenimentele) **viitoare**

SEMNALE

Semnale în VHDL

- operaţia de **atribuire a unei valori** se poate realiza:
  - prin conectare la un port de ieşire a unei componente
  - în domeniul concurent (corespunde descrierii flux de date)
  - în domeniul secvenţial

SEMNALE

Declararea semnalelor

- **semnale externe**
  - **port** - canal de comunicare dinamică între o entitate (sau un bloc) şi mediul înconjurător
  - caracteristici:
    - nume
    - mod - sensul fluxului de informaţie
    - tip
    - eventual valoare iniţială
- **semnale interne**
  - cuvânt cheie **signal**
  - fără declaraţie de mod

SEMNALE

Vizibilitatea semnalelor

- determinată de **locul declaraţiei**
- **reguli**:
  - semnal declarat în pachet - văzut de unităţile de proiectare care utilizează pachetul
  - orice port - văzut în toate arhitecturile entităţii
  - semnal declarat în zona de declaraţii a arhitecturii - văzut numai în arhitectura respectivă
  - semnal declarat într-un bloc din arhitectură - văzut doar în acel bloc

SEMNALE

Asignarea semnalelor

- instrucţiunile de asignare de valori **modifică valoarea viitoare** (modifică piloţii)
- simbolul asignării: **<=** (“primeşte”)
- **element de formă de undă** = o pereche valoare + **after** + întârziere
- valoare
  - tip compatibil cu semnalul
  - poate fi:
    - constantă
    - rezultatul unei expresii

SEMNALE

Asignarea semnalelor

- **întârzieri**
  - obligatoriu de tip Time
  - apar obligatoriu în ordine crescătoare
- Exemplu:
  - Y <= X after 10 ns, ‘1’ after 20 ns, ‘0’ after 30 ns;
- **întârziere nulă (delta)**
  - nu există dispozitive fizice care nu au timpi de propagare a semnalelor electrice (întârzieri)
  - întârziere delta - **reprezintă o cauzalitate** - este o întârziere nulă pentru simulare

SEMNALE

Asignarea semnalelor

- 2 dimensiuni ale timpului
  - **timp real**
    - văzut de proiectant
    - se măsoară în paşi de simulare
    - foloseşte unităţi de timp Time
  - **timp delta**
    - gestionat de simulator
    - în fiecare pas de simulare se alocă felii de timp infinitezimal pt. a gestiona succesiunea asignărilor
    - **exprimarea cauzalităţii** generate de succesiunea asignărilor

SEMNALE

Asignarea semnalelor

- modele de transmisie
  - inerţial
    - filtrare impulsuri mai mici decât timpul de transmisie
    - modul implicit
  - transport
    - cuvânt cheie: **transport**
    - transmiterea oricărui impuls, indiferent de durata sa

PARAMETRI GENERICI

Scop

- transmiterea unei **informaţii statice** unui bloc
- blocuri generice = blocuri parametrizate
- în interiorul blocurilor
  - văzuţi ca şi constante
  - manipulaţi ca şi constante
- pot fi utilizaţi în mod dinamic

PARAMETRI GENERICI

Blocuri generice

- entitate
  - se poate compila
  - se găseşte în bibliotecă
  - perechea entitate/arhitectură nu se poate simula
- bloc intern declarat cu instrucţiunea **block**
  - nu poate fi instanţiat din exteriorul arhitecturii în care se găseşte şi nici din interior
  - se poate instanţia doar în momentul declarării

PARAMETRI GENERICI

Parametri declaraţi generici

- dimensiuni de obiecte complexe (vectori, magistrale ...)
- iteratori pt. bucle **for**
- parametri de temporizare:
  - întârzieri
  - timp de setup
  - timp de hold
  - timpi de comutare

PARAMETRI GENERICI

Sintaxa

generic (parametru1 {, alt_parametru} : tip_ parametru
{:= valoare_implicită};

parametru2 {, alt_parametru} : tip_ parametru
{:= valoare_implicită};

.....

parametruN {, alt_parametru} : tip_ parametru
{:= valoare_implicită});

PARAMETRI GENERICI

Exemplu

```
POARTA_ŞI_NU: block
generic (nr_intrări: Natural := 3);
port (intrări: in Bit_Vector (1 to nr_intrări);
        ieşire: out Bit);
generic map (nr_intrări => 4); -- Instanţierea unei porţi ŞI-NU cu 4 intrări
begin
--Zona de instrucţiuni din cadrul blocului
process (intrări)
    variable V: Bit := '1';
    begin
    for I in 1 to nr_intrări loop
            V:= V nand intrări(I);
    end loop;
    ieşire <= V;
end process;
end POARTA_ŞI_NU;
```

CONSTANTE

Scop

- informaţie statică declarată în interiorul modelului → arhitectură
- valoare de iniţializare care nu mai poate fi modificată
- tipul valorii de iniţializare
  - identic cu cel din declaraţie
  - nu poate fi de tip acces sau fişier
- declarare de constantă cu valoare amânată - în specificaţie de pachet

OPERATORI

Clase şi priorităţi

- 7 clase, cu prioritate crescătoare de la 1 la 7:
  - 1. operatori logici: and, or, nand, nor, xor, xnor
  - 2. operatori relaţionali: =, /=, <, <=, >, >=
  - 3. operatori de deplasare: sll, srl, sla, sra, rol, ror
  - 4. operatori de adunare: +, -, &
  - 5. operatori de semn: +, -
  - 6. operatori de înmulţire: *, /, mod, rem
  - 7. operatori diverşi: **, abs, not

OPERATORI

Caracteristici

- operatorii logici
  - **predefiniţi** pentru realizarea operaţiilor logice: ŞI, SAU, ŞI-NU, SAU-NU, SAU-EXCLUSIV, COINCIDENŢĂ
  - operanzi de tip Boolean (False, True) şi Bit (‘0’, ‘1’)
  - funcţionali pe vectori de elemente de tip Boolean sau Bit, dacă au aceeaşi lungime

OPERATORI

Caracteristici

- operatorii relaţionali
  - rezultat de tip Boolean (False, True)
  - = şi /= nu sunt definiţi pt. tip fişier
  - la tipurile enumerate, primul element este considerat cel mai mic
    - exemplu: la tipul Boolean, False e mai mic decât True

OPERATORI

Caracteristici

- operatorii de deplasare - binari
  - operează pe vectori cu elemente de tip Bit sau Boolean
- operatorii de adunare - binari
  - & - definit pe vectori (tipul String = vector de caractere)
- operatorii de semn - unari
  - au prioritate mai mică decât înmulţirea → utilizarea parantezelor pt. evitarea erorilor

OPERATORI

Caracteristici

- operatori de înmulţire - binari
- operatori diverşi
  - ** - ridicare la putere
    - operandul din stânga de tip întreg sau flotant
    - puterea - obligatoriu tip întreg
  - abs - pe orice tip numeric
  - not
    - operator logic, unar
    - operează pe obiecte de tip Boolean şi Bit şi pe vectori de astfel de elemente

TIPURI DE DATE

Generalităţi

- VHDL puternic **tipizat**
  - fiecare semnal, variabilă, constantă are un tip (definit înainte de utilizare)
  - parametrii procedurilor şi funcţiilor şi rezultatul returnat de funcţii - au obligatoriu un tip
- tipizarea obiectelor - protejează instrucţiunile de atribuire
- nivel de abstractizare relativ la implementarea structurilor de date → prin asociere de reprezentare simbolică independentă de partea hardware

TIPURI DE DATE

Tipuri de date

- 4 tipuri de date:
  - scalare - valoarea constituită dintr-un element
  - compuse - valoarea constituită din mai multe elemente
  - acces (pointeri)
  - fişier

TIPURI DE DATE

Clase de obiecte - familii de tipuri

	Constante	Variabile	Semnale	Fişiere
Scalare	DA	DA	DA	NU
Compuse	DA	DA	DA	NU
Acces	NU	DA	NU	NU
Fişier	NU	NU	NU	DA

TIPURI DE DATE

Tipuri scalare

- 4 tipuri: enumerate, întregi, flotante, fizice
- ordonate (pot fi comparate)
- **interval de validitate** - restrânge valorile posibile
  - **range** expresie1 **to** expresie2;
  - **range** expresie3 **downto** expresie4;

TIPURI DE DATE

Tipuri scalare enumerate

- **type** Nume **is** (valoare_simbolică1, valoare simbolică2, ...);
- simbolurile: identificatori sau caractere
- predefinite în pachetul Standard:
  - **type** Boolean **is** (False, True);
  - **type** Bit **is** (‘0’, ‘1’);
  - **type** Severity_Level **is** (Note, Warning, Error, Failure);
  - Character - toate caracterele admise în VHDL
- poziţia - induce relaţia de ordine între elemente

TIPURI DE DATE

Tipuri scalare întregi

- sunt definiţi operatorii aritmetici
- în declaraţie se indică domeniul (**range**)
  - exemplu: **type** Nume **is range** 1 **to** 15;
- tipul **Integer** predefinit în pachetul Standard
- tipurile **Positive** şi **Natural** - subtipuri ale Integer
- conversie implicită în Universal_Integer - tip virtual

TIPURI DE DATE

Tipuri scalare flotante

- sunt definiţi operatorii aritmetici
- în declaraţie se indică domeniul (**range**)
- tipul **Real** predefinit în pachetul Standard
- intervalul domeniului - cu precizie de minimum 6 cifre după virgulă
- conversie implicită în Universal_Real - tip virtual

TIPURI DE DATE

Tipuri scalare fizice

- în VHDL - noţiune de unitate de cantitate
- caracteristici:
  - unitatea de bază
  - intervalul valorilor autorizate
  - eventual, colecţie de subunităţi cu corespondenţele lor
- sunt definiţi operatorii aritmetici
- între număr şi unitatea de măsură - spaţiu!!!
- toate valorile unui tip fizic şi toate valorile de conversie trebuie să fie numere întregi

TIPURI DE DATE

Tipuri scalare fizice

- Sintaxa

```
type nume is                                    -- definiţie_tip_fizic
range valoare_limită1 to valoare limită2        -- limita_domeniu_valori
                                  units
      declaraţie_unitate_fundamentală           -- un identificator
      { declaraţie_unitate_secundară }          -- identificator =
                                                -- multiplicator_unitate;
                                                -- multiplicator_unitate =
                                                --[numeric] nume_unitate
end units;
```

TIPURI DE DATE

Tipuri scalare fizice

- **Time** - singurul tip fizic predefinit în pachetul Standard → tip utilizat de simulator
  - unitatea de bază - femptosecunda (10^{-15} sec)
  - definit de un interval cu capetele exprimate pe 64 biţi
  - unităţi: fs, ps, ns, us, ms, sec, min, hr

TIPURI DE DATE

Tipuri compuse

- 2 tipuri compuse:
  - tablouri - colecţie de obiecte de acelaşi tip
  - articole - colecţie de obiecte de tipuri diferite

TIPURI DE DATE

Tipuri compuse

- tablouri
  - structuri omogene
  - elementele accesibile pe baza unor indecşi
  - definirea:
    - specificarea tipului
    - indicarea numărului indecşilor (tip discret)
    - specificarea tipului elementelor (fără tipul fişier)
  - vector - tablou cu un singur index

TIPURI DE DATE

Tipuri compuse

- tablouri
  - constrânse
    - intervalul de variaţie al indecşilor cunoscut cu anticipaţie
    - sensul de variaţie a indecşilor - specificat cu **to** şi **downto**
    - exemple:
      - **type** Funcţii_ALU **is** (dezactivată, adunare, scădere, înmulţire, împărţire);
      - **type** Adresă **is array** (0 **to** 15) **of** Bit;
      - **type** Word **is array** (31 **downto** 0) **of** Bit;
      - **type** Memory **is array** (Adresă) **of** Word;

TIPURI DE DATE

Tipuri compuse

- tablouri
  - neconstrânse
    - intervalul de variaţie al indecşilor cunoscut numai în timpul simulării
    - **<>** (box) amână definirea intervalului de indexare şi a direcţiei de variaţie
    - 2 tablouri neconstrânse în pachetul Standard:
      - **type** Bit_vector **is array** (Natural **range** <>) **of** Bit;
      - **type** String **is array** (Positive **range** <>) **of** Character;

TIPURI DE DATE

Tipuri compuse

- articole
  - elemente (câmpuri) diferite
  - enumerare câmpuri între **record** … **end record**
  - selectarea prin notaţia cu punct
- exemplu:

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

TIPURI DE DATE

Tipuri compuse

- indicarea valorilor - notaţia prin agregare
  - asociere poziţională
    - contează ordinea în lista elementelor
  - asociere prin denumire
    - nume => valoare
    - ordinea câmpurilor nu contează
  - asociere mixtă
    - începe cu partea poziţională
  - pentru iniţializare se foloseşte “**others**”

TIPURI DE DATE

Tipuri acces (pointeri)

- indică spre obiecte definite anterior
- pt. crearea dinamică de obiecte → alocarea dinamică a memoriei
- alocarea - **new**; dezalocarea - **deallocate**
- la declarare - iniţializare cu valoarea implicită **null**

TIPURI DE DATE

Tipuri fişier

- fişiere cu acces secvenţial
- la declarare trebuie precizat tipul de date din fişier
- 3 subprograme create automat la declararea de tip fişier: Read, Write, Endfile
- de la VHDL'93 - proceduri de deschidere şi închidere: File_Open, File_Close

TIPURI DE DATE

Conversii de tip

- conversia - **foarte restrictivă**
- posibilă în 3 cazuri:
  - tipuri cu reprezentare întreagă sau flotantă
  - pentru tablouri
    - aceleaşi dimensiuni
    - aceleaşi tipuri de elemente
    - indecşii trebuie să fie convertibili
  - conversia unui tip în el însuşi

ATRIBUTE

Generalităţi

- caracteristică asociată unui tip sau unui obiect, care poate fi cunoscută în mod dinamic, în timpul rulării
- notaţie - adăugarea unui **apostrof** după numele tipului sau obiectului
- atribute:
  - predefinite
  - definite de proiectant

ATRIBUTE

Atribute predefinite

- pot fi: valori (tipizate), funcţii, tipuri, intervale de variaţie
- se aplică unor prefixe care pot fi: valori, tipuri, etichete
- simplifică scrierea
- apar în funcţii utilitare

ATRIBUTE

Atribute predefinite

- atribute definite pe tipuri
  - T desemnează un tip (obiectul asupra căruia acţionează atributul), prefix al atributului
  - obiect'nume_atribut(parametri)

ATRIBUTE

Atribute predefinite

- atribute definite pe tipuri
  - tip sau subtip **scalar** T; X = tip scalar
    - **Atribut** — **Rezultat**

T'left	Limita la stânga a lui T
T'right	Limita la dreapta a lui T
T'low	Limita inferioară a lui T
T'high	Limita superioară a lui T
T'base	Tipul de bază a lui T
T'image(X)	Şirul X
T'value(X)	Valoare de tip T

ATRIBUTE

Atribute predefinite

- atribute definite pe tipuri
  - tip sau subtip **discret** sau **fizic** T; X membru al lui T; N număr întreg
    - **Atribut** **Rezultat**

T'pos(X)	Numărul poziţiei lui X în T
T'val(N)	Valoarea la poziţia N în T
T'leftof(X)	Valoarea în T, cu o poziţie în stânga lui X
T'rightof(X)	Valoarea în T, cu o poziţie în dreapta lui X
T'pred(X)	Valoarea în T, cu o poziţie mai mică decât X
T'succ(X)	Valoarea în T, cu o poziţie mai mare decât X
T'ascending	Valoare booleană pt. interval crescător sau descrescător

ATRIBUTE

Atribute predefinite

- atribute definite pe tipuri sau subtipuri **tablou**
  - A = tablou; N = număr întreg între 1 şi numărul dimensiunilor lui A
    - **Atribut** **Rezultat**

A'left(N)	Limita stânga a domeniului indicelui dimensiunii N a lui A
A'right(N)	Limita dreapta a domeniului indicelui dimensiunii N a lui A
A'low(N)	Limita inferioară a domeniului indicelui dimensiunii N a lui A
A'high(N)	Limita superioară a domeniului indicelui dimensiunii N a lui A
A'range(N)	Domeniul indicelui dimensiunii N a lui A
A'reverse_range(N)	Inversul domeniului indicelui dimensiunii N a lui A
A'length(N)	Lungimea domeniului indicelui dimensiunii N a lui A
A'ascending(N)	Valoare booleană pentru direcţia indicelui dimensiunii N a lui A

ATRIBUTE

Atribute predefinite

- atribute definite pe **semnale** S
  - atribute semnal
    - **Atribut** **Rezultat**

S'delayed(T)	Semnal S întârziat cu T unităţi de timp
S'stable(T)	Valoare booleană True dacă S e fără evenimente în T
S'quiet(T)	Valoare booleană True dacă S e inactiv în T
S'transaction	Modificare valoare Bit de câte ori S este activ

ATRIBUTE

Atribute predefinite

- atribute definite pe **semnale** S
  - atribute funcţie
    - **Atribut** **Rezultat**

S'event	Valoare booleană True pt. eveniment pe S
S'active	Valoare booleană True dacă S e activ
S'last_event	Timpul trecut de la ultimul eveniment pe S (valoare Time)
S'last_active	Timpul trecut de la ultima activare a lui S (valoare Time)
S'last_value	S imediat înainte de ultima modificare
S'driving_value	Permite o operaţie de atribuire
S'driving	Valoare booleană dacă S nu este deconectat

ATRIBUTE

Atribute predefinite

- atribute definite pe **obiecte** în sens larg X
  - utilizate pentru elaborare de mesaje
    - **Atribut** **Rezultat**

X'simple_name	Numele X
X'path_name	Numele X şi etichetele de revenire la X
X'instance_name	Numele X, etichetele de revenire la X, informaţii de configurare

ATRIBUTE

Atribute definite de utilizator

- declarare atribut
  - **attribute** nume_atribut: tip_atribut;
- specificare atribut (primeşte valoare)
  - **attribute** nume_atribut **of** obiect **is** expresie;
  - nu pot fi decât constante, deci sunt statice
- utilizare atribut - cu notaţia cu apostrof
  - obiect'nume_atribut

ATRIBUTE

Atribute definite de utilizator

- se pot raporta la:
  - entităţi, arhitecturi, configuraţii, pachete, proceduri, funcţii, tipuri, subtipuri, constante, semnale, variabile, componente, etichete
- raportarea se poate face:
  - la anumite elemente, care trebuie numite în câmpul “obiect”
    - nume_element, {nume_element}: clasă_element
    - pentru toate celelalte elemente ale unei clase:
      - **others**: clasă_element
  - la toată clasa de elemente
    - **all**.clasă_elemente