Limbajul VHDL
Partea 1 – Structura si elemente de baza

S.l. Ing. Vlad-Cristian Miclea

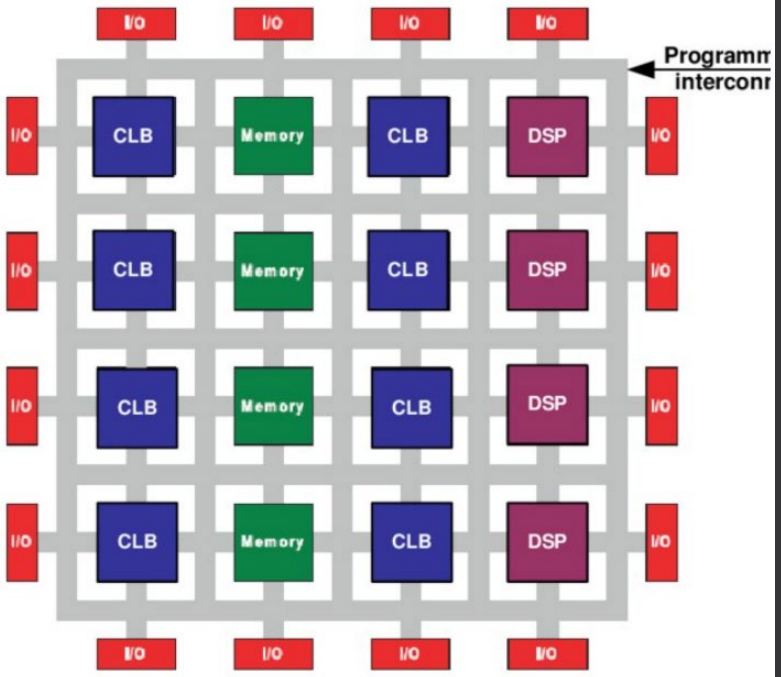
Universitatea Tehnica din Cluj-Napoca

Departamentul Calculatoare

CUPRINS

Saptamana trecuta

- ASIC
  - Procesoare
  - Microcontrollere
  - GPU
- FPGA
  - Field Programmable Gate Array
  - Circuite reconfigurabile
- Structura unui FPGA
  - CLB
  - LUT
  - Interconexiuni
- Generarea circuitului
  - Sintetizare
  - Translatare
  - Mapare
  - Plasare
  - Rutare



VHDL - INTRODUCERE

VHDL

- VHSIC - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
- HDL - **H**ardware **D**escription **L**anguage
- Prima varianta - 1980; standard 1987; extins 1993; variantă 2004; acum **1076/2008**

Cum am putea descrie "algoritmic" circuite hardware?

- Descriere folosind diagrame bloc (exemplu sem 1)
  - Probleme la design-uri mari (ex. Conexiuni)
  - Anumite structuri – repetitive, pot fi descrise mai usor "algoritmic"
  - Exemplu: MUX (simplu if); counter (incrementare)
- Limbaje de programare
  - Permit doar excutia secventiala a unor instructiuni
    - Can "se termina" executia lor, nu mai exista
    - De obicei se executa pe un circuit fizic/procesor
  - Avem nevoie de o descriere care sa "creeze" circuitul

INTRODUCERE VHDL

HDL: limbaj de descriere a sistemelor electronice hardware

- Contine:
  - structură de blocuri
  - relaţii
  - interconexiuni
- VHDL definit şi integrat în instrumentele CAD (Computer-Aided Design)
- toate instrumentele CAE (Computer-Aided Engineering) - produse cu intrări / ieşiri
- Usor de interpretat si “citit” pentru a fi implementat pt circuite fizice

Atentie: INTOTDEAUNA TREBUIE PRIVIT CA UN LIMBAJ DE DESCRIERE HARDWARE, NU UN LIMBAJ DE PROGRAMARE!!!

- Limbaj de programare – instructiuni – cod masina – ruleaza pe un processor/circuit existent
- Limbaj de descriere – instructiuni – genereaza un nou circuit hardware!

DOMENII DE APLICARE

Obiective VHDL

- **Specificare** sisteme hardware
  - Descrierea functionalitatii/structurii
  - Pregatire pentru etapele de sintetizare + implementare
- **Simulare** evoluţie temporală a descrierilor
  - Instrumentele de simulare – tot structuri VHDL
  - Realizează simularea (“execuţia”) codului VHDL în paralel
  - Codul nu descrie modul de proiectare sau de realizare a funcţiei, ci doar ce trebuie să facă aceasta

DOMENII DE APLICARE

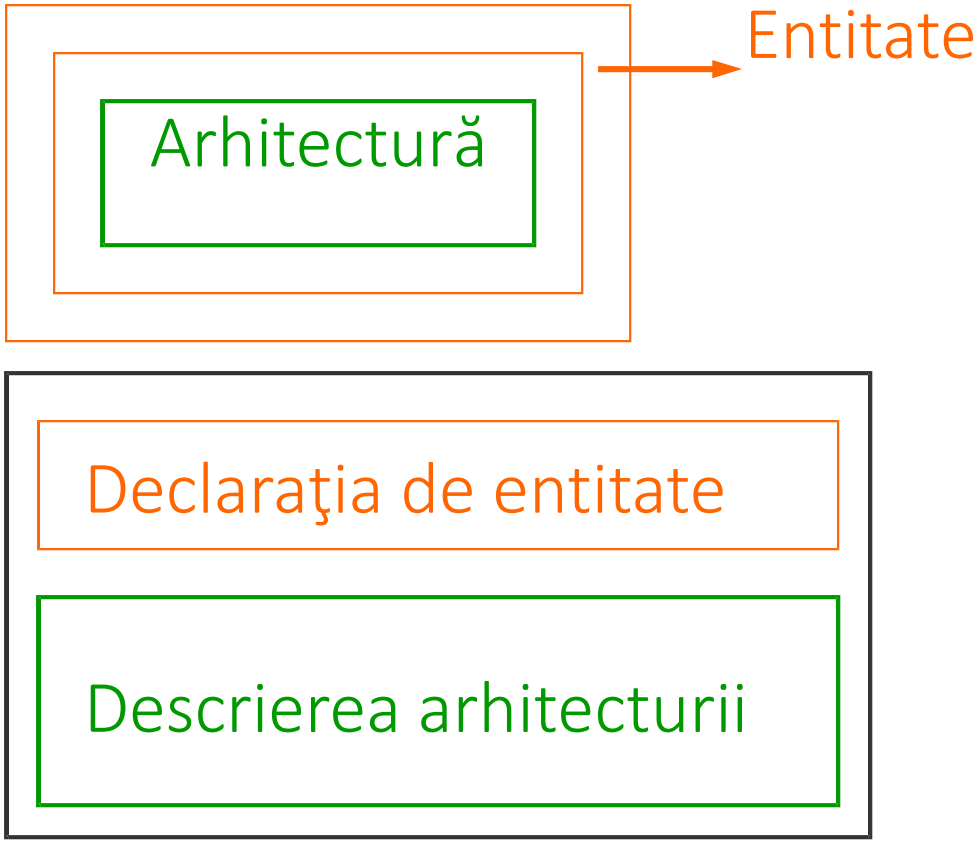
Obiective VHDL (continuare)

- **Sinteza logică** în cadrul instrumentelor CAD care integrează VHDL (fază automatizată)
  - Descrierea proiectării unui system
    - Descrierea funcţionării
    - Descrierea structurii exacte a fiecărei părţi
  - Descrierea realizării finale - interconexiuni de componente logice elementare
  - Porneşte de la **o descriere VHDL sintetizabilă**
  - Conduce la o **schemă logică clasică** (porţi logice + bistabile)

STRUCTURA VHDL

Proiectare ierarhică

- model VHDL: pereche entitate + arhitectură



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 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 SI cu 2 intrări

```
entity and2 is
        port (a, b: in bit; y: out bit);
end and2;
```

- Poartă logică de tip SI – 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/algoritmica
  - **flux de date** = descriere de interconexiuni
  - **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

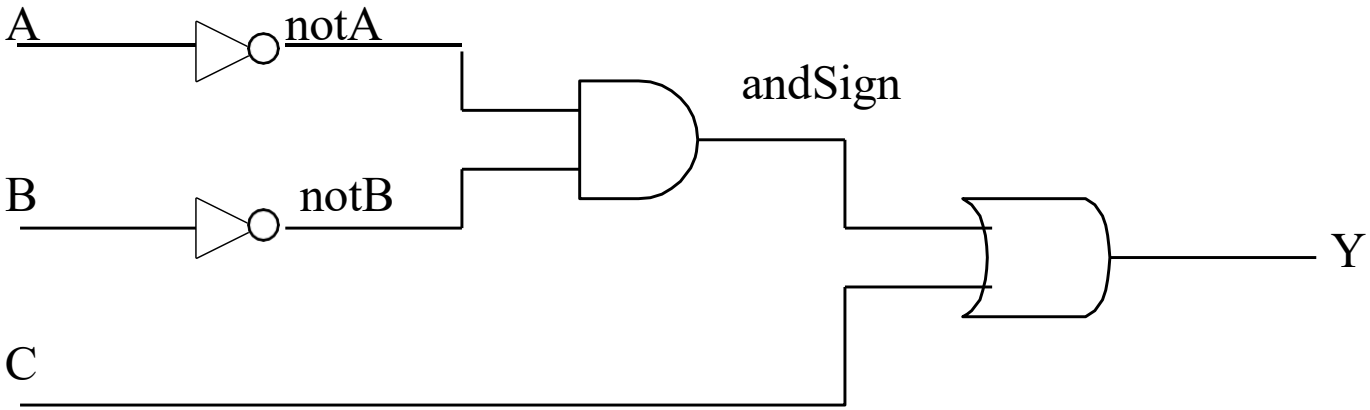
Arhitectură

- **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**

UNITĂŢI DE PROIECTARE

Exemple de arhitecturi

- Descrieri pentru circuitul din figură:




```
library IEEE;
use IEEE.std_logic_1164.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

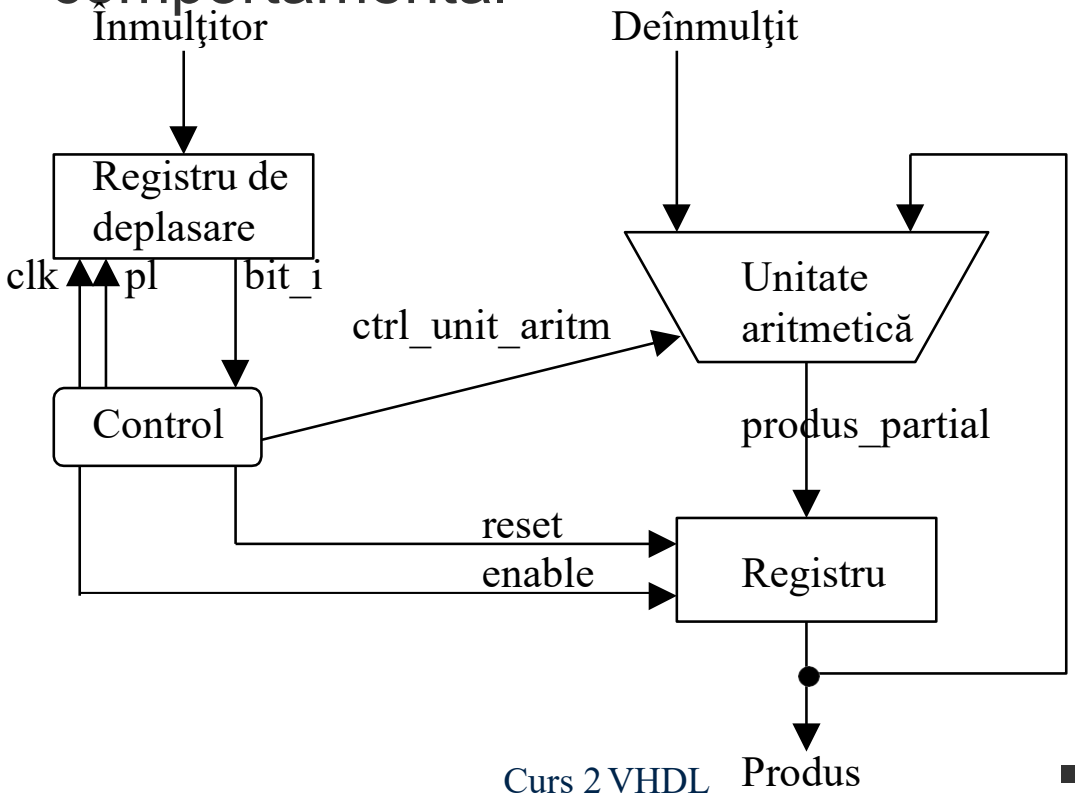
- Arhitectură **flux de date**

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

UNITĂŢI DE PROIECTARE

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;
```

UNITĂŢI DE PROIECTARE

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;
```

UNITĂŢI DE PROIECTARE

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

UNITĂŢI DE PROIECTARE

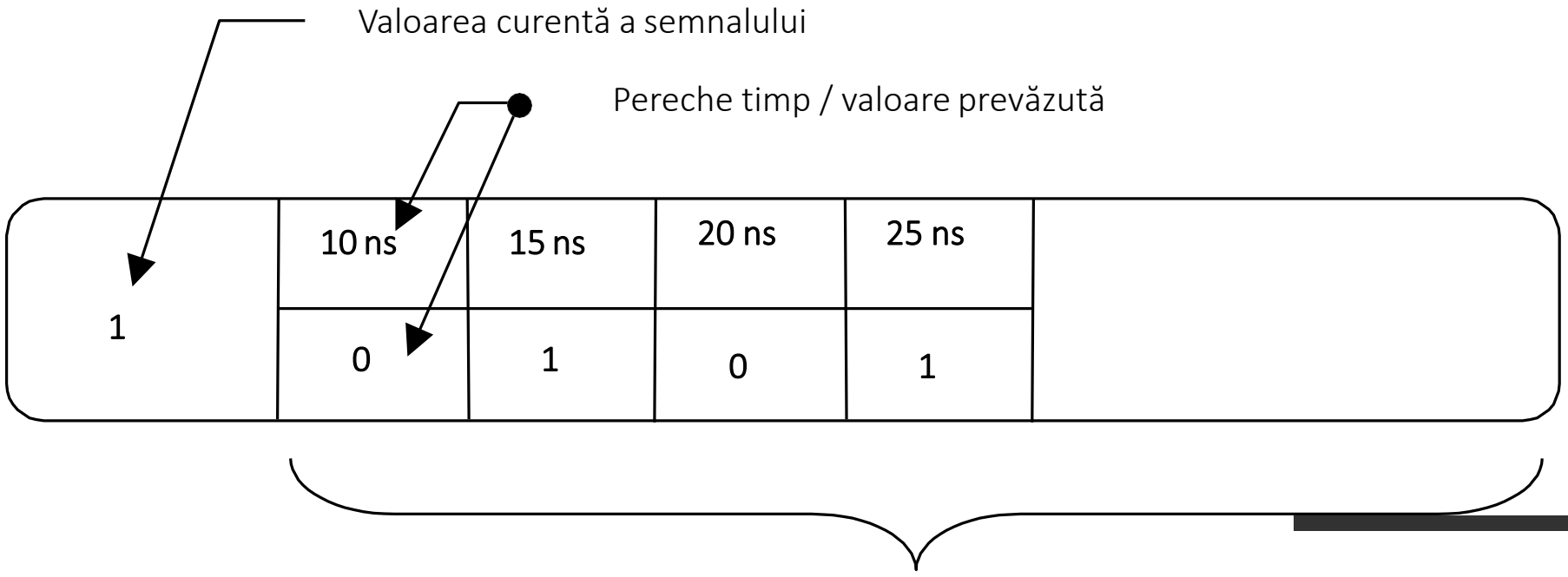
Exemplu de corp de pachet

- 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;
```

OBIECTE IN VHDL

- Constante si variabile (mai putin utilizate)
- **Semnale – s**pecifice sistemelor hardware
  - Modelează informaţia care tranzitează între componente (legătură fizică prin fire)
  - Există tot timpul Pot modela intrari/iesiri sau informatii intermediare
  - Pilot (driver) de semnal:



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**
  - fara declaratie 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 intarziere nula pt simulare

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

Concluzii

- Limbajul VHDL
  - Generalitati
  - Domenii de aplicare – specificare, simulare, sinteza
- Structura unui program VHDL
  - Structura ierarhica
  - Entitate
  - Arhitectura
  - Structurala vs comportamentala vs flux-de-date
- Obiecte in VHDL
- Semnale
- Parametrii generic, constante

Multumesc pt atentie!