PROIECTARE SISTEMELOR DIGITALE

Masterat ICCP - an 1

LABORATOR 1

NOȚIUNI INTRODUCTIVE DESPRE LIMBAJUL VHDL

Inițialele **VHDL** provin de la *VHSIC Hardware Description Language*, iar **VHSIC** înseamnă *Very High Speed Integrated Circuits*, şi a fost un program dezvoltat în anii 1980 de către Departamentul Apărării al Statelor Unite, cu scopul de a se realize progrese semnificative în priectarea şi dezvoltarea de circuite integrate pe scară largă şi foarte largă.

La programul VHSIC au luat parte un număr mare de companii, care utilizau diverse formalisme (de obicei limbaje sau unelte proprii) pentru descrierea, simularea sau sinteza circuitelor integrate. Au apărut astfel probleme legate de schimbul de informații între formele participante la proiect. Soluția propusă a fost dezvoltarea unui limbaj de descriere hardware (HDL - Hardware Description Language) care să permită:

- descrierea unor proiecte sau părti ale unor proiecte într-o formă standardizată
- simularea circuitelor la diverse niveluri de descriere
- sinteza automată a circuitelor descrise în limbajul respectiv.

Astfel s-a dezvoltat limbajul VHDL, care a fost adoptat ca standard IEEE în anul 1987, devenind standardul IEEE Std. 1076-1987. Standardul (limbajul) a fost re-actualizat în anul 1993: IEEE Std. 1076-1993.

Descrierea (reprezentarea) unui sistem digital poate fi facută la diferite niveluri de abstractizare şi în domenii (planuri) distincte (domeniul comportamental, domeniul structural si domeniul fizic). Nivelurile de descriere ale unui sistem sau circuit sunt, în ordine crescatoare a gradului de abstractizare:

1. **circuit**: în domeniul comportamental structural, elementele componente sunt tranzistoare, rezistoare şi capacitoare, iar în domeniul comportamental se utilizează ecuații diferențiale pentru a descrie funcționarea acestora.
2. **logic**: componentele structurale sunt reprezentate de porți logice şi bistabile, iar în planul comportamental, funcționarea sistemului este descrisă de ecuații logice booleene.
3. **nivelul transferului între regiştri** (*Register Transfer* sau *RT-level*): în plan comportamental funcționarea e descrisă de operațiile de transfer între regiştri, iar în plan structural, elementele sunt: registre, ALU (unități aritmetice şi logice), multiplexoare, etc.
4. **nivelul sistem**: este cel mai abstract nivel de reprezentare. Funcționarea e descrisă prin algoritmi sau procese (de exemplu procese VHDL), iar elementele structurale sunt unități centrale (CPU), memorii, magistrale.

1.1 Unitatea de design

Un circuit sau sistem poate fi modelat în VHDL sub forma unui set de elemente de proiectare (**design units**). Elementele de proiectare existente în VHDL sunt următoarele:

1. **entity declaration** - declarația entității sau pe scurt, entitatea;
2. **architecture body** - corpul arhitecturii sau arhitectura;

3. **package declaration**;
4. **package body** - corpul package-ului;
5. **configuration declaration** – configurația;

1.1.1 Entități (Entity declaration)

O declarație de entitate specifică numele entității şi interfața acesteia cu exteriorul, reprezentată prin porturile sale.

Sintaxa unei declarații de entitate este:

entity_declaration::=
ENTITY entity_name IS
entity_header
[entity_declarative_part]
[BEGIN
entity_statement_part]
END [ENTITY][entity_name];

entity_header::=
[PORT_clause][GENERIC_clause]

In care semnificația simbolurilor este:

- ::= – se defineşte
- [] – secvență opțională
- {} – secvență opțională repetabilă
- | – alternative logice

Limbajul VHDL nu este "case-sensitive", adică nu face deosebirea între litere mari şi mici, dar pe parcursul acestui laborator vom scrie cu majuscule cuvintele cheie ale limbajului. Deci, o declarație de entitate conține cuvântul cheie ENTITY şi numele entității, urmată de un header şi, în mod opțional, de o parte de declarații şi de una de instrucțiuni, tot opțională.
Header-ul entității conține clauza PORT, prin care se declară porturile entității (semnalele prin care aceasta comunică cu mediul exterior) şi clauza GENERIC. Parametrii generici aparțin clasei constantelor şi sunt vizibili în toate arhitecturile care aparțin unei entități. Un exemplu tipic de utilizare a lor este pentru a specifica întârzierile care apar pe circuit, între intrările şi ieşirile sale.

Declarația de entitate se încheie cu cuvântul cheie END urmat eventual de cuvântul cheie ENTITY sau/şi de numele entității. Simbolul ; de la sfârşitul declarației este obligatoriu.

1.2.2 Arhitecturi (Architecture body)

Declarația de entitate nu precizează nimic despre implementarea entității. Acest lucru se face utilizând arhitecturile. O entitate poate avea oricâte arhitecturi, de exemplu o arhitectură care o descrie din punct de vedere comportamental şi o altă arhitectură care o descrie din punct de vedere structural.

Sintaxa:

architecture_body::=
ARCHITECTURE architecture_name OF entity_name IS
architecture declarative part

BEGIN
architecture statement part
END [ARCHITECTURE][architecture_name];

După cuvântul cheie ARCHITECTURE urmează numele arhitecturii şi entitatea căreia îi este asociată, iar apoi o parte de declarații.
Corpul arhitecturii se află între BEGIN si END şi conține instrucțiuni concurente, şi anume:

- instrucțiuni PROCESS;
- instrucțiuni BLOCK;
- apeluri concurente de proceduri;
- instrucțiuni ASSERT concurente;
- asignări concurente de semnale;
- instanțieri de componente;
- instrucțiuni GENERATE;

Observație: În VHDL o arhitectură poate conține orice combinație de descrieri structurale şi comportamentale, dar pentru claritate în exemplele următoare vom face distincție între stilul comportamental (concurent sau secvențial) şi cel structural.

Exemplu: vom descrie un circuit generator de paritate în trei moduri: **structural**, **comportamental concurent** şi **comportamental secvențial**. Circuitul are 4 intrări de tip BIT (un tip enumerare predefinit în limbajul VHDL, având valorile '0' şi '1', corespunzător celor două valori din logica booleană) şi o ieşire, care va avea valoarea '1' daca un număr par de intrări au valoarea '1', respectiv va avea valoarea '0' daca un număr impar de intrări au valoarea '1'.

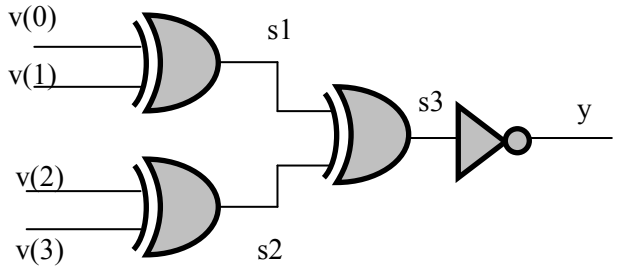


Figura 1.1 Schema circuitului generator de paritate

```
-- fisierul porti.vhd
-------------------------------
ENTITY poarta_xor IS
        GENERIC(delay: TIME:=1ns);
                PORT(x1,x2: IN BIT;
                        y: OUT BIT);
END poarta_xor;
---------------------------------
ARCHITECTURE behave OF poarta_xor IS
        -- simbolul pentru comentariu !
        -- declaratii de semnale, etc,
        -- NU se pot declara variabile in arhitectura
BEGIN
```

```
        y <= x1 XOR x2 AFTER delay;
END behave;
---------------------------------
ENTITY inversor IS
                PORT(x: IN BIT;
                        y: OUT BIT);
END inversor;
---------------------------------
ARCHITECTURE behave OF inversor IS
BEGIN
        y <= NOT x;
END behave;
-----------------------------------
```

Primul fişier conţine descrierile celor două tipuri de porţi, **sau-exclusiv** şi **inversor**, care vor fi utilizate în descrierea structurală a generatorului de paritate. Cele două entităţi conţin câte o clauză GENERIC (folosită pentru a declara parametrul generic *del*, de tip TIME, ce reprezintă timpul de propagare al fiecărei porţi), precum şi o clauză PORT, care specifică porturile fiecărei entităţi. Fiecare din arhitecturi conţine o singură instrucţiune, şi anume o asignare concurentă de semnal.

Entitatea *circ_paritate* are o ieşire *y*, pe un bit şi o intrare *v*, care este un vector de 4 biţi (BIT VECTOR care este un tip predefinit în VHDL şi reprezintă un tablou de biţi).

Arhitectura *struct* prezintă o descriere structurală a circuitului. În partea de declaraţii a arhitecturii se declară componentele care vor fi utilizate, adică o poartă de tip sau-exclusiv, componenta *xor_gate* şi o componentă inversor, *inv_gate*. Se observă că numele componentelor diferă de numele entităţilor, *poarta_ xor* şi *inversor*.

Numele porturilor sunt la fel ca la entităţi, dar în general pot să difere. De asemenea se mai declară 3 semnale interne, *s1*, *s2* şi *s3*, care sunt folosite în corpul arhitecturii, pentru interconectarea componentelor.

Corpul arhitecturii *struct* conţine doar instrucţiuni de instanţiere de componente, şi anume 3 instanţieri ale porţii sau-exclusiv şi o instanţiere a inversorului. Fiecare instrucţiune de instanţiere de componente are o etichetă (*label*), care este obligatorie, urmată de precizarea tipului componentei care este instanţiată şi de maparea genericelor şi a porturilor.

Dacă se face şi maparea genericelor, atunci pentru instanţa respectivă a componentei, valoarea parametrului generic va fi cea precizată prin GENERIC MAP, iar dacă GENERIC MAP lipseşte, atunci se păstrează valoarea de la declararea parametrului generic.

Maparea porturilor constă în a stabili corespondenţa între portul formal din declaraţia de componentă şi portul actual, care este semnalul care se leagă la portul respectiv.

Portul actual poate fi sau un semnal intern sau un port al entităţii care este descrisă (*circ_paritate* în cazul nostru). Dacă portul formal este un port al entităţii, atunci există reguli care precizează ce fel de port actual îi corespunde unui port formal (trebuie să corespundă tipul lor şi direcţia : IN, OUT, INOUT, etc). Nu se poate lega un port formal OUT la un port actual IN (sau un IN la un OUT).

Maparea porturilor se poate face după **nume**, ca în cazul instrucţiunilor cu etichetele *xor1* şi *inv1* sau **poziţional**, cum se întâmplă la instrucţiunile cu etichetele *xor2* şi *xor3*. La maparea după nume se precizează numele portului formal, simbolul =>, urmat de numele portului actual, iar ordinea în care se mapează porturile nu contează. La maparea poziţională nu mai apar numele porturilor formale, ci pe poziţia corespunzătoare portului formal din declaraţia componentei se trece numele portului actual care se mapează la acel port actual.

Observații:

1. Este important să nu se confunde simbolul =>, folosit la maparea după nume, cu simbolul <=, utilizat la instrucțiunile de asignare de semnal.
2. Direcția simbolului => este **întotdeauna** de la portul formal spre portul actual, **indiferent** de direcția porturilor (IN, OUT sau INOUT).
3. Maparea poziţională poate fi o sursă de erori foarte greu de detectat !! Compilatorul nu semnalează eroare dacă unui port formal i se asociază un alt port decât cel dorit, dar având modul (direcția) şi tipul compatibile cu portul actual (de exemplu, dacă la intrarea de *reset* a unui circuit se mapează semnalul extern de *clock*).

La descrierea structurală se foloseşte şi o **configurație** (*configuration declaration*), care face legătura între componentele declarate în arhitectura *struct* şi perechile entitate - arhitectură asociate componentelor respective. În exemplul nostru, componentei *xor_gate* i se asociază entitatea *poarta_xor* cu arhitectura *behave*, iar componentei *inv_gate* i se asociază entitatea inversor cu arhitectura *behave*. Se observă că nu contează ordinea în care apar instrucțiunile de instanțiere de componente.

```
----------------------------------------
ENTITY circ_paritate IS
        PORT(v: IN BIT_VECTOR(3 DOWNTO 0);
                y: OUT BIT);
END circ_paritate;
----------------------------------------

ARCHITECTURE struct OF circ_paritate IS
        COMPONENT xor_gate IS
                GENERIC(delay: TIME:=1ns);
                PORT(x1,x2: IN BIT;
                        y: OUT BIT);
        END COMPONENT;
        COMPONENT inv_gate IS
                PORT(x: IN BIT; y: OUT BIT);
        END COMPONENT;
FOR ALL: xor_gate USE ENTITY WORK.poarta_xor(behave);
FOR inv1: inv_gate USE ENTITY WORK.inversor(behave);

SIGNAL s1, s2, s3: BIT;
        BEGIN
-- mapare dupa nume
xor1: xor_gate PORT MAP(y => s1, x2=> v(1), x1=> v(0));

--mapare pozitionala
xor2: xor_gate PORT MAP(v(2), v(3), s2);



-- maparea genericelor
xor3: xor_gate GENERIC MAP(delay => 1ns) PORT MAP(s1, s2, s3);
```

```
inv1: inv_gate PORT MAP(x=>s3, y=>y);
END ARCHITECTURE struct;
-----------------------------------
```

Arhitectura *comport_concurent1* foloseşte 4 instrucţiuni de asignare concurentă de semnale şi se observă că se poate stabili o corespondență între ecuațiile logice din această arhitectură şi porțile logice din arhitectura *struct*. Deoarece cele 4 instrucţiuni de asignare de semnal sunt concurente, nu contează ordinea în care sunt scrise.

```
-----------------------------------
ARCHITECTURE comport_concurent1 OF circ_paritate IS
        SIGNAL t1, t2, t3: BIT;
        BEGIN
                y<=NOT t3;
                t3<=t2 XOR t1;
                t2<=v(2) XOR v(3);
                t1<=v(0) XOR v(1);
END;
-----------------------------------
```

Arhitectura *comport_concurent2* foloseşte o singură asignare concurentă de semnal, scriind într-o formă mai condensată ecuațiile logice din arhitectura *comport_concurent1*.

```
-----------------------------------
ARCHITECTURE comport_concurent2 OF circ_paritate IS
        BEGIN
                y<= NOT(v(0) XOR v(1) XOR v(2) XOR v(3));
END ARCHITECTURE;
-----------------------------------
```

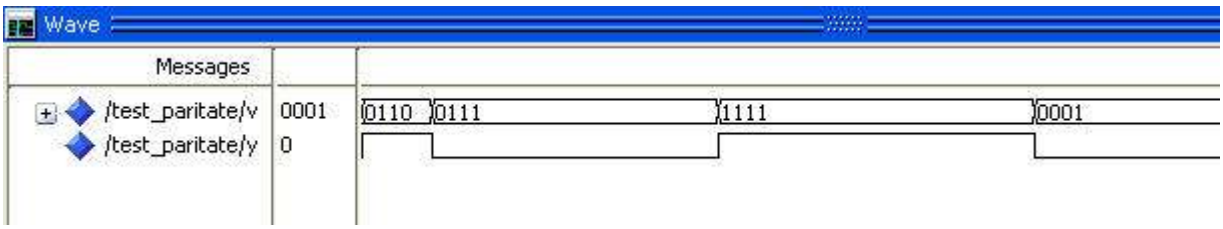


Figura 1.2 Simularea circuitului generator de paritate

1.5 TEME PROPUSE

1.5.1 Să se realizeze o descriere structurală a circuitului din *figura 1.3* ce reprezintă structura internă a unui comparator numeric pe 1 bit.

A	B	mic	egal	mare
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$A < B = \overline{A} \cdot B$$
$$A = B = \overline{A} \cdot \overline{B} + A \cdot B = \overline{A \oplus B}$$
$$A > B = A \cdot \overline{B}$$

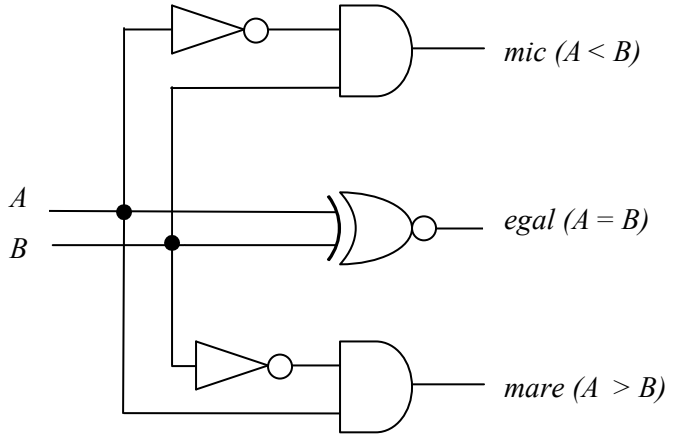
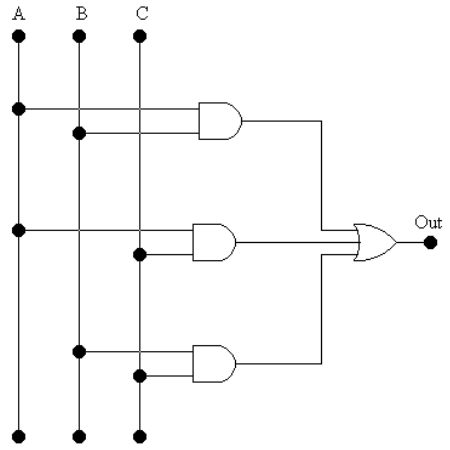


Figura 1.3 Tabel de adevăr, ecuații de funcționare şi schema logică a unui comparator cu două intrări

1.5.2 Să se implementeze la nivel structural, funcția de vot majoritar a cărei schemă logică este prezentată în *figura 1.4*.



A	B	C	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figura 1.4 Schema funcției de vot majoritar şi tabela de adevăr

1.5.3 Să se realizeze o descriere structurală a circuitului din *figura 1.5* , ce reprezintă structura internă a unui decodificator de 2 biți.

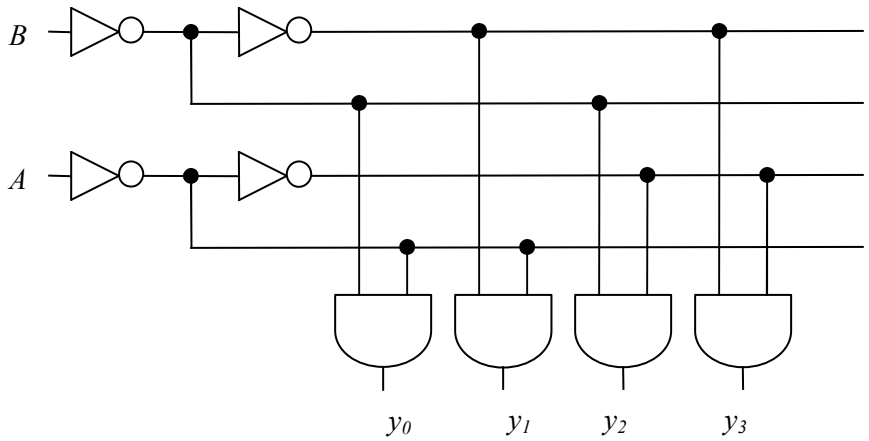


Figura 1.5 Decodificatorul cu două intrări pe 1 bit

A	B	y0	y1	y2	y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figura 1.6 Tabela de adevăr a decodificatorului cu două intrări pe 1 bit