PROIECTAREA SISTEMELOR DIGITALE

Masterat ICCP - an 1

LABORATOR 4

INSTRUCȚIUNI SECVENȚIALE ÎN LIMBAJUL VHDL
Partea a 2-a

In aceasta lucrare de laborator sunt prezentate o serie de instrucțiuni secvențiale care pot apare într-un proces sau subprogram: instrucțiunea **case**, instrucțiunile de buclare (**loop, while loop, for loop, next, exit**) şi instrucțiunea **assert** secvențială.

4.1. Instrucțiuni secvențiale

4.1.1. Instrucțiunea case

Ca şi instrucțiunea **if**, instrucțiunea **case** selectează pentru execuție una din mai multe secvențe alternative de instrucțiuni pe baza valorii unei expresii. Spre deosebire de instrucțiunea **if**, în cazul instrucțiunii **case** expresia nu trebuie să fie booleană, ci poate fi reprezentată de un semnal, o variabilă sau o expresie de orice tip discret (un tip enumerat sau întreg) sau un tablou unidimensional de caractere (inclusiv **bit_vector** sau **std_logic_vector**). Instrucțiunea **case** se utilizează atunci când există un număr mare de alternative posibile.

Această instrucțiune este mai lizibilă decât o instrucțiune **if** cu un număr mare de ramuri, permițând identificarea uşoară a unei valori şi a secvenței de instrucțiuni asociate.

Sintaxa instrucțiunii case este următoarea:

case *expresie* **is**
    when *opțiuni_1* =>
        secvență_de_instrucțiuni
        ...
    when *opțiuni_n* =>
        secvență_de_instrucțiuni
    [**when others** =>
        secvență_de_instrucțiuni]
end case;

Instrucțiunea **case** conține mai multe clauze **when**, fiecare specificând una sau mai multe opțiuni. Opțiunile reprezintă fie o valoare individuală, fie un set de valori cu care se compară expresia instrucțiunii **case**. În cazul în care expresia este egală cu valoarea individuală sau cu una din valorile setului, se execută secvența de instrucțiuni specificată după simbolul =>. O secvență de instrucțiuni poate fi formată şi din instrucțiunea nulă, **null**. Spre deosebire de anumite limbaje de programare, instrucțiunile dintr-o secvență nu trebuie incluse între cuvintele cheie **begin** şi **end**. Clauza **others** se poate utiliza pentru a specifica execuția unei secvențe de instrucțiuni în cazul în care valoarea expresiei nu este egală cu nici una din valorile specificate în clauzele **when**.

În cazul în care o opțiune este reprezentată de un set de valori, se pot specifica fie valorile individuale din set, separate prin simbolul "|" având semnificația "sau", fie domeniul valorilor, fie o combinație a acestora, după cum se arată în exemplul următor.

```
case expresie is
        when val =>
                secvență_de_instrucțiuni
        when val1 | val2 | ... | valn =>
                secvență_de_instrucțiuni
        when val3 to val4 =>
                secvență_de_instrucțiuni
        when val5 to val6 | val7 to val8 =>
                secvență_de_instrucțiuni
                ...
        when others =>
                secvență_de_instrucțiuni
        end case;
```

Observații

- Într-o instrucțiune **case** trebuie enumerate toate valorile posibile pe care le poate avea expresia de selecție, ținând cont de tipul sau subtipul acesteia. De exemplu, dacă expresia de selecție este de tip **std_logic_vector** de 2 biți, trebuie enumerate 81 de valori, deoarece pentru un singur bit pot exista 9 valori posibile. Dacă nu sunt enumerate toate valorile, trebuie utilizată clauza **others**.
- Clauza **others** trebuie să fie ultima dintre toate opțiunile.

Exemplul 1 prezintă un proces pentru secvențierea valorilor unui tip enumerat reprezentând stările unui semafor. Procesul este descris cu ajutorul unei instrucțiuni **case**.

Exemplul 1:

```
type tip_culoare is (rosu, galben, verde);
process (culoare)
        case culoare is
                when rosu =>
                        culoare_urm <= verde;
                when galben =>
                        culoare_urm <= rosu;
                when verde =>
                        culoare_urm <= galben;
        end case;
end process;
```

Exemplul 2 definește o entitate şi o arhitectură pentru o poartă SAU EXCLUSIV cu două intrări. Poarta este descrisă în mod funcțional cu ajutorul unui proces care conține o instrucțiune **case**.

Exemplul 2:

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
        port (a, b: in std_logic;
                x: out std_logic);
```

```
end xor2;
architecture arh_xor2 of xor2 is
begin
process (a, b)
variable temp: std_logic_vector (1 downto 0);
begin
temp := a & b;
        case temp is
                when "00" => x <= '0';
                when "01" => x <= '1';
                when "10" => x <= '1';
                when "11" => x <= '0';
                when others => x <= '0';
        end case;
end process;
end arh_xor2;
```

4.1.2. Instrucțiuni de buclare

Instrucțiunile de buclare permit execuția repetată a unei secvențe de instrucțiuni, de exemplu, prelucrarea fiecărui element al unui tablou în acelaşi mod. În limbajul VHDL, există trei tipuri de instrucțiuni de buclare: instrucțiunea de buclare simplă **loop**, instrucțiunea **while loop** şi instrucțiunea **for loop**.

Instrucțiunea de buclare simplă **loop** specifică repetarea unor instrucțiuni în mod nedefinit. În cazul instrucțiunii **while loop**, instrucțiunile care formează corpul buclei sunt repetate cât timp condiția specificată este adevărată, iar în cazul instrucțiunii **for loop** instrucțiunile din corpul buclei sunt repetate de un număr de ori specificat de un contor.

Observație

- Singura instrucțiune de buclare care poate fi utilizată pentru sinteza logică este instrucțiunea **for loop**, deoarece la aceasta numărul de iterații este fix.

4.1.2.1. Instrucțiunea loop

Instrucțiunea de buclare simplă **loop** are sintaxa următoare:

[*etichetă*:] **loop**
    secvență_de_instrucțiuni
end loop [*etichetă*];

Instrucțiunea are o etichetă opțională, care poate fi utilizată pentru identificarea instrucțiunii. Instrucțiunea **loop** are ca efect repetarea instrucțiunilor din corpul buclei de un număr nelimitat de ori. În cazul acestei instrucțiuni, singura posibilitate de terminare a execuției este utilizarea unei instrucțiuni **exit**.

4.1.2.2. Instrucțiunea while loop

Instrucțiunea **while loop** este o instrucțiune de buclare condiționată. Sintaxa acestei instrucțiuni este următoarea:

[*etichetă*:] **while** *condiție* **loop**
    secvență_de_instrucțiuni

end loop [*etichetă*];

Condiția este testată înaintea fiecărei execuții a buclei. Dacă această condiție este adevărată, se execută secvența de instrucțiuni din corpul buclei, după care controlul este transferat la începutul buclei. Execuția buclei se termină atunci când condiția testată devine falsă, caz în care se execută instrucțiunea care urmează după clauza **end loop**.

Exemplul 3:

```
process
        variable contor: integer := 0;
begin
        wait until clk = '1';
        while nivel = '1' loop
                contor := contor + 1;
        wait until clk = '0';
        end loop;
end process;
```

În procesul din Exemplul 3 se numără fronturile crescătoare ale semnalului de ceas *clk* atât timp cât semnalul nivel are valoarea '1'. Stabilitatea semnalului nivel nu este testată. Se testează doar dacă acest semnal are valoarea '1' la detectarea unui front crescător al semnalului *clk*, şi nu se testează dacă valoarea semnalului nivel s-a modificat de la testarea anterioară.

Corpul buclei unei instrucțiuni **while loop** poate conține o altă instrucțiune de buclare, în particular o instrucțiune **while loop**, după cum se arată în exemplul următor.

```
E1: while i < 10 loop
E2: while j < 20 loop
...
end loop E2;
end loop E1;
```

4.1.2.3. Instrucțiunea for loop

Pentru execuția repetată a unei secvențe de instrucțiuni de un număr de ori specificat se poate utiliza instrucțiunea **for loop**. Sintaxa acestei instrucțiuni este următoarea:

[*etichetă*:] **for** *contor* **in** *domeniu* **loop**
        secvență_de_instrucțiuni
end loop [*etichetă*];

Într-o instrucțiune **for loop** se specifică un contor de iterații şi un domeniu. Instrucțiunile din corpul buclei sunt executate cât timp contorul se află în domeniul specificat. După terminarea unei iterații, contorului i se asignează următoarea valoare din domeniu. Domeniul poate fi unul crescător, specificat prin cuvântul cheie **to**, sau descrescător, specificat prin cuvântul cheie **downto**. Acest domeniu poate fi specificat şi sub forma unui tip enumerat sau subtip, caz în care nu se specifică în mod explicit limitele domeniului în cadrul instrucțiunii **for loop**. Limitele domeniului vor fi determinate de compilator din declarația tipului sau subtipului respectiv.

Instrucțiunea **for loop** din Exemplul 4 calculează pătratele valorilor întregi cuprinse între 1 şi 10, pe care le depune în tabloul i_patrat.

Exemplul 4:

```
for i in 1 to 10 loop
```

```
        i_patrat (i) <= i * i;
end loop;
```

Contorul de iterații din acest exemplu este în mod implicit de tip **integer**, deoarece tipul acestuia nu a fost definit în mod explicit. Forma completă a declarației domeniului pentru contorul de iterații este similară cu cea a unui tip. Pentru exemplul anterior, clauza for poate fi scrisă şi sub forma următoare:

```
for i in integer range 1 to 10 loop
```

În unele limbaje de programare, în cadrul buclei se poate asigna o valoare contorului de iterații (în exemplul anterior, i) pentru a-i modifica valoarea. Limbajul VHDL nu permite însă asignarea unei valori contorului de iterații sau utilizarea acestuia ca parametru de intrare sau de ieşire al unei proceduri. Contorul poate fi utilizat însă într-o expresie, cu condiția să nu i se modifice valoarea. Un alt aspect legat de contorul de iterații este că acesta nu trebuie declarat în mod explicit în cadrul procesului. Acest contor este declarat în mod implicit ca o variabilă locală a instrucțiunii de buclare prin specificarea sa după cuvântul cheie **for**. Dacă există o altă ariabilă cu acelaşi nume în cadrul procesului, cele două vor fi tratate ca variabile separate.

Interpretarea sintezei instrucțiunii **for loop** este aceea că se realizează o nouă copie a circuitului descris de conținutul instrucțiunii la fiecare iterație a buclei. Utilizarea instrucțiunii **for loop** pentru generarea unui circuit este ilustrată în Exemplul 5.

Exemplul 5:

```
entity potrivire_biti is
        port (a, b: in bit_vector (7 downto 0);
                potriviri: out bit_vector (7 downto 0));
end potrivire_biti;
architecture functional of potrivire_biti is
begin
process (a, b)
begin
        for i in 7 downto 0 loop
                potriviri (i) <= not (a(i) xor b(i));
        end loop;
end process;
end functional;
```

Prin procesul din acest exemplu, se generează un set de comparatoare de câte un bit, care compară biții de acelaşi rang ai vectorilor *a* şi *b*. Rezultatul este înscris în vectorul potriviri, care va conține '1' în pozițiile în care biții celor doi vectori sunt identici şi '0' în celelalte poziții. Procesul din exemplul anterior este echivalent cu următorul proces:

```
process (a, b)
begin
        potriviri (7) <= not (a(7) xor b(7));
        potriviri (6) <= not (a(6) xor b(6));
        potriviri (5) <= not (a(5) xor b(5));
        potriviri (4) <= not (a(4) xor b(4));
        potriviri (3) <= not (a(3) xor b(3));
        potriviri (2) <= not (a(2) xor b(2));
        potriviri (1) <= not (a(1) xor b(1));
```

```
        potriviri (0) <= not (a(0) xor b(0));
end process;
```

În acest caz, ordonarea domeniului contorului de iteraţii nu are importanţă, deoarece nu există conexiune între blocurile generate ale circuitului. Această ordonare devine importantă atunci când există o conexiune între aceste blocuri. Această conexiune este creată de obicei de către o variabilă care păstrează o valoare într-o iteraţie a buclei, valoare care este citită apoi într-o altă iteraţie. De obicei, este necesară iniţializarea unei asemene variabile înaintea intrării în buclă. Exemplul 6 prezintă un asemenea circuit.

Exemplul 6:

```
library ieee;
use ieee.numeric_bit.all;
entity contorizare_unu is
        port (v: in bit_vector (15 downto 0);
                num: out signed (3 downto 0));
end contorizare_unu;
architecture functional of contorizare_unu is
begin
process (v)
        variable rez: signed (3 downto 0);
begin
        rez := (others => '0');
        for i in 15 downto 0 loop
                if v(i) = '1' then
                        rez := rez + 1;
                end if;
        end loop;
num <= rez;
end process;
end functional;
```

Acest exemplu este un circuit combinaţional care contorizează numărul biţilor din vectorul *v* care sunt '1'. Rezultatul este acumulat pe parcursul execuţiei procesului într-o variabilă numită *rez* şi este asignată apoi semnalului de ieşire *num* la sfârşitul procesului.

În Exemplul 6, domeniul contorului de iteraţii al instrucţiunii **for loop** a fost specificat în mod explicit ca fiind **15 downto 0**. În practică, această specificare explicită este utilizată foarte rar pentru accesarea tablourilor, fiind recomandată utilizarea atributelor pentru tablouri.

4.1.2.4. Instrucţiunea next

Există situaţii în care este necesară oprirea execuţiei instrucţiunilor dintr-o buclă în iteraţia curentă şi trecerea la următoarea iteraţie. Pentru aceasta se poate utiliza instrucţiunea **next**. Sintaxa acestei instrucţiuni este următoarea:

next [*etichetă*] [**when** *condiţie*];

La întâlnirea unei instrucţiuni **next** în cadrul corpului unei bucle, execuţia iteraţiei curente este oprită şi controlul este transferat la începutul instrucţiunii de buclare, fie necondiţionat, dacă clauza **when** nu este prezentă, fie condiţionat, dacă această clauză este prezentă. Contorul de iteraţii va fi actualizat, iar dacă limita domeniului nu a fost atinsă,

execuția va continua cu prima instrucțiune din corpul buclei. În caz contrar, execuția instrucțiunii de buclare se va termina.

În cazul în care există mai multe nivele de instrucțiuni de buclare (o buclă conținută într-o altă buclă), în cadrul instrucțiunii **next** se poate specifica o etichetă, aceasta fiind eticheta instrucțiunii de buclare de nivel imediat superior în care este cuprinsă instrucțiunea **next**. Această etichetă are doar rolul de a creşte claritatea descrierii şi nu poate fi diferită de cea a instrucțiunii de buclare curente.

O instrucțiune **next** se poate utiliza în locul unei instrucțiuni **if** pentru execuția condiționată a unui grup de instrucțiuni. Pentru sinteza instrucțiunii **next** sunt necesare aceleaşi circuite ca şi cele necesare pentru sinteza instrucțiunii **if.** Alegerea uneia din cele două instrucțiuni rămâne la latitudinea proiectantului.

Ca un exemplu, considerăm acelaşi circuit care contorizează numărul biților de '1' dintr-un vector. Exemplul 7 prezintă descrierea modificată a acestui circuit. În locul instrucțiunii **if** se utilizează o instrucțiune **next**, prin care se abandonează o iterație în cazul în care valoarea elementului curent este '0', astfel că nu se execută incrementarea contorului.

Exemplul 7:

```
library ieee;
use ieee.numeric_bit.all;
entity contorizare_unu is
        port (v: in bit_vector (15 downto 0);
                num: out signed (3 downto 0));
end contorizare_unu;
architecture functional of contorizare_unu is
begin
process (v)
        variable rez: signed (3 downto 0);
begin
        rez := (others => '0');
        for i in v'range loop
                next when v(i) = '0';
                        rez := rez + 1;
        end loop;
        num <= rez;
end process;
end functional;
```

4.1.2.5. Instrucțiunea exit

Există situații în care execuția unei instrucțiuni de buclare trebuie oprită complet, fie datorită apariției unei erori în timpul execuției unui model, fie datorită faptului că prelucrarea trebuie terminată înaintea depăşirii domeniului de către contorul de iterații. În asemenea situații, se poate utiliza instrucțiunea **exit**. Sintaxa acestei instrucțiuni este următoarea:

exit [*etichetă*] [**when** *condiție*];

Există trei forme ale instrucțiunii **exit**. Prima este cea în care nu apare o etichetă şi nici o condiție specificată printr-o clauză **when**. În acest caz, se va opri în mod necondiționat execuția instrucțiunii curente de buclare.

În cazul în care instrucțiunea **exit** apare într-o instrucțiune de buclare aflată în interiorul unei alte instrucțiuni de buclare, va fi oprită doar execuția instrucțiunii interioare de buclare, dar execuția instrucțiunii exterioare de buclare va continua.

Dacă în instrucțiunea **exit** se specifică o etichetă a unei instrucțiuni de buclare, la întâlnirea instrucțiunii **exit** controlul va fi transferat la eticheta specificată.

Dacă instrucțiunea **exit** conține o clauză **when**, execuția instrucțiunii de buclare va fi oprită numai în cazul în care condiția specificată de această clauză este adevărată. Următoarea instrucțiune executată depinde de prezența sau absența unei etichete în cadrul instrucțiunii. Dacă se specifică o etichetă a unei instrucțiuni de buclare, următoarea instrucțiune executată va fi prima din instrucțiunea de buclare specificată de acea etichetă. Dacă nu se specifică o etichetă, următoarea instrucțiune executată va fi cea de după clauza **end loop** a instrucțiunii de buclare curente.

Instrucțiunea **exit** se poate utiliza pentru a termina execuția unei instrucțiuni **loop** simple, după cum se arată în Exemplul 8.

Exemplul 8:

```
E3: loop
        a := a + 1;
        exit E3 when a > 10;
end loop E3;
```

Exemplul 9 prezintă descrierea unui circuit pentru contorizarea numărului de zerouri de la sfârşitul unui vector de biți. Se testează fiecare element al vectorului reprezentând o valoare întreagă, iar dacă un element este '1', bucla se termină cu ajutorul instrucțiunii **exit**.

Exemplul 9:

```
library ieee;
use ieee.numeric_bit.all;
entity contorizare_zero is
        port (v: in bit_vector (15 downto 0);
                num: out signed (3 downto 0));
end contorizare_zero;
architecture functional of contorizare_zero is
begin
process (v)
        variable rez: signed (3 downto 0);
begin
        rez := (others => '0');
        for i in v'reverse_range loop
                exit when v(i) = '1';
                rez := rez + 1;
        end loop;
        num <= rez;
end process;
end functional;
```

4.1.3. Instrucțiunea secvențială assert

Instrucțiunea **assert** este utilă pentru afişarea unor mesaje de avertisment sau de eroare în timpul simulării unui model. Această instrucțiune testează valoarea unei condiții booleene

şi afişează mesajul specificat în cazul în care condiția este falsă. Sintaxa instrucțiunii este următoarea:

assert *condiție*
[**report** *şir_de_caractere*]
[**severity** *nivel_de_severitate*];

Condiția specificată este o expresie care trebuie să se evalueze la o valoare booleană. Dacă această valoare este TRUE, instrucțiunea nu are nici un efect. Dacă valoarea este FALSE, se afişează textul specificat în clauza **report**.

Clauza opțională **report** poate avea ca argument un şir de caractere, cu tipul predefinit **string**. Dacă această clauză nu este specificată, şirul de caractere care va fi afişat în mod implicit va fi "*Assertion violation*".

Clauza opțională **severity** permite specificarea nivelului de severitate al violării aserțiunii. Nivelul de severitate trebuie să fie o expresie cu tipul predefinit **severity_level**. Acest tip conține următoarele valori, în ordinea crescătoare a nivelului de severitate: **note, warning, error şi failure**. Dacă această clauză este omisă, se va presupune nivelul de severitate implicit **error**. Utilizarea nivelelor de severitate este descrisă pe scurt în continuare.

- Nivelul **note** poate fi utilizat pentru afişarea unor informații despre modul de desfăşurare al simulării.
- Nivelul **warning** poate fi utilizat în situațiile în care simularea poate fi continuată, dar este posibil ca rezultatele să fie imprevizibile.
- Nivelul **error** se utilizează atunci când violarea aserțiunii reprezintă o eroare care determină funcționarea incorectă a modelului, în acest caz execuția simulării fiind oprită.
- Nivelul **failure** se utilizează atunci când violarea aserțiunii reprezintă o eroare fatală, cum este împărțirea la zero sau adresarea unui tablou cu un index care depăşeşte domeniul permis. Şi în acest caz, execuția simulării va fi oprită.

Exemplul 10

```
assert not (R = '1' and S = '1')
    report "Ambele semnale R şi S au valoarea '1'"
    severity error;
```

Atunci când ambele semnale R şi S au valoarea '1', se afişează mesajul specificat şi simularea va fi oprită.

Pentru afişarea unui mesaj în mod necondiționat, se va utiliza condiția FALSE, de exemplu:

assert (FALSE) **report** "Start simulare";

Observații

- Instrucțiunea **assert** descrisă este o instrucțiune secvențială, presupunând că ea apare într-un proces sau subprogram. Există însă şi o versiune concurentă a acestei instrucțiuni. Aceasta are un format identic cu versiunea secvențială, dar poate apare numai în afara unui proces sau subprogram.
- De obicei, sistemele de sinteză ignoră instrucțiunea **assert**.

4.3. Teme propuse

4.3.1. Analizați exemplele prezentate. Simulați descrierea circuitului pentru compararea biților de acelaşi rang a doi vectori (Exemplul 5), a circuitului pentru contorizarea biților de '1' ai unui vector (Exemplul 6). Utilizați sistemul *Active-HDL* pentru compilarea cu succes a acestor descrieri.

4.3.2. Realizați un comparator de 4 biți cu trei ieşiri (egal, mai mic, mai mare) utilizând:

a) Operatori logici;
b) Instrucțiunea de asignare selectivă;
c) Instrucțiunea condițională if.

Compilați descrierile comparatorului şi simulați funcționarea acestora.

4.3.2. Descrieți un multiplexor 4:1 de 8 biți utilizând o instrucțiune case. Intrările multiplexorului sunt a[7:0], b[7:0], c[7:0], d[7:0], s[1:0], iar ieşirile sunt q[7:0].

4.3.3. Să se realizeze o descriere a unui decodificator pe 3 biti, cu o intrare de 3 biti, intrare de activare a iesirii si 8 iesiri de câte 1 bit. Introducerea unui cod pe intrare generează activarea iesirii corespunzătoare codului aplicat.

- dacă In=0 atunci iesirea corespunzătoare codului se activează y_0=1, iar celelalte iesiri sunt 0;
- dacă In=1 atunci iesirea corespunzătoare codului se activează y_1=1, s.a.m.d.

4.3.4. Realizați un decodificator din codul BCD pentru afişajul cu 7 segmente. Intrările decodificatorului sunt bcd[3:0], iar ieşirile sunt led[6:0].