PROIECTAREA SISTEMELOR DIGITALE

Masterat ICCP - an 1

LABORATOR 3

INSTRUCȚIUNI SECVENȚIALE ÎN LIMBAJUL VHDL
Partea 1

O descriere în limbajul VHDL are două domenii: un domeniu secvențial şi un domeniu concurent. Domeniul *secvențial* este reprezentat de un proces sau subprogram care conține instrucțiuni secvențiale. Aceste instrucțiuni sunt executate în ordinea în care apar în cadrul procesului sau subprogramului, ca şi în cazul limbajelor de programare. Domeniul *concurent* este reprezentat de o arhitectură, care conține procese, apeluri concurente de proceduri, asignări concurente ale semnalelor şi instanțieri de componente. Toate activitățile descrise de acestea au loc simultan.

În această lucrare de laborator se prezintă formatul şi modul de utilizare al unor instrucțiuni secvențiale. Pentru exemplificare sunt descrise unele componente combinaționale şi secvențiale de bază, ca de exemplu: multiplexoare, codificatoare, decodificatoare, numaratoare.

3.1. Instrucțiuni secvențiale

În această secțiune se descriu mai întâi unele aspecte legate de procese, cum sunt modul de specificare a unui proces, execuția unui proces, instrucțiunea **wait** şi deosebirea dintre procesele combinaționale şi cele secvențiale. Sunt prezentate apoi instrucțiunile secvențiale care pot apare într-un proces sau subprogram: asignarea secvențială a semnalelor, asignarea variabilelor, instrucțiunea **if**.

3.1.1. Procese

Un proces este o secvență de instrucțiuni care sunt executate în ordinea specificată. Declarația unui proces delimitează un domeniu secvențial al arhitecturii în care apare declarația. Procesele se utilizează pentru descrieri funcționale.

3.1.1.1. Structura şi execuția unui proces

Un proces poate apare oriunde în corpul unei arhitecturi (partea care începe după cuvântul cheie **begin**). Structura de bază a declarației unui proces este următoarea:

[*nume*:] **process** [(*listă_de_sensibilitate*)]
    [*declarații_de_tipuri*]
    [*declarații_de_constante*]
    [*declarații_de_variabile*]
    [*declarații_de_subprograme*]
begin
    instrucțiuni_secvențiale
end process [*nume*];

Declarația unui proces este cuprinsă între cuvintele cheie **process** şi **end process**. Unui proces i se poate asigna în mod opțional un nume pentru identificarea mai uşoară a procesului în textul sursă. Numele este un identificator şi trebuie urmat de caracterul ':' (cu rol de etichetă). Acest nume este util şi pentru simulare, de exemplu, pentru setarea unui punct de întrerupere a execuției simulării. Numele poate fi repetat la sfârşitul declarației, după cuvintele cheie **end process**.

Lista de sensibilitate (opțională) este lista semnalelor la a căror modificare procesul este "sensibil". Producerea unui eveniment asupra unuia din semnalele specificate în lista de sensibilitate va determina execuția instrucțiunilor secvențiale din cadrul procesului, similar cu instrucțiunile dintr-un program obişnuit. Spre deosebire de un limbaj de programare, în limbajul VHDL clauza **end process** nu specifică terminarea execuției procesului.

Procesul va fi executat într-un ciclu infinit, iar în cazul în care se specifică o listă de sensibilitate, procesul va fi doar suspendat după execuția ultimei instrucțiuni, până la producerea unui nou eveniment asupra semnalelor din listă. Se menționează că un eveniment are loc numai la modificarea valorii unui semnal. Astfel, asignarea aceleiaşi valori la un semnal nu reprezintă un eveniment.

În cazul în care lista de sensibilitate lipseşte, procesul va fi executat în mod continuu. În acest caz, procesul trebuie să conțină o instrucțiune **wait** pentru a determina suspendarea procesului şi activarea acestuia la apariția unui eveniment sau îndeplinirea unei condiții. În cazul în care lista de sensibilitate este prezentă, procesul nu poate conține instrucțiuni **wait**.

Partea declarativă a procesului este cuprinsă între cuvintele cheie **process** şi **begin**. În această parte se pot declara tipuri, constante, variabile şi subprograme (proceduri şi funcții) care sunt locale procesului. Elementele declarate se pot utiliza deci numai în interiorul procesului.

Observație

- În cadrul unui proces nu pot fi declarate semnale, ci numai constante şi variabile.

Partea de instrucțiuni a procesului începe după cuvântul cheie **begin**. Această parte conține instrucțiunile care vor fi executate la fiecare activare a procesului. Nu este permisă utilizarea instrucțiunilor concurente în interiorul unui proces.

În Exemplul 1 se prezintă declarația unui proces simplu format dintr-o singură instrucțiune de asignare secvențială.

Exemplul 1:

```
proc1: process (a, b, c)
begin
        x <= a and b and c;
end process proc1;
```

3.1.1.2. Procese cu liste de sensibilitate incomplete

Unele sisteme de sinteză nu verifică listele de sensibilitate ale proceselor. Aceste sisteme presupun că toate semnalele din partea dreaptă a asignărilor secvențiale la semnale se află în lista de sensibilitate. Astfel, aceste sisteme vor interpreta cele două procese din Exemplul 2 ca fiind identice.

Exemplul 2:

```
proc1: process (a, b, c)
begin
        x <= a and b and c;
end process proc1;
```

```
proc2: process (a, b)
begin
        x <= a and b and c;
end process proc2;
```

Toate sistemele de sinteză vor interpreta procesul *proc1* ca o poartă ŞI cu 3 intrări. Unele sisteme de sinteză vor interpreta procesul *proc2* de asemenea ca o poartă ŞI cu 3 intrări, chiar dacă la simularea acestui cod procesul nu se va comporta ca atare. La simulare, o modificare a valorii semnalului *a* sau *b* va determina execuţia procesului, iar valoarea funcţiei ŞI logic între semnalele *a*, *b* şi *c* se va asigna semnalului *x*. Totuşi, dacă se modifică valoarea semnalului *c*, procesul nu este executat, iar semnalul *x* nu este actualizat.

Deoarece nu este clar modul în care un sistem de sinteză ar trebui să genereze un circuit pentru care o tranziţie a semnalului *c* nu determină o modificare a semnalului *x*, dar o modificare a semnalelor *a* sau *b* determină actualizarea semnalului *x* cu valoarea funcţiei ŞI logic între semnalele *a*, *b* şi *c*, există următoarele alternative pentru sistemele de sinteză:

- Interpretarea procesului *proc2* în mod identic cu procesul *proc1* (cu o listă de sensibilitate care cuprinde toate semnalele din partea dreaptă a instrucţiunilor de asignare a semnalelor din cadrul procesului);
- Indicarea unei erori la compilare, specificând faptul că nu se poate realiza sinteza procesului fără o listă de sensibilitate completă.

A doua variantă este preferabilă, deoarece proiectantul va trebui să modifice codul sursă astfel încât funcţionarea circuitului generat să fie aceeaşi cu rezultatul simulării funcţionale a codului sursă.

Deşi din punct de vedere sintactic procesele pot fi declarate fără o listă de sensibilitate şi fără o instrucţiune **wait**, asemenea procese nu vor fi suspendate niciodată. De aceea, dacă se va simula un asemenea proces, timpul de simulare nu va avansa, deoarece faza de iniţializare, în care toate procesele sunt executate până când acestea sunt suspendate, nu se va termina niciodată.

3.1.1.3. Instrucţiunea wait

În locul unei liste de sensibilitate, un proces poate conţine o instrucţiune **wait**. Utilizarea unei instrucţiuni **wait** are două scopuri:

- Suspendarea execuţiei unui proces;
- Specificarea condiţiei care va determina activarea procesului suspendat.

La întâlnirea unei instrucţiuni **wait**, procesul în care apare această instrucţiune este suspendat. Atunci când se îndeplineşte condiţia specificată în cadrul instrucţiunii **wait**, procesul este activat şi se execută instrucţiunile acestuia până când se întâlneşte din nou instrucţiunea **wait**. Limbajul VHDL permite ca un proces să conţină mai multe instrucţiuni **wait**. Atunci când se utilizează pentru modelarea logicii combinaţionale în vederea sintezei, un proces poate conţine însă o singură instrucţiune **wait**.

Dacă un proces conţine o instrucţiune **wait**, nu poate conţine o listă de sensibilitate. Procesul din Exemplul 3, care conţine o instrucţiune **wait** explicită, este echivalent cu procesul din Exemplul 1, care conţine o listă de sensibilitate. Ambele procese se vor executa atunci când apare o modificare a valorii semnalelor *a*, *b* sau *c*.

Exemplul 3:

```
proc3: process
begin
        x <= a and b and c;
wait on a, b, c;
end process proc3;
```

Există trei forme ale instrucțiunii wait:

wait on *listă_de_sensibilitate*;
wait until *expresie_condițională*;
wait for *expresie_de_timp*;

Instrucțiunea **wait on** a fost ilustrată în exemplele precedente. Instrucțiunea **wait until** suspendă un proces până când condiția specificată devine adevărată datorită modificării unuia din semnalele care apar în expresia condițională. Se menționează că dacă nici un semnal din această expresie nu se modifică, procesul nu va fi activat, chiar dacă expresia condițională este adevărată. Exemplele următoare prezintă mai multe forme ale instrucțiunii **wait until**:

wait until *semnal* = *valoare*;
wait until *semnal*'**event and** *semnal* = *valoare*;
wait until not *semnal*'**stable and** *semnal* = *valoare*;

unde *semnal* este numele unui semnal, iar *valoare* este valoarea care se testează. Dacă semnalul este de tip bit, atunci pentru valoarea '1' se aşteaptă frontul crescător al semnalului, iar pentru '0' frontul descrescător.

Instrucțiunea **wait until** se poate utiliza pentru implementarea unei funcționări sincrone. În mod obişnuit, semnalul testat este un semnal de ceas. De exemplu, aşteptarea frontului crescător al unui semnal de ceas se poate exprima în următoarele moduri:

wait until clk = '1';
wait until clk'**event and** clk = '1';
wait until not clk'**stable and** clk = '1';

Pentru descrierile destinate sintezei, instrucțiunea **wait until** trebuie să fie prima din cadrul procesului. Din această cauză, logica sincronă descrisă cu o instrucțiune **wait until** nu poate fi resetată în mod asincron.

Instrucțiunea **wait for** permite suspendarea execuției unui proces pentru un timp specificat, de exemplu:

wait for 10 ns;

Se pot combina mai multe condiții ale instrucțiunii **wait** într-o condiție mixtă. În Exemplul 4, procesul *proc4* va fi activat la modificarea valorii unuia din semnalele *a* sau *b*, dar numai atunci când valoarea semnalului *clk* este '1'.

Exemplul 4:

```
proc4: process
begin
        wait on a, b until clk = '1';
...
end process proc4;
```

De obicei, instrucțiunea **wait** apare fie la începutul unui proces, fie la sfârşitul acestuia. În Exemplul 3, instrucțiunea **wait** apare la sfârşitul procesului. S-a menționat că

această formă a procesului este echivalentă cu forma care conține o listă de sensibilitate. Această echivalență se datorează modului în care se execută simularea unui model. În faza de inițializare a simulării, se execută toate procesele modelului. Dacă procesul conține o listă de sensibilitate, toate instrucțiunile procesului vor fi executate o singură dată. În forma procesului care conține o instrucțiune wait şi aceasta apare la sfârşit, în faza de inițializare se execută o singură dată toate instrucțiunile până la instrucțiunea **wait**, ca şi în cazul formei care conține o listă de sensibilitate.

În cazul în care instrucțiunea **wait** apare la începutul procesului, simularea se va executa în mod diferit, deoarece în faza de inițializare procesul va fi suspendat fără a se executa nici o instrucțiune a acestuia. Deci, un proces cu instrucțiunea **wait** plasată la început nu este echivalent cu un proces care conține o listă de sensibilitate.

Deoarece faza de inițializare a simulării nu are echivalent hardware, nu vor exista diferențe la sinteza celor două procese cu instrucțiunea **wait** plasată la sfârşit, respectiv la început.

Totuşi, din cauza diferenței între simulare şi sinteză în ceea ce priveşte faza de inițializare a simulării, pot exista diferențe între funcționarea modelului la simulare şi funcționarea circuitului rezultat prin sinteză. Este posibil ca modelul care funcționează corect la simulare să fie sintetizat într-un circuit a cărui funcționare este eronată.

3.1.1.4. Procese combinaționale şi procese secvențiale

Atât procesele combinaționale, cât şi procesele secvențiale sunt interpretate în acelaşi fel, singura deosebire dintre acestea fiind că la procesele secvențiale semnalele de ieşire se înscriu în bistabile. Un process combinațional simplu este prezentat în Exemplul 5.

Exemplul 5:

```
proc5: process
begin
        wait on a, b;
        z <= a and b;
end process proc5;
```

Acest proces va fi implementat prin sinteză ca o simplă poartă ŞI cu două intrări. Pentru ca un proces să modeleze un circuit combinațional, acesta trebuie să conțină în lista de sensibilitate toate semnalele care sunt intrări ale procesului. Cu alte cuvinte, procesul trebuie reevaluat de fiecare dată când una din intrările circuitului pe care îl modelează se modifică. În acest fel se modelează în mod corect un circuit combinațional.

Dacă un proces nu este sensibil la toate intrările sale şi nu este un proces secvențial, atunci nu poate fi sintetizat, deoarece nu există un echivalent hardware al unui asemenea proces. Nu toate sistemele de sinteză impun o asemenea regulă, astfel încât trebuie acordată atenție la scrierea proceselor combinaționale pentru a nu introduce erori. Asemenea erori vor avea ca efect diferențe subtile între modelul simulat şi circuitul obținut prin sinteză, deoarece un proces non-combinațional este interpretat de sistemul de sinteză ca un circuit combinațional.

Dacă un proces conține o instrucțiune **wait** sau o instrucțiune if ***semnal*'event**, procesul va fi interpretat ca un proces secvențial. Astfel, procesul din Exemplul 6 va fi interpretat ca un proces secvențial.

Exemplul 6:

```
proc6: process
begin
```

```
        wait until clk = '1';
        z <= a and b;
end process proc6;
```

3.1.2. Instrucțiunea secvențială de asignare a semnalelor

Semnalele reprezintă interfața dintre domeniul concurent al unui model în limbajul VHDL şi domeniul secvențial din cadrul unui proces. Un model VHDL este format dintr-un număr de procese care comunică între ele prin intermediul semnalelor. La simulare, întreaga ierarhie din cadrul modelului este eliminată, rămânând numai procesele şi semnalele. Simulatorul execută în mod alternativ actualizarea valorii unor semnale şi rularea proceselor activate de modificarea valorii semnalelor aflate în listele lor de sensibilitate.

3.1.2.1. Execuția instrucțiunii secvențiale de asignare

Asignarea valorilor la semnale se poate realiza printr-o instrucțiune secvențială sau o instrucțiune concurentă. Instrucțiunea secvențială poate apare doar în interiorul unui proces, iar cea concurentă poate apare doar în exteriorul proceselor.

Instrucțiunea secvențială de asignare are o singură formă, cea simplă, care este o asignare necondiționată. Instrucțiunea concurentă are, pe lângă forma sa simplă, încă două forme: asignarea condițională şi asignarea selectivă.

Instrucțiunea secvențială de asignare are aceeaşi sintaxă cu forma simplă a instrucțiunii concurente de asignare, deosebirea dintre acestea rezultând din context.

Instrucțiunea secvențială de asignare a unui semnal are sintaxa următoare:

semnal <= *expresie* [**after** *întârziere*];

Ca rezultat al execuției acestei instrucțiuni într-un proces, se evaluează expresia din dreapta simbolului de asignare şi se planifică un eveniment care constă din modificarea valorii semnalului. Simulatorul va modifica însă valoarea semnalului doar în momentul în care procesul va fi suspendat, iar în cazul în care se utilizează clauza **after**, după întârzierea specificată din acest moment. Deci, într-un proces semnalele vor fi actualizate doar după execuția tuturor instrucțiunilor din cadrul procesului sau la întâlnirea unei instrucțiuni **wait**.

În mod tipic, sistemele de sinteză nu permit utilizarea clauzelor **after**, sau ignoră aceste clauze. Clauzele **after** sunt ignorate nu numai deoarece interpretarea lor pentru sinteză nu este specificată de standarde, ci şi pentru că ar fi dificilă garantarea rezultatelor unor asemenea întârzieri. De exemplu, nu este clar dacă întârzierea ar trebui interpretată ca o întârziere de propagare minimă sau maximă. De asemenea, nu este clar cum trebuie să procedeze programul de sinteză dacă o întârziere specificată în codul sursă nu poate fi asigurată.

O consecință a modului în care se execută asignarea semnalelor în cadrul proceselor este că în cazul în care există mai multe asignări a unor valori diferite la acelaşi semnal, va avea efect doar ultima asignare. Astfel, cele două procese din Exemplul 7 sunt echivalente.

Exemplul 7:

```
proc7: process (a)
begin
        z <= '0';
        z <= a;
end process proc7;
proc8: process (a)
```

```
begin
        z <= a;
end process proc8;
```

În concluzie, trebuie să se țină cont de următoarele aspecte importante la utilizarea instrucțiunilor de asignare a semnalelor în interiorul proceselor:

- Orice asignare a unei valori la un semnal devine efectivă numai atunci când procesul este suspendat. Până în acel moment, toate semnalele îşi păstrează vechea valoare.
- Numai ultima asignare a unei valori la un semnal va fi executată efectiv. Deci, nu are sens să se asigneze mai mult de o valoare la un semnal în acelaşi proces.

3.1.2.2. Reacții inverse

O altă consecință a modului în care se execută asignarea semnalelor este că prin citirea unui semnal căruia i se asignează o valoare în acelaşi proces se va obține valoarea care i-a fost asignată semnalului la execuția precedentă a procesului. Citirea unui semnal şi asignarea unei valori la acel semnal în acelaşi proces este echivalentă cu o *reacție inversă*. Într-un proces combinațional, valoarea precedentă este o ieşire a logicii combinaționale, astfel încât reacția este asincronă. Într-un proces secvențial, valoarea precedentă este valoarea memorată într-un bistabil sau registru, astfel încât reacția este sincronă.

În Exemplul 8 se prezintă un proces în care se utilizează reacția inversă sincronă. Procesul descrie un numărător de 4 biți, semnalul *num* fiind de tip **unsigned**, tip care este declarat în pachetul **numeric_bit**. Se observă că semnalul *num* este declarat în afara procesului.

Exemplul 8:

```
library ieee;
use ieee.numeric_bit.all;
signal num: unsigned (3 downto 0);
process
begin
        wait until clk = '1';
        if reset = '1' then
                num <= "0000";
        else
                num <= num + "0001";
end if;
end process;
```

3.1.2.3. Întârzierea inerțială

În limbajul VHDL există două tipuri de întârzieri care pot fi utilizate pentru modelarea sistemelor reale. Acestea sunt întârzierea inerțială şi întârzierea de transport. Menționăm că aceste întârzieri nu se pot utiliza pentru sinteza logică. Întârzierea inerțială este implicită, fiind utilizată atunci când nu se specifică tipul întârzierii.

Clauza **after** presupune în mod automat întârzierea inerțială. Într-un model cu întârziere inerțială, două modificări consecutive ale valorii unui semnal de intrare nu vor modifica valoarea unui semnal de ieşire dacă timpul dintre aceste modificări este mai scurt decât întârzierea specificată. Această întârziere reprezintă inerția circuitului real.

Dacă, de exemplu, apar anumite impulsuri de durată scurtă ale semnalelor de intrare, semnalele de ieşire vor rămâne neschimbate.

3.1.2.4. Întârzierea de transport

Întârzierea de transport trebuie indicată în mod explicit prin cuvântul cheie **transport**. Aceasta reprezintă întârzierea unei conexiuni, în care efectul apariţiei unui impuls al unui semnal de intrare este propagat la ieşire cu întârzierea specificată, indiferent de durata acestui impuls. Întârzierea de transport este utilă în special pentru modelarea liniilor de transmisie şi a conexiunilor dintre componente.

Exempu de întârzierea de transport:

b <= **transport** a **after** 20 ns;

3.1.3. Variabile

Restricţiile impuse asupra semnalelor reduc posibilităţile de utilizare ale acestora. Deoarece semnalele pot păstra numai ultima valoare asignată, ele nu pot fi utilizate pentru memorarea rezultatelor intermediare sau temporare în cadrul unui proces. Un alt inconvenient este că noile valori sunt asignate semnalelor nu în momentul execuţiei instrucţiunii de asignare, ci după suspendarea execuţiei procesului. Aceasta determină ca analiza unei descrieri să fie dificilă.

Spre deosebire de semnale, variabilele pot fi declarate în interiorul unui proces şi se pot utiliza pentru memorarea rezultatelor intermediare. Variabilele pot fi utilizate însă numai în domeniul secvenţial al limbajului VHDL, deci în interiorul proceselor sau subprogramelor, şi nu pot fi declarate sau utilizate direct într-o arhitectură. Ele sunt deci locale procesului sau subprogramului respectiv.

3.1.3.1. Declararea şi iniţializarea variabilelor

Ca şi semnalele, variabilele trebuie declarate înainte de a fi utilizate. Declaraţia unei variabile este similara cu cea a unui semnal, cu deosebirea că se utilizează cuvântul cheie **variable**. În mod opţional, în cazul variabilelor scalare se poate specifica un domeniu restrâns, iar în cazul variabilelor de tip tablou se poate specifica un index restrâns. Pentru ambele tipuri de variabile, se poate specifica o valoare iniţială. Exemplul 9 ilustrează declararea şi iniţializarea variabilelor.

Exemplul 9:

variable a, b, c: bit;
variable x, y: integer;
variable index **range** 1 **to** 10 := 1;
variable t_ciclu: time **range** 10 ns **to** 50 ns := 10 ns;
variable mem: bit_vector (0 **to** 15);

Unei variabile i se atribuie o valoare iniţială în faza de iniţializare a simulării. Această valoare iniţială este fie cea specificată în mod explicit la declararea variabilei, fie o valoare implicită, care este valoarea cea mai din stânga a tipului. De exemplu, pentru tipul bit valoarea iniţială implicită este '0', iar pentru tipul integer această valoare este -2.147.483.647.

Pentru sinteză, nu există o interpretare hardware a valorilor iniţiale, astfel încât sistemele de sinteză ignoră valorile iniţiale sau semnalează eroare.

3.1.3.2. Instrucțiunea de asignare a variabilelor

O instrucțiune de asignare a unei variabile înlocuieşte valoarea curentă a variabilei cu o nouă valoare specificată de o expresie. Expresia poate conține variabile, semnale şi literale. Variabila şi rezultatul evaluării expresiei trebuie să fie de acelaşi tip. Instrucțiunea de asignare a unei variabile are sintaxa următoare:

variabilă := *expresie*;

Această instrucțiune este similară cu asignările din majoritatea limbajelor de programare. Spre deosebire de o instrucțiune secvențială de asignare a semnalelor, asignarea unei variabile este executată instantaneu, deci într-un timp de simulare zero.

Există următoarele diferențe principale între asignarea semnalelor şi a variabilelor:

- În cazul asignării semnalelor, se planifică un eveniment pentru actualizarea valorii acestora, iar actualizarea se execută numai la suspendarea procesului, în timp ce pentru asignarea variabilelor nu se planifică un eveniment, iar actualizarea se execută instantaneu.
- La asignarea semnalelor se poate specifica o întârziere, în timp ce asignarea variabilelor nu poate fi întârziată.
- Într-un proces, doar ultima asignare a unei valori la un semnal este efectivă. În schimb, pot exista numeroase asignări la o variabilă în acelaşi proces, toate fiind efective.

Exemplul 10 ilustrează utilizarea variabilelor pentru memorarea rezultatelor intermediare.

Exemplul 10

```
entity add_1 is
        port (a, b, cin: in bit;
                s, cout: out bit);
end add_1;
architecture functional of add_1 is
begin
process (a, b, cin)
        variable s1, s2, c1, c2: bit;
begin
        s1 := a xor b;
        c1 := a and b;
        s2 := s1 xor cin;
        c2 := s1 and cin;
        s <= s2;
        cout <= c1 or c2;
end process;
end functional;
```

Exemplul anterior descrie un sumator elementar. Acesta nu este modul optim de a descrie un sumator elementar, dar ilustrează utilizarea variabilelor. Rezultatul se generează prin crearea a două semisumatoare, primul generând ieşirile *s1* şi *c1*, iar al doilea generând ieşirile *s2* şi *c2*. În final, ieşirile sunt asignate celor două porturi de ieşire *s* şi *cout* prin instrucțiuni de asignare a semnalelor, deoarece porturile sunt semnale.

3.1.4. Instrucțiunea if

3.1.4.1. Sintaxa şi execuția instrucțiunii if

Instrucțiunea **if** selectează pentru execuție una sau mai multe secvențe de instrucțiuni, în funcție de valoarea unei condiții corespunzătoare secvenței respective. Sintaxa acestei instrucțiuni este următoarea:

if *condiție* **then** *secvență_de_instrucțiuni*
[**elsif** *condiție* **then** *secvență_de_instrucțiuni* ...]
[**else** *secvență_de_instrucțiuni*]
end if;

Fiecare condiție este o expresie booleană, care se evaluează la valoarea TRUE sau FALSE. Pot exista mai multe clauze **elsif**, dar poate exista o singură clauză **else**. Se evaluează mai întâi condiția de după cuvântul cheie **if**, şi dacă este adevărată, se execută secvența de instrucțiuni corespunzătoare. În caz contrar, dacă este prezentă clauza **elsif**, se evaluează condiția de după această clauză şi dacă această condiție este adevărată, se execută secvența de instrucțiuni corespunzătoare. În caz contrar, dacă sunt prezente alte clauze **elsif**, se continuă cu evaluarea condiției acestor clauze. Dacă nici o condiție evaluată nu este adevărată, se execută secvența de instrucțiuni corespunzătoare clauzei **else**, dacă aceasta este prezentă.

Exemplul 11:

```
process (a, b)
begin
        if a = b then
        rez <= 0;
elsif a < b then
        rez <= -1;
else
        rez <= 1;
end if;
end process;
```

3.1.4.2. Interpretarea sintezei instrucțiunii if

Exemplul 12 prezintă utilizarea unei asemenea instrucțiuni pentru descrierea unui comparator.

Exemplul 12:

```
library ieee;
use ieee.numeric_bit.all;

entity comp is
        port (a, b: in unsigned (7 downto 0);
                egal: out bit);
end comp;
architecture functional of comp is
begin
process (a, b)
begin
        if a = b then
                egal <= '1';
```

```
        else
                egal <= '0';
        end if;
end process;
end functional;
```

În exemplul anterior, se testează egalitatea a două semnale de tip **unsigned**, reprezentând două valori întregi de câte 8 biți, rezultând o valoare de tip **bit**.

O instrucțiune **if** cu ramuri multiple, în care apare cel puțin o clauză **elsif**, este implementată prin mai multe etaje de multiplexoare. Considerăm instrucțiunea **if** din Exemplul 13.

Exemplul 13:

```
process (a, b, c, s0, s1)
begin
        if s0 = '1' then
                z <= a;
        elsif s1 = '1' then
                z <= b;
        else
                z <= c;
        end if;
end process;
```

Condițiile din ramurile succesive ale unei instrucțiuni **if** sunt evaluate independent. În exemplul anterior, condițiile implică cele două semnale *s0* şi *s1*. Pot exista oricâte condiții, fiecare din acestea fiind independent de celelalte. Structura instrucțiunii **if** asigură faptul că primele condiții vor fi testate înaintea celorlalte. În acest exemplu, semnalul *s0* a fost testat înaintea semnalului *s1*. Această prioritate se reflectă în circuitul rezultat, în care multiplexorul controlat de semnalul *s0* este mai apropiat de ieşire decât multiplexorul controlat de semnalul *s1*.

Este important să se rețină existența acestei priorități cu care se testează condițiile, astfel încât să fie eliminate condițiile redundante. Considerăm instrucțiunea **if** din Exemplul 14, care este echivalentă cu instrucțiunea **if** din Exemplul 13.

Exemplul 14:

```
process (a, b, c, s0, s1)
begin
        if s0 = '1' then
                z <= a;
        elsif s0 = '0' and s1 = '1' then
                z <= b;
        else
                z <= c;
        end if;
end process;
```

Condiția suplimentară *s0 = '0'* este redundantă, deoarece ea va fi testată numai dacă prima condiție a instrucțiunii **if** este falsă. Se recomandă evitarea unor asemenea redundanțe, deoarece nu există garanția că ele vor fi detectate şi eliminate de către sistemul de sinteză.

În cazul instrucțiunilor **if** cu ramuri multiple, în mod normal fiecare condiție va fi dependentă de semnale şi variabile diferite. Dacă fiecare ramură este dependentă de acelaşi semnal, atunci este mai avantajos să se utilizeze o instrucțiune **case**.

3.1.4.3. Instrucțiuni if incomplete

În exemplele prezentate până acum, toate instrucțiunile **if** au fost complete. Cu alte cuvinte, semnalului destinație i s-a asignat o valoare în toate condițiile posibile. Sunt posibile însă două situații în care unui semnal nu i se asignează o valoare: atunci când lipseşte clauza **else** a instrucțiunii **if** sau atunci când unui semnal nu i se asignează o valoare într-o ramură a instrucțiunii **if**. În ambele cazuri, interpretarea este aceeaşi. În situațiile în care unui semnal nu i se asignează o valoare, semnalul îşi păstrează valoarea precedentă.

Se pune însă problema care este valoarea precedentă a semnalului. Dacă există o instrucțiune anterioară de asignare în care semnalul apare ca destinație, valoarea precedentă provine de la acea instrucțiune de asignare. În caz contrar, valoarea provine din execuția precedentă a procesului, ceea ce conduce la existența unei reacții inverse în cadrul circuitului. Primul caz este ilustrat prin Exemplul 15.

Exemplul 15:

```
process (a, b, enable)
begin
        z <= a;
        if enable = '1' then
                z <= b;
        end if;
end process;
```

În acest caz, instrucțiunea **if** este incompletă deoarece lipseşte clauza **else**. În instrucțiunea **if**, semnalul **z** primeşte o valoare în cazul în care condiția *enable = '1'* este adevărată, dar rămâne la valoarea precedenta în cazul în care condiția este falsă. Valoarea precedentă provine din instrucțiunea de asignare aflată înaintea instrucțiunii **if**.
Instrucțiunea **if** din Exemplul 15 este echivalentă cu instrucțiunea **if** din exemplul următor.

Exemplul 16:

```
process (a, b, enable)
begin
        if enable = '1' then
                z <= b;
        else
                z <= a;
        end if;
end process;
```

În cazul în care instrucțiunea **if** este incompletă şi nu există o instrucțiune anterioară de asignare, va exista o reacție inversă de la ieşirea circuitului la intrare. Aceasta deoarece valoarea semnalului de la execuția precedentă a procesului este păstrată şi aceasta devine valoarea semnalului la execuția curentă a procesului.

O asemenea formă a instrucțiunii **if** este utilizată pentru a descrie un bistabil sau un registru cu o intrare de validare, ca în Exemplul 17.

Exemplul 17:

```
process
begin
        wait until clk = '1';
        if en = '1' then
                q <= d;
        end if;
end process;
```

Semnalul *q* este actualizat cu noua valoare a semnalului *d* în cazul în care condiția este adevărată, dar nu este actualizat în cazul în care condiția este falsă. În acest caz, valoarea precedentă a semnalului *q* este păstrată prin reacția inversă secvențială a semnalului *q*.

Instrucțiunea **if** din Exemplul 17 este echivalentă cu instrucțiunea **if** completă din exemplul următor.

Exemplul 18:

```
process
begin
        wait until clk = '1';
        if en = '1' then
                q <= d;
        else
                q <= q;
        end if;
end process;
```

În cazul în care condiția este falsă, se asignează semnalului *q* valoarea anterioară a acestuia, ceea ce este echivalent cu păstrarea valorii sale anterioare.

Una din cele mai obişnuite erori întâlnite în descrierile VHDL destinate sintezei logice este introducerea neintenționată a reacției inverse în circuit datorită unei instrucțiuni **if** incomplete. Pentru a evita introducerea reacției inverse, proiectantul trebuie să se asigure că fiecare semnal căruia i se asignează o valoare într-un process (care este deci un semnal de ieşire din proces) primeşte o valoare în fiecare combinație posibilă a condițiilor.

În practică, există două posibilități pentru aceasta: asignarea unei valori unui semnal de ieşire în fiecare ramură a instrucțiunii **if** şi includerea clauzei **else**, sau inițializarea semnalului printr-o instrucțiune de asignare necondiționată înaintea instrucțiunii **if**.

3.1.4.4. Instrucțiuni if în care apar variabile

Până acum, în instrucțiunile **if** au fost utilizate doar semnale. Aceleaşi reguli se aplică şi în cazul utilizării variabilelor, cu o singură diferență. Ca şi în cazul unui semnal, dacă unei variabile i se asignează valori numai în anumite ramuri ale instrucțiunii **if**, se păstrează valoarea anterioară a variabilei prin reacție inversă. Spre deosebire de cazul utilizării unui semnal, citirea şi scrierea unei variabile în acelaşi proces va determina o reacție inversă numai dacă citirea apare înaintea scrierii. În acest caz, valoarea citită este valoarea precedentă a variabilei. În cazul citirii şi scrierii în acelaşi proces a unui semnal, va rezulta întotdeauna o reacție inversă.

Această observație se poate utiliza pentru a crea registre sau numărătoare utilizând variabile. Reamintim că un proces secvențial este interpretat la sinteză prin plasarea unui bistabil sau registru înaintea ieşirii fiecărui semnal căruia i s-a asignat o valoare în procesul

respectiv. Aceasta înseamnă că în mod normal variabilele nu sunt înscrise în bistabile sau registre. Dacă există însă o reacție inversă a valorii precedente a unei variabile, această reacție se realizează printr-un bistabil sau registru pentru ca procesul să fie sincron.

În Exemplul 21 se descrie un numărător utilizând tipul întreg **unsigned**. La incrementarea unei valori de tip **unsigned**, dacă valoarea este cea maximă a domeniului se obține valoarea minimă a domeniului.

Exemplul 21:

```
process
variable num: unsigned (7 downto 0);
begin
        wait until clk = '1';
        if reset = '1' then
                num := "00000000";
        else
                num := num + 1;
        end if;
        rez <= num;
end process;
```

În acest exemplu, în ramura **else** a instrucțiunii **if** se citeşte valoarea precedentă a variabilei *num* pentru a calcula valoarea următoare. Rezultă astfel o reacție inversă.

De notat că în acest exemplu se crează de fapt două registre. Conform regulilor pentru reacția inversă, variabila *num* va fi înscrisă într-un registru. Semnalul *rez* va fi de asemenea înscris într-un registru, deoarece toate semnalele cărora li se asignează o valoare într-un proces secvențial vor fi înscrise în registre. Acest registru suplimentar va conține întotdeauna aceeaşi valoare ca şi registrul variabilei *num*. Sistemul de sinteză va elimina în mod normal acest registru redundant.

3.2 Teme propuse

3.2.1 Analizați exemplele prezentate care se referă la instrucțiunile secvențiale. Simulați descrierea comparatorului (Exemplul 12), a unui bistabil (Exemplul 18), a numărătorului (Exemplul 21). Utilizați sistemul *Active-HDL* pentru compilarea cu succes a acestor descrieri.

3.2.2 Descrieți un multiplexor 4:1 de 4 biți utilizând o instrucțiune IF. Intrările multiplexorului sunt a[3:0], b[3:0], c[3:0], d[3:0], s[1:0], iar ieşirile sunt q[3:0].