LUCRAREA NR. 6
DOMENIUL SECVENŢIAL. PROCESE

1. Scopul lucrării

Lucrarea oferă o perspectivă detaliată asupra noţiunilor fundamentale referitoare la domeniul comportamental al descrierilor VHDL. Se prezintă noţiunea de proces, modul de execuţie, de suspendare şi de reactivare al unui proces, problemele legate de instrucţiunea **wait** şi structurile echivalente ei, precum şi *lista de sensibilitate* a unui proces. Se studiază rolul şi importanţa variabilelor în cadrul proceselor, cu exemple concludente şi bogat comentate.

2. Consideraţii teoretice

2.1 Procese

Domeniul secvenţial este strâns legat de stilul de descriere *comportamental*. Specificaţia comportamentală constă de fapt într-o listă de operaţii care trebuie efectuate pentru a se obţine rezultatele dorite. *Procesul* constituie o modalitate formală de a realiza o listă de operaţii secvenţiale în VHDL. Un proces are un format foarte bine structurat, chiar dacă el reprezintă comportamentul unei părţi minore a proiectului.

Procesul este obiectul fundamental manipulat de către simulator, pentru care orice descriere VHDL se rezumă la un set de procese caracterizate de semnalele la care sunt sensibile şi de operaţiile secvenţiale executate de către fiecare dintre ele.

Specificarea unui proces se va face întotdeauna utilizând cuvântul cheie **process**, după care urmează o zonă de declaraţii locale procesului şi apoi cuvântul cheie **begin** (care arată momentul de început al secvenţei de instrucţiuni care se va executa), iar în final, cuvintele cheie **end process** indică terminarea procesului. Sintaxa completă a unui proces este prezentată în continuare.

În partea declarativă a procesului nu se admite declararea oricăror obiecte. Cel mai important aspect este acela că în această zonă este interzisă

declararea semnalelor; în schimb, este permisă declararea variabilelor şi a sub-programelor interne.

```
{etichetă:} process {listă_de_sensibilitate}
      ...
      ... Zona de declaraţii locale procesului
      ...
begin
      ...
      ... Instrucţiuni secvenţiale
      ...
end process {etichetă};
```

Imediat înaintea cuvântului cheie process, se permite (opţional) cuvântul cheie **postponed**, pentru a arăta caracterul „amânat" al procesului. Acelaşi cuvânt cheie **postponed** poate să apară şi la sfârşitul procesului, în instrucţiunea **end process** (aceasta, în scopuri de documentare).

```
{etichetă:} {postponed} process {listă_de_sensibilitate}
      ...
      ... Zona de declaraţii locale procesului
      ...
begin
      ...
      ... Instrucţiuni secvenţiale
      ...
end {postponed} process {etichetă};
```

De vreme ce construcţiile VHDL reprezintă nişte sisteme reale, conceptul de *terminare a procesului* este diferit de cel din limbajele de programare clasice. În VHDL procesul este conceput astfel încât după executarea ultimei instrucţiuni el va relua imediat execuţia primei instrucţiuni din lista de instrucţiuni secvenţiale. Prin urmare, *un proces nu se termină niciodată*. Figura 6.1 ilustrează aceste noţiuni într-o formă grafică.

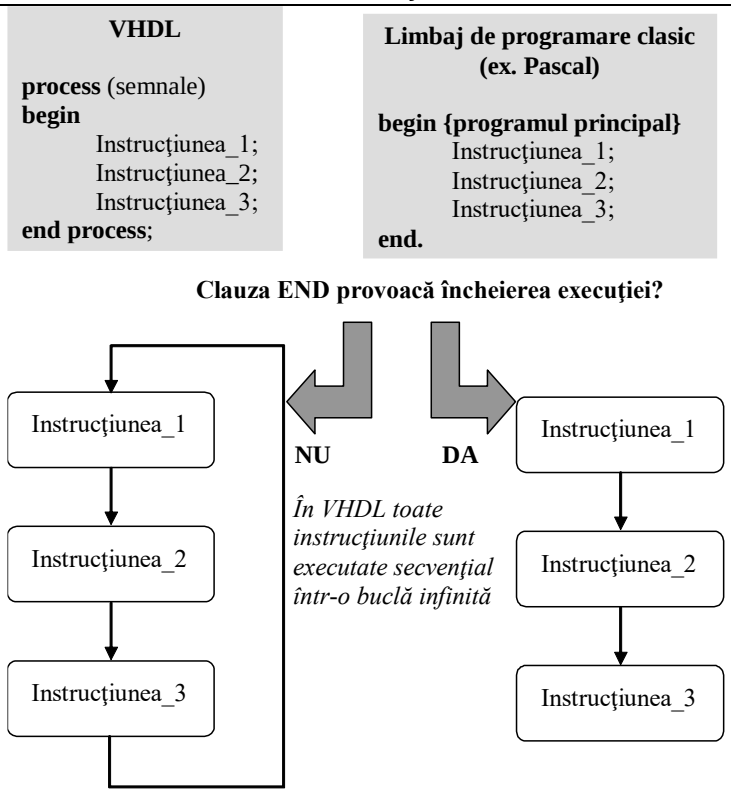

Figura 6.1 *Execuţia unui proces în VHDL*

Standardul VHDL afirmă că:

- Un proces nu poate conţine decât instrucţiuni secvenţiale;
- Orice instrucţiune concurentă poate întotdeauna să fie tradusă în termenii unui proces (este „procesul său echivalent”);
- Un proces "trăieşte" nedefinit (este „global”); durata sa de viaţă este cea a simulării. El nu se poate termina. Într-adevăr, o dată ajuns în punctul terminal (după executarea ultimei instrucţiuni, **end process**) el se execută din nou de la început (de la cuvântul cheie **begin**). Procesul este întotdeauna ciclic (iterativ). Dacă procesul posedă o instrucţiune **wait** fără nici o condiţie, el se va executa o singură dată şi apoi se va suspenda definitiv. Dacă procesul nu are nici o instrucţiune **wait**, atunci el se va executa în buclă infinită.

Vom vedea în continuare că un proces poate fi *suspendat* şi apoi *reactivat* - astfel, comportarea lui devine mai apropiată de cea a sistemelor din lumea reală.

2.2 Suspendarea şi reactivarea proceselor

În mod obişnuit, dispozitivele electronice operează în buclă infinită. După activarea lor, ele efectuează operaţiile specifice pentru care au fost proiectate, după care revin la o stare de *aşteptare a îndeplinirii condiţiilor de activare*. După încheierea operaţiilor curente, dispozitivele îşi suspendă funcţionarea şi îşi reiau activitatea atunci când condiţiile specificate sunt din nou îndeplinite.

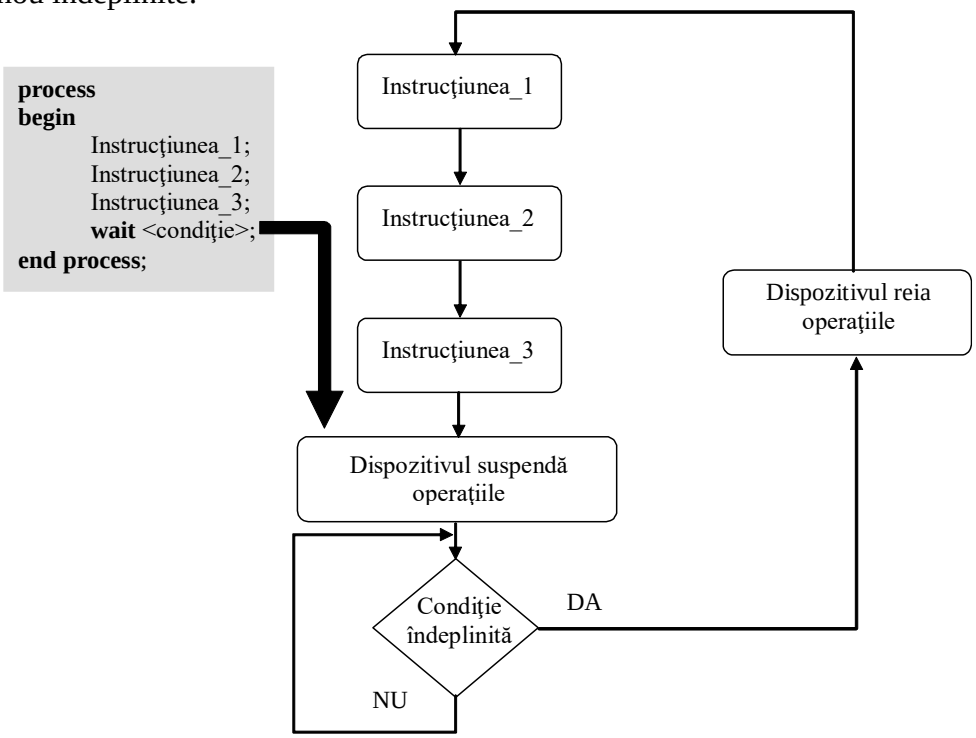

Figura 6.2 *Modul de operare al instrucţiunii* ***wait***

Acest mod de funcţionare a dispozitivelor este descris în VHDL de către instrucţiunea **wait**, care suspendă procesul atunci când toate operaţiile prevăzute au fost efectuate şi apoi îl reactivează când condiţiile specificate sunt din nou îndeplinite. Reactivarea unui proces poate fi realizată prin

intermediul unei singure instrucţiuni de forma „*aşteaptă (**wait**) îndeplinirea condiţiilor specifice*”.

2.3 Instrucţiunea *wait*

Înainte ca un proces suspendat să poată fi reactivat, el trebuie să aştepte îndeplinirea condiţiei specifice. În această direcţie, instrucţiunea **wait** acţionează astfel:

a) Stopează necondiţionat (suspendă) execuţia procesului;
b) Evaluează condiţiile care vor reactiva procesul.

Dacă procesul conţine o instrucţiune **wait**, el va executa toate instrucţiunile, în ordine, până la întâlnirea instrucţiunii **wait**. Apoi, procesul va fi suspendat şi se va aştepta îndeplinirea condiţiei specificate în cadrul instrucţiunii **wait**. În momentul în care condiţiile respective sunt îndeplinite, procesul este reactivat şi îşi execută toate instrucţiunile până când întâlneşte din nou instrucţiunea **wait**.

De vreme ce în mediul industrial real există o mare varietate de condiţii care pot declanşa reactivarea unui proces, limbajul VHDL oferă trei tipuri de instrucţiuni **wait**:

- ***wait for*** *expresie_de_tip_TIME*. În cadrul acestei instrucţiuni, se aşteaptă scurgerea unui anumit interval de timp până la reactivarea procesului;
- ***wait until*** *condiţie_de_tip_BOOLEAN*. În cadrul acestei instrucţiuni se aşteaptă până la îndeplinirea unei anumite condiţii de tip BOOLEAN (până când condiţia respectivă devine adevărată, TRUE);
- ***wait on*** *listă_de_sensibilitate*. În cadrul acestei instrucţiuni se aşteaptă până când unul dintre semnalele din cadrul listei de sensibilitate suferă o modificare a valorii sale.

Prin combinarea acestor condiţii se poate forma o a patra construcţie numită *condiţie complexă*.

Sintaxa completă a instrucţiunii **wait** este prezentată mai jos:

```
wait {on listă_de_semnale}{until condiţie_booleană} {for timp};
```

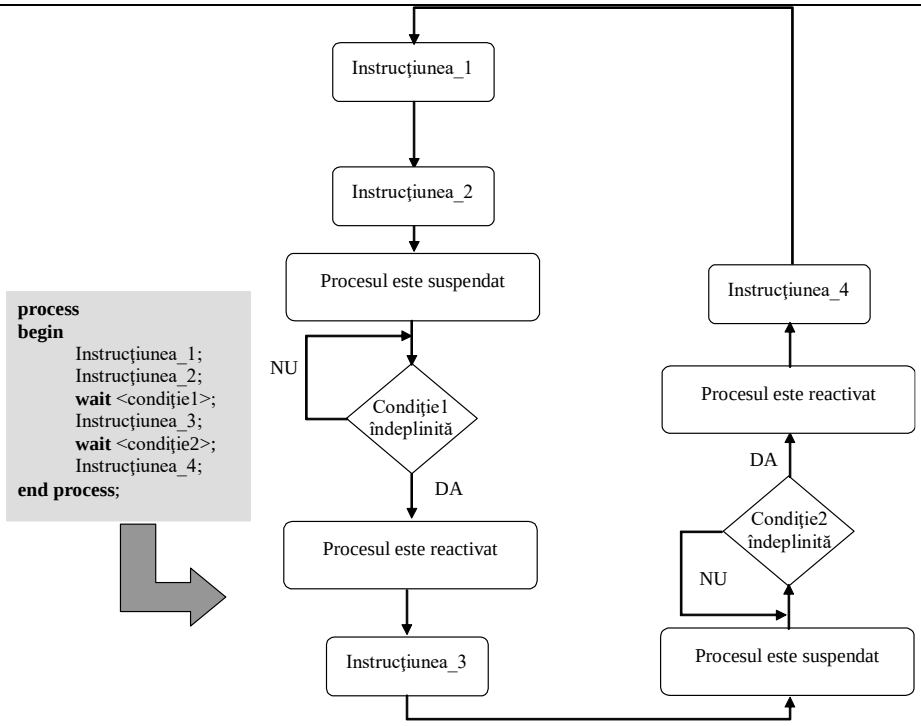

Figura 6.3 *Execuţia unui proces care conţine mai multe instrucţiuni* ***wait***

Observaţie

Instrucţiunea ***wait until*** *condiţie* suspendă procesul până când respectiva condiţie DEVINE adevărată datorită unei MODIFICĂRI a oricărui semnal dintre cele care participă la formarea condiţiei. Trebuie subliniată diferenţa dintre termenii „devine” şi „se modifică”: dacă un semnal rămâne neschimbat, instrucţiunea **wait until** nu va reactiva procesul, chiar dacă se îndeplineşte condiţia!

Toate cele patru cazuri sunt ilustrate în exemplele de mai jos:

```
signal SIG1, SIG2: BIT;
...
proc1: process
begin
      ...
      wait on SIG1, SIG2;
end process proc1;
```

După executarea tuturor instrucţiunilor, procesul va fi suspendat în momentul în care se ajunge la instrucţiunea **wait** şi va fi reactivat atunci când unul dintre semnalele SIG1 sau SIG2 îşi va schimba valoarea.

```
wait until ENABLE = '1';
```

În acest exemplu instrucţiunea **wait** va reactiva procesul atunci când semnalul ENABLE îşi va modifica valoarea la '1'.

```
wait for 50 ns;
```

În acest exemplu, procesul va fi suspendat pe o durată de 50 de nanosecunde.

```
proc2: process
begin
      wait on SIG1, SIG2 until CLOCK = '1';
      ...
end process proc2;
```

Procesul proc2 este reactivat după o modificare a semnalului SIG1 sau după o modificare a semnalului SIG2, însă numai atunci când valoarea semnalului CLOCK va fi egală cu '1'.

2.4 Localizarea instrucţiunii *wait*

După executarea ultimei instrucţiuni a procesului (care este localizată imediat înaintea instrucţiunii **end process**), urmează să fie executată prima instrucţiune (localizată imediat după instrucţiunea **begin**). La prima vedere, acest mod de funcţionare poate sugera ideea că nu contează unde este localizată instrucţiunea **wait**: la începutul sau la sfârşitul procesului. În realitate, de cele mai multe ori această presupunere nu este adevărată. De exemplu, dacă simulatorul întâlneşte o instrucţiune **wait** chiar la începutul procesului, atunci procesul va fi suspendat imediat fără ca vreo instrucţiune să fie executată. Pe de altă parte, dacă instrucţiunea **wait** este localizată undeva pe la mijlocul secvenţei de instrucţiuni a procesului, vor exista câteva instrucţiuni care vor fi executate înainte de suspendarea acestuia.

Instrucţiunea **wait** poate apărea oriunde în cadrul unui proces. De regulă, ea este plasată la începutul sau la sfârşitul secvenţei de instrucţiuni. În plus, este permisă apariţia mai multor instrucţiuni **wait** în cadrul unui proces.

```
process
      begin
            wait on SIGA;
            Instrucţiunea1;
            Instrucţiunea2;
            Instrucţiunea3;
end process;
```

În acest caz, dacă instrucţiunea **wait** apare ca prima instrucţiune din proces, atunci procesul va fi suspendat imediat după lansarea lui în execuţie.

```
process
      begin
            Instrucţiunea1;
            Instrucţiunea2;
            Instrucţiunea3;
            wait on SIGB;
end process;
```

În acest caz, dacă instrucţiunea **wait** apare ca ultima instrucţiune din proces, atunci toate instrucţiunile vor fi executate o dată înainte de suspendarea procesului.

2.5 Lista de sensibilitate a unui proces

Instrucţiunea ***wait on*** *listă_de_sensibilitate* este probabil cea mai des folosită condiţie pentru reactivarea proceselor; prin urmare, VHDL oferă o construcţie numită *lista de sensibilitate a unui proces*. Această listă va fi specificată alături de cuvântul cheie **process** şi este întru totul identică cu lista de sensibilitate din cadrul instrucţiunii ***wait on*** *listă_de_sensibilitate* care apare *la sfârşitul procesului*.

Observaţii

Datorită regulilor de execuţie a proceselor (descrise anterior), o instrucţiune **wait on** plasată la începutul procesului NU este echivalentă cu un proces cu listă de sensibilitate, chiar dacă semnalele din lista de sensibilitate a procesului sunt aceleaşi cu cele care ar apărea în lista de sensibilitate a instrucţiunii **wait on**.

Un proces înzestrat cu listă de sensibilitate nu poate să conţină nici o altă instrucţiune **wait** explicită.

În exemplul de mai jos, procesul BISTABIL_D este sensibil la semnalele RST şi CLK, ceea ce înseamnă că orice eveniment survenit pe oricare dintre aceste două semnale va provoca reactivarea procesului. Cele două procese de mai jos sunt echivalente:

```
-- Primul proces, cu listă de sensibilitate
BISTABIL_D: process (CLK, RST)
      begin
            if RST = '1'
                  then Q <= '0';
            elsif (CLK'EVENT) and (CLK = '1')
                  then Q <= D;
            end if;
end process BISTABIL_D;
-- Procesul echivalent, exprimat fără listă de sensibilitate
BIST_D: process
      begin
            if RST = '1'
                  then Q <= '0';
            elsif (CLK'EVENT) and (CLK = '1')
                  then Q <= D;
            end if;
            wait on CLK, RST;
end process BIST_D;
```

În general, cele mai multe procese descrise în VHDL posedă listă de sensibilitate.

La demararea simulării, procesul este executat o singură dată, deoarece o listă de sensibilitate este echivalentă cu o instrucţiune **wait** localizată la sfârşitul procesului, iar instrucţiunea **wait** opreşte execuţia procesului.

În continuare, procesul este suspendat până când oricare dintre semnalele din lista de sensibilitate îşi modifică valoarea. O astfel de modificare va declanşa reactivarea procesului, deci toate instrucţiunile din cadrul procesului, de la început până la sfârşit, vor fi executate în ordinea specificată.

Observaţie

„Toate instrucţiunile" înseamnă într-adevăr toate instrucţiunile din cadrul procesului, nu numai cele legate de semnalul care a reactivat procesul!

După executarea ultimei instrucţiuni, procesul va fi din nou suspendat.

Există o serie de restricţii care trebuie cunoscute:

- Semnalele care fac parte din lista de sensibilitate a procesului trebuie să fie *statice*. Un semnal este static dacă numele său este cunoscut la compilarea unităţii de proiectare în care se găseşte. De exemplu, semnalul TAB(I), unde TAB este un tablou şi I este valoarea returnată de o funcţie sau un parametru de procedură, nu este static;
- Instrucţiunea **wait** nu poate fi utilizată în interiorul unui proces decât dacă acesta nu are listă de sensibilitate după cuvântul cheie **process**. De asemenea, instrucţiunea **wait** nu poate fi utilizată în procedurile eventual apelate de procesul respectiv (aceasta, tot în cazul în care lista de sensibilitate nu este vidă). Această restricţie este riguros verificată de către analizoare. Într-adevăr, o funcţie trebuie să returneze *imediat* un rezultat (în sensul simulării). Orice altă variantă ar deschide calea unor efecte laterale deosebit de greu de controlat.

În practică se foloseşte deseori scrierea *fără* listă de sensibilitate, căci în acest mod se obţine un câştig în privinţa generalităţii.

2.6 Procese pasive

În partea de specificare a unei entităţi nu sunt admise decât anumite instrucţiuni concurente. De fapt, nu sunt autorizate decât acele instrucţiuni al căror proces este *pasiv*.

Să ne imaginăm că avem un fişier sursă VHDL în care există o mulţime de procese. Să presupunem că la un moment dat sosesc stimuli de intrare. Aceste semnale se află pe lista de sensibilitate a anumitor procese, care vor fi declanşate. Procesele sunt activate; execuţia lor modifică eventual anumite alte semnale (cele care se află în membrul stâng al instrucţiunilor de asignare). Aceste semnale modificate se află la rândul lor în lista de sensibilitate a altor procese, care sunt la rândul lor activate etc.

Se numesc *pasive* acele procese în care nici un semnal nu va apărea în membrul stâng al unei instrucţiuni de asignare. Aceste procese se numesc pasive nu pentru că nu se execută, ci pentru că execuţia lor nu va antrena execuţia altor procese. Instrucţiunea concurentă **assert** şi, în anumite condiţii (neutilizarea parametrilor de mod **out** sau **inout**) apelul concurent de proceduri, au procese echivalente pasive.

2.7 Procese amânate

Pentru asimilarea cunoştinţelor expuse în această secţiune este indicată reluarea informaţilor din lucrarea nr. 3 referitoare la întârzierea „delta”.

În continuare vom lucra pe următorul exemplu:

```
S2 <= not S1; --Teoretic, S1 nu este niciodată egal cu S2
S1 <= '0', '1' after 10 ns;--Impunem o formă de undă pt. S1
-- Testăm totuşi...
assert S2 = not S1 report "Comportare neaşteptată!";
```

Instrucţiunea **assert** va genera mesajul său aparent neobişnuit la momentele indicate în figura 6.4.

Într-adevăr, verificarea condiţiei din instrucţiunea **assert** se efectuează la fiecare schimbare a valorii lui S1 sau a lui S2. În consecinţă, în acest caz precis, acest lucru se întâmplă în decursul anumitor întârzieri „delta”, care nu sunt, de fapt, decât nişte etape tranzitorii de calcul ale simulatorului, care pot provoca alarme false.

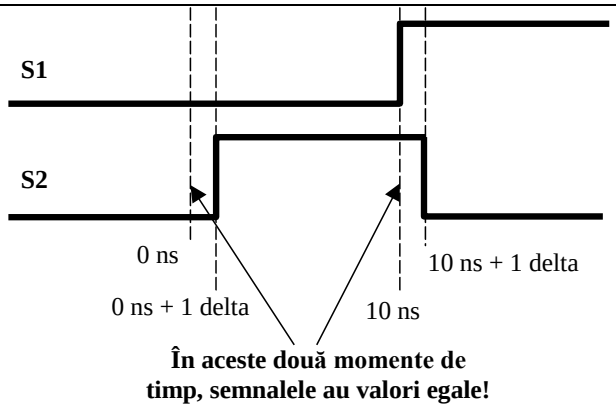

Figura 6.4 *Întârzierile „delta” provoacă alarme false*

Pentru evitarea acestor situaţii au fost introduse *procesele amânate*. Un proces amânat nu este activat decât în momentul ultimei întârzieri „delta” a unui ciclu de simulare, adică chiar înainte ca simulatorul să facă să avanseze timpul de simulare curent (TIME).

Pe lângă procesul în sine, mai există încă trei instrucţiuni concurente care pot fi transformate în procese amânate:

- instrucţiunea concurentă **assert**;
- apelul concurent de procedură;
- asignarea concurentă de valori unui semnal.

Sintaxa este foarte simplă: nu trebuie decât să se adauge cuvântul cheie **postponed** imediat după etichetă (opţional).

```
P: postponed process ...
postponed assert ...
postponed PROCEDURA_MEA...
```

Prin urmare, următoarea linie de cod permite evitarea alarmelor false din schema precedentă:

```
postponed assert S2 = not S1 report „Acest mesaj nu va mai
                                    apărea!”;
```

Există două restricţii de care este necesar să ţinem seama când utilizăm acest gen de procese:

- Un proces amânat nu poate să conţină instrucţiuni de asignare cu întârziere nulă. Dacă un asemenea proces ar conţine o astfel de

instrucţiune, ar putea să genereze la rândul lui o nouă întârziere „delta”… chiar după ultima!

- Valoarea atributelor predefinite asupra semnalelor nu poate fi utilizată în cadrul acestor procese. În practică, ea creează numeroase şi dificile probleme de definiţie.

2.8 Semnale şi variabile în procese

Scopul principal al descrierii comportamentale VHDL este acela de a descrie reacţia ieşirilor la intrări. Atât intrările cât şi ieşirile sunt semnale, astfel că reacţiile semnalelor de ieşire sunt date sub forma unor instrucţiuni de asignare de valori unor semnale. Semnalele şi instrucţiunile de asignare apar în interiorul proceselor. Cu toate acestea, utilizarea semnalelor în procese este guvernată de trei restricţii importante:

1. Semnalele nu pot fi declarate în interiorul proceselor;
2. Orice asignare a unei valori unui semnal are efect numai atunci când procesul se suspendă. Până în momentul suspendării, toate semnalele îşi păstrează valorile anterioare;
3. Numai ultima asignare a unei valori unui anumit semnal din cadrul procesului este luată în considerare. Aşadar, nu are rost să asignăm mai mult decât o singură valoare unui semnal în cadrul procesului respectiv.

Dacă unui semnal îi este asignată o valoare într-un proces, iar semnalul respectiv se află pe lista de sensibilitate a acelui proces, atunci orice schimbare a valorii semnalului va provoca reactivarea procesului:

```
signal A, B, C, X, Y, Z: INTEGER;
process (A, B, C)
       begin
              X <= A + 1;
              Y <= A * B;
              Z <= C - X;
              B <= Z * C;
              Y <= B;
end process;
```

Atunci când procesul va fi activat de o modificare a valorii semnalului C, aceasta va provoca o modificare a valorii semnalului B din interiorul procesului. Acest eveniment survenit pe semnalul B va provoca la rândul lui o nouă reactivare a procesului, deoarece semnalul B face parte din lista de sensibilitate a procesului.

La execuţia acestui proces, asignările de semnale sunt efectuate secvenţial, în ordine, însă cea de-a doua asignare (`Y <= A * B;`) nu va fi executată niciodată deoarece se va activa numai *ultima asignare* a semnalului Y. Mai mult, în cea de-a treia asignare (`Z <= C - X;`) va fi folosită numai valoarea anterioară a lui X, deoarece prima asignare (`X <= A + 1;`) va avea efectiv loc numai la suspendarea procesului, adică după parcurgerea integrală şi efectuarea tuturor instrucţiunilor din lista de instrucţiuni secvenţiale.

Restricţiile impuse asupra semnalelor au un impact deosebit de puternic asupra aplicaţiilor lor practice. Imposibilitatea de a declara semnale în cadrul proceselor nu constituie o problemă majoră, însă regulile de asignare de valori semnalelor au consecinţe dintre cele mai serioase.

De vreme ce semnalele pot stoca numai ultima lor valoare asignată, ele nu pot fi utilizate pentru stocarea datelor temporare sau intermediare în interiorul proceselor.

Un alt inconvenient major îl constituie faptul că noile valori le sunt asignate semnalelor nu în cadrul execuţiei instrucţiunilor de asignare, ci doar după suspendarea procesului. Aceasta face ca analiza proiectului să devină destul de dificilă, aşa cum se poate vedea din exemplul prezentat mai jos.

```
signal A, B, C, D, E: INTEGER;
process (C, D)
      begin
            A <= 2;           -- Instrucţiunea 1
            B <= A + C;       -- Instrucţiunea 2
            A <= D + 1;       -- Instrucţiunea 3
            E <= A * 2;       -- Instrucţiunea 4
end process;
```

Să presupunem că, iniţial, toate cele 5 semnale (A, B, C, D şi E) au valoarea 1 (A = 1, B = 1, C = 1, D = 1 şi E = 1). La un anumit moment de timp, valoarea semnalului D se modifică de la 1 la 2. Acest eveniment declanşează activarea procesului, pentru că semnalul D se află pe lista de sensibilitate a acestuia. Prin urmare, instrucţiunile din cadrul procesului vor fi executate secvenţial, după cum urmează:

Instrucţiunea 1 (`A <= 2;`): semnalul A este pregătit să ia valoarea 2, însă această asignare nu are deocamdată nici un efect asupra valorii

semnalului A. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1).

Instrucţiunea 2 (`B <= A + C;`): semnalul B este pregătit să ia valoarea (A + C = 2), însă această asignare nu are deocamdată nici un efect asupra valorii semnalului B. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1).

Instrucţiunea 3 (`A <= D + 1;`): semnalul A este pregătit să ia valoarea (D + 1 = 3). Această asignare va „suprascrie" (sau se mai poate spune că va înlocui) asignarea precedentă a semnalului A (cea din cadrul instrucţiunii 1), însă ea nu are deocamdată nici un efect asupra valorii semnalului A. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1).

Instrucţiunea 4 (`E <= A * 2;`): semnalul E este pregătit să ia valoarea (A * 2 = 2), însă această asignare nu are deocamdată nici un efect asupra valorii semnalului E. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1).

Instrucţiunea 5 este „ascunsă": este vorba despre o instrucţiune **wait on** C, D; (ştim că prezenţa listei de sensibilitate este echivalentă cu o instrucţiune **wait on** plasată la sfârşitul procesului). În acest punct, procesul este suspendat şi toate asignările de semnale, care până acum au fost doar „pregătite", vor deveni efective. Prin urmare,

- semnalul A va căpăta valoarea 3 (pregătită în cadrul instrucţiunii 3);
- semnalul B va căpăta valoarea 2 (pregătită în cadrul instrucţiunii 2);
- semnalul E va căpăta valoarea 2 (pregătită în cadrul instrucţiunii 4).

Situaţia finală va fi deci: (A = 3, B = 2, C = 1, D = 2 şi E = 2).

Deoarece nici A, nici B şi nici E nu se află pe lista de sensibilitate a procesului, procesul rămâne suspendat până la o nouă activare a sa, care va fi dată de apariţia unui eveniment pe semnalele C sau D (aceste evenimente pot fi provocate de către asignările de valori semnalelor C şi D în cadrul altor procese).

De vreme ce semnalele nu pot fi declarate în interiorul proceselor şi întrucât asignarea valorilor lor nu are loc decât după suspendarea procesului din care fac parte, a apărut necesitatea unui obiect care să poată fi declarat în interiorul unui proces şi care să ofere posibilitatea stocării temporare a datelor. Un asemenea obiect există în VHDL şi se numeşte *variabilă*.

În exemplul de mai jos, unuia dintre semnale i se asignează de mai multe ori noi valori. Cu toate acestea, respectivele asignări nu sunt vizibile

în exteriorul procesului. Această problemă poate fi rezolvată prin introducerea variabilelor:

```
entity E1 is
      port (A: in NATURAL := 1;
            B : inout NATURAL := 1);
end entity E1;
architecture ARH1 of E1 is
begin
P1: process (A)
      begin
       B <= A + 2;
       B <= B + 3;
       B <= B * 2;
       B <= B + 1; -- Numai această ultimă asignare are loc
end process P1;
end architecture ARH1;
-- Rezolvarea problemei prin introducerea de variabile
architecture ARH2 of E1 is
begin
P2: process (A)
variable B_VAR: NATURAL;
      begin
       B_VAR := A + 2;
       B_VAR := B_VAR + 3;
       B_VAR := B_VAR * 2;
       B <= B_VAR + 1;
end process P2;
end architecture ARH2;
```

Variabilele sunt asemănătoare semnalelor, însă cu deosebirile expuse mai sus. Datorită acestor caracteristici, ele pot fi utilizate în anumite aplicaţii în care utilizarea semnalelor nu poate fi satisfăcătoare. De exemplu, ele pot fi utilizate pentru descrierea algoritmilor în interiorul proceselor.

Întrucât variabilele sunt limitate numai la procese, ele sunt declarate în interiorul proceselor (în zona de declaraţii a proceselor) şi nu pot fi folosite în afara acestora. O declaraţie de variabilă seamănă cu declaraţia de semnal, cu singura diferenţă că se foloseşte cuvântul cheie **variable**, în locul cuvântului cheie **signal**.

Asignarea de valori unei variabile se face prin intermediul simbolului „:=”. Instrucţiunea de asignare de valori unei variabile modifică pe loc valoarea acelei variabile (instantaneu - nu mai are loc amânarea

executării sale efective până la suspendarea procesului) şi fiecărei variabile i se pot asigna noi valori ori de câte ori este necesar.

O variabilă poate fi declarată ca având orice tip sau subtip posibil, atât constrâns cât şi neconstrâns:

```
variable MEM is array(NATURAL range<>,NATURAL range<>) of
STD_LOGIC;
variable DELTA1, DELTA2: TIME;
variable RAM1: MEM (0 to 1023, 0 to 7);
```

Valoarea iniţială a unei variabile poate fi dată de o expresie statică globală. Expresia trebuie să fie de acelaşi tip ca şi variabila însăşi:

```
variable COND: BOOLEAN := TRUE;
```

În exemplul de mai jos, se prezintă mai întâi variabile cu nume simple, apoi variabile cu nume parţiale (*slice names*) şi în final variabile cu nume indexate.

```
variable X, Y: REAL;
variable A, B: BIT_VECTOR (0 to 7);
-- Variabile cu nume simple
X := 108.0;
A := B;
A := "11110000";
-- Variabile cu nume parţiale
A(3 to 6) := ('1', '1', '1', '1');
A(0 to 5) := B(2 to 7);
-- Variabile cu nume indexate
A(7) := '0';
B(0) := A(6);
```

Revenind la un exemplu anterior, să vedem modificările care apar în cazul introducerii variabilelor:

```
signal A, B, C, D, E: INTEGER;
process (C, D)
variable A_VAR, B_VAR, E_VAR: INTEGER := 0;
      begin
            A_VAR := 2;             -- Instrucţiunea 1
            B_VAR := A_VAR + C;     -- Instrucţiunea 2
```

```
            A_VAR := D + 1;          -- Instrucţiunea 3
            E_VAR := A_VAR * 2;      -- Instrucţiunea 4
            A <= A_VAR;              -- Instrucţiunea 5
            B <= B_VAR;              -- Instrucţiunea 6
            E <= E_VAR;              -- Instrucţiunea 7
end process;
```

Să presupunem din nou că, iniţial, toate cele 5 semnale (A, B, C, D şi E) au valoarea 1 (A = 1, B = 1, C = 1, D = 1 şi E = 1). La un anumit moment de timp, valoarea semnalului D se modifică de la 1 la 2. Acest eveniment declanşează activarea procesului, pentru că semnalul D se află pe lista de sensibilitate a acestuia. Prin urmare, instrucţiunile din cadrul procesului vor fi executate secvenţial, după cum urmează:

Instrucţiunea 1 (`A_VAR := 2;`): variabila A_VAR ia valoarea 2. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 2, B_VAR = 0 şi E_VAR = 0).

Instrucţiunea 2 (`B_VAR := A_VAR + C;`): variabila B_VAR ia valoarea (A_VAR + C = 3). În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 2, B_VAR = 3 şi E_VAR = 0).

Instrucţiunea 3 (`A_VAR := D + 1;`): variabila A_VAR ia valoarea (D + 1 = 3). Această asignare va „suprascrie” (sau se mai poate spune că va înlocui) valoarea precedentă a variabilei A (cea din cadrul instrucţiunii 1). În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 3, B_VAR = 3 şi E_VAR = 0).

Instrucţiunea 4 (`E_VAR := A_VAR * 2;`): variabila E ia valoarea (A_VAR * 2 = 6). În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 3, B_VAR = 3 şi E_VAR = 6).

Instrucţiunea 5 (`A <= A_VAR;`): semnalul A este pregătit să ia valoarea A_VAR = 3, însă această asignare nu are deocamdată nici un efect asupra valorii semnalului A. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 3, B_VAR = 3 şi E_VAR = 6).

Instrucţiunea 6 (`B <= B_VAR;`): semnalul B este pregătit să ia valoarea B_VAR = 3, însă această asignare nu are deocamdată nici un efect

asupra valorii semnalului B. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 3, B_VAR = 3 şi E_VAR = 6).

Instrucţiunea 7 (`E <= E_VAR;`): semnalul E este pregătit să ia valoarea E_VAR = 6, însă această asignare nu are deocamdată nici un efect asupra valorii semnalului E. În continuare, semnalele îşi păstrează aceleaşi valori ca la activarea procesului (A = 1, B = 1, C = 1, D = 2 şi E = 1; A_VAR = 3, B_VAR = 3 şi E_VAR = 6).

Instrucţiunea 8 este „ascunsă": este vorba despre o instrucţiune **wait on** C, D; (ştim că prezenţa listei de sensibilitate este echivalentă cu o instrucţiune **wait on** plasată la sfârşitul procesului). În acest punct, procesul este suspendat şi toate asignările de semnale, care până acum au fost doar „pregătite", vor deveni efective. Prin urmare:

- semnalul A va căpăta valoarea 3 (pregătită în cadrul instrucţiunii 5);
- semnalul B va căpăta valoarea 3 (pregătită în cadrul instrucţiunii 6);
- semnalul E va căpăta valoarea 6 (pregătită în cadrul instrucţiunii 7).

Situaţia finală va fi deci: (A = 3, B = 3, C = 1, D = 2 şi E = 6).

Deoarece A, B şi E nu se află pe lista de sensibilitate a procesului, procesul rămâne suspendat până la o nouă activare a sa, care va fi dată de apariţia unui eveniment pe semnalele C sau D (aceste evenimente pot fi provocate de către asignările semnalelor C şi D în cadrul altor procese).

În tabelul de mai jos este prezentată o comparaţie între semnale şi variabile.

SEMNALE	**VARIABILE**
Semnalul are 3 trăsături caracteristice: - tip - valoare - timp	*Variabila* are numai 2 trăsături: - tip - valoare

Există o legătură foarte strânsă între valoare şi timp: fiecare semnal are o *istorie* a valorilor pe care le-a avut în timp, în decursul simulării.	Variabila nu păstrează decât valoarea sa curentă. Nu are *istorie*.
Atâta timp cât semnalele şi variabilele sunt de acelaşi tip, ele pot fi asignate reciproc.	

3. Desfăşurarea lucrării

3.1 Se vor testa cele 2 variante de descriere prin procese pentru BISTABIL_D.

3.2 Se vor verifica cele două variante de definire ale arhitecturii pentru entitatea E1. Se vor modifica valorile semnalelor şi ale variabilelor şi se vor evidenţia diferenţele.

3.3 Se va proiecta şi simula sistemul numeric care comandă funcţionarea unei firme luminoase care funcţionează astfel:
   a) firma rămâne aprinsă o perioadă de timp $T_A = 5$ secunde.
   b) firma „pâlpâie" (sistemul de iluminare este aprins şi, alternativ, stins de 5 ori) o perioadă $T_B = T_A$
   c) procesul se reia de la a).

3.4 Se va proiecta un sistem logic secvenţial care citeşte date de pe o linie serială şi detectează apariţia secvenţei **0110** din şirul de intrare, afişând numărul de asemenea secvenţe detectate. Atenţie: în cazul apariţiei secvenţei 0110110, sistemul va detecta secvenţa dată de 2 (două) ori.