PROIECTAREA SISTEMELOR DIGITALE
Masterat ICCP - an 1

LABORATOR 5

INSTRUCȚIUNI CONCURENTE ÎN LIMBAJUL VHDL

În aceast laborator se prezintă mai întâi structura şi execuția unei arhitecturi, iar apoi sunt descrise principalele instrucțiuni concurente ale limbajului VHDL. Cea mai importantă instrucțiune concurentă este declarația unui proces. Procesele au fost prezentate în laboratorul 3, astfel încât în acesta lucrare de laborator vor fi prezentate doar principalele caracteristici ale proceselor. Alte instrucțiuni concurente sunt instrucțiunea concurentă de asignare a semnalelor, instrucțiunea **block**, instrucțiunea concurentă **assert**.

5.1. Instrucțiuni concurente

Operațiile din sistemele reale se execută în mod concurent. Limbajul VHDL modelează sistemele reale sub forma unui set de subsisteme care funcționează în mod concurent. Fiecare din aceste subsisteme poate fi specificat sub forma unui proces separat, iar comunicația dintre procese se poate realiza prin semnale. Complexitatea diferitelor procese poate fi foarte variată, de la o simplă poartă logică până la un procesor. Modelarea sistemelor reale sub această formă poate fi realizată cu ajutorul instrucțiunilor concurente.

5.1.1. Structura şi execuția unei arhitecturi

O arhitectura are două părți: o parte declarativă şi o parte descriptivă. În partea declarativă se pot declara obiecte care sunt interne arhitecturii. Partea descriptivă conține instrucțiuni concurente. Acestea definesc procesele sau blocurile interconectate care descriu funcționarea sau structura globală a sistemului.

Toate procesele dintr-o arhitectură se execută în paralel unele față de altele, dar instrucțiunile din cadrul unui anumit proces se execută secvențial. Un proces suspendat este activat din nou atunci când unul din semnalele din lista sa de sensibilitate îşi modifică valoarea. Atunci când există mai multe procese într-o arhitectură, la modificarea valorii unui semnal sunt activate toate procesele care conțin acest semnal în lista lor de sensibilitate.

Instrucțiunile din cadrul proceselor activate sunt executate secvențial, dar independent de instrucțiunile din alte procese.

În Exemplul 1 fiecare poartă din cadrul schemei unui sumator pe 1 bit este descrisă printr-un proces separat care se execută în mod concurent cu celelalte procese.

Exemplul 1:

```
entity add_1 is
        port (a, b, cin: in bit;
                s, cout: out bit);
end add_1;
architecture procese of add_1 is
        signal s1, s2, s3, s4: bit;
begin
p1: process (b, cin)
begin
```

```
        s1 <= b xor cin;
end process p1;
p2: process (a, b)
begin
        s2 <= a and b;
end process p2;
p3: process (a, cin)
begin
        s3 <= a and cin;
end process p3;
p4: process (b, cin)
begin
        s4 <= b and cin;
end process p4;
p5: process (a, s1)
begin
        s <= a xor s1;
end process p5;
p6: process (s2, s3, s4)
begin
        cout <= s2 or s3 or s4;
end process p6;
end procese;
```

Comunicația între procese poate fi realizată cu ajutorul instrucțiunilor de asignare a semnalelor. Acestea se pot utiliza atât pentru activarea proceselor, cât şi pentru sincronizarea între procese. Astfel, un semnal poate aştepta un eveniment asupra unui semnal de intrare, semnal care este asignat într-un alt proces. Acest semnal este declarat în partea declarativă a arhitecturii, şi astfel este vizibil pentru toate procesele din cadrul arhitecturii.

Observație

- Pentru comunicația între procese se pot utiliza doar semnalele, nu şi variabilele, deoarece acestea sunt obiecte locale în procesul în care sunt declarate.

5.1.2. Procese

Procesele sunt compuse din instrucțiuni secvențiale, dar declarațiile proceselor reprezintă instrucțiuni concurente. Se pot formula următoarele caracteristici ale unui proces:

- Se execută în paralel cu alte procese;
- Nu poate conține instrucțiuni concurente;
- Definește o regiune a arhitecturii unde instrucțiunile se execută secvențial;
- Trebuie să conțină o listă de sensibilitate explicită sau o instrucțiune **wait**;
- Permite descrieri funcționale, asemănătoare limbajelor de programare;
- Permite accesul la semnalele definite în arhitectura în care apare procesul şi la cele definite în entitatea cu care este asociată arhitectura.

5.1.3. Instrucțiuni concurente de asignare a semnalelor

O instrucțiune concurentă de asignare a valorii unui semnal este echivalentă cu un proces conținând acea instrucțiune. O asemenea instrucțiune este executată în paralel cu alte instrucțiuni concurente sau alte procese.

Există trei tipuri ale instrucțiunilor concurente de asignare a semnalelor:

1. instrucțiunea de asignare simplă;
2. instrucțiunea de asignare condițională;
3. instrucțiunea de asignare selectivă.

5.1.3.1. Instrucțiunea de asignare simplă

Această instrucțiune este versiunea concurentă a instrucțiunii secvențiale de asignare a semnalelor, având aceeaşi formă cu aceasta. Ca şi în cazul versiunii secvențiale, asignarea concurentă defineşte un nou driver pentru semnalul asignat. Deosebirea față de versiunea secvențială este că instrucțiunea concurentă de asignare apare în afara unui proces, în cadrul unei arhitecturi. O instrucțiune concurentă de asignare reprezintă o formă simplificată de scriere a unui proces, fiind echivalentă cu un proces care conține o singură instrucțiune secvențială de asignare.

Descrierea sumatorului de 1 bit din Exemplul 1 poate fi simplificată prin utilizarea instrucțiunilor concurente de asignare, după cum se arată în Exemplul 2.

Exemplul 2:

```
entity add_1 is
        port (a, b, cin: in bit;
                s, cout: out bit);
end add_1;
architecture concurent of add_1 is
signal s1, s2, s3, s4: bit;
begin
        s1 <= b xor cin;
        s2 <= a and b;
        s3 <= a and cin;
        s4 <= b and cin;
        s <= a xor s1;
        cout <= s2 or s3 or s4;
end concurent;
```

După cum se observă din exemplul anterior, instrucțiunile concurente de asignare apar direct în cadrul arhitecturii, şi nu în interiorul unui proces. Ordinea în care sunt scrise instrucțiunile nu are importanță. La simulare toate instrucțiunile se execută în acelaşi ciclu de simulare.

În cazul proceselor, activarea execuției acestora este determinată de modificarea valorii unui semnal din lista de sensibilitate a acestora sau de întâlnirea unei instrucțiuni **wait**. În cazul instrucțiunilor concurente de asignare, modificarea valorii unuia din semnalele care apar în partea dreaptă a asignării activează execuția asignării, fără să se specifice în mod explicit o listă de sensibilitate. Activarea unei instrucțiuni de asignare este independentă de activarea altor instrucțiuni concurente din cadrul arhitecturii.

Instrucțiunile concurente de asignare se utilizează pentru descrieri de tipul fluxului de date. Prin sinteza acestor instrucțiuni se obțin circuite combinaționale.

Observaţie

- Dacă într-o arhitectură există mai multe asignări concurente la acelaşi semnal, vor fi create drivere multiple pentru acel semnal. În asemenea cazuri, trebuie să existe o funcţie de rezoluţie predefinită sau definită de utilizator pentru tipul semnalului respectiv. Spre deosebire de asignările concurente, dacă într-un proces există mai multe asignări secvenţiale la acelaşi semnal, va avea efect doar ultima dintre acestea.

5.1.3.2. Instrucţiunea de asignare condiţională

Instrucţiunea de asignare condiţională este echivalentă funcţional cu instrucţiunea condiţională **if**, având sintaxa următoare:

semnal <= [*expresie* **when** *condiţie* **else** ...]
expresie;

Valoarea uneia din expresiile sursă se atribuie semnalului destinaţie. Expresia atribuită va fi prima a cărei condiţie booleană asociată este adevărată. La execuţia unei instrucţiuni de asignare condiţională, condiţiile sunt testate în ordinea în care ele sunt scrise. La întâlnirea primei condiţii care se evaluează la valoarea booleană TRUE, expresia corespunzătoare acesteia se asignează semnalului destinaţie. Dacă nici o condiţie nu se evaluează la valoarea TRUE, semnalului destinaţie i se asignează ultima expresie, cea a ultimei clauze **else**. Dacă există două sau mai multe condiţii care se evaluează la valoarea TRUE, va fi luată în considerare doar prima dintre acestea.

Deosebirile dintre instrucţiunea de asignare condiţională şi instrucţiunea condiţională **if** sunt următoarele:

- Instrucţiunea de asignare condiţională este o instrucţiune concurentă, deci poate fi utilizată într-o arhitectură, în timp ce instrucţiunea **if** este secvenţială, astfel că poate fi utilizată numai în interiorul unui proces.
- Instrucţiunea de asignare condiţională poate fi utilizată numai pentru asignarea valorii unor semnale, în timp ce instrucţiunea **if** poate fi utilizată pentru execuţia oricărei instrucţiuni secvenţiale.

Exemplul 3 defineşte o entitate şi două arhitecturi pentru o poartă SAU EXCLUSIV cu două intrări. Prima arhitectură utilizează o instrucţiune de asignare condiţională, iar a doua utilizează o instrucţiune if echivalentă.

Exemplul 3:

```
entity xor2 is
        port (a, b: in bit;
                x: out bit);
end xor2;
architecture arh1_xor2 of xor2 is
begin
        x <= '0' when a = b else '1';
end arh1_xor2;

architecture arh2_xor2 of xor2 is
begin
process (a, b)
begin
```

```
        if a = b then x <= '0';
        else x <= '1';
        end if;
end process;
end arh2_xor2;
```

5.1.3.3. Instrucțiunea de asignare selectivă

Ca şi instrucțiunea de asignare condițională, instrucțiunea de asignare selectivă permite selectarea unei expresii sursă pe baza unei condiții. Deosebirea constă în faptul că instrucțiunea de asignare selectivă utilizează o singură condiție pentru selecția dintre diferite opțiuni. Această instrucțiune este echivalentă funcțional cu instrucțiunea secvențială **case**. Sintaxa este următoarea:

with *expresie_de_selecție* **select**
semnal <= *expresie_1* **when** *opțiuni_1*,
...
expresie_n **when** *opțiuni_n*,
[*expresie* **when others**];

Semnalului destinație i se atribuie valoarea uneia din expresii. Expresia selectată este prima dintre cele ale căror opțiuni includ valoarea expresiei de selecție. Sintaxa opțiunilor este aceeaşi ca şi în cazul instrucțiunii **case**. Astfel, fiecare opțiune poate fi reprezentată de o valoare individuală sau de un set de valori. În cazul în care o opțiune este reprezentată de un set de valori, se pot specifica fie valorile individuale din set, separate prin simbolul "|", fie domeniul valorilor, fie o combinație a acestora. Tipul expresiei de selecție determină tipul fiecărei opțiuni.

Fiecare valoare din domeniul expresiei de selecție trebuie să fie acoperită de o opțiune. Ultima opțiune poate fi indicată prin cuvântul cheie **others**, care specifică toate valorile din domeniul expresiei de selecție rămase neacoperite de opțiunile anterioare.

Există următoarele restricții pentru diferitele opțiuni:

- Valorile din cadrul opțiunilor nu se pot suprapune.
- Dacă opțiunea **others** nu este prezentă, toate valorile posibile ale expresiei de selecție trebuie acoperite de setul de opțiuni.

Observație

- Opțiunile din cadrul instrucțiunii de asignare selectivă sunt separate prin virgule.

În Exemplul 4 se reia definiția porții SAU EXCLUSIV cu două intrări, dar în cadrul arhitecturii se utilizează o instrucțiune de asignare selectivă.

Exemplul 4:

```
entity xor2 is
        port (a, b: in bit;
                x: out bit);
end xor2;
architecture arh_xor2 of xor2 is
        signal temp: bit_vector (1 downto 0);
begin
        temp <= a & b;
```

```
        with temp select
                x <= '0' when "00",
                x <= '1' when "01",
                x <= '1' when "10",
                x <= '0' when "11";
end arh_xor2;
```

5.1.3.4. Instrucțiunea block

O instrucțiune **block** defineşte un grup de instrucțiuni concurente. Această instrucțiune este utilă pentru organizarea instrucțiunilor concurente în mod ierarhic sau pentru partiționarea unei liste de conexiuni structurale în scopul creşterii lizibilității descrierii. Sintaxa instrucțiunii **block** este următoarea:

etichetă: **block** [(*expresie_de_gardă*)]
    [*declarații*]
begin
    instrucțiuni_concurente
end block [*etichetă*];

Eticheta obligatorie denumeşte blocul. În partea de declarații se pot declara obiecte locale blocului. Declarațiile posibile sunt cele care pot apare şi în partea declarativă a unei arhitecturi, şi anume:

- Clauze **use**;
- Declarații de porturi şi generice, ca şi declarații pentru maparea acestora;
- Declarații şi corpuri de subprograme;
- Declarații de tipuri şi subtipuri;
- Declarații de constante, variabile şi semnale;
- Declarații de componente;
- Declarații de fişiere, atribute şi configurații.

Ordinea instrucțiunilor concurente dintr-un bloc nu este semnificativă, deoarece toate instrucțiunile sunt întotdeauna active. Într-un bloc pot fi declarate alte blocuri, pe mai multe nivele ierarhice. Obiectele declarate într-un bloc sunt vizibile în acel bloc şi în toate blocurile interioare. Atunci când într-un bloc interior se declară un obiect cu acelaşi nume ca şi un obiect dintr-un bloc exterior, este valabilă declarația din blocul interior.

Exemplul 5 ilustrează utilizarea blocurilor pe mai multe nivele ierarhice.

Exemplul 5:

```
B1: block
signal s: bit; -- declarația "s" în blocul B1
begin
        s <= a and b; -- "s" din blocul B1
B2: block
signal s: bit; -- declarația "s" în blocul B2
begin
        s <= c and d; -- "s" din blocul B2
B3: block
begin
        x <= s; -- "s" din blocul B2
```

```
end block B3;
end block B2;
        y <= s; -- "s" din blocul B1
end block B1;
```

Introducerea blocurilor în cadrul unei descrieri nu afectează execuția unui model la simulare, ci are doar rol de organizare a descrierii.

La declararea unui bloc se poate specifica o expresie booleană, numită *expresie de gardă*. Această expresie, specificată în paranteze după cuvântul cheie **block**, crează în mod implicit un semnal boolean numit **guard**, care se poate utiliza pentru controlul unor operații din cadrul blocului. Acest semnal poate fi citit ca orice alt semnal din cadrul instrucțiunii **block**, dar nu poate fi actualizat printr-o instrucțiune de asignare. De fiecare dată când apare o tranzacție asupra unuia din semnalele dintr-o expresie de gardă, expresia este evaluată şi semnalul **guard** este actualizat imediat. Acest semnal ia valoarea TRUE dacă valoarea expresiei de gardă este adevărată şi FALSE în caz contrar.

Semnalul **guard** poate fi declarat şi în mod explicit ca un semnal de tip boolean în cadrul instrucțiunii **block**. Avantajul acestei declarări explicite este că se poate utiliza un algoritm mai complex pentru controlul semnalului **guard** decât cel permis de o expresie booleană. În particular, se poate utiliza un proces separat pentru controlul acestui semnal.

Dacă într-un bloc nu se specifică o expresie de gardă şi semnalul **guard** nu este declarat în mod explicit, atunci acest semnal are întotdeauna valoarea TRUE.

Semnalul **guard** poate fi utilizat pentru controlul instrucțiunilor de asignare a semnalelor din cadrul blocului. O asemenea instrucțiune de asignare conține cuvântul cheie **guarded** după simbolul de asignare, care determină ca execuția instrucțiunii de asignare să fie condițională:

semnal <= **guarded** *expresie*;

Această asignare se execută numai dacă semnalul **guard** al blocului care conține expresia de gardă are valoarea TRUE.

În Exemplul 6, semnalului *out1* i se va asigna valoarea *not in1* numai dacă valoarea expresiei *clk'event and clk = '1'* va fi adevărată.

Exemplul 6:

```
front_crescator: block (clk'event and clk = '1')
begin
        out1 <= guarded not in1 after 5 ns;
        ...
end block front_crescator;
```

În Exemplul 7, semnalul **guard** este declarat in mod explicit, astfel încât i se poate asigna o valoare ca şi oricărui alt semnal.

Exemplul 7:

```
UAL: block
signal guard: boolean := FALSE;
begin
        out1 <= guarded not in1 after 5 ns;
        ...
p1: process
begin
        guard <= TRUE;
```

```
    ...
end process p1;
end block UAL;
```

Observații

- În general, sistemele de sinteză nu permit utilizarea blocurilor cu expresii de gardă. Un asemenea bloc este echivalent cu un proces cu o listă de sensibilitate şi care conține instrucțiuni condiționale. Se poate utiliza un asemenea proces în locul unui bloc cu o expresie de gardă.
- De obicei, blocurile simple sunt ignorate de sistemele de sinteză.
- Deşi blocurile se pot utiliza pentru partiționarea descrierilor, limbajul VHDL permite utilizarea unui mecanism mai puternic pentru partiționare, şi anume instanțierea componentelor.

5.1.3.5. Instrucțiunea concurentă assert

Această instrucțiune este versiunea concurentă a instrucțiunii secvențiale **assert**, având aceeaşi sintaxă ca şi versiunea sa secvențială:

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

Instrucțiunea concurentă **assert** se va executa ori de câte ori se modifică unul din semnalele din cadrul expresiei condiționale, spre deosebire de instrucțiunea secvențială **assert**, care se va executa atunci când se ajunge la această instrucțiune în cadrul unui proces sau subprogram.

5.2. Teme propuse

5.2.1. Analizați exemplele prezentate. Simulați Exemplele 2, 3, 4. Utilizați sistemul *Active-HDL* pentru compilarea cu succes a acestor descrieri.