PROIECTAREA SISTEMELOR DIGITALE
Masterat ICCP - an 1

LABORATOR 6

DESCRIEREA UNOR CIRCUITE SECVENȚIALE

6.1. Circuite secvențiale sincrone şi asincrone

Circuitele secvențiale reprezintă acea categorie de circuite logice care cuprind elemente de memorare.

Efectul de memorare se datorează unor legături inverse (bucle de reacție) prezente în schemele logice ale acestor circuite. Semnalele generate la ieşirile unui circuit secvențial depind atât de semnalele de intrare, cât şi de starea circuitului.

Starea prezentă a circuitului este determinată de o stare anterioară şi de valorile semnalelor de intrare. În cazul circuitelor secvențiale sincrone, modificarea stării se poate realiza la momente bine definite de timp sub controlul unui semnal de ceas. În cazul circuitelor secvențiale asincrone, modificarea stării poate fi cauzată de schimbarea aleatoare în timp a valorii unui semnal de intrare. Comportamentul unui circuit asincron este în general mai puțin sigur, evoluția stării fiind influențată şi de timpii de întârziere ai componentelor circuitului. Trecerea între două stări stabile se poate realiza printr-o succesiune de stări instabile, aleatoare.

Circuitele secvențiale sincrone sunt mai fiabile şi au un comportament predictiv. Toate elementele de memorare ale unui circuit sincron îşi modifică simultan starea, ceea ce elimină apariția unor stări intermediare instabile. Prin testarea semnalelor de intrare la momente bine definite de timp se reduce influența întârzierilor şi a eventualelor zgomote.

Există două tehnici de proiectare a circuitelor secvențiale: *Mealy* şi *Moore*. În cazul circuitelor secvențiale Mealy, semnalele de ieşire depind atât de starea curentă, cât şi de intrările prezente. În cazul circuitelor secvențiale Moore, ieşirile sunt dependente numai de starea curentă, fără să depindă în mod direct de intrări. Metoda Mealy permite implementarea unui anumit circuit printr-un număr minim de elemente de memorare (bistabile), însă eventualele variații necontrolate ale semnalelor de intrare se pot transmite semnalelor de ieşire. Proiectarea prin metoda Moore necesită mai multe elemente de memorare pentru acelaşi tip de comportament, dar funcționarea circuitului este mai sigură.

6.1.2. Bistabile

În Exemplul 1 se descrie un bistabil sincron de tip D acționat pe frontul crescător al semnalului de ceas

Exemplul 1:

```
library ieee;
use ieee.std_logic_1164.all;
entity bist_d is
        port (d: in std_logic;
                clk: in std_logic;
                q: out std_logic);
end bist_d;
```

```
architecture exemplu of bist_d is
begin
process (clk)
begin
        if (clk'event and clk = '1') then
                q <= d;
end if;
end process;
end exemplu;
```

Procesul utilizat pentru descrierea bistabilului este sensibil numai la modificările semnalului de ceas *clk*. Tranziția semnalului de intrare *d* nu determină activarea procesului. Expresia *clk'event* şi lista de sensibilitate sunt redundante, deoarece ambele detectează modificarea semnalului de ceas. Unele sisteme de sinteză ignoră însă lista de sensibilitate a procesului, motiv pentru care trebuie inclusă expresia *clk'event* pentru descrierea evenimentelor acționate pe frontul semnalului de ceas.

Pentru descrierea unui circuit *latch* acționat pe nivel, se elimină condiția *clk'event* şi se inserează intrarea de date *d* în lista de sensibilitate a procesului, după cum se arată în Exemplul 2.

Exemplul 2:

```
architecture exemplu of latch_d is
begin
process (clk, d)
begin
        if (clk = '1') then
                q <= d;
        end if;
end process;
end exemplu;
```

În exemplele 1 şi 2 nu există o condiție **else**. Fără această condiție, este specificat în mod implicit un element de memorie (care va păstra valoarea semnalului *q*). Cu alte cuvinte, următorul fragment:

```
if (clk'event and clk = '1') then
        q <= d;
end if;
```

are aceeaşi semnificație pentru simulare ca şi fragmentul:

```
if (clk'event and clk = '1') then
        q <= d;
else
        q <= q;
end if;
```

Aceasta este în concordanță cu modul în care funcționează un bistabil de tip D. Cele mai multe sisteme de sinteză nu permit utilizarea unei expresii else după condiția **if** (*clk'event and clk = '1'*), deoarece implementarea unei asemenea descrieri poate fi ambiguă. Pentru a evita generarea unor elemente de memorare (atunci când nu este necesar), trebuie să se asigneze valori variabilelor sau semnalelor în fiecare ramură condițională.

6.1.3. Registre

În Exemplul 3 se descrie un registru de 8 biți printr-un proces similar celui din Exemplul 2, cu deosebirea că *d* şi *q* sunt vectori.

Exemplul 3:

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
        port (d: in std_logic_vector (7 downto 0);
                clk: in std_logic;
                q: out std_logic_vector (7 downto 0));
end reg8;
architecture ex_reg of reg8 is
begin
process (clk)
begin
        if (clk'event and clk = '1') then
                q <= d;
        end if;
end process;
end ex_reg;
```

6.1.4. Numărătoare

În Exemplul 4 se descrie un numărător de 3 biți.

Exemplul 4:

```
library ieee;
use ieee.std_logic_1164.all;
entity num3 is
        port (clk: in std_logic;
                num: buffer integer range 0 to 7);
end num3;
architecture num3_integer of num3 is
begin
count: process (clk)
begin
        if (clk'event and clk = '1') then
                num <= num + 1;
        end if;
end process count;
end num3_integer;
```

În exemplul anterior, operatorul de adunare este utilizat pentru semnalul *num*, care este de tip **integer**. Majoritatea sistemelor de sinteză permit această utilizare, convertind tipul integer la tipul **bit_vector** sau **std_logic_vector**. Utilizarea tipului **integer** pentru porturi pune însă unele probleme:

- Pentru a utiliza valoarea *num* într-o altă porțiune a proiectului pentru care interfața are porturi de tip **std_logic**, trebuie efectuată o conversie de tip.

- Vectorii aplicați în timpul simulării codului sursă nu pot fi utilizați pentru simularea modelului generat în urma sintezei. Pentru codul sursă, vectorii trebuie să fie valori întregi. Modelul generat în urma sintezei necesită vectori de tip **std_logic**.

Deoarece operatorul nativ + al limbajului VHDL nu este definit pentru tipurile **bit** sau **std_logic**, acest operator trebuie redefinit înainte de adunarea operanzilor care au aceste tipuri. Standardul IEEE 1076.3 defineşte funcții pentru redefinirea operatorului + pentru următoarele perechi de operanzi: (**unsigned, unsigned**), (**unsigned, integer**), (**signed, signed**) şi (**signed, integer**). Aceste funcții sunt definite în pachetul **numeric_std** al standardului 1076.3.

Exemplul 5 reprezintă Exemplul 4 modificat pentru a se utiliza tipul **unsigned** pentru ieşirea numărătorului.

Exemplul 5:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity num3 is
        port (clk: in std_logic;
                num: buffer unsigned (3 downto 0));
end num3;
architecture num3_unsigned of num3 is
begin
count: process (clk)
begin
        if (clk'event and clk = '1') then
                num <= num + 1;
        end if;
end process count;
end num3_unsigned;
```

De obicei, sistemele de sinteză pun la dispoziție pachete suplimentare care redefinesc operatorii pentru tipul **std_logic**. Deşi acestea nu sunt pachete standard, ele se utilizează adesea de către proiectanți, deoarece permit de obicei operații aritmetice şi relaționale cu tipul **std_logic**, din acest punct de vedere fiind chiar mai utile decât pachetul **numeric_std**. De asemenea, aceste pachete nu necesită utilizarea a două tipuri suplimentare (signed, unsigned) în plus față de tipul **std_logic_vector** şi nici a funcțiilor necesare conversiei între aceste tipuri. La utilizarea unuia din aceste pachete pentru operațiile aritmetice, sistemul de sinteză va utiliza pentru tipul **std_logic_vector** o reprezentare fără semn sau una cu semn (de obicei în C2), şi va genera componentele aritmetice corespunzătoare.

6.1.5. Resetarea componentelor sincrone

Exemplele anterioare nu fac referire la resetarea componentelor descrise sau la condițiile inițiale. Standardul VHDL nu specifică faptul că un circuit trebuie resetat sau inițializat. Pentru simulare, standardul specifică faptul că, dacă un semnal nu este inițializat explicit, acesta va fi inițializat cu valoarea având atributul **'left** a tipului semnalului respectiv. Pentru ca circuitele reale să fie aduse într-o stare cunoscută la inițializare, trebuie utilizate semnale de resetare şi setare (preset).

Un bistabil de tip D cu un semnal de resetare asincronă este descris în modul prezentat în Exemplul 6.

Exemplul 6:

```
architecture exemplu_r of bist_d is
begin
process (clk, reset)
begin
        if (reset = '1') then
                q <= '0';
        elsif rising_edge (clk) then
                q <= d;
        end if;
end process;
end exemplu_r;
```

Dacă semnalul *reset* este activat, semnalul *q* va fi setat la '0', indiferent de valoarea semnalului de ceas. Funcția **rising_edge** este definită în pachetul **std_logic_1164**, având rolul de a detecta frontul crescător al unui semnal. Această funcție se poate utiliza în locul expresiei (*clk'event and clk = '1'*), dacă semnalul *clk* este de tip **std_logic**. În acelaşi pachet este definită şi funcția **falling_edge**, care detectează fronturile descrescătoare ale semnalelor. Aceste funcții sunt preferate de către unii proiectanți deoarece la simulare funcția **rising_edge**, de exemplu, va asigura că tranziția este de la '0' la '1', şi nu va ține cont de alte tranziții cum este cea de la 'U' la '1'.

Pentru a descrie un bistabil cu un semnal de setare asincronă, exemplul anterior se modifică astfel:

```
if (set = '1') then
        q <= '1';
elsif rising_edge (clk) then
```

Se pot utiliza semnale de resetare (sau de setare) sincrone prin includerea condiției respective în interiorul porțiunii procesului care este sincronă cu ceasul, după cum se indică în Exemplul 7.

Exemplul 7:

```
architecture exemplu_r_sinc of bist_d is
begin
process (clk)
begin
        if rising_edge (clk) then
        if (reset = '1') then
                q <= '0';
        else
                q <= d;
        end if;
        end if;
end process;
end exemplu_r_sinc;
```

Execuția procesului din exemplul anterior depinde numai de modificările semnalului de ceas. În urma sintezei se generează un bistabil D care resetat în mod sincron atunci când semnalul *reset* este activ şi apare un front crescător al semnalului de ceas. Deoarece

bistabilele circuitelor PLD nu dispun de obicei de intrări de setare sau resetare sincronă, implementarea acestor intrări necesită utilizarea unor resurse logice suplimentare.

Se pot utiliza de asemenea combinații de semnale sincrone şi asincrone de resetare (sau setare). Uneori sunt necesare două semnale asincrone: atât un semnal de resetare, cât şi unul de setare.

Exemplul 8 prezintă un numărător de 8 biți cu semnale asincrone de resetare şi setare.

Exemplul 8:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity num8 is
        port (clk: in std_logic;
                reset, set: in std_logic;
                enable, load: in std_logic;
                data: in unsigned (7 downto 0);
                num: buffer unsigned (7 downto 0));
end num8;
architecture arh_num8 of num8 is
begin
count: process (reset, set, clk)
begin
        if (reset = '1') then
                num <= (others => '0');
        elsif (set = '1') then
                num <= (others => '1');
        elsif (clk'event and clk = '1') then
        if (load = '1') then
                num <= data;
        elsif (enable = '1') then
                num <= num + 1;
        end if;
        end if;
end process count;
end arh_num8;
```

În exemplul anterior, ambele semnale *reset* şi *set* sunt utilizate pentru asignarea asincronă a unor valori la registrele numărătorului. Combinația de semnale de resetare şi setare din acest exemplu ridică o problemă legată de sinteză. Construcția **if-then-else** utilizată în cadrul procesului implică o precedență – faptul că numărătorului trebuie să i se asigneze valoarea "11111111" numai atunci când semnalul *set* este activ şi semnalul *reset* nu este activ.

Există posibilitatea ca aceasta să nu reprezinte comportarea dorită. Unele sisteme de sinteză pot recunoaşte faptul că acesta nu reprezintă efectul dorit şi că prin construcția bistabilelor este dominant fie semnalul *reset*, fie semnalul *set*.

Multe circuite CPLD care permit resetarea sau setarea prin termeni produs pot implementa ambele variante. De asemenea, majoritatea circuitelor FPGA dispun de resursele necesare pentru implementarea resetării sau setării prin termeni produs. Totuşi, în timp ce majoritatea circuitelor FPGA permit resetarea sau setarea eficientă prin semnale globale, de obicei acestea nu dispun de resurse pentru resetarea sau setarea eficientă prin termeni produs.

În toate exemplele anterioare în care există semnale de resetare sau setare s-a utilizat instrucțiunea **if** sau funcția **rising_edge** pentru descrierea circuitelor sincrone. Pentru descrierea acestor circuite se poate utiliza şi instrucțiunea **wait until**, dar în acest caz semnalele de resetare şi setare trebuie să fie sincrone. Aceasta deoarece pentru descrierile destinate sintezei instrucțiunea **wait** trebuie să fie prima din cadrul unui proces, astfel încât toate instrucțiunile care urmează vor descrie o logică sincronă.

6.1.6. Semnale cu trei stări si semnale bidirecționale

Majoritatea circuitelor programabile dispun de ieşiri cu trei stări sau semnale bidirecționale de I/E. În plus, anumite circuite dispun de buffere interne cu trei stări. Un semnal cu trei stări poate avea valorile '0', '1' şi 'Z', toate acestea fiind permise de tipul **std_logic**.

Exemplul 9 prezintă descrierea modificată a numărătorului din Exemplul 8 pentru a utiliza ieşiri cu trei stări. Acest numărător nu dispune de un semnal de setare asincronă. De data acesta s-a utilizat pachetul **std_arith** şi tipul **std_logic_vector** pentru *data* şi *num*.

Exemplul 9:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_arith.all;
entity num8 is
        port (clk, reset: in std_logic;
                enable, load: in std_logic;
                oe: in std_logic;
                data: in std_logic_vector (7 downto 0);
                num: buffer std_logic_vector (7 downto 0));
end num8;
architecture arh_num8 of num8 is
signal num_tmp: std_logic_vector (7 downto 0);
begin
count: process (reset, clk)
begin
        if (reset = '1') then
                num <= (others => '0');
        elsif rising_edge (clk) then
        if (load = '1') then
                num_tmp <= data;
        elsif (enable = '1') then
                num_tmp <= num_tmp + 1;
        end if;
        end if;
end process count;
oep: process (oe, num_tmp)
begin
        if (oe = '0') then
                num <= (others => 'Z');
        else
                num <= num_tmp;
        end if;
```

```
end process oep;
end arh_num8;
```

În această descriere se utilizează două semnale suplimentare față de descrierea din Exemplul 9: semnalul *oe*, prevăzut pentru controlul ieşirilor cu trei stări, şi semnalul local *num_tmp* declarat în cadrul arhitecturii. Procesul etichetat cu *oep* este utilizat pentru a descrie ieşirile cu trei stări ale numărătorului. Dacă semnalul *oe* nu este activat, ieşirile sunt trecute în starea de înaltă impedanță. Descrierea procesului *oep* este în concordanță cu operarea unui buffer cu trei stări.

Numărătorul din exemplele precedente poate fi modificat astfel încât pentru ieşirile acestuia să se utilizeze semnale bidirecționale. În acest caz, numărătorul poate fi încărcat cu valoarea curentă a ieşirilor acestuia, ceea ce înseamnă că valoarea încărcată atunci când semnalul *load* este activ va fi valoarea precedentă a numărătorului sau o valoare aplicată din exterior, în funcție de starea semnalului *oe*.

6.2. Teme propuse

6.2.1. Simulați funcționarea circuitelor combinaționale descrise în exemplele 1 - 9. Utilizați sistemul *Active-HDL* pentru compilarea cu succes a acestei descrieri.

6.2.2. Modificați descrierea registrului de 8 biți din Exemplul 3 astfel încât să se utilizeze două semnale suplimentare de intrare, reset şi init. La activarea semnalului reset, registrul va fi resetat la "00000000" în mod asincron. La activarea semnalului init, registrul va fi setat la "11111111" în mod sincron. Verificați apoi funcționarea registrului.

6.2.3. Compilați descrierea numărătorului de 3 biți din Exemplul 4 şi verificați funcționarea numărătorului. Modificați apoi descrierea pentru a utiliza tipul std_logic_vector pentru vectorul de ieşire num.