Lucrarea de laborator 3
Sinteza si implementarea circuitelor logice secvențiale

Scopul lucrării: Proiectarea, testarea şi simularea circuitelor logice secvențiale în mediul de dezvoltare software Altera Quartus II. Descrierea circuitelori va fi efectuată în limbajul VHDL, folosind codificarea comportamentală.

1. INTRODUCERE TEORETICĂ

Circuitele secvențiale conțin atât logică combinațională cât şi elemente de memorare. Efectul de memorare se datorează unor legături inverse (bucle de reacţie) prezente în schemele logice ale acestor circuite. În figura 1 se prezintă diagrama unui sistem secvențial. În acest sistem secvențial elementul de memorare este circuitul bistabil.

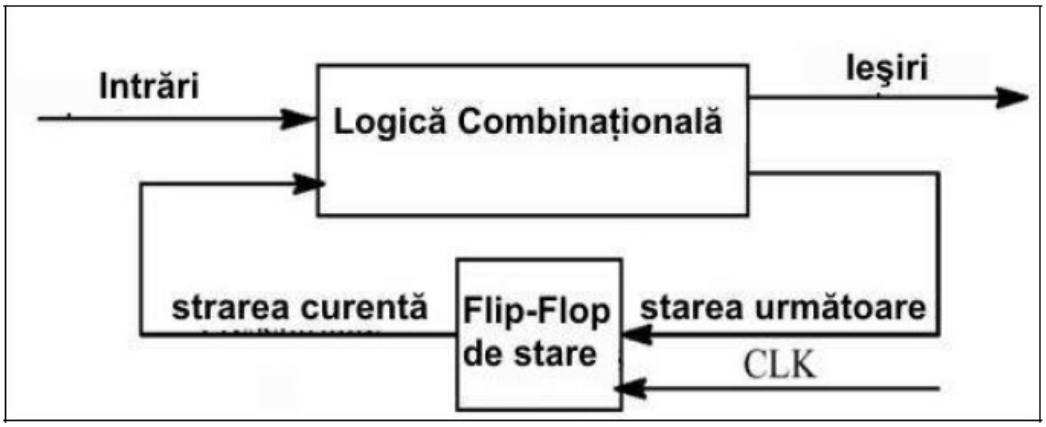


Figura 1 . Sistem secvențial

În cazul circuitelor secvenţiale sincrone, modificarea stării se poate realiza la momente bine definite de timp sub controlul unui semnal de ceas. În cazul circuitelor secvenţiale asincrone, modificarea stării poate fi cauzată de schimbarea aleatoare în timp a valorii unui semnal de intrare.

Comportamentul unui circuit asincron este mai puţin sigur, evoluţia stării fiind influenţată şi de timpii de întârziere ai componentelor circuitului. Trecerea între două stări stabile se poate realiza printr-o succesiune de stări instabile, aleatoare.

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

Comportamentul circuitelor secvențiale poate fi descris folosind construcția **proces.**

Regulile de proiectare a CLS:

1. Utilizarea **proceselor** care nu includ toate porturile de intrare în lista de sensibilitate (în caz contrar se va însera un circuit combinațional).
2. Utilizarea instrucțiunilor **if-then-elsif** incomplet specificate pentru a sugera că unele semnale trebuie să-și păstreze valorile în anumite condiții.

Pentru modelarea funcționării CLS în conformitate cu semnalul de ceas se folosesc următoarele construcții:

1. Construcţia ***if (clk`event)*** întoarce valoarea true pe oricare front al semnalului de tact.
2. Construcţia ***if (clk='1')*** întoarce valoarea true pe nivelul ‚1' al semnalului de tact.
3. Construcţia ***if (clk`event and clk='1')*** întoarce valoarea true doar pe frontul **crescător** al semnalului de tact.
4. Construcţia ***if (clk`event and clk='0')*** întoarce valoarea true doar pe frontul **descrescător** al semnalului de tact.

Construcții alternative:

```
if rising_edge(Clk)
if falling_edge(Clk)
wait until clk='1' and clk'event
wait until clk='0' and clk'event
```

În continuare se vor analiza câteva modele VHDL ale circuitelor secvențiale de bază:
- Bistabile;
- Registre;
- Numărătoare.

Registrele pot avea următoarele funcții:

- Setare/resetare asincronă
- Setare/resetare sincronă
- Validarea semnalului de ceas (Clock Enable)
- Deplasare la stânga/dreapta, bidirecțional

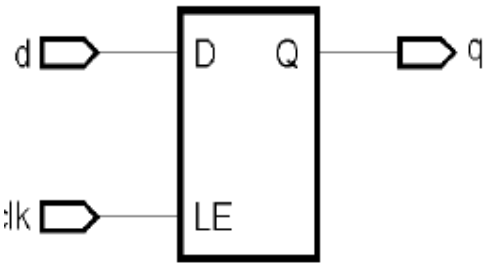
Numaratoare pot avea următoarele funcții:

- Setare/resetare asincronă
- Setare/resetare sincronă
- Încărcare sincronă/asincronă
- Validarea semnalului de ceas (Clock Enable)
- Modul de lucru (Direct, invers, reversibil)

Bistabile

1. Bistabilul D acționat pe nivelul semnalului de ceas.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (d: in std_logic;
        clk: in std_logic;
        q: out std_logic);
end dff;
architecture example of d_latch is
  begin
    process (clk, d)
      begin
        if (clk = '1') then
        q <= d;
        end if;
   end process;
end example;
```



În acest exemplu lista de sensibilitate include toate porturile de intrare.

Instrucțiunea ***if*** nu conține condiția ***else***, ceea ce va duce la înserarea automată a unui bistabil de tip latch. Bistabilul creat va păstra valoarea ***q*** între apelările procesului.

Cu alte cuvinte fragmentul :

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

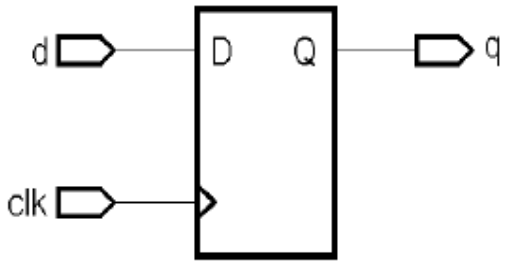
este identic pentru simulare cu fragmentul:

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

În general, majoritatea utilitarelor de proiectare nu admit utilizarea condiției ***else*** după construcția **if** (clk = '1') **then** sau **if (clk`event and clk='1'),** din cauza posibilității de implementare ambiguie.

2. Bistabilul D acționat pe frontul crescător al semnalului de ceas (flip-flop).

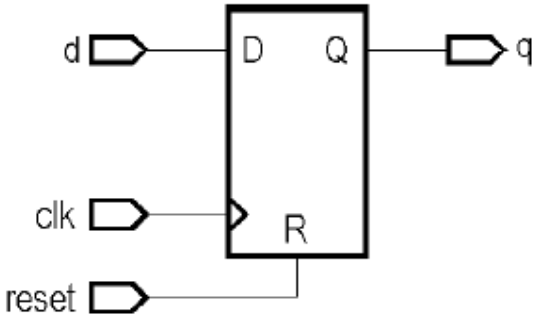
```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (d: in std_logic;
        clk: in std_logic;
        q: out std_logic);
end dff;
architecture example of dff is
  begin
    process (clk)
      begin
        if (clk'event and clk = '1') then
        q <= d;
       end if;
    end process;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



Lista de sensibilitate conține doar semnalul CLK, orice schimbare ale intrărilor de date ***d*** nu determină activarea procesului.

3. Bistabilul Dff cu resetare asincronă

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



În acest model intrarea asincronă de ştergere CLR este dominantă faţă de orice comportare determinată de semnalul de tact CLK, deci este prima evaluată în clauza „if”. Când ***reset*** este negat, se evaluează clauza „elsif”. Funcția ***rising_edge*** este definită în pachetul std_logic_1164 şi poate fi utilizată în locul expresiei (clk'**event and** clk = '1') în cazul când semnalul de ceas este de tipul std_logic. Unii proiectanâi preferă această funcție deoarece, în procesul simulării, funcția ***rising_edge*** va reacționa doar la tranziția din ‘0’ în ‘1’ şi nu va reacționa la oricare altă tranziție, de exemlu din ‘U’ în ‘1’.

Pentru a descrie un bistabil cu setare asincronă, fragmentul:

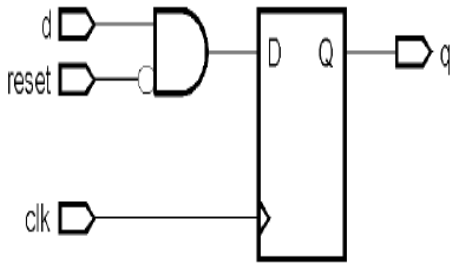
```
begin if (reset = '1') then
q <= '0';
```

va fi înlocuit cu următorul fragment:

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

4. Bistabilul dff cu resetare sincronă

```
architecture example_r_sync of dff 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 example_r_sync;
```



În acest exemplu resetarea bistabilului are loc atunci când semnalul reset este activ în acelaşi timp cu frontul crescător al semnalului de ceas. Deoarece majoritatea bistabilelor în circuitele PLD nu dispun de setare/resetare sincronă, implementarea acestei logici necesită porți suplimentare.

Registre

5. Registru pe 8 biți, activ pe frontul crescător al semnalului de ceas.

```
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. Registru pe 8 biți cu încărcare paralelă şi resetare asincronă

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8bit is
  port ( clk, reset, ld: in std_logic;
         din: in std_logic_vector(7 downto 0);
        dout: out std_(7 downto 0));
end reg8bit;
architecture behavior of reg8bit is
 signal n_state: std_logic_vector(7 downto 0);
 signal p_state : std_logic_vector(7 downto 0);
  begin
   process(clk, reset)
     begin
      if (reset = '0') then p_state <= (others => '0'); --reset asincron
       elsif (clk'event and clk = '1') then
        if (ld='1') then n_state <= din; --încărcare paralelă
          else null;
       end if;
      p_state <= n_state;
   end if;
  end process;
dout <= p_state;
end behavior;
```

Expresia ***p_state <= (others => '0')*** este folosită pentru atribuirea valorii 0 tuturor biților din semnalul ***p_state***.

Registre de deplasare

Un registru de deplasare este un circuit secvenţial care deplasează la stânga sau la dreapta conţinutul registrului cu o poziţie în fiecare ciclu de ceas. De obicei, intrările unui registru de deplasare sunt reprezentate de semnalul de ceas, o intrare serială de date, un semnal de setare/resetare sincronă sau asincronă şi un semnal de validare a ceasului. În plus, un registru de deplasare poate avea semnale de control şi de date pentru încărcarea paralelă sincronă sau asincronă. Datele de ieşire ale unui registru de deplasare pot fi accesate fie serial, atunci când este accesibil numai conţinutul ultimului bistabil pentru restul circuitului, fie în paralel, atunci când este accesibil conţinutul mai multor bistabile.

Există mai multe posibilităţi pentru descrierea registrelor de deplasare în limbajul VHDL:

- Utilizarea operatorului de concatenare: ***reg <= reg (6 downto 0) & SI;***
- Utilizarea construcţiilor ***for loop;***
- Utilizarea operatorilor de deplasare predefiniţi ***(sll, srl, sla, sra).***

7. Descrierea unui registru de deplasare la stânga de 8 biţi cu semnale de validare a ceasului, intrare serială şi ieşire serială. Pentru descrierea registrului de deplasare se utilizează o construcţie ***for loop***.

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8_depl is
  port (clk, ce, si: in std_logic;
        so:          out std_logic);
end reg8_depl;
architecture reg_depl of reg8_depl is
   signal tmp: std_logic_vector (7 downto 0);
    begin
     process (clk)
        begin
         if (clk'event and clk = '1') then
           if (ce = '1') then
           for i in 0 to 6 loop
             tmp(i+1) <= tmp(i);
           end loop;
             tmp(0) <= si; -- încărcare serială
           end if;
        end if;
     end process;
  so <= tmp(7); -- ieşire serială
end reg_depl;
```

8. Registru de deplasare bidirecţional

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity reg_bidirect is
port (clk, si, left_right: in std_logic;
        Po: out std_logic_vector (7 downto 0));
end reg_bidirect;
architecture beh of reg_bidirect is
signal tmp: std_logic_vector (7 downto 0);
  begin
    process (clk)
       begin
          if (clk'event and clk='1') then
            if (left_right='0') then
               tmp<=tmp(6 downto 0) & si;
```

```
                else
                  tmp<=si & tmp(7 downto 1);
                end if;
              end if;
            end process;
           po<=tmp; --ieșire paralelă
        end beh;
```

Exemplu de încărcare paralelă

```
process (CLK, LOAD, D)
   begin
      if (LOAD='1') then
         tmp <= D; --încărcare paralelă
       elsif (CLK'event and CLK='1') then
          tmp <= tmp(6 downto 0) & SI;
```

Numărătoare

9. Numărător direct de 8 biţi cu semnale asincrone de resetare şi setare.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity num8 is
  port (clk: in std_logic;
        rst, set: in std_logic;
        en, load: in std_logic;
        data: in std_logic_vector (7 downto 0);
         num: out std_logic_vector (7 downto 0));
end num8;
architecture arh_num8 of num8 is
   signal tmp: std_logic_vector (7 downto 0);
     begin
      cnt: process (rst, set, clk)
          begin
             if (rst = '1') then
                tmp <= (others => '0');
            elsif (set = '1') then
                tmp <= (others => '1');
            elsif (clk'event and clk = '1') then
                if (load = '1') then
                    tmp <= data;
            elsif (en = '1') then
                    tmp <= tmp + 1;
            end if;
```

```
            end if;
        end process cnt;
          num <= tmp;
end arh_num8;
```

10. Numărător pe 4 biți reversibil cu încărcare sincronă

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
port (clk, clr, load, up, down: in std_logic;
        data: in std_logic_vector(3 downto 0);
       count: out std_logic_vector(3 downto 0));
end counter;
architecture count4 of counter is
signal cnt: std_logic_vector(3 downto 0);
  begin
     process (clr, clk)
        begin
         if clr='1' then cnt<='0000';
         elsif (clk'event and clk='1') then
               if load='1' then cnt<=data
                  elsif up='1' then
                      if cnt='1111' then cnt <='0000';
                     else cnt<=cnt+1;
                    end if;
                 elsif down='1' then
                   if cnt='0000' then cnt<='1111';
                 else cnt<=cnt-1;
             end if;
            else
        cnt<=cnt;
end if;
end if;
count<=cnt;
end process;
end count4;
```

11. Numărător pentru codul 8421+3

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
entity V74x163 is
  port ( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
         D: in UNSIGNED (3 downto 0);
         Q: out UNSIGNED (3 downto 0);
         RCO: out STD_LOGIC );
end V74x163;
architecture V74xs3_arch of  V74x163  is
  signal IQ: UNSIGNED (3 downto 0);
    begin
      process (CLK, ENT, IQ)
        begin
         if CLK'event and CLK='1' then
           if CLR_L='0' then IQ <= (others => '0');
           elsif LD_L='0' then IQ <= D;
           elsif (ENT and ENP)='1' and (IQ=12) then IQ <= ('0','0','1','1');
           elsif (ENT and ENP)='1' then IQ <= IQ + 1;
          end if;
        end if;
           if (IQ=12) and (ENT='1') then  RCO <= '1';
           else RCO <= '0';
           end if;
       Q <= IQ;
      end process;
end V74xs3;
```

Desfăşurarea lucrării

1. Se vor modela circuitele pentru exemplele Nr. 3, 5, 7 şi 9. Pentru fiecare circuit se va crea un proiect nou.
2. Se va proiecta un registru conform variantei din tabelul 1.
3. Se va proiecta un numărător conform variantei din tabelul 2.

Tabelul 1.

Nr.	Num. de biţi	CLK	Resetare	Setare	Direcţie deplasare	Încărcare	Iesire
1.	4	Front crescător	sincronă	nu	dreapta	serială	serială
2.	8	Front descrescător	asincronă	nu	stânga	paralelă	paralelă
3.	4	Front crescător	nu	sincronă	bidirecţional	serială	serială
4.	8	Front descrescător	asincronă	nu	dreapta	paralelă	paralelă
5.	6	Front crescător	sincronă	nu	stânga	serială	serială
6.	8	Front descrescător	nu	asincronă	bidirecţional	paralelă	paralelă
7.	5	Front crescător	sincronă	nu	dreapta	serială	serială
8.	7	Front	nu	asincronă	stânga	paralelă	paralelă

		descrescător					
9.	4	Front crescător	sincronă	nu	bidirecțional	serială	serială
10.	5	Front descrescător	asincronă	nu	dreapta	paralelă	paralelă
11.	8	Front crescător	nu	asincronă	stânga	serială	serială
12.	7	Front descrescător	asincronă	nu	bidirecțional	paralelă	paralelă

Tabelul 2.

Nr.	mod	CLK	Resetare	Tip numărător	Secvența de numărare	Încărcare
1.	12	Front descrescător	sincronă	Direct	1-12	paralelă
2.	9	Front crescător	asincronă	Invers	14-6	paralelă
3.	11	Front descrescător	nu	Direct	2-12	paralelă
4.	10	Front crescător	asincronă	Invers	13-4	paralelă
5.	9	Front descrescător	sincronă	Direct	3-11	paralelă
6.	13	Front crescător	nu	Invers	14-2	paralelă
7.	10	Front descrescător	sincronă	Direct	3-12	paralelă
8.	12	Front crescător	nu	Invers	12-1	paralelă
9.	13	Front descrescător	sincronă	Direct	2-14	paralelă
10.	14	Front crescător	asincronă	Invers	14-1	paralelă
11.	14	Front descrescător	nu	Direct	0-13	paralelă
12.	11	Front crescător	asincronă	Invers	13-3	paralelă