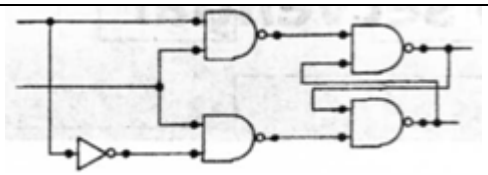
LABORATOR 4

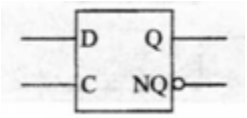
1. **Proiectarea registrelor şi a latchurilor**
2. **Proiectarea numărătoarelor**
3. **Proiectarea registrelor de deplasare**

1. Proiectarea registrelor şi a latchurilor

Latchurile şi registrele pot fi modelate prin metode structurale. De exemplu, latch-ul de tip D din Figura 1 poate fi modelat structural prin programul din Figura 2.



a.



b.

C	D	Q	NQ
1	0	0	1
1	1	1	0
0	–	Ultimul Q	Ultimul NQ

c.

Fig.1 Latch de tip D. a) schema logică; b) schema bloc; c) tabelul de adevăr.

```
entity dlatch is
port   (D,  C:  in STD_LOGIC;
        Q, NQ: buffer STD_LOGIC);
end entity;
architecture dlatch_str of dlatch is
signal DN,  SN,  RN:  STD_LOGIC;
component inv
     port   (I:  in STD_LOGIC; O:  out STD_LOGIC);
     end component;
component nand2b
 port   (10,  II:  in STD_LOGIC; 0: buffer STD_LOGIC);
 end component;

 begin
    U1: inv port map ( .... );
    U2: nand2b port map ( .... ) ;
    U3: nand2b port map  ( ... );
    U4: nand2b port map ( .... ) ;
    U5: nand2b port map ( .... );
end architecture dlatch str;
```

Fig. 2 Modelul structural al unui latch de tip D

O altă variantă de model pentru un latch de tip D este aceea din Figura 3. Este un model comportamental care foloseşte numai o linie de cod VHDL. Deoarece în program nu se precizează ce se întâmplă dacă **C** nu este 1, compilatorul va forma un model care reține valoarea lui **Q** între două execuții ale procesului.

```
architecture dlatch_beh of dlatch is
begin
process  (C,  D, Q)
begin
     if  (C =  '1' )   then …      ;
    end if;
end process;
end architecture;
```

Fig. 3 Un model comportamental pentru un latch de tip D

Pentru a descrie comportarea bistabilelor care comută pe frontul unui semnal de ceas, trebuie să se folosească *atributele semnalelor* în VHDL (de exemplu, atributul **'event).** Dacă **SEM** este numele unui semnal, atunci construcție **SEM'event** rerumează valoarea true dacă **SEM** se schimbă de la o valoare la alta (eveniment în **SIM),** false în caz contrar.

Cu această observație, se poate obține un model pentru un bistabil D comutat pe frontul pozitiv al semnalului de ceas (Figura 4).

```
entity d_ff is            . ..
port   (D,  CLK:  in STD_LOGIC; Q:  out
STD_LOGIC);
end entity;
architecture d_ff_behav of d_ff is
begin
    process  (CLK) begin
     if (CLK'event and .....) then  .... ;
     end if;
    end process;
end architecture;
```

Fig. 4 Modelul unui bistabil D comutat pe front pozitiv

Modelul unui bistabil D poate fi complicat pentru a include şi *intrări asincrone* (preset, clear) şi o ieşire complementată, ca în circuitul 74x74. Acest model este prezentat în Figura 5.

```
entity d_ff_7 4 is
port   (D,  CLK,   PR_L,  CLR_L:  in STD_LOGIC;
        Q,  NQ:  out STD_LOGIC);
end entity;
architecture d_ff_74_behav of d_ff_74 is
  signal PR,  CLR:    .;

begin
    process  (CLR_L,  CLR,  PR_L,  PR, CLK)
    begin
     PR <= not PR_L; CLR <= not CLR_L;
     if  (CLR and PR)  =  '1'  then  ;
                                      -- Q si NQ vor fi ,0'
       elsif CLR = '1'  then ... ;
       elsif PR = '1' then .... ;
       elsif  (......)  then Q <= .... ;
              NQ <=  ... ; -- acțiunea pe front de tact pozitiv
       end if;
     end process;
end architecture;
```

Fig. 5 Bistabil D cu intrări asincrone preset şi clear

Modelele comportamentale ale registrelor se pot obține definind intrările şi ieşirile ca vectori şi incluzând şi funcții suplimentare. De exemplu, în Figura 6 se prezintă modelul unui registru de 16 biți, cu ieşiri three-state, cu intrări clock-enable, output-enable şi clear. Se foloseşte un semnal intern **IQ** pentru a reține valorile ieşirilor bistabilelor.

```
 entity regl6bit is
     port   (CLK,  CLKEN,  OE_L,  CLR_L:  in STD_LOGIC;
             D:  in  ..........;     Q:  out       ) ;
 end entity;
architecture regl6bit_beh of regl6bit is
 signal CLR,  OE:  STD_LOGIC;
 signal IQ:  STD_LOGIC_VECTOR  (....);
begin
 process  (CLK,  CLR_L,  CLR,  OE_L,  OE,  IQ)
 begin
   CLR <= not CLR_L; OE <= ...;
   if  (CLR = '1') then IQ <= (others => .....) ;
   elsif  (CLK'event and .....)  then
       if  (CLKEN =  '1')  then IQ <= ...;
       end if;
   end if;
   if OE = '1' then Q <= ...;
   else Q <= (others => ...); --iesire tristate
   end if;
  end process;
 end architecture;
```

Fig. 6 Modelul comportamental al unui registru de 16 biți

Codul VHDL din figura 6. este foarte util pentru implementarea unui port de intrare sincron pentru o unitate de procesare.

În Figura 7 se prezintă *modelul algoritmic al unui flip-flop de tip JK.* Setarea şi resetarea sunt operații prioritare față de funcționarea normală prin intermediul semnalului de ceas. Dacă **R = S = 1,** nu se întâmplă nimic. Pentru că aceasta este o condiție ilegală, ieşirea, în general, nu se specifică. Deci, modul de funcționare "nu se întâmplă nimic" este consistent cu specificația.

```
architecture ALG of JKFF is
begin
  process (CLK, S,R)
  begin
    if S = '1' and R = '0' then
      Q <= ..... after ..... ;
      CN <= ..... after  --
    elsif S = '0' and R = '1' then
    ……..
    elsif CLK'EVENT and CLK = '1' and S='0' and R='0' then
     if J = '1' and K = '0' then
    ……..
      elsif J = '0' and K ='1'   then
     ……
-- toate variantele
       end if;
     end if;
    end process;
  end architecture ALG;
```

Fig. 7 Modelul unui flip-flop JK.

1. Completați programele din Figura 2, Figura 3, Figura 4, Figura 5, Figura 6, Figura 7. Compilați aceste programe şi simulați funcționarea circuitelor respective.
2. Elaborați un raport scris care să conțină modelele VHDL complete (comentate) şi formele de undă care demonstrează funcționarea corectă a circuitelor în toate cazurile posibile.

2 Proiectarea numărătoarelor

Operația de numărare este fundamentală în circuitele digitale. în Figura 8 se prezintă modelul unui numărător care poate fi resetat **(RESET** = '1'), încărcat **(LOAD** = ' 1') şi poate număra înainte sau înapoi (up/down), după cum se stabileşte prin semnalul de control **UP,** pe frontul crescător al semnalului de ceas. Ordinea priorităților este (1) reset, (2) încărcare, (3) numărare, (4) activarea ieşirii. Modelul pentru numărător foloseşte funcțiile **INC** şi **DEC** din pachetul **PRIMS** din laboratorul anterior.

```
entity COUNTER is port (RESET, LOAD, COUNT, UP, OE, CLK: in
  ...;
DATA_IN: in ....( 7 downto 0);
CNT: out std_logic_vector (  7 downto 0 )) ;
end entity COUNTER;
use work.PRIMS.all;
architecture ALG of COUNTER is
signal cnt_v : BIT_VECTOR (  ....);
begin
process(CLK) begin
  if CLK'event and .....then
      if RESET = '1' then
            cnt_v <= ...); --resetare
      elsif LOAD ='1' then
  cnt_v <= to_bitvector(...); -- conversie din std_logic_vector
--funcţia de incarcare paralela
      elsif COUNT ='1' then
            if UP = '1' then
-- apelare funcţii descrise în pachetul PRIMS
                  cnt_v <= INC (...); --numărare inainte
            else
                  cnt_v <= ... (cnt_v);--numărare inapoi
            end if;
      end if;

      if OE = '1' then            -- activare ieşire
      CNT <= to_stdlogicvector(...); --converise din bit_vector
      end if;
 end if;
end process;
end architecture ALG;
```

Fig 8 Modelul unui numărător cu funcţii de *reset* şi *load*

Secvența de cod VHDL prezentată în figura 9 poate fi folosită pentru implementarea modulelor de contor de program sau indicator de stivă în cazul unei unități de microprocesare.

Atenţie – La proiect trebuie adăugat pachetul PRIMS pentru a putea utiliza funcțiile de incrementare şi decrementare. Procedura de adăugare a fost prezentată în laboratorul anterior, vezi ALU.

În Figura 9 este prezentat modelul unui numărător similar cu 74x163. Programul foloseşte biblioteca **IEEE. std_logic_arith. all,** care include tipul **UNSIGNED.** Această bibliotecă include definițiile pentru operatorii + şi - care realizează adunarea şi scăderea asupra operanzilor **UNSIGNED.** Programul declară intrarea şi ieşirea numărătorului ca vectori **UNSIGNED** şi foloseşte + pentru a incrementa valoarea numărătorului.

```
entity num_74x163 is
       port   (CLK,  CLR_L,  LD_L,  ENP,  ENT:  in STD_LOGIC;
       D:  in UNSIGNED  (3 downto 0);
       Q:  out UNSIGEND  (3 downto 0) ;
       RCO:  out STD_LOGIC);
 end entity;
 architecture num_74x163_arch of num_74x163 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 LDJL =  "0'  then IQ <= D;
      elsif  (ENT and ENP)  =  "1'  then IQ <= IQ + 1;
      endif;
     end if;

     if  (IQ = 15)  and  (ENT =  "1')  then RCO <= "1';
     else RCO <= '0';
     end if;

   Q <= IQ;
  end process;
 end architecture;
```

Fig. 9 Modelul VHDL al unuinumărător de tip 74x163

Din păcate, unele programe de sinteză au tendința să sintetizeze numărătoarele cu ajutorul unui sumator binar care adună 1 la valoarea curentă a numărătorului. Această modalitate conduce la o logică de tip combinațional mai complexă decât ceea ce se observă la structurile clasice de numărătoare. De aceea, este util să dezvoltăm modele structurale ale numărătoarelor, modele în care să conectăm celule elementare de numărare.

De exemplu, putem modela o celulă elementară de numărare pentru numărătorul 74x163, folosind circuitul din Figura 10.

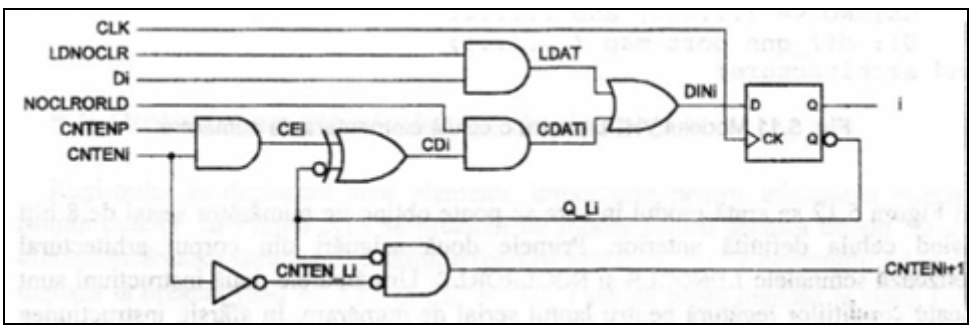


Fig. 10 Celulă elementară de numărare pentru numărătorul 74x163

Semnalele din celula elementară au următoarele semnificații:

CLK	(comun) Intrarea de ceas pentru toate etajele
LDNOCLR	(comun) Activat dacă intrarea LD a numărătorului este activată şi CLR este negat
NOCLRORLD	(comun) Activat dacă intrările CLR şi LD sunt negate
SNTENP	(comun) Activat dacă intrarea ENP a numărătorului este activată
Di	(pentru fiecare celulă) Intrarea de încărcare paralelă pentru celula i
CNTENi	(pentru fiecare celulă) intrare de validare pentru celula i
CNTENi +1	(pentru fiecare celulă) ieşire de validare pentru celula i
Qi	(pentru fiecare celulă) Ieşirea de numărare pentru celula i

În Figura 11 se prezintă programul VHDL pentru celula din Figura 10. în acest program, componenta **dff_qnn** (bistabil de tip D) se presupune definită; este similar cu bistabilul D din Figura 5, cu adăugarea unei ieşiri complementate QN.

```
entity sync_ser_cell
port   (CLK,  LDNOCLR,  NOCLRORLD,  CNTENP,   D,  CNTEN:  in     ;
    CNTENO,  Q:  out STD_LOGIC);
end entity;
architecture sync_ser_cell_arch of sync_ser_cell is
component dff_qnn
      port  (CLK,  D:  in.. ;    Q, QN:  out ..) ;
end component;
signal LDAT,  CDAT,  DIN,  Q_L:  STD_LOGIC;
begin
      LDAT <= LDNOCLR ....   D;
      CDAT <= NOCLRORLD and  ( (.... )  xor not   ...) ;
      DIN <= LDAT  ...   CDAT;
      CNTENO <=  ( ... )  and .... ;
      UI:  dff_qnn port map  ( .. );
  end architecture;
```

Fig. 12 Modelul VHDL pentru o celulă elementară de numărare

În Figura 12 se arată modul în care se poate obține un numărător serial de 8 biți folosind celula definită anterior. Primele două asignări din corpul arhitectural sintetizează semnalele **LDNOCLR** şi **NOCLRORLD.** Următoarele două instrucțiuni sunt dedicate condițiilor legătură pentru lanțul serial de numărare. În sfârşit, instrucțiunea generate instanțiază opt celule de numărare de 1 bit şi conectează lanțul de validare.

```
entity 74xl63_b is
    port   (CLK,,  CLR_L,  LD_L,  ENP,  ENT:  in STD_LOGIC;
     D:  in STD__LOGIC_VECTOR  (7 downto 0);
     Q:  out STD_LOGIC_VECTOR  (7 downto 0)
     RCO:  out STD_LOGIC);
end entity;
architecture 74xl63_b_arch of 74xl63_b is
component sync_ser_cell
port   (CLK,  LDNOCLR,  NOCLRORLD,  CNTENP,  D,  CNTEN:  in ...
         CNTENO,  Q:  out   . );
end component;
signal LDNOCLR, NOCLRORLD:  STD_LOGIC;
signal SCNTEN:  STD_LOGIC_VECTOR  (8 downto 0);
begin
    LDNOCLR <= .....;
    NOCLRORLD <= .....;
    SCNTEN (0)  <=.....;
    RCO <=......;
    gl:  for i in 0 to 7 generate
       Ul: sync_ser__cell     port map  (    );
    end generate;
end architecture;
```

Fig. 12 Modelul VHDL pentru un numărător serial care foloseşte celule elementare de numărare.

1. Completați programele din Figura 8, Figura 9, Figura 11, Figura 12. Compilați aceste programe şi simulați funcționarea circuitelor respective.
2. Elaborați un raport scris care să conțină modelele VHDL complete (comentate) şi formele de undă care demonstrează funcționarea corectă a circuitelor în toate cazurile posibile

3. Proiectarea registrelor de deplasare

Registrele de deplasare sunt elemente importante pentru efectuarea operațiilor asupra datelor. În Figura 13 este prezentat tabelul de funcționare pentru un registru de deplasare de 8 biți cu un set extins de funcții. În plus față de funcțiile **Hold, Load** şi **Shift,** acest registru realizează şi deplasarea circulară şi deplasarea aritmetică. În operațiile de deplasare circulară, bitul care se află la un capăt al vectorului se transferă înapoi, la celălalt capăt. In operațiile de deplasare aritmetică, intrarea se setează pentru înmulțirea cu 2 sau împărțirea la 2. Pentru o deplasare la stânga, intrarea din dreapta este 0. Pentru o deplasare la dreapta, este copiat bitul cel mai din stânga (semn).

Un model comportamental pentru acest registru cu funcții extinse este prezentat în Figura 14. Se defineşte un proces şi se foloseşte atributul **event** pentru a obține comutarea pe frontul dorit al semnalului **CLK.** Alte caracteristici ale programului sunt:

- Se foloseşte un semnal intern, **IQ,** imagine a lui **Q,** semnal care poate fi citit/scris prin instrucțiunile procesului. O altă posibilitate este definirea ieşirilor **Q** de tip *buffer*.

- Intrarea **CLR** este asincronă; întrucât acest semnal este în lista de sensibilitate, el este testat ori de câte ori se schimbă. Iar instrucțiunea IF este structurată astfel încât **CLR** are precedență față de orice altă condiție.

- Se foloseşte o instrucțiune **CASE** pentru a defini operația registrului pentru cele opt combinațiii posibile ale intrărilor de selecție **S (2 downto 0).**

- În **CASE,** cazul „when others" este necesar pentru a evita erori la compilare.

- Instrucțiunea **NULL** arată că, în anumite cazuri, nu are loc nici o acțiune. în cazul 1 nu se impune nici o acțiune; acțiunea implicită pentru un semnal este menținerea valorii până când se indică altceva.

- În cele mai multe cazuri se foloseşte operația de concatenare & pentru a construi un tablou de 8 biți dintr-un subset de 7 biți format din **IQ,** plus un alt bit.

- Întrucât VHDL este puternic tipizat, se foloseşte **CONVINTEGER** pentru a converti intrarea **S (STD_LOGIC__VECTOR**) la un întreg în instrucțiunea **CASE.**

Funcția	Intrări			Starea următoare							
	S2	S1	S0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
Hold	0	0	0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
Load	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Shift Right	0	1	0	RIN	Q7	Q6	Q5	Q3	Q2	Q1	Q0
Shift Left	0	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	LIN
Shift Circular Right	1	0	0	Q0	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Shift Circular Left	1	0	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	Q7
Shift Arithmetic Right	1	1	0	Q7	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Shift Arithmetic Left	1	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	0

Fig. 13 Tabelul de adevăr pentru un registru de deplasare cu funcții extinse.

```
entity shift_reg is
      port   (CLK,   CLR,   RIN,   LIN:   in STD_LOGIC;
               S:  in ...    ;    D:  in  ;    Q: buffer    ;
end entity;
architecture schift_reg_arch of shift_reg is
signal IQ STD_LOGIC_VECTOR  (7 downto 0);
begin
    process  (CLK, CLR,  IQ)
    begin
    if  (CLR = '1') then IQ <= (others => '0');
```

```
        elsif  (CLK'event and CLK =  '1')  then
          case CONV_INTEGER  (S)  is
           when 0 => null;
          when 1 => IQ <= D;
          when 2 => IQ <= RIN & IQ(7 downto 1);
             .....  -- toate cazurile;
          when others => null;
          end case;
      end if;
      Q <= IQ;
      end process;
      end architecture;
```

Fig. 14 Modelul VHDL pentru un registru de deplasare cu funcții extinse.

1. Completați programul din Figura 14. Compilați programul şi simulați funcționarea.
2. Elaborați un raport scris care să conțină modelele VHDL complete (comentate) şi formele de undă care demonstrează funcționarea corectă a circuitelor în toate cazurile posibile.