Programarea structurilor hardware recprogramabile prin descrieri concurente

I. SCOPUL LUCRĂRII

În această lucrare sunt prezentate implementări ale HDL utilizând descrieri cu specificații concurente. Exemplele propuse cuprind o parte din specificațiile concurente ce pot fi sintetizate şi implementate pe sistemul reconfigurabil din laborator, utilizând descrieri de tip flux de date sau de tip structural.

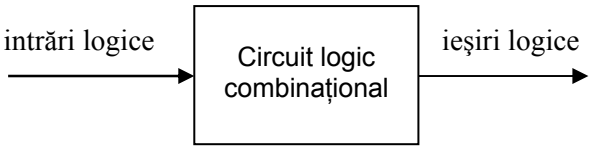
II. INTRODUCERE TEORETICĂ

1. Descrierea cu specificații concurente

Prin intermediul limbajelor de descriere hardware pot fi proiectate module digitale independente, interconectate între ele prin semnale şi care funcționează în paralel.

Limbajul de descriere hardware prezintă mecanisme de descriere paralelă cu specificații concurente a modulelor digitale combinaționale.

Logica combinațională este o structură digitală în care ieşirile depind numai de intrări.



În cadrul unui program scris în limbajul VHDL, zona de descriere concurentă se regăseşte între specificațiile **begin** şi **end** ale unei arhitecturi.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- alte incluziuni de librării şi pachete;

entity modul_digital is
        generic ( --declarații de constante generice )
```

```
        port(
                - - -
          );
end modul_digital;


architecture descriere of modul_digital is
begin

    - - - zona de specificații concurente

end descriere;
```

1.1. Atribuirea condițională a semnalelor

Atribuirea semnalelor se face în interiorul arhitecturii sau a proceselor.

Atribuirea simplă a unui semnal este realizată prin operatorul „<=”.

Atribuirea condițională în domeniul concurent se realizează prin specificația WHEN/ELSE.

Sintaxă:

```
LABEL1:    -- etichetă optională
        SIG_NAME <= <expresie> when <condiție> else
                     ---
                    <expresie> when <condiție> else
                    <expresie>;
```

Modificarea valorii logice a unui semnal se face numai dacă este îndeplinită o anumită condiție booleană. Altfel, este luată în considerare condiția următoare care apare după clauza ELSE. Întotdeauna, o atribuire condițională trebuie să se termine cu specificația ELSE.

1.2. Atribuirea selectivă a semnalelor

Atribuirea selectivă a semnalelor este realizată cu specificația WITH/SELECT. În acest caz, spre deosebire de atribuirea condițională, trebuie incluse toate combinațiile posibile în declarație.

```
LABEL1:    -- etichetă opțională
        with <expresie de selecție> select
            SIG_NAME <= <expresie> when <selectie>,
                        <expresie> when <selectie>,
                         ---
                        <expresie> when others;
```

Pentru eliminarea tuturor posibilităţilor de selecţie din expresia condiţională, la sfârşitul specificaţiei de atribuire este obligatorie introducerea clauzei WHEN OTHERS.

2. Descrierea structurală

Componenta reprezintă o pereche entitate/arhitectură şi specifică un susbsistem care poate fi introdus şi interconectat (instanţiat) în altă arhitectură pe o metodologie ierarhică. Pentru a fi utilizată, componenta este *declarată* în zona declarativă a arhitecturii, după care inserarea acesteia în alte module se realizează prin *instanţiere*.

Declararea unei componente

Declaraţia unei componente reprezintă primul pas în procesul de utilizare al acesteia într-un modul digital.

Sintaxa:

```
component component_name [ is ]
   generic (generic_list);
   port (port_list);
end component component_name;
```

Declaraţia componentei (sintaxa de mai sus) defineşte interfaţa virtuală (soclul în care va fi introdus circuitul), dar nu indică direct componenta.

O componentă poate fi definită în package-uri, entităţi, arhitecturi sau declaraţii de blocuri. În cazul în care, componenta este declarată într-o arhitectură, aceasta trebuie să fie plasată în zona declarativă înainte de clauza **begin**.

Instanţierea unei componente

Al doilea pas în utilizarea componentei constă în instanţierea acesteia, adică realizarea asocierilor de semnale şi atribuirii de valori generice specifice, în cadrul arhitecturii modulului digital.

Sintaxă:

```
eticheta : [ component ] nume_componenta
generic map ( lista_valori_generice )
port map ( lista_porturi );

eticheta : entity nume_entitate [(identificator_arhitectura)]
generic map ( lista_valori_generice )
port map ( lista_porturi );
```

```
eticheta : configuration nume_configuratie
generic map ( lista_valori_generice )
port map ( lista_porturi );
```

Instanțierea unei componente permite păstrarea referințelor unității instanțiate şi valorile actuale ale genericelor, respectiv porturilor acesteia.

Numele componentei instanțiate trebuie să fie acelaşi cu numele componentei declarate. Lista de asociere poate fi realizată după nume sau poziționarea porturilor.

În *asocierea poziţională*, parametrii actuali sunt conectați în aceeaşi ordine cu cea a porturilor în care a fost declarată componenta.

```
U1: poarta PORT MAP(a, b, c);
```

Asocierea după nume dă posibilitatea porturilor şi valorilor generice să fie puse într-o ordine diferită de cea declarată în componentă. Asocierea porturilor sau valorilor generice se face prin operatorul „=>”.

```
U1: poarta PORT MAP(in1 =>a, in2 => b,iesire => c);
```

Specificaţia GENERATE

Specificația GENERATE reprezintă o facilitate furnizată de VHDL pentru realizarea iterativă sau condițională a unor porțiuni de program.

Sintaxă:

```
eticheta: for parametru in interval
generate
[ { declaratii }
begin ]
    { specificatii concurente }
end generate [ eticheta ] ;
```

```
eticheta: if conditie generate
[ { declaratii }
begin ]
    { specificatii concurente }
end generate [ eticheta ] ;
```

Specificația de tip **generate** este utilizată pentru simplificarea descrierii unor porțiuni de program repetitive. De obicei, este utilizată pentru interconectarea unui grup de componente identice folosind o singură componentă care este multiplicată.

O specificație **generate** constă în:

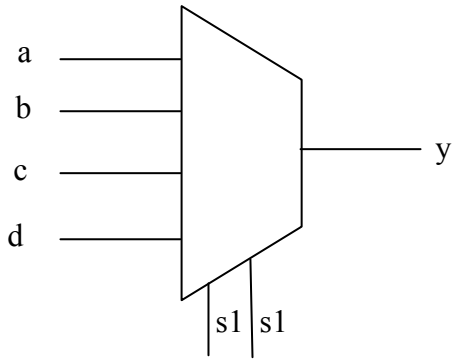
- generarea de scheme (for generate sau if generate);
- parte declarativă (declarații locale de subprograme, tipuri, semnale, constante, componente, atribute, configurații, fişiere şi grupuri);
- specificații concurente.

Elementele limbajului VHDL prezentate în partea teoretică a acestei lucrări de laborator vor fi regăsite în aplicațiile următoare.

III. APLICAȚII

a) Descrieți în limbajul VHDL un multiplexor de tipul 4:1 cu ajutorul specificației condiționale concurente WHEN/ELSE.

Soluție



Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_a is
   Port ( a,b,c,d : in std_logic;
          s1,s2 : in std_logic;
          y : out std_logic);
end mux_a;

architecture Behavioral of mux_a is
          signal s_temp : std_logic_vector(1 downto 0);
begin
          s_temp <= s2 & s1;
          y <=  a when s_temp="00" else
                          b when s_temp="01" else
                          c when s_temp="10" else
                          d;
end Behavioral;
```

b) Realizați un multiplexor 4:1 utilizând implementarea prin specificația selectivă WITH / SELECT / WHEN.

Soluție

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_b is
   Port ( a,b,c,d : in std_logic;
          s1, s2 : in std_logic;
          y : out std_logic);
```

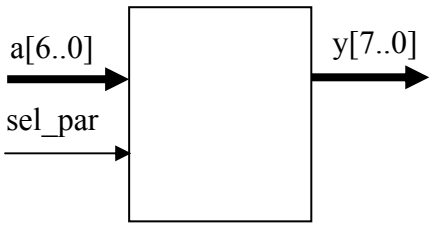
```
end mux_b;

architecture Behavioral of mux_b is
   signal s_temp : std_logic_vector(1 downto 0);
begin
        WITH s_temp SELECT
                y <=  a when "00",
                              b when "01",
                              c when "10",
                              d when OTHERS;

end Behavioral;
```

c) Prezentați descrierea hardware a unui corector de paritate în care datele de ieşire vor avea numai paritate pară sau impară. Alegerea parității se va face printr-un semnal de intrare **sel_par**.

Soluție

Corectarea parității se realizează prin ataşarea unui bit pe poziția cea mai semnificativă a informațiilor de intrare.

În figura de mai jos este prezentată entitatea modulului digital:



sel_par = 0 – paritate pară

sel_par = 1 – paritate impară

De exemplu:

Infirmații la intrare **a[6..0]**	Bit paritate **sel_par**	Informații la ieşire **y[7..0]**
1001001	0 – paritate pară	**1**1001001
1001001	1 – paritate impară	**0**1001001
0001100	0 – paritate pară	**0**0001100
0001100	1 – paritate impară	**1**0001100

Un număr binar este considerat ca având paritate *pară*, dacă numărul de biți conținuți cu valoarea „1” logic este par (inclusiv valoarea 0).

Un număr binar este considerat ca având paritate *impară*, dacă numărul de biți conținuți cu valoarea „1” logic este impar .

Determinarea parității unui număr binar se realizează prin efectuarea operației logice XOR între toți biții acestuia.

În cazul exemplului nostru, relația de determinare a parității este următoarea:

$$paritate = a[6] XOR\, a[5] XOR\, a[4] XOR\, a[3] XOR\, a[2] XOR\, a[1] XOR\, a[0]$$

Dacă se doreşte ca informația de ieşire să aibă paritate pară, atunci la informația de intrare se ataşează bitul de paritate corespunzător $y = paritate \,\&\, a$;

Dacă se doreşte paritate impară, la informația de intrare se ataşează bitul de paritate obținut prin complementare $y = not(paritate) \,\&\, a$;

Programul VHDL este următorul:

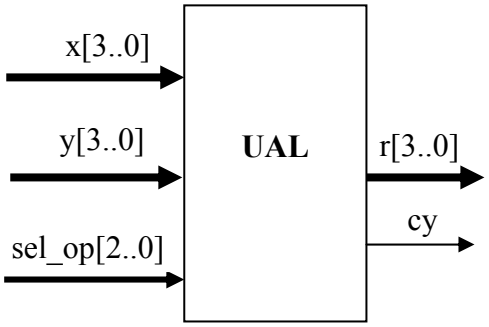
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Corect_Parit is
   Port ( a : in std_logic_vector(6 downto 0);
          sel_par : in std_logic;
          y : out std_logic_vector(7 downto 0));
end Corect_Parit;

architecture Behavioral of Corect_Parit is
signal paritate : std_logic;
begin

        paritate <= a(6)XOR a(5)XOR a(4)XOR a(3)XOR a(2)XOR a(1)XOR a(0);

        with sel_par select
                y <= paritate & a when '0',
                        (not paritate) & a when others;

end Behavioral;
```

d) Descrieți în limbajul VHDL o structură digitală ce implementează o unitate aritmetico-logică (UAL) ce procesează doi operanzi de intrare **x**, **y** reprezentați pe 4 biți şi obține rezultatul **r** prezentat tot pe patru biți. Operațiile realizate de către UAL sunt prezentate în tabelul de mai jos:

sel_op[2..0]	**operație**	**funcție**	**unitate**
000	r <= x	Transferă x la ieşire	Aritmetică
001	r <= y	Transferă y la ieşire	
010	r <= x + y	Adună x cu y (fără transport)	
011	cy,r <= x + y	Adună x cu y (cu transport)	
100	r <= x AND y	ŞI logic	Logică
101	r <= x OR y	SAU logic	
110	r <= NOT x	Complement x	
111	r <= NOT y	Complement y	

Soluție

Entitatea unității aritmetico-logice este simbolizată în figura de mai jos:



Selecția între unitatea logică şi unitatea aritmetică se face cu bitul cel mai semnificativ al semnalului **sel_op**. Biții **sel_op[0]** şi **sel_op[1]** sunt utilizați în selecția funcțiilor logice sau aritmetice corespunzătoare celor două unități de calcul.

Soluția prezentată în programul următor este realizată integral în domeniul concurent. Cele două unități de calcul (aritmetic şi logic) operează, de asemenea, în paralel realizând calculele cu cei doi operanzi de intrare **x**, respectiv **y**.

Implementările acestor operații sunt posibile datorită includerii pachetului de funcții **std_logic_unsigned** din biblioteca **IEEE**. În cadrul programului mai sunt utilizate două semnale care fac legătura dintre unitatea aritmetică, respectiv unitatea logică cu multiplexorul de la ieşire.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UAL is
   Port ( x, y : in std_logic_vector(3 downto 0);
          sel_op : in std_logic_vector(2 downto 0);
          r : out std_logic_vector(3 downto 0);
          cy : out std_logic);
end UAL;

architecture Behavioral of UAL is
          signal t_arith : std_logic_vector(4 downto 0);
          signal t_logic : std_logic_vector(3 downto 0);

begin

          --descrierea unitatii aritmetice
          with sel_op(1 downto 0) select
                    t_arith <= '0'& x when "00",
                               '0'& y when "01",
                               '0'& x + '0'& y when "10",
                               '0'& x - '0'& y when others;

          --descrierea unitatii logice
          with sel_op(1 downto 0) select
                    t_logic <= x and y when "00",
                               x or y when "01",
                               not x when "10",
                               not y when others;

          --multiplexarea intre unitatea logica si unitatea aritmetica

          r  <= t_arith(3 downto 0)  when sel_op(2)='0' else
                    t_logic;
  cy <= t_arith(4) when sel_op(2)='0' else
                    '0';

end Behavioral;
```

Cele două unități de calcul, aritmetic şi logic, au la bază specificații concurente selective. Pentru multiplexor s-a folosit o specificație concurentă condițională.

În unitatea aritmetică s-a lucrat cu operanzii pe 5 biți astfel încât să se obțină rezultatul operației pe 4 biți şi transport pe 1 bit. Operanzii au fost măriți cu un bit (de la 4 biți la 5 biți) prin operatorul de concatenare "**&**".

De exemplu, semnalul **x** este reprezentat pe 3 biți, dar scriindu-l sub forma **'0' & x** rezultă un semnal pe 4 biți, bitul cel mai semnificativ fiind 0 logic.

Bitul $\mathbf{c_y}$ este extras numai atunci când are loc operația aritmetică, adică **sel_par[2]='1'**.

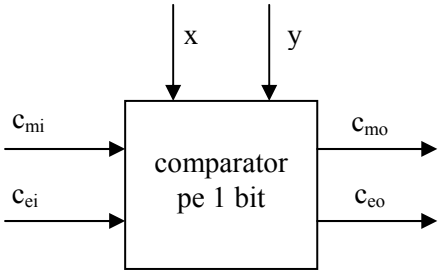
e) Realizați implementarea în cod VHDL a unui modul digital care compară două valori binare **x**, **y** reprezentate pe 4 biți şi semnalizează relația dintre acestea.

Soluție

Realizarea unui comparator direct pe mai mulți biți utilizând ecuații booleene este dificilă. Pentru simplificarea soluției, se foloseşte un comparator general pe un singur bit ce poate fi plasat în cascadă cu alte module de acelaşi tip, formând comparatorul pe mai mulți biți. Comparația se va face secvențial începând cu biții cei mai semnificativi.

Etapa 1. Realizarea comparatorului pe un singur bit.

Comparatorul pe un singur bit trebuie să poată semnaliza următoarele relații dintre operatorii x şi y : $x < y$, $x > y$ şi $x = y$. În plus, comparatorul conține două porturi de intrare, c_{mi} şi c_{ei}, care indică starea modulului comparator anterior (pentru biții cu grad mai mare de semnificație). Dacă $c_{mi} = 1$ atunci $x > y$, iar dacă $c_{ei} = 1$, între operatorii de intrare există egalitate. Acest modul conține şi două porturi de ieşire, c_{mo} şi c_{eo} prin care se semnalizează relațiile de ordine dintre operatorii de intrare către modulele de comparație ulterioare (pentru biții cu grad mai mic de semnificație). Modulul digital are următoarea schemă:



c_{mi}	c_{ei}	x	y	relație	c_{mo}	c_{eo}
0	0	X	X	<	0	0
0	1	0	0	=	0	1
0	1	0	1	<	0	0
0	1	1	0	>	1	0
0	1	1	1	=	0	1
1	0	X	X	>	1	0
1	1	X	X	invalidă	1	1

Ecuațiile booleene simplificate rezultate din tabelul de adevăr sunt următoarele:

$$c_{mo} = c_{mi} + c_{ei} \cdot x \cdot \overline{y}$$

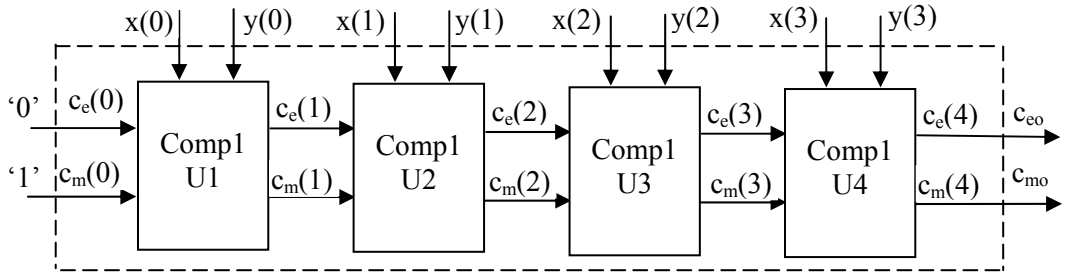
$$c_e = c_{mi} \cdot c_{ei} + c_{ei} \cdot \overline{x} \cdot \overline{y} + c_{ei} \cdot x \cdot y$$

Programul sursă în VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp1 is
        port(
                x : in STD_LOGIC;
                y : in STD_LOGIC;
                cmi : in STD_LOGIC;
                cei : in STD_LOGIC;
                cmo : out STD_LOGIC;
                ceo : out STD_LOGIC
                );
end comp1;

architecture comp1 of comp1 is
begin

        cmo <= cmi or (cei and x and (not y));
       ceo <= (cmi and cei)or(cei and(not x)and (not y))or(cei and x and y);

end comp1;
```

Etapa 2. Realizarea comparatorului pe 4 biți.

O primă soluție constă în realizarea comparatorului pe 4 biți utilizând comparatoare pe un singur bit conectate în cascadă, după cum se prezintă în figura de mai jos:



Cascadarea se realizează prin declararea unei componente **comp1** şi interconectarea (instanțierea) acesteia prin intermediul specificației PORT MAP. Programul sursă pentru descrierea comparatorului pe 4 biți este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity comp4 is
        port(
                x : in STD_LOGIC_VECTOR(3 downto 0);
                y : in STD_LOGIC_VECTOR(3 downto 0);
                cmo : out STD_LOGIC;
                ceo : out STD_LOGIC
                );
end comp4;

architecture comp4 of comp4 is
component comp1
port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        cmi : in STD_LOGIC;
        cei : in STD_LOGIC;
        cmo : out STD_LOGIC;
        ceo : out STD_LOGIC
        );
end component;
signal cm,ce:std_logic_vector(4 downto 0);
begin
 cm(0)<='0';
 ce(0)<='1';
 U1: comp1 port map(x(0),y(0),cm(0),ce(0),cm(1),ce(1));
 U2: comp1 port map(x(1),y(1),cm(1),ce(1),cm(2),ce(2));
 U3: comp1 port map(x(2),y(2),cm(2),ce(2),cm(3),ce(3));
 U4: comp1 port map(x(3),y(3),cm(3),ce(3),cm(4),ce(4));
 cmo<=cm(4);
 ceo<=ce(4);
end comp4;
```

O a doua soluție, optimizată, ce permite generalizarea constă în realizarea comparatorului pe *n* biți utilizând specificația FOR-GENERATE.

În structura comparatorului pe 4 biți se observă că este utilizat doar modulul **comp1** şi este interconectat repetitiv de patru ori. Acest lucru dă posibilitatea ca descrierea structurală a comparatorului pe 4 biți să fie realizată cu specificația FOR GENERATE ca în programul de mai jos:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp4 is
        port(
                x : in STD_LOGIC_VECTOR(3 downto 0);
```

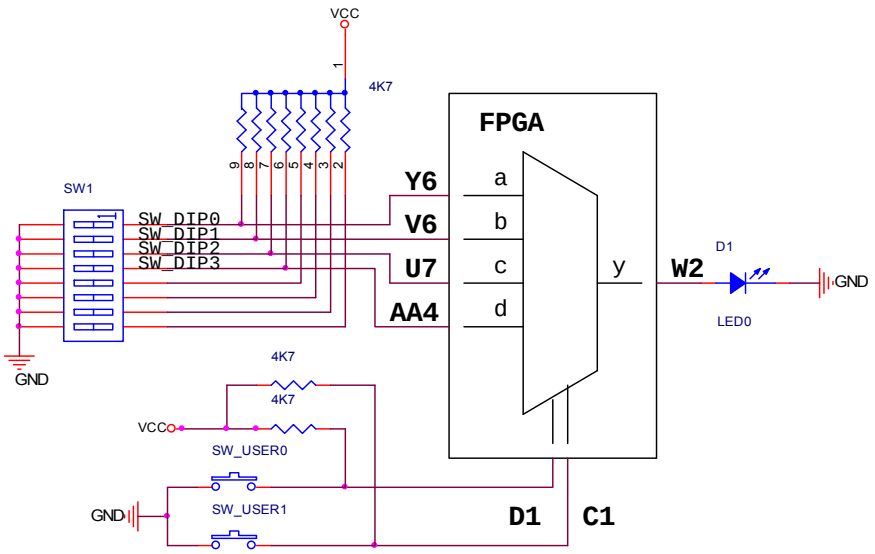
```
                    y : in STD_LOGIC_VECTOR(3 downto 0);
                    cmo : out STD_LOGIC;
                    ceo : out STD_LOGIC
                    );
end comp4;

architecture comp4 of comp4 is
component comp1
port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        cmi : in STD_LOGIC;
        cei : in STD_LOGIC;
        cmo : out STD_LOGIC;
        ceo : out STD_LOGIC
        );
end component;
signal cm,ce:std_logic_vector(4 downto 0);

begin

cm(0)<='0';
ce(0)<='1';

unit: for i in 0 to 3 generate
U1: comp1 port map(x(i),y(i),cm(i),ce(i),cm(i+1),ce(i+1));
end generate;

cmo<=cm(4);
ceo<=ce(4);

end comp4;
```

IV. DESFĂŞURAREA LUCRĂRII

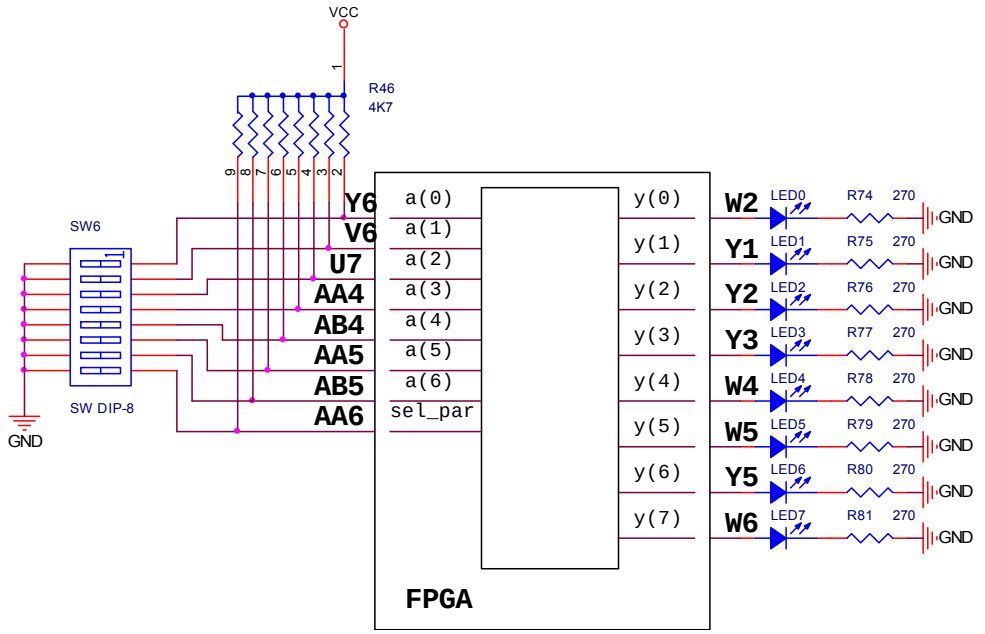
1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se vor implementa programele din exemplele a) … e) folosind schemele hardware de mai jos (2.1 … 2.4). Implementarea se va realiza în următorii paşi:
   - pentru fiecare aplicație în parte se realizează un proiect nou;
   - se introduc sursele VHDL;
   - se simulează logic proiectul;
   - se creează fişierele de constrângeri ale pinilor după schemele hardware de mai jos (2.1 … 2.4);
   - se realizează sinteza acestora;
   - se vizualizează schemele logice, respectiv tehnologice;
   - se simulează modulul rezultat după sinteză;
   - se implementează proiectul;
   - se vizualizează rutarea structurii fizice;
   - se realizează simularea Post-Place & Post Route a modulului;
   - se configurează circuitul fizic.

Notă: Circuitul configurabil FPGA este de tipul **3s400fg456** cu speed grade **-4**

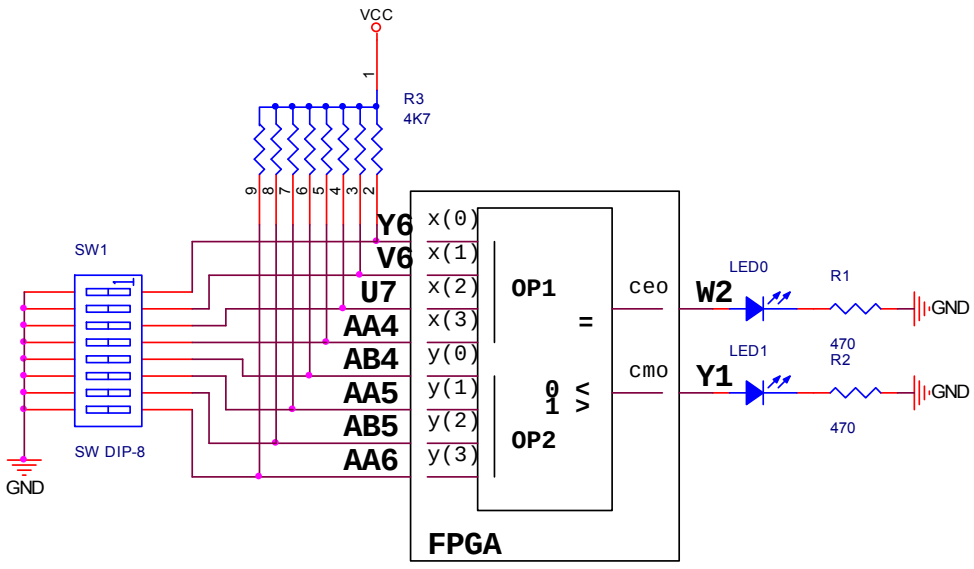
2.1. Multiplexor 4:1 (exemplele a şi b)



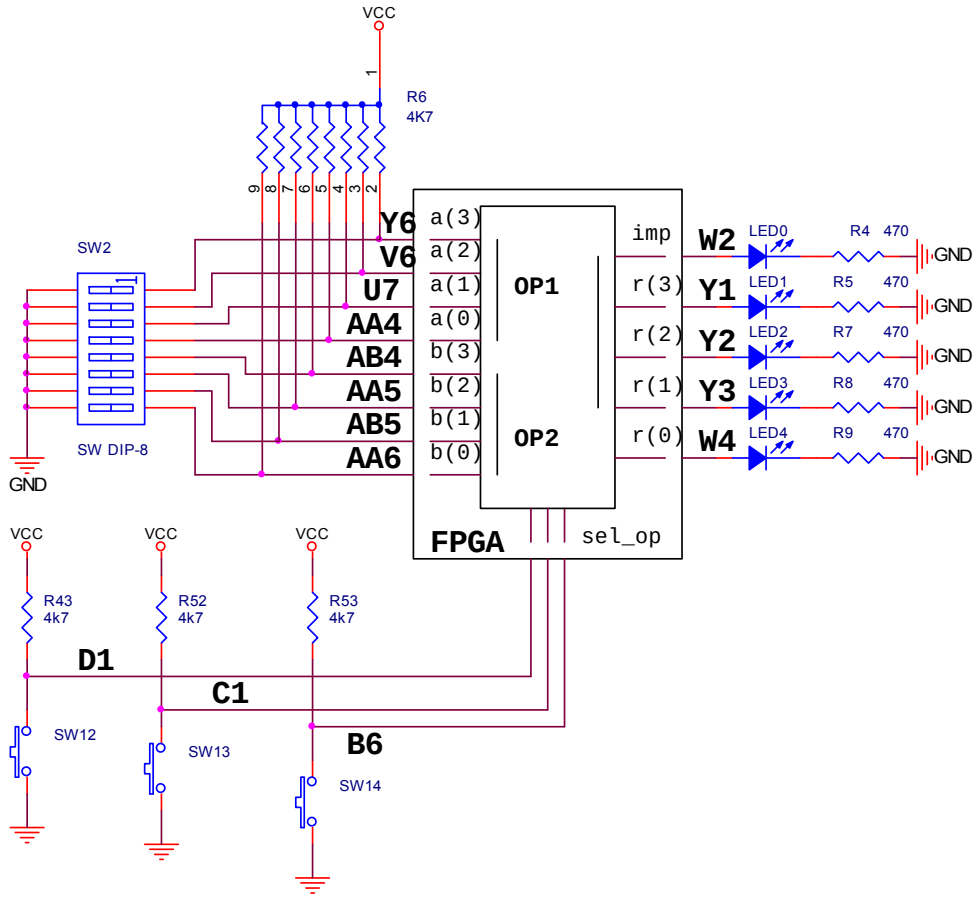
2.2. Corectorul de paritate (exemplul c)



2.3. Comparatorul pe 4 biți (exemplul d)



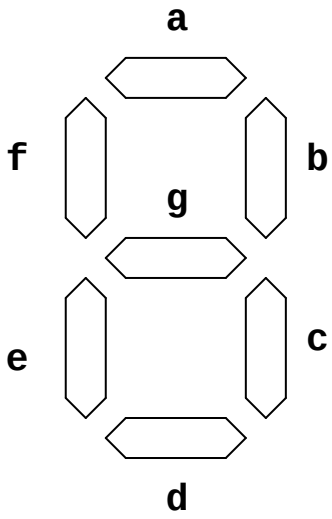
2.4. Unitate aritmetico-logică (exemplul e)

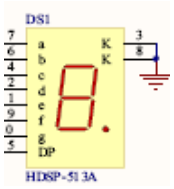


Legăturile dintre pinii fizici ai structurii reconfigurabile şi porturile din entitățile modulelor descrise în VHDL sunt specificate în schemele electrice anterioare.

Componentele din schemele electrice (butoane, LED-uri, rezistențe) sunt prezente în schema sistemului de dezvoltare. Entitatea modulului digital se regăseşte în chenarul notat: **FPGA**.

3. Să se realizeze un decodor pe 4 biți ce va permite afişarea valorii în format zecimal a informației binare de intrare, sub forma numerelor de la 0 la 9, utilizând un digit cu şapte segmente. Valorile binare de intrare sunt aplicate cu ajutorul butoanelor USER0, USER1, USER2, USER3 de pe sistemul de dezvoltare.

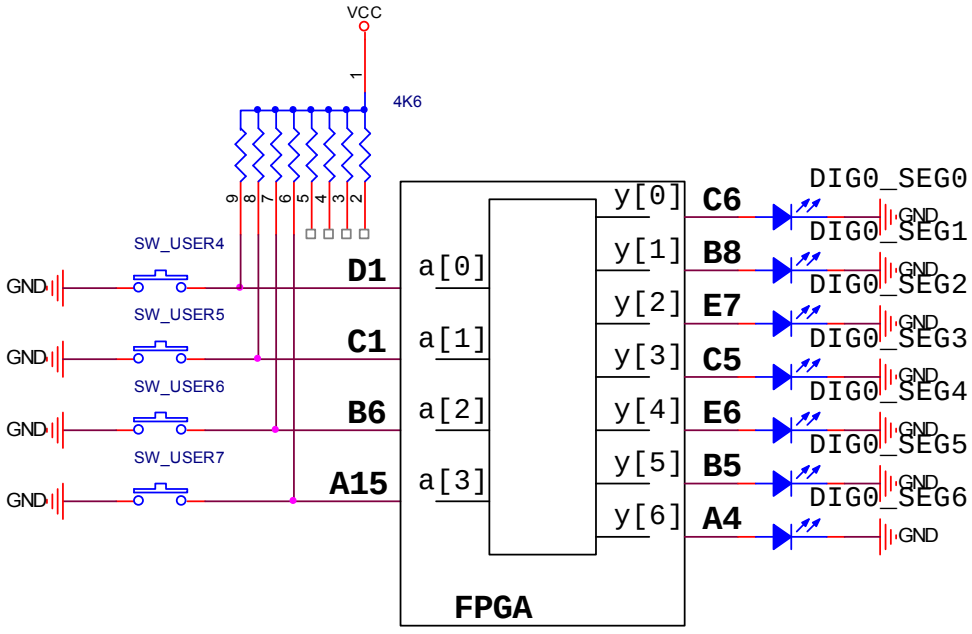




Corespondența dintre liniile circuitului fpga şi semnalele ce acționează cele 7 segmente este următoarea:

D_7	D_6	D_5	D_5	D_3	D_2	D_1	D_0
dp	g	f	e	d	c	b	a

dp –dot point



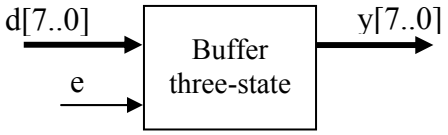
Să se realizeze acest modul prin două metode: descriere de tip flux de date, respectiv descriere structurală.

4. Să se realizeze în limbajul VHDL un buffer inversor pe 8 biţi cu ieşirile de tip three-state (tree-state sau TS) utilizând numai specificaţii concurente.

e	y
0	Z
1	d

Numele de buffer three-state provine de la ieşirea acestuia care poate lua una din cele trei stări posibile: ‘1’ logic, ‘0’ logic sau ‘Z’, aceasta fiind starea de înaltă impedanţă - HiZ. În general buffer-ele de tip three-state sunt utilizate pentru conectarea unor module digitale la magistralele de date. Un etaj de ieşire three-state permite cuplarea în paralel a mai multor ieşiri. Într-o astfel de configuraţie, fiecare ieşire poate fi activă la un moment dat (are acces la magistrala de ieşire comună), dar celelalte ieşiri trebuie să fie în starea de înaltă impedanţă.

Tabelul de adevăr şi porturile de interconectare ale unui astfel de buffer sunt date în figura de mai jos:



5. Să se realizeze în limbaj VHDL un modul digital care afişează numărul de biţi aflaţi în starea 1 logic dintr-o informaţie pe 7 biţi ce se aplică la un port de intrare.

Corelarea cu schema fizică a porturilor de intrare/ieşire se va face după schema electrică a sistemului reconfigurabil.

Intern, modulul digital va fi realizat din două module descrise cu specificaţii concurente şi interconectate prin descriere structurală.