Programarea structurilor hardware reprogramabile prin descrieri cu specificații secvențiale

I. SCOPUL LUCRĂRII

În această lucrare sunt prezentate implementări cu descrieri secvențiale ale modulelor digitale ce conțin logică sincronă sau asincronă. Sunt realizate mai multe exemple în care sunt cuprinse o parte din specificațiile secvențiale ce vor fi sintetizate şi implementate pe sistemul reconfigurabil din cadrul laboratorului. În final sunt propuse probleme din aceeaşi categorie în vederea implementării.

II. INTRODUCERE TEORETICĂ

1.1. Semnalele şi variabilele în domeniul secvențial

Transportul datelor în VHDL poate fi realizat prin semnale sau prin variabile. În timp ce semnalele pot fi declarate în domeniul concurent, variabilele pot fi declarate numai în domeniul secvențial. Semnalul poate fi utilizat global, în domeniile concurente şi secvențiale, iar variabila este numai locală domeniului secvențial. Domeniul secvențial poate fi descris prin procese, funcții, respectiv proceduri.

Un proces defineşte o listă de specificații secvențiale prin care se descrie comportamental o anumită structură dintr-un modul digital. Într-un proces, dacă nu este necesară utilizarea unui semnal, se poate utiliza o variabilă. Semnalul nu poate fi declarat într-un proces. Valoarea unui semnal este afectată numai la ieşirea din proces de ultima atribuire, chiar dacă asupra lui s-au făcut alte atribuiri în timpul procesului.

Declarația unei variabile se poate face prin utilizarea următoarei sintaxe:

```
variable nume_variabilă : tip;
variable nume_variabilă : tip := valoare_inițială;
```

Atribuirea unei variabile se realizează după sintaxa:

```
variable_name := expression ;
```

1.2. Introducerea unui proces

Specificațiile secvențiale pot fi introduse prin intermediul clauzei „PROCESS". Procesele sunt activate printr-o listă de senzitivități. Dacă lista de senzitivități lipseşte, activarea procesului se realizează prin specificația WAIT.

De menționat faptul că procesele conțin descrieri secvențiale, dar între ele sunt concurente.

Sintaxă:

```
[eticheta:] PROCESS (lista de senzitivitati)
        [VARIABILE nume: tip [dimensiune] [:= valoare_initiala;]]
BEGIN
        (cod secvential)
END PROCESS [eticheta];
```

Între clauzele **process** şi **begin** se găseşte zona declarativă în care pot fi declarate variabile, tipuri, subprograme, atribute, etc. Zona de descriere secvențială este definită între clauzele **begin** şi **end**.

1. 3. Specificații secvențiale

Specificația IF

Specificația IF este utilizată în structuri condiționale şi are următoarea sintaxă:

```
IF conditie THEN specificatii_secventiale;
ELSIF conditie THEN specificatii_secventiale;
…..
ELSE specificatii_secventiale;
END IF;
```

Datorită influenței puternice a mediilor de programare software (de exemplu C++, PASCAL, etc.), tendința programatorilor este de a utiliza structurile condiționale în descrierea comportamentului unui modul digital, fără a mai face recurs la descrierile de tip flux de date prin ecuații booleene sau alte specificații care ocupă o arie hardware mult mai mică. Totuşi, utilizarea specificației IF nu afectează, în principiu, structura hardware foarte mult. Explicația este dată de faptul că în procesul de sinteză se produce o optimizare a ecuațiilor logice şi evitându-se astfel mărirea complexității hardware nejustificate. Totuşi, este indicat ca imbricarea specificațiilor **IF** să nu se facă pe prea multe nivele.

IF este o specificație secvențială care nu poate fi utilizată în zona concurentă a unei arhitecturi şi este diferită de specificația **IF GENERATE** din domeniul concurent.

Exemplu de utilizare a unei structuri condiționale IF

```
IF (reset = '1') THEN data_out <= (others => ,1')
ELSIF (clk='1' AND clk'event) THEN data_out <= data_in;
ELSE data_out <= (others => ,Z');
END IF;
```

În exemplul de mai sus dacă semnalul **RESET** este activ în 1 logic, semnalul de ieşire va pune toate liniile acestuia în 1 logic. Dacă semnalul **RESET** nu este activ şi a avut loc un eveniment pe frontul pozitiv al semnalului **CLK**, semnalul de ieşire primeşte valorile semnalului de intrare. În caz contrar semnalul de ieşire este trecut în starea de înaltă impedanță.

Specificația WAIT

Specificația **WAIT** este utilizată în cazurile în care procesul nu are o listă de senzitivități. Specificația poate fi utilizată sub trei forme, după cum se prezintă în sintaxele următoare:

```
WAIT UNTIL conditie_semnal
WAIT ON semnal1 [, semnal2, …];
WAIT FOR time;
```

Prima sintaxă este utilizată în general pentru modelele digitale sincrone, fiind mai puțin folosită la cele asincrone. Acest lucru se datorează faptului că prin specificația WAIT UNTIL este introdusă o condiție asupra unui semnal de care nu se poate trece până când condiția respectivă nu este îndeplinită.

Cea de-a doua sintaxă este utilizată atunci când sunt monitorizate mai multe semnale. Procesul devine activ numai când unul din semnalele aflate în lista specificației WAIT ON îşi schimbă starea.

În final, ultima specificație WAIT FOR este introdusă numai pentru simularea modulelor digitale în fişierele test. Această specificație nu este sintetizabilă.

De exemplu: WAIT FOR 100ns

Specificațiile WAIT sunt plasate imediat după BEGIN în cadrul unui proces.

Specificația CASE

Specificația CASE este utilizată pentru selectarea unei alternative în funcție de valoarea unei expresii.

```
CASE identificator IS
        WHEN value => atribuire;
        WHEN value => atribuire;
        …
```

```
        WHEN OTHERS => atribuire;
END CASE;
```

Specificația CASE evaluează o expresie şi selectează una din alternative în concordanță cu valoarea acesteia. Expresia de evaluare poate fi de natură discretă (numere întregi) sau şir de caractere. CASE conține o listă de alternative care încep cu specificația WHEN. Este urmată de valoarea corespunzătoare alternativei respective şi de specificațiile secvențiale care trebuie executate în cazul în care este aceasta este aleasă.

Clauza OTHERS este folosită atunci când sunt luate în considerare şi alte valori ale identificatorului ce nu sunt prevăzute în alternativele cuprinse în WHEN.

Specificația LOOP

Specificația LOOP este utilizată pentru repetarea unor secvențe de cod VHDL, după o anumită condiție WHILE/LOOP sau repetitiv cu specificația FOR/LOOP.

FOR/LOOP – bucla este repetată de un număr de ori predefinit care nu se mai poate schimba după intrarea în aceasta.

```
[eticheta:] FOR identificator IN interval LOOP
        (specificatii secventiale)
END LOOP [eticheta:];
```

WHILE-LOOP – bucla este repetată până când nu mai este îndeplinită condiția.

```
[eticheta:] WHILE conditie LOOP
        (specificatii secventiale)
END LOOP [eticheta:];
```

EXIT – este utilizată pentru terminarea forțată a unei bucle.

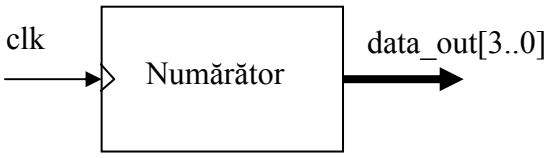
```
[eticheta:] EXIT [eticheta] [WHEN conditie];
```

NEXT – este folosită pentru omiterea unui pas într-o buclă.

```
[eticheta:] NEXT [eticheta bucla] [WHEN conditie];
```

III. APLICAȚII

a) Folosind limbajul VHDL, să se realizeze un numărător BCD către înainte având porturile de intrare/ieşire precizate în figura de mai jos:



Soluție

Modulul digital conține doar un semnal de intrare **clk** şi la ieşire un semnal pe patru biți denumit **data_out**. Exemplul realizat utilizează specificația **IF THEN**.

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

entity numarator is
	port(
		clk : in STD_LOGIC;
		data_out : out STD_LOGIC_VECTOR(3 downto 0)
		);
end numarator;

architecture numarator of numarator is
begin
	PROCESS (clk)
		VARIABLE  temp_num:  STD_LOGIC_VECTOR(3  downto
0):="0000";
	BEGIN
		IF (clk'event AND clk='1')THEN
			temp_num:=temp_num+1;
			IF (temp_num="1010") THEN
				temp_num:=(OTHERS => '0');
			END IF;
		END IF;

		data_out<=temp_num;
	END PROCESS;
end numarator;
```

Descrierea numărătorului este de tip comportamental prin specificații secvențiale în cadrul unui proces. Procesul este activat numai la apariția unui eveniment pe frontul pozitiv al semnalului **clk**, fiind plasat în lista de senzitivități a acestuia. În zona declarativă a procesului este introdusă variabila **temp_num**, cu valoarea inițială 0 şi ce contorizează stările la numărare. Numărarea se efectuează pe frontul crescător a semnalului de ceas prin linia de cod:

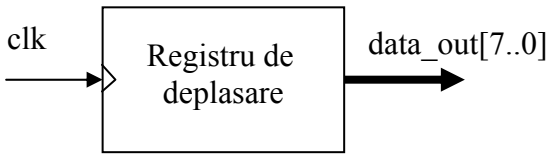
```
IF (clk'event AND clk='1')THEN
```

Menținerea în domeniul zecimal a valorii variabilei **temp_num** (în gama de la 0 la 9) se obține prin linia de cod:

```
IF (temp_num="1010") THEN
```

Cum semnalele se pot actualiza numai la ieşirea din proces, atribuirea semnalului de ieşire **data_out** cu variabila **temp_num** s-a făcut la sfârşitul procesului.

b) Folosind limbajul VHDL, să se realizeze un registru de deplasare în inel pe 8 biți, de la dreapta la stânga, acționat pe frontul pozitiv al semnalului de ceas **clk**. Valoarea inițială a registrului este „00000001". La fiecare impuls de ceas deplasarea se face cu un bit, iar bitul cel mai puțin semnificativ rămâne în starea 0 logic.



Soluție

Programul VHDL este următorul:

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

entity process_shift is
        port(
                clk : in STD_LOGIC;
                data_out : out STD_LOGIC_VECTOR(7 downto 0)
                );
end process_shift;

architecture process_shift of process_shift is
begin
```

```
PROCESS
        VARIABLE temporar:STD_LOGIC_VECTOR(7 downto
0):="00000001";
        BEGIN
                WAIT UNTIL (clk'event AND clk='1');
                        data_out<=temporar;
                        temporar:=temporar(6 downto 0)&'0';
        END PROCESS;
end process_shift;
```

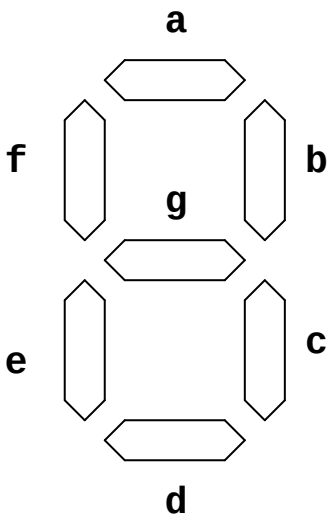
Procesul nu prezintă o listă de senzitivități datorită utilizării specificației WAIT UNTIL. Respectiva specificație nu permite intrarea în proces dacă nu a fost îndeplinită condiția de activare (clk'event AND clk='1'). Variabila **temporar** este deplasată la stânga cu un bit de fiecare dată când este activat procesul. La ieşirea din proces, semnalul **data_out** primeşte valoarea curentă a variabilei **temporar**.

În zona declarativă a procesului, variabila **temporar** este inițializată cu valoarea „00000001”.

c) Folosind limbajul VHDL să se realizeze un numărător zecimal cu afişare pe un digit de tip display cu 7 segmente. Numărarea se va face pe frontul pozitiv al semnalului de ceas aplicat, de la 0 la 9.

Soluție

Un display cu 7 segmente este format din leduri ordonate ca în figura de mai jos. Tabela de adevăr pentru translația din codul binar în cod corespunzător celor 7 segmente este următoarea:



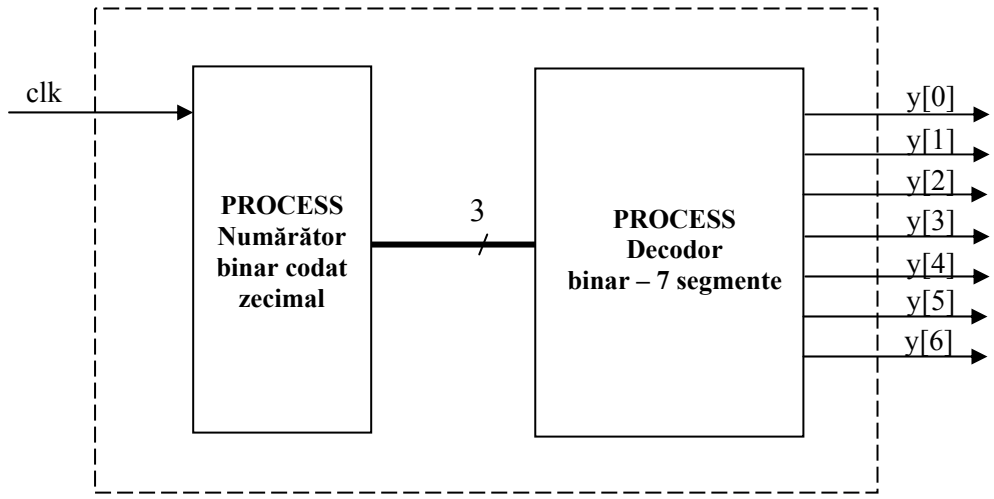
nr. binar	cod 7 segmente gfedcba
0000	0111111
0001	0000110
0010	1011011
0011	1001111
0100	1100100
0101	1101101
0110	1111101
0111	0000111
1000	1111111
1001	1101111

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
-	g	f	e	d	c	b	a

Ledurile se aprind atunci când pe liniile de comandă ale display-ului pe 7 segmente se generează valoarea "1" logic.

Soluția prezentată exemplifică utilizarea specificației CASE. Entitatea corespunzătoare structurii descrise are următoarea configurație.



Descrierea comportamentală a numărătorului se realizează prin două procese. În primul proces, numărătorul zecimal este descris printr-o logică secvențială. Al doilea proces descrie decodorul binar - 7 segmente, în mod secvențial, printr-o logică combinațională care respectă tabela de adevăr prezentată anterior.

Programul VHDL este următorul:

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

entity num_7seg is
        port(
                clk : in STD_LOGIC;
                y : out STD_LOGIC_VECTOR(6 downto 0)
                );
end num_7seg;

architecture num_7seg of num_7seg is
        SIGNAL semnal_temp:STD_LOGIC_VECTOR(3 downto 0);

begin
      PROCESS (clk)
        VARIABLE temp_num: STD_LOGIC_VECTOR(3 downto 0):="0000";
      BEGIN
```

```
                IF (clk'event AND clk='1')THEN
                        temp_num:=temp_num+1;
                        IF (temp_num="1010") THEN
                                temp_num:=(OTHERS => '0');
                        END IF;
                END IF;
                semnal_temp<=temp_num;
        END PROCESS;

        PROCESS (semnal_temp)
                VARIABLE iesire_temp: STD_LOGIC_VECTOR(6 downto 0);
        BEGIN
                CASE semnal_temp IS
                        WHEN "0000"=> iesire_temp:="0111111";
                        WHEN "0001"=> iesire_temp:="0000110";
                        WHEN "0010"=> iesire_temp:="1011011";
                        WHEN "0011"=> iesire_temp:="1001111";
                        WHEN "0100"=> iesire_temp:="1100100";
                        WHEN "0101"=> iesire_temp:="1101101";
                        WHEN "0110"=> iesire_temp:="1111101";
                        WHEN "0111"=> iesire_temp:="0000111";
                        WHEN "1000"=> iesire_temp:="1111111";
                        WHEN "1001"=> iesire_temp:="1101111";
                        WHEN OTHERS=> iesire_temp:="ZZZZZZZ";
                END CASE;
                y<=iesire_temp;
        END PROCESS;
end num_7seg;
```

Cele două procese pot fi văzute ca două module digitale separate. Acestea sunt legate între ele printr-un semnal denumit **semnal_temp**.

Primul proces este senzitiv la semnalul de ceas şi incrementează variabila **temp_num** de fiecare dată când apare un front crescător pe **clk**. Dacă variabila **temp_num** a ajuns la valoarea binară 1010 este automat resetă în 0000. La ieşirea din proces, semnalul **semnal_temp** preia valoarea variabilei **temp_num**.

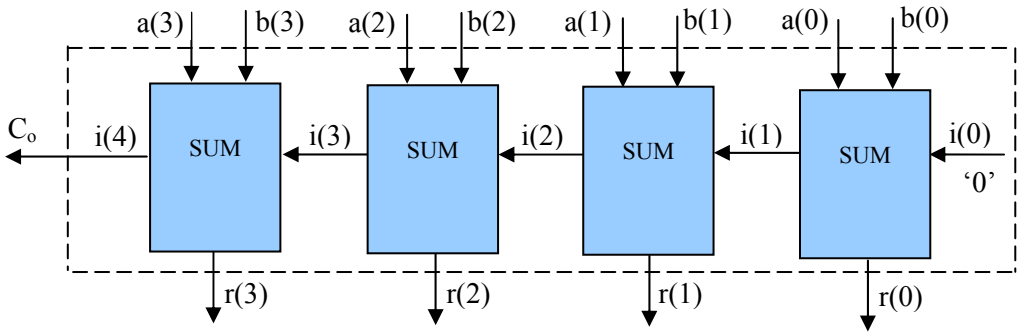
Al doilea proces este asincron şi este senzitiv la semnalul intern **semnal_temp**. În acest proces, prin specificația CASE se face transformarea din codul BCD în 7 segmente. Valoarea corespunzătoare cifrei ce trebuie afişată pe display-ul cu 7 segmente este salvată în variabila **ieşire_temp**. La ieşirea din proces, semnalul **y** preia valoarea variabilei **ieşire_temp**.

d) Folosind limbajul VHDL să se implementeze un sumator binar pe 4 biți, utilizând specificația secvențială LOOP.

Soluție

În limbajul VHDL, un sumator se poate realiza prin mai multe metode. Una dintre metode (descriere stucturală) a fost utilizată la modulul comparator pentru două numere binare, reprezentate pe patru biți, din lucrarea de laborator 2. În acel exemplu s-au utilizat specificații concurente. S-a realizat operația de comparare pentru un singur bit, după care a fost multiplicată de patru ori prin instanțierea de componente.

Soluția ce urmează prezintă o altă metodă de rezolvare (descriere comportamentală) ce utilizează specificații secvențiale. În acest caz nu mai este necesară multiplicarea sumatorului pe un singur bit (vezi cazul comparatorului), ci utilizarea repetitivă, ce are în vedere faptul că adunarea se realizează secvențial, rezultatul obținându-se bit după bit, cu folosirea transportului. Procedura de adunare utilizată este cunoscută sub denumirea Ripple Carry. Schema bloc a modului de implementare a sumatorului digital este prezentată în figura de mai jos:



Ecuațiile booleene ale celulei de sumare pe un singur bit sunt următoarele:

$$r = a \oplus b \oplus c_i$$
$$c_o = a \cdot b + c_i (a \oplus b)$$

Se constată faptul că ieşirea $\mathbf{c_o}$ a celulei k este conectată la intrarea $\mathbf{c_i}$ de la celula k+1 prin semnalul intern **i(k+1)**.

Aplicând ecuațiile anterioare pentru fiecare celulă a sumatorului se obțin relațiile recurente:

$$r(k) = a(k) \oplus b(k) \oplus i(k)$$
$$c_o = i(k+1) = a(k) \cdot b(k) + i(k)(a(k) \oplus b(k))$$

pentru $k \in [0,3]$

Implementarea relațiilor booleene de mai sus poate fi realizată cu specificația LOOP. Sumatorul pe 4 biți este descris prin următorul cod sursă VHDL.

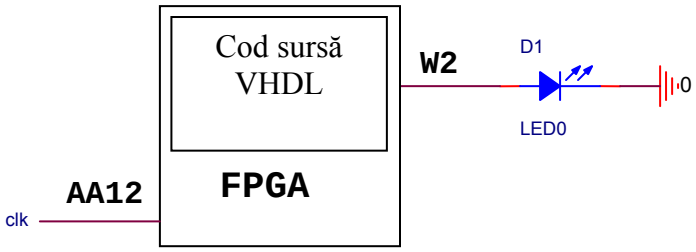
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sumator_loop is
        port(
                a : in STD_LOGIC_VECTOR(3 downto 0);
                b : in STD_LOGIC_VECTOR(3 downto 0);
                co : out STD_LOGIC;
                r : out STD_LOGIC_VECTOR(3 downto 0)
                );
end sumator_loop;

architecture sumator_loop of sumator_loop is
begin
        process(a,b)
                variable i:     STD_LOGIC_VECTOR(4 downto 0);
                variable rez:   STD_LOGIC_VECTOR(3 downto 0);
        begin

                i(0):='0';
                for k in 0 to 3 loop
                        rez(k):=a(k) xor b(k) xor i(k);
                        i(k+1):=(a(k) and b(k)) or (i(k) and (a(k) xor b(k)));
                end loop;
                r<= rez;
                co <= i(4);

        end process;

end sumator_loop;
```

Programul de mai sus nu utilizează specificații concurente pentru realizarea sumatorului pe patru biți. Practic, au fost utilizate două relații booleene (pentru rezultat şi pentru transportul de ieşire), care au fost repetate pentru fiecare celulă de sumare elementară, plecând de la bitul cel mai puțin semnificativ.

e) Să se prezinte codul VHDL ce descrie comportamental un temporizator digital folosit pentru aprinderea alternativa a unei diode led, cu frecvența de 1Hz.

Soluție

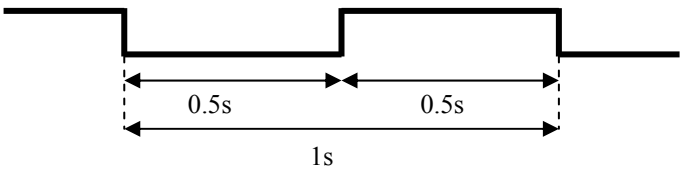
Ideea de realizare a temporizatorului se bazează pe utilizarea unui numărător digital. Semnalul de ceas al numărătorului este preluat de la un oscilator cu cuarț pe frecvența de 50MHz, existent pe sistemul de dezvoltare. Prin divizări repetate se poate obține un semnal cu o frecvență stabilă de 1Hz. Schema bloc de interconectare este dată în figura următoare.



Numărătorul operează cu o constantă dată de raportul dintre frecvența semnalului de intrare **clk** şi o frecvență de două ori mai mare a semnalului de ieşire, conform relației de mai jos:

$$C_{div} = \frac{f_{clk}}{f_{out}} = \frac{50MHz}{2Hz} = 25 \cdot 10^{6}{}_{(10)} = 17D7840_{(16)}$$

Semnalul de ieşire are frecvența de 1Hz şi factor de umplere de 1/2, astfel încât 0.5s este „1” logic, iar 0.5s „0” logic. Perioada de 0.5s corespunde unui semnal cu frecvența de 2Hz.



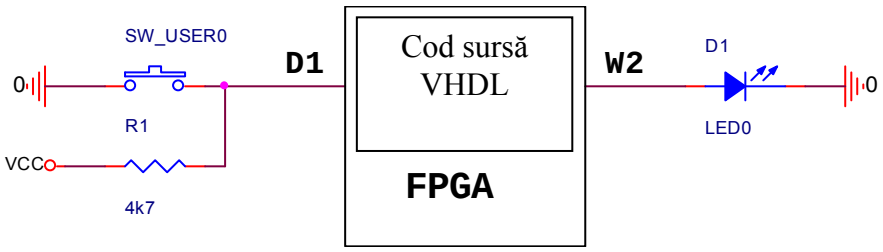
În acest caz, constanta de divizare este:
50MHz/2Hz = $25000000_{(10)}$ = $17D7840_{(h)}$, număr reprezentat pe 25 de biți.

Programul sursă VHDL este următorul:

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

entity temporizator is
        port(
                clk : in STD_LOGIC;
                out_led : buffer STD_LOGIC
                );
end temporizator;
```

```
architecture temporizator of temporizator is
begin
        process (clk,out_led)
                variable Qint: STD_LOGIC_VECTOR (24 downto 0);

                variable temp: STD_LOGIC;
        begin
                temp := out_led;
                if (CLK'event and CLK='1') then
                        Qint:=Qint+1;
                        if (Qint=x"17D7840") THEN
                                Qint:=(others=>'0');
                                out_led <= not temp;
                        end if;
                end if;
        end process;
end temporizator;
```

f) Se consideră un circuit format dintr-un led, o tastă si o structură FPGA conectate ca în schema de mai jos:



Butonul **sw_user0** este conectat la linia **D1** şi ledul **led0** la linia **W2** a structurii de tip FPGA. Se cere să se implementeze un modul digital în cod VHDL care la fiecare apăsare a butonului să schimbe starea ledului din aprins în stins sau invers. Se va ţine seama de efectele parazite introduse de contactul butonului.

Soluţie

În mod normal, apăsarea şi relaxarea unei taste provoacă generarea unui număr nedeterminabil de impulsuri parazite care pot conduce la interpretări eronate legate de starea acesteia. În schimb, se poate aprecia timpul cât durează acest proces tranzitoriu, şi anume aproximativ 10 ms, atât pentru situaţia apăsării tastei cât şi la relaxarea acesteia. În limba engleză, termenul folosit pentru acest timp este *debounce time*. Evitarea efectelor parazite menţionate se realizează prin folosirea de temporizări adecvate. Procedura de lucru constă în determinarea unui prim impuls apărut ca urmare a contactului făcut la tastă, aşteptarea a 10 ms şi reluarea testării tastei apăsate, după scurgerea acestui interval de timp. În cazul în

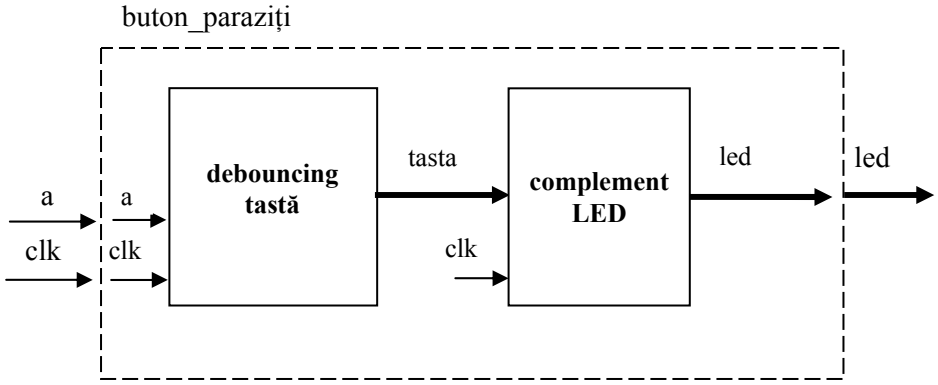
care se detectează menţinerea contactului la tastă, decizia care se ia este "tastă apăsată" şi se tratează ca atare. În caz contrar, se consideră că iniţial a apărut un impuls parazit la apăsarea tastei (sau aceasta nu a fost apăsată ferm), iar decizia care se ia este "tastă neapăsată". Procedura este similară şi pentru tratarea situaţiei de relaxare a respectivei taste cu diferenţa dată de faptul că după trecerea intervalului de timp de aşteptare menţionat, se verifică starea deschisă a contactului.

În condiţiile problemei propuse vom putea detecta apăsarea tastei aflate pe linia **D1** prin faptul că la apăsare semnalul de intrare devine "0" logic. Pauza de 10 ms o vom realiza prin introducerea unui modul temporizator comandat de un semnal de ceas **clk** aflat pe pinul AA12 al circuitului SPARTAN3.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity buton_paraziti is
        port(
                a : in STD_LOGIC;
                led : out STD_LOGIC;
                clk: in STD_LOGIC
                );
end buton_paraziti;

architecture buton_paraziti of buton_paraziti is
        signal tasta: STD_LOGIC:='1';
begin
        process (clk)
                variable Qint: STD_LOGIC_VECTOR (19 downto
0):=(others =>'0');
                variable val_veche: STD_LOGIC:='0';
        begin
                if (CLK'event and CLK='1') then
                        if (a xor val_veche)='1' then
                                Qint:=(others=>'0');
                                val_veche:=a;
                        else
                                Qint:=Qint+'1';
                                if ((Qint="00111111111111111111")
and ((val_veche xor a)='0')) THEN
                                        tasta <= val_veche;
                                end if;
                        end if;
                end if;
        end process;

        process (clk)
                variable temp: std_logic:='0';
```

```
                variable tasta_veche: std_logic:='1';
        begin
                if (CLK'event and CLK='1') then
                        if ( tasta='0' and tasta_veche='1') then
                                temp:=not temp;
                        end if;
                        tasta_veche :=tasta;
                end if;
                        led <=temp;
        end process;
end buton_paraziti;
```

Programul este format din două procese. În primul proces este implementată testarea butonului prin tehnica descrisă anterior. Este utilizat un numărător care se resetează la fiecare apăsare sau relaxare a tastei. Dacă o tastă este apăsată sau relaxată, noua stare a acesteia diferă de cea veche şi atunci se resetează contorul numărătorului. Dacă tasta ținută într-o stare mai mult de 10ms, terminarea temporizării conduce la atribuirea valorii logice a tastei la semnalul **tasta**.



Practic, semnalul **tastă** preia valoarea logică a tastei, dacă aceasta a fost menținută minimum 10ms în aceeaşi stare.

În al doilea proces se realizează complementarea semnalului de comandă al ledului. La tranziția pe frontul crescător al semnalului **tastă**, semnalul de ieşire este complementat prin intermediul variabilei **temp**. S-a folosit variabila **temp** care atribuie valoarea semnalului **led,** după care o returnează acestuia, dar complementată. Pentru evitarea creării unui semnal de ieşire de tip *buffer*, s-a declarat aceasta variabila asupra căruia se fac operațiile de complementare, ce se atribuie ulterior ieşirii **led**.

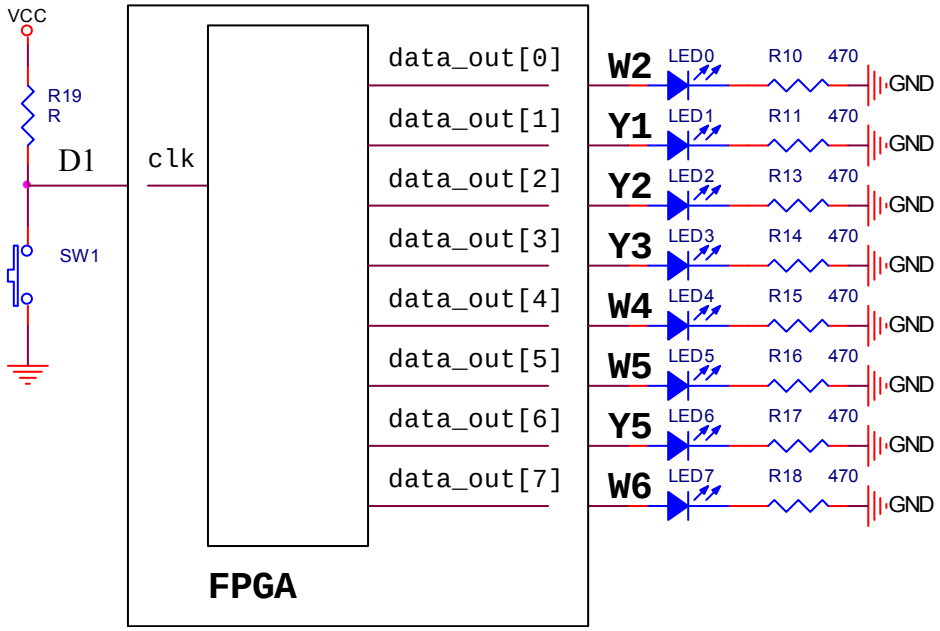
IV. DESFĂŞURAREA LUCRĂRII

1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se vor implementa structurile propuse prin exemplele anterioare a) … f). 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 corespunzătoare;
   - se realizează sinteza acestora;
   - se simulează modulul rezultat după sinteză;
   - se implementează proiectul;
   - se realizează simularea Post-Place & Post Route a modulului;
   - se configurează circuitul fizic.

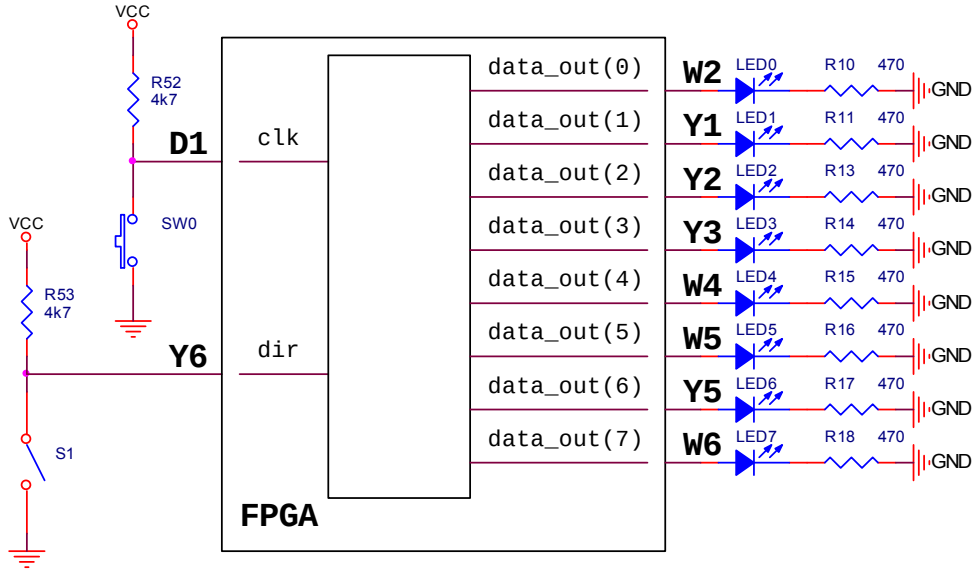
Notă: Circuitul configurabil FPGA este de tipul **3s400fg456** cu speed grade **-4**

Folosindu-vă de exemplele prezentate anterior sa se rezolve următoarele probleme:

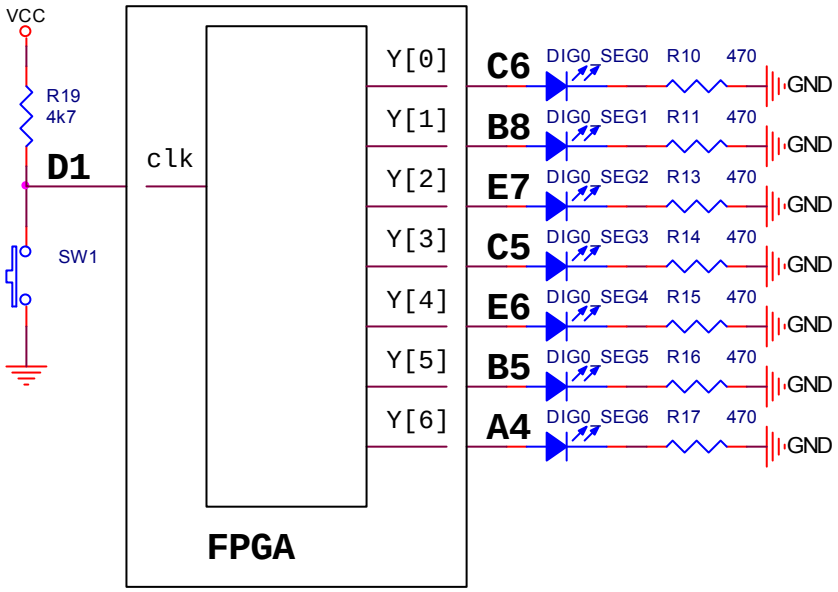
2. Să se implementeze un registru cu deplasare serială stânga (către biții cu grad mai mare de semnificație), în inel, având 8 ieşiri paralele ce acționează leduri, conform schemei de mai jos. Starea inițială a registrului este „00000001”, corespunzătoare ledului de pe ieşirea 0 aprins, celelalte fiind stinse. Schimbarea stării registrului se face la apăsarea tastei SW1.



3. Să se modifice modulul anterior prin adăugarea unui semnal de intrare care să schimbe direcția de deplasare în inel, denumit **dir**. Dacă **dir=1**, deplasarea se va realiza la stânga (către biții cu grad mai mare de semnificație), iar pentru **dir=0**, deplasarea se va realiza la dreapta (către biții cu grad mai mic de semnificație). Schema electrică este următoarea:

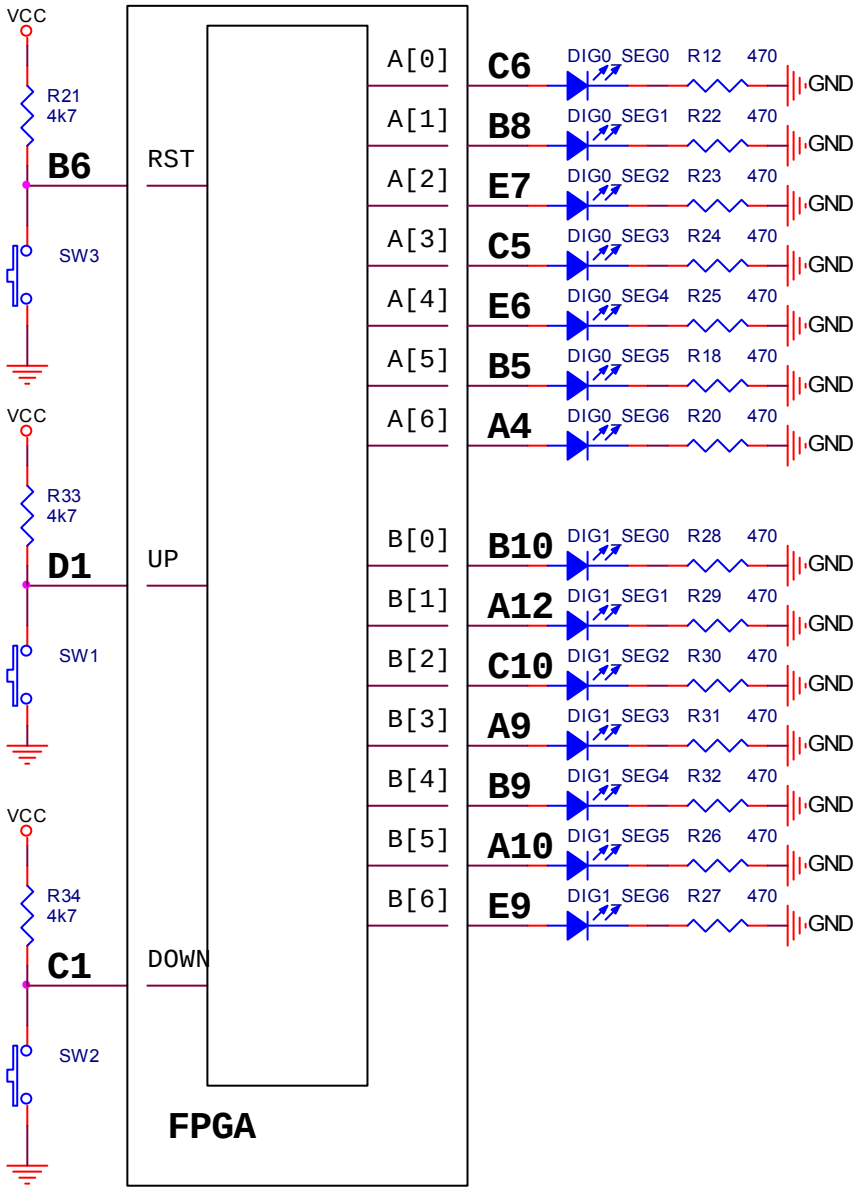


4. Să se implementeze un numărător zecimal cu afişarea stării pe un digit având 7 segmente led în sistemul de dezvoltare după schema următoare. Numărătorul va fi acționat de tasta SW1. Soluția cerută face abstracție de fenomenul de debouncing.

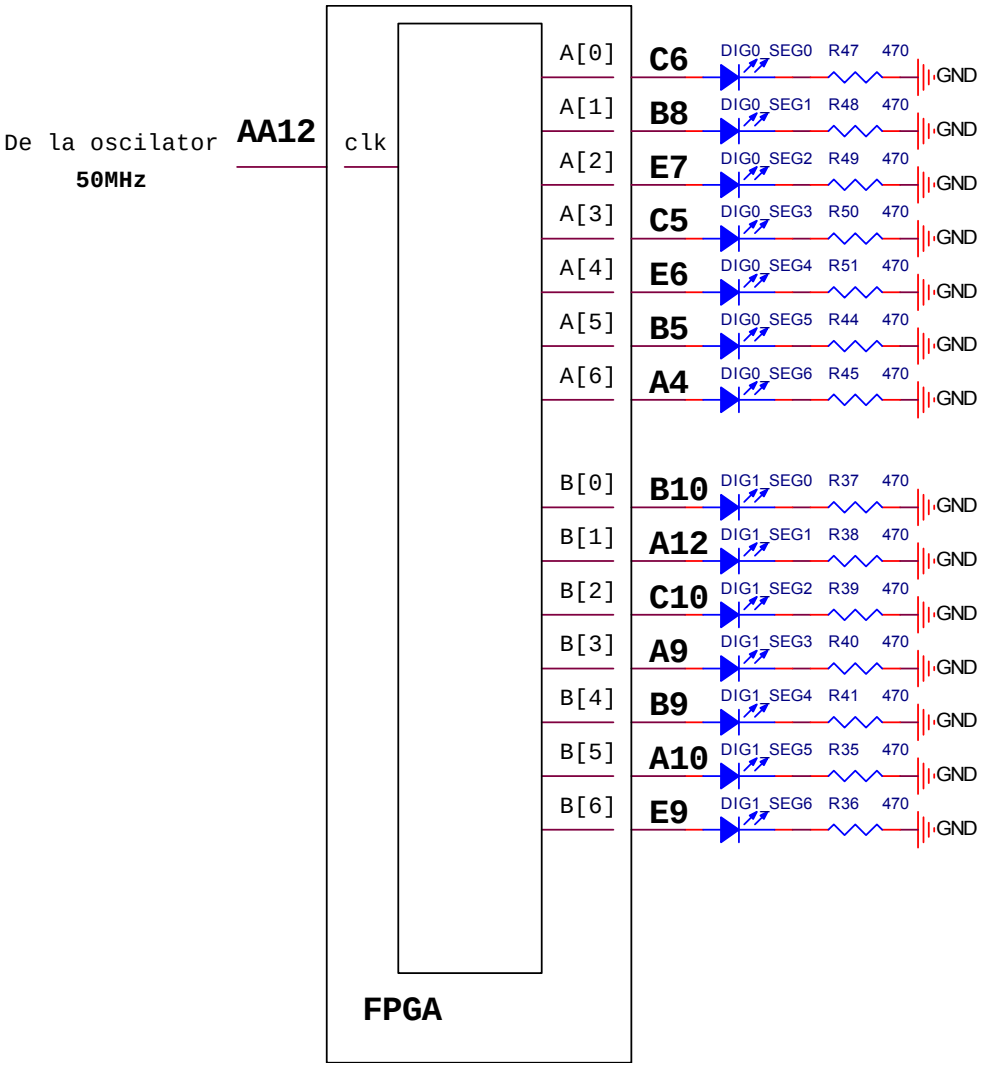


5. Să se rezolve problema anterioară ținând seama de efectul de debouncing.

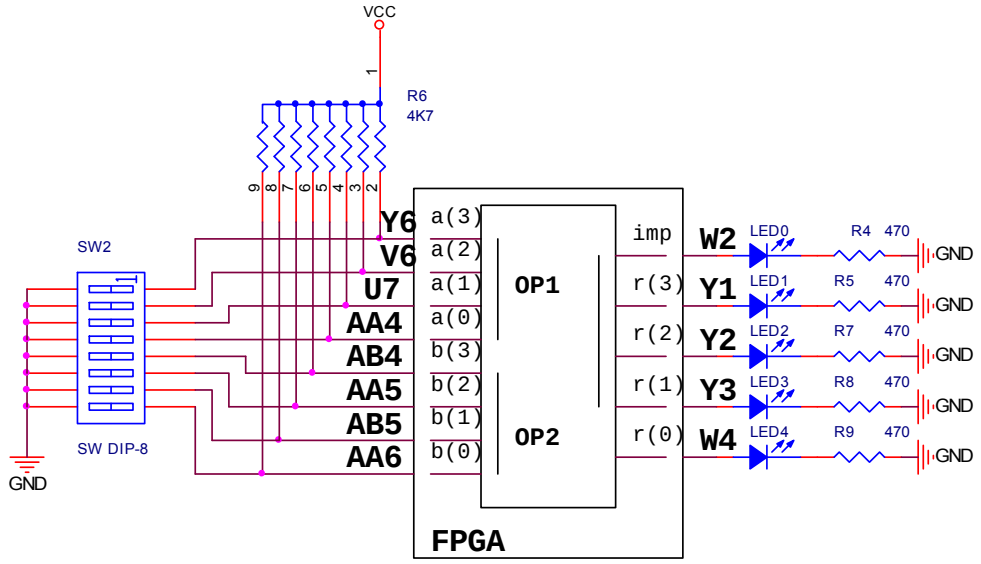
6. Să se realizeze un numărător reversibil, zecimal, cu afişare pe doi digiți, după schema de mai jos. Numărarea către înainte se va realiza prin apăsarea tastei SW2 (UP), iar numărarea către înapoi se va face prin apăsarea tastei SW1 (DOWN). La apăsarea tastei SW3 (RST) se va realiza aducerea la 0 a numărătorului. Se va ține seama de efectul de debouncing introdus taste.



7. Să se implementeze în limbaj VHDL un numărător digital la nivel de secundă. Referința de frecvență provine de la oscilatorul local al platformei având 50MHz. Afişarea timpului se face pe doi digiți (00 - 99). După atingerea stării finale 99 se va continua cu starea inițială 00. Schema electrică a circuitului este următoarea.



8. Să se implementeze un sumator simplu pentru doi operanzi exprimați pe 4 biți utilizând următoarea schemă:



Implementarea sumatorului se va realiza prin specificații concurente. Să se realizeze o comparație între acest sumator şi cel prezentat în lucrare, din punct de vedere al vitezei de operare, respectiv al numărului blocuri logice configurabile utilizate.