Laboratorul 4 - VHDL

Regişti şi numărătoare

Un simplu registru paralel

Regiştrii sunt circute digitale secvenţiale de stocare a informaţiei sub formă binară ce permit scrierea sau citirea simultană a tuturor biţilor.

Din punct de vedere structural, regiștri, sunt constituiţi din mai multe circute de tip flip-flop ce sunt grupate împreună. Descrierea unui proces secvenţial in limbaj VHDL se face prin urmărirea evenimentelor unui semnal de tact (a unui semnal de ceas pe care îl vom denumi generic **clk**).

Evenimentele pot avea loc pe „frontul crescător” (momentul de trecere a semnalului din starea de „0” logic în starea de „1” logic) sau pe „frontul descrescător” (momentul de trecere a semnalului din starea de „1” logic în starea de „0” logic) pot fi puse în evidenţă în limbajul VHDL prin intermediul uneia din funcțiile „**rising_edge**”, respectiv „**falling_edge**”. Aceste două funcții sunt implementate în librăria **std_logic_1164** – librărie ce este inclusă în mod implicit atunci când fișierul **.vhd** este generat automat.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

În blocul de cod de mai jos (ARCHITECTURE I) este reprezentat în limbaj VHDL descrierea unui registru paralel pe 4 biți implementat cu 4 circuite basculante bistabile de tip D.

```
-----------------------------------------BEGIN ARCHITECTURE I -----------------------------

architecture Register_behavioral of Register is

        signal D : std_logic_vector(3 downto 0);
        signal Q : std_logic_vector(3 downto 0);

begin
        process (clk) begin
                if rising_edge(clk) then
                        Q <= D;
                end if;
        end process;
end Register_behavioral;
------------------------------------------ END ARCHITECTURE I ----------------------------------
```

Transferul de date de la intrare la ieșire pentru acest registru are loc „pe evenimentul” unui semnal de tact (*clock*) respectiv pe front negativ sau pe front pozitiv. În unele situații mai apar și alte condiționări ale transferului respectiv este necesară îndeplinirea unei condiții impuse de un semnal de control. În exemplul de mai jos transferul de date are loc pe frontul pozitiv și doar

atunci când pe portul de intrare „load” se aplică valoarea logică „1”. Putem vorbi și în acest caz de un transfer sincron de date.

```
-------------------------------------BEGIN ARCHITECTURE II ----------------------------

architecture Registru_cu_incarcare_behavior of Registru_cu_incarcare is

        signal D : std_logic_vector(3 downto 0);
        signal Q : std_logic_vector(3 downto 0);

begin
        process (clk) begin
                if rising_edge(clk) then
                        if (load = '1') then
                                Q <= D;
                        end if;
                end if;
        end process;
end Registru_cu_incarcare_behavior;

---------------------------------- END ARCHITECTURE II ----------------------------------------
```

În momentul în care ați implementa arhitectura de mai sus veți obține un mesaj de eroare gen: “*Poor placement for routing between an IO pin and BUFG. If this sub optimal condition is acceptable for this design, you may use the CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote this message to a WARNING. ...*”. Acest mesaj este generat în principal deoarece utilizăm un pin care nu este dedicat pentru semnalul de tact drept un pin de tact (*clock*).

În principal FPGA-ul are așa zișii pini *clock capable pins* (CC pins). Doar acești pini pot fi conectați la porțile globale de *buff*-erare (BUFG) – acestea au ieșiri cu un *fanout* mare și sunt gândite tocmai pentru a fi capabile să distribuie diferite semnale în cât mai multe locuri în FPGA, un astfel de semnal este și semnalul de tact (*clock*).

Pentru a elimina eroarea de mai sus și pentru a o transforma într-o atenționare, introduceți în fișierul constrângerilor următoarea linie:

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk]
```

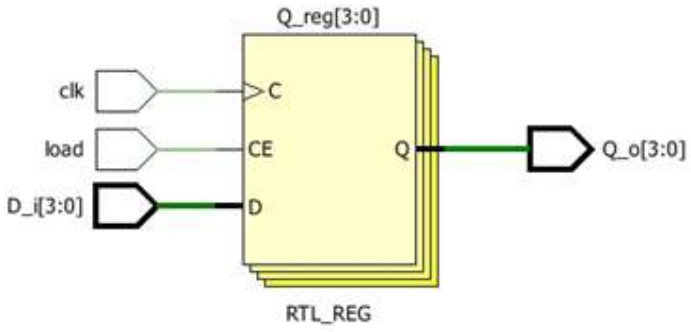

Figura 1. Registru paralel generat în urma procesului de sinteză a proiectului REG_LOAD

Exercițiu 4.1: Să se realizeze următoarele cerințe:

1. Să se creeze un proiect în programul VIVADO care se va numi **REG_LOAD**
2. Să se editeze în proiectul nou creat configurația „entity” pentru arhitectura II descrisă mai sus

3. Să se editeze fişierul de constrângeri pentru placa de dezvoltare BASYS3 astfel încât intrările de date ale registrului (D_i) să fie mapate la comunatoarele SW1…SW4, intrarea de LOAD va fi mapată la comutatorul SW0 iar ieşirile Q_o vor fi mapate la LED-urile LD1 … LD4.
4. Se vor executa procesele de sinteză, de implementare şi de generare a fişierului bit-steam iar fişierul bit-stream rezultat va fi încărcat în circuitl FPGA al plăcii de dezvoltare BASYS3.
5. Se va verifica funcţionalitate circuitului prin editarea tabelului de adevăr

Regişti de deplasare

Exercițiu 4.2: Să se realizeze un registru de deplasare pe 4 biți în care: **deplasarea informației** să se poată face într-un singur sens (atunci când starea liniei de intrare ***shift_n*** este pe 0 logic), **încărcarea informației** să se poată face serial (bit după bit, toti bitii cuvântului de 4 biți – atunci când starea liniei de intrare ***load_n*** este pe 1 logic) sau **paralel** (cei 4 biți se scriu simultan în registru – atunci când starea liniei de intrare ***load_n*** este pe 0 logic) iar **citirea registrului** să se poate face paralel (toți biții simultan).

Pentru atingerea acestor obiective se vor urma pașii:

1. Să se creeze un proiect în programul VIVADO care se va numi **REG_SHIFT_4**
2. Să se editeze în proiectul nou creat configuraţia „entity” pentru arhitectura III descrisă mai jos:

```
--------------------------------------------------------------------------------
----------------------------------BEGIN ARCHITECTURE III ---------------------

ARCHITECTURE reg_shift_4bits OF shf_reg_4 IS

 signal reg : std_logic_vector(3 downto 0);

BEGIN

 PROCESS(clk)
 BEGIN
  IF (clk'EVENT AND (clk = '1')) THEN
   IF (load_n = '0') THEN
    reg <= p_in;
   ELSIF (shift_n = '0') THEN
    reg <= reg(2 DOWNTO 0) & s_in;
   END IF;
  END IF;
 END PROCESS;

 p_out <= reg;

END reg_shift_4bits;
---------------------------------END ARCHITECTURE III ----------------------
--------------------------------------------------------------------------------
```

3. Să se editeze fişierul de constrângeri pentru placa de dezvoltare BASYS3 astfel încât intrările de date ale registrului (***p_in***) să fie mapate la comunatoarele **SW3**…**SW6**, intrarea ***load_n*** va fi mapată la comutatorul **SW1**, intrarea ***s_in*** va fi mapată la comutatorul **SW2**, intrarea ***shift_n*** va fi

mapată la comutatorul **SW0** iar ieşirile ***Q_o*** vor fi mapate la LED-urile **LD3** … **LD6**. Intrarea ***clk*** va fi conectată la **SW15**.

4. Se vor executa procesele de sinteză, de implementare şi de generare a fişierului bit-steam iar fişierul bit-stream rezultat va fi încărcat în circuitl FPGA al plăcii de dezvoltare BASYS3.
5. Se va verifica funcţionalitate circuitului prin editarea tabelului de adevăr

În codul anterior se observă că în primul **IF** avem următoarea secvență de condiții: clk'**EVENT AND** (clk = '1'). Primul operator **'event** are semnificația apariției unui eveniment notabil în semnalul digital,. Acest eveniment poate fi doar frontul pozitiv sau cel negativ – deoarece semnalul este unul digital. Prin combinarea (**AND**) cu condiția ca valoarea semnalului să fie '1' logic expresia din primul **IF** identifică frontul pozitiv al semnalului de clock – deci orice schimbare a cărei rezultat este un nivel 1 logic poate fi doar frontul pozitiv.

Există mai multe modalități prin intermediul cărora un front pozitiv poate fi identificat într-o condiție de tip **IF**, astfel:

```
RISING_EDGE( nume_clk_semnal )
nume_clk_semnal'EVENT and nume_clk_semnal = '1'
nume_clk_semnal = '1' and nume_clk_semnal'EVENT
not nume_clk_semnal'STABLE and nume_clk_semnal = '1'
nume_clk_semnal = '1' and not nume_clk_semnal'STABLE
```

O secvență de tipul Semnal'**STABLE** este adevărată dacă nici un eveniment nu are loc în cadrul semnalului Semnal.

Dacă implementarea este corectă circuitul rezultant va fi similar cu următorul prezentat în **Figura 2**.

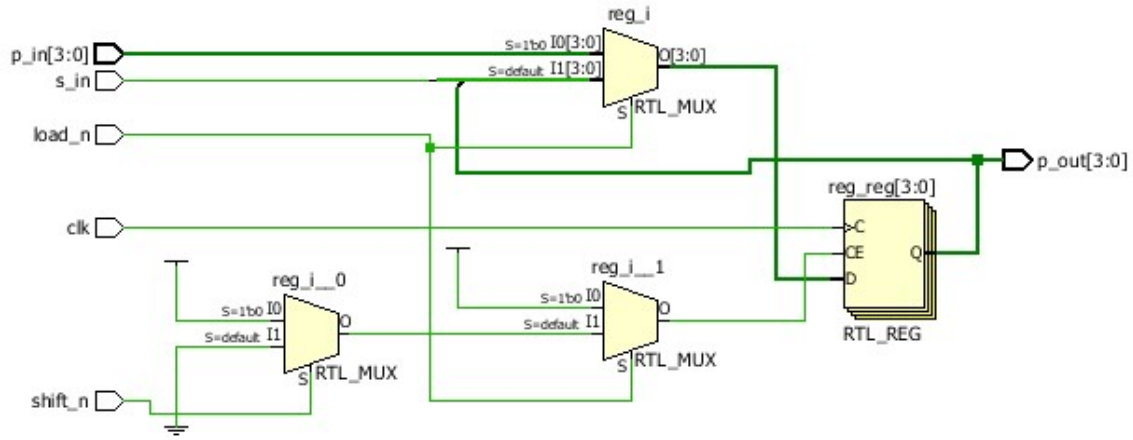

Figura 2. Registrul de deplasare generat în urma procesului de sinteză a proiectului REG_LOAD

Exercițiu 4.3: Veți observa în cel mai scurt timp că utilizarea butonului **SW15** drept generator a semnalului de tact nu este o alegere prea fericită. De foarte multe ori se vor obține rezultate similare ca în situația în care s-ar genera mai multe pulsuri consecutive ale semnalului de tact. Acest fenomen este dat de oscilațiile închis-descis-închis ... ale contactului comutatorului atunci cand el este închis sau

deschis. Pentru a obține un contact mai ferm utilizați unul din cele 5 *push-buttons* existente pe placa.

Exercițiu 4.4: Să se implementeze în limbajul VHDL funcţiile circuitului 54/74-95 (registru de deplasare combinat serie-paralel) ce a fost studiat în unul din laboratoarele precedente.

Divizor de frecvenţă

Placa de dezvoltare BASYS3 are un genetator de clock cu o frecvenţă de 100 MHz a cărui ieşire este conectată la pinul **W5** al circuitului FPGA.

Exercițiu 4.5: Se cere să se descrie în limbaj VHDL un divizor de frecvență, astfel încât frecvența de ieșire să fie redusă de 10^8 ori. Ieșirea out_div se va conecta la unul din LED-urile existente pe placa de dezvoltare BASYS3.

Soluția exercițiului precedent se prezintă mai jos:

```
----------------------------------------------------------------------------------------------------BEGIN ARCHITECTURE
IV ---------------------------

	library IEEE;
	use IEEE.STD_LOGIC_1164.ALL;

	entity clk_divizat is
		port
			(
			clk_in  : in  std_logic;
			out_div : out std_logic
			);
	end clk_divizat;

	architecture Behavioral of clk_divizat is

	begin

		process(clk_in)
			variable  n : integer range 0 to 1000000000;
		begin
			if clk_in'event and clk_in='1' then
				if n < 100000000 then
					n := n+1;
				else
					n := 0;
				end if;

				if n <= 50000000 then
					out_div <= '1';
				else
					out_div <= '0';
				end if;
			end if;
		end process;
```

```
end Behavioral;
------------------------------ END ARCHITECTURE IV ----------------------------------------------------------------------------
----------------------------
```

NUMĂRĂTOARE

Numărătoarele sunt circuite secvențiale de „integrare” a semnalului de clock ce se aplică la intrare. În funcție de modul de efectuare a resetului se evidențiaza numărătoare cu reset asincron (atunci când semnalul de reset este prioritat față de semnalul de clock) și numărătoare cu reset sincron (resetarea are loc în situația îndeplinirii a două condiții: activarea semnalului de reset și apariția unei eveniment de clock.

Exercițiu 4.6: Se cere să se descrie în limbaj VHDL un circuit numărător binar pe 4 biți care să ofere facilitatea schimbării sensului de numărare, prin intermediul unui pin extern (ud_in) și, în plus, să permită utilizatorului resetarea circuitului (reset_in).

Soluția exercițiului precedent se prezintă mai jos:

```
----------------------------------------------------------------------------------------------------BEGIN ARCHITECTURE V
----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity my_counter is
   Port ( counter_clk : in  STD_LOGIC;
          ud_in       : in  STD_LOGIC;
          reset_in    : in  STD_LOGIC;
          Data_out    : out STD_LOGIC_VECTOR (3 downto 0)
         );
end my_counter;

architecture Behavioral of my_counter is
   signal a : std_logic_vector(3 downto 0);
begin
          process(counter_clk, ud_in, reset_in)
          begin
                    if reset_in = '1' then        --1
                              a <= (others => '0');
                    elsif (counter_clk'event and counter_clk = '1') then
            if ud_in = '1' then             --2
               a <=  a + 1;
            elsif ud_in = '0' then
               a <=  a - 1;
            end if;                         --2
       end if;                          --1
          end process;

          Data_out <= a;

end Behavioral;
--------------------------------END ARCHITECTURE V ----------------------------------------------------------------------------
----------------------------
```

Pentru a nu primi erori din partea compilatorului VHDL pe funcțiile de adunare (`a <= a + 1`) și scădere (`a <= a - 1`) trebuie să introduceți librăria **std_logic_unsigned**.

Exercițiu 4.7: Având la dispoziție implementările divizorului de frecvență și a numărătorului, prezentate anterior, grupațile într-o structură ierarhică, similară cu cea de mai jos din **Figura 3**, asfel încât semnalul de tact de 1 Hz să intre în tactul numărătorului. În acest mod evitați folosirea unui buton, care prin oscilațiile existente la închiderea sau deschiderea sa determină apariția unor pulsuri parazite pe semnalul de tact.

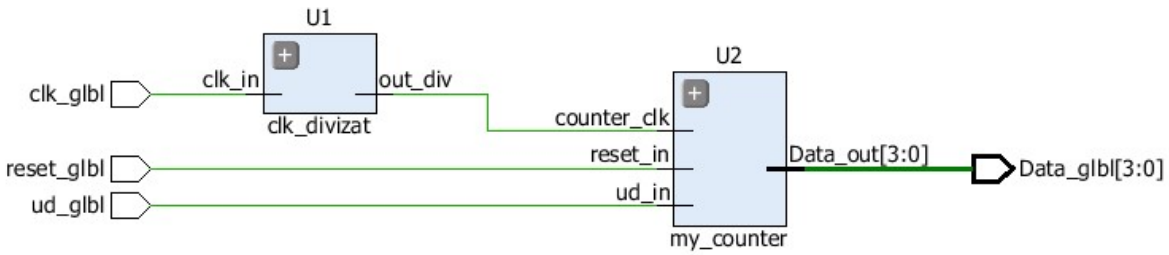

Figura 3. Schema globală a circuitului ce trebuie implementat în cadrul **exercițiului 4.7**

Exercițiu 4.8: Pentru eliminarea problemelor generate de oscilația contactactelor butoanelor se poate folosi un circuit precum cel din **Figura 4**. Restricțiile în funcționarea acestui circuit sunt date de necesitatea utilizării unei perioade a semnalului de tact (CLK) mai mare decât intervalul de oscilație a contactului de intrare.

Având gata implementate circuitul divizor și numărătorul realizați o descriere în VHDL a unui circuit numărător, identic ca funcționare cu cel din **Exercițiu 4.6**, a cărui semnal de tact să fie dat de comutatorul **SW0** (de pe placa BASYS3), iar acestă intrare să fie "deparazitată" de oscilațiile parazite cu ajutorul unui circuit ca cel prezentat în **Figura 4**.

Știind că oscilațiile parazite ce apar la închiderea sau deschiderea unui contact țin în jur de 20-30 ms reproiectați circuitul divizor pentru a furniza semnalul de tact optim circuitului din **Figura 4**.

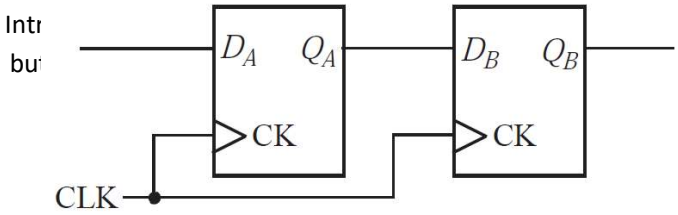

Figura 4. Circuit de sincronizare și eliminare a oscilațiilor parazite