

Asignatura: Electrónica Digital

Prueba de Evaluación Continua (PEC)

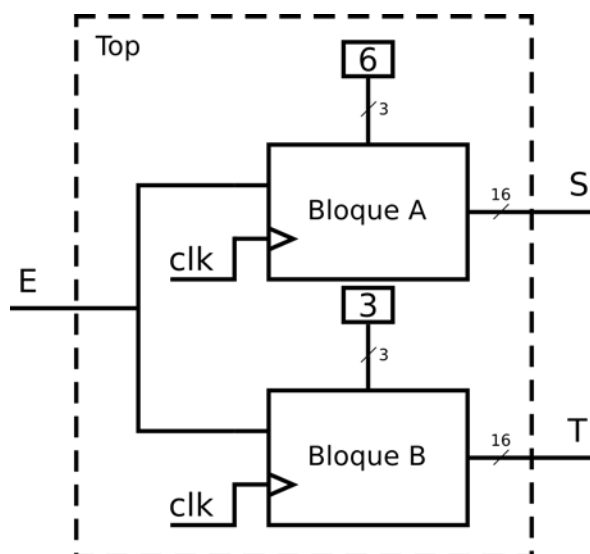
Convocatoria: Julio 2019 Fecha: 27/06/2019

Duración: 1 hora y 30 minutos

Ejercicio 1 (6 Puntos)

Por una línea serie se reciben, de manera síncrona con el reloj, paquetes de 8 bits, en los que el primer bit recibido vale siempre '1', los tres siguientes indican una dirección y los cuatro restantes, un valor numérico. Mientras no hay mensaje, la línea permanece a cero. Los bits de la dirección y del valor numérico se reciben ordenados del más significativo al menos significativo. Se pide:

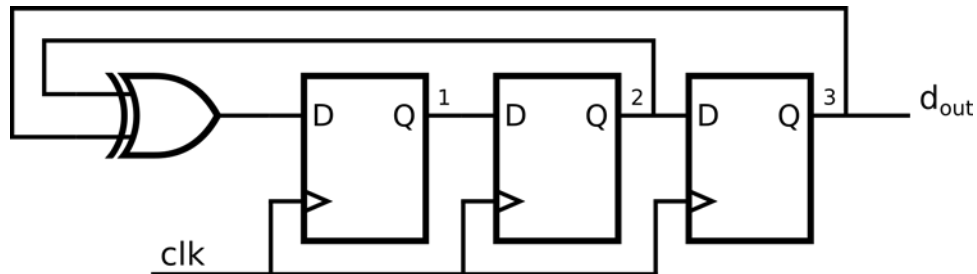
- Descripción VHDL de un bloque (bloque A en la figura) que reciba la señal serie, interprete el paquete, y en caso de que los bits de dirección coincidan con un valor fijo, que se le indica al bloque por otra entrada (en paralelo, de tres bits), acumule el valor numérico recibido sobre un registro interno de 16 bits. Este valor interno se saca al exterior del módulo. De forma particular, si la dirección recibida es 111, no se atiende al valor numérico del paquete, sino que se resetea el contador de forma síncrona.
- Con dos de los bloques arriba descritos y empleando para ello una descripción estructural, se desea implementar un sistema de acumuladores que muestre por salidas separadas el valor de un acumulador para las direcciones 3 y 6, tal como muestra la figura. La entrada serie es común a ambos bloques.
- Implemente también en VHDL un *test bench* que permita validar la acumulación de más de un número en uno solo de los módulos y su función de reset con la dirección 111.



.../... (sigue en la cara posterior)

Ejercicio 2 (4 puntos)

Dado el circuito de la figura:



Se pide:

- Implementación en VHDL del circuito de la figura, sabiendo que no puede inicializarse a 000 (el sistema no funcionaría).
- Implementación en VHDL de un circuito similar, pero con longitud de 65 bits, aplicando la XOR entre los bits con posiciones 65 y 47. Nótese que la salida de este circuito puede considerarse un generador de números aleatorios de un bit.
- Empleando un bloque como el de la figura, asumiendo que se le añade una señal de habilitación (un puerto *enable*), y sabiendo que tenemos un reloj de 8 MHz, diseñe un circuito que genere un número aleatorio (de un bit) nuevo cada segundo.

Nota: No está permitido el uso de calculadora en todo el examen.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity accumulator is
6      port(
7          clk      : in  std_logic;
8          reset    : in  std_logic;
9          addr     : in  std_logic_vector(2 downto 0);
10         din      : in  std_logic;
11         dout     : out std_logic_vector(15 downto 0)
12     );
13 end accumulator;
14
15 architecture behavioral of accumulator is
16
17     signal shift_reg : std_logic_vector(6 downto 0);
18     signal data_cnt  : unsigned(2 downto 0);
19
20     -- Simple FSM: '1' - operating, '0' - idle
21     signal state      : std_logic;
22     signal load       : std_logic;
23     signal reset_s    : std_logic;
24
25     signal acc        : unsigned(15 downto 0);
26
27 begin
28
29     -- Input shift register
30     process(clk,reset)
31     begin
32         if reset = '1' then
33             shift_reg <= (others => '0');
34         elsif clk'event and clk = '1' then
35             shift_reg <= shift_reg(5 downto 0) & din;
36         end if;
37     end process;
38
39     -- Input data counter
40     process(clk,reset)
41     begin
42         if reset = '1' then
43             data_cnt <= (others => '0');
44         elsif clk'event and clk = '1' then
45             if state = '0' then
46                 data_cnt <= (others => '0');
47             else
48                 data_cnt <= data_cnt + 1;
49             end if;
50         end if;
51     end process;
52
53     -- Control logic (simple FSM)
54     process(clk,reset)
55     begin
56         if reset = '1' then
57             state <= '0';
58         elsif clk'event and clk = '1' then
59             if state = '0' and din = '1' then
60                 state <= '1';
61             elsif state = '1' and data_cnt = 7 then

```

```
62         state <= '0';
63     end if;
64 end if;
65 end process;
66
67 load    <= '1' when shift_reg(6 downto 4) = addr and data_cnt = 7 else '0';
68 reset_s <= '1' when shift_reg(6 downto 4) = "111" and data_cnt = 7 else '0';
69
70 -- Accumulator logic
71 process(clk,reset)
72 begin
73     if reset = '1' then
74         acc <= (others => '0');
75     elsif clk'event and clk = '1' then
76         if reset_s = '1' then
77             acc <= (others => '0');
78         else
79             if load = '1' then
80                 acc <= acc + unsigned(shift_reg(3 downto 0));
81             end if;
82         end if;
83     end if;
84 end process;
85
86 -- Output connection
87 dout <= std_logic_vector(acc);
88
89 end behavioral;
90
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity top is
5      port(
6          clk      : in  std_logic;
7          reset    : in  std_logic;
8          e        : in  std_logic;
9          s        : out std_logic_vector(15 downto 0);
10         t        : out std_logic_vector(15 downto 0);
11     );
12 end top;
13
14 architecture structural of top is
15
16     signal addr_a : std_logic_vector(2 downto 0);
17     signal addr_b : std_logic_vector(2 downto 0);
18
19 begin
20
21     addr_a <= "110";
22     addr_b <= "011";
23
24     bloque_a: entity work.accumulator
25     port map(
26         clk  => clk,
27         reset => reset,
28         addr => addr_a,
29         din  => e,
30         dout => s
31     );
32
33     bloque_b: entity work.accumulator
34     port map(
35         clk  => clk,
36         reset => reset,
37         addr => addr_b,
38         din  => e,
39         dout => t
40     );
41
42 end structural;
43
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity accumulator_tb is
5  end accumulator_tb;
6
7  architecture testbench of accumulator_tb is
8
9      signal clk      : std_logic;
10     signal reset     : std_logic;
11     signal addr      : std_logic_vector(2 downto 0);
12     signal din        : std_logic;
13     signal dout       : std_logic_vector(15 downto 0);
14
15     signal aux        : std_logic_vector(6 downto 0);
16
17     constant clk_period : time := 10 ns;
18
19 begin
20
21     uut: entity work.accumulator
22     port map(
23         clk      => clk,
24         reset    => reset,
25         addr     => addr,
26         din      => din,
27         dout     => dout
28     );
29
30     proc_clk: process
31     begin
32         clk <= '1';
33         wait for clk_period/2;
34         clk <= '0';
35         wait for clk_period/2;
36     end process;
37
38     proc_stim: process
39     begin
40         reset <= '1';
41         addr <= "000";
42         aux <= "0000000";
43         din <= '0';
44         wait for clk_period*100.5;
45
46         reset <= '0';
47         wait for clk_period*100;
48
49         addr <= "011";
50         aux <= "0111010";
51
52         din <= '1';
53         wait for clk_period;
54         for i in 6 downto 0 loop
55             din <= aux(i);
56             wait for clk_period;
57         end loop;
58         din <= '0';
59         wait for clk_period*100;
60
61         addr <= "110";
```

```
62         aux <= "1100011";
63
64         din <= '1';
65         wait for clk_period;
66         for i in 6 downto 0 loop
67             din <= aux(i);
68             wait for clk_period;
69         end loop;
70         din <= '0';
71         wait for clk_period*100;
72
73         addr <= "110";
74         aux <= "1111111";
75
76         din <= '1';
77         wait for clk_period;
78         for i in 6 downto 0 loop
79             din <= aux(i);
80             wait for clk_period;
81         end loop;
82         din <= '0';
83         wait for clk_period*100;
84
85         wait;
86     end process;
87
88 end testbench;
89
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity lfsr is
5      port(
6          clk      : in  std_logic;
7          reset    : in  std_logic;
8          dout     : out std_logic
9      );
10 end lfsr;
11
12 architecture behavioral of lfsr is
13
14     constant C_DEPTH : integer := 3; -- In b is 65
15     constant C_TAP_1  : integer := 3; -- in b is 65
16     constant C_TAP_2  : integer := 2; -- In b is 47
17
18     signal shift_register : std_logic_vector(C_DEPTH-1 downto 0);
19     signal din             : std_logic;
20
21 begin
22
23     process (clk, reset)
24     begin
25         if reset = '1' then
26             shift_register <= (0 => '1', others => '0');
27         elsif clk'event and clk = '1' then
28             shift_register <= shift_register(C_DEPTH-2 downto 0) & din;
29         end if;
30     end process;
31
32     din  <= shift_register(C_TAP_1-1) xor shift_register(C_TAP_2-1);
33     dout <= shift_register(C_DEPTH-1);
34
35 end behavioral;
36
```



```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity random is
5      port(
6          clk      : in  std_logic;
7          reset    : in  std_logic;
8          dout     : out std_logic
9      );
10 end random;
11
12 architecture behavioral of random is
13
14     constant C_DEPTH : integer := 3; -- In b is 65
15     constant C_TAP_1  : integer := 3; -- in b is 65
16     constant C_TAP_2  : integer := 2; -- In b is 47
17
18     signal shift_register : std_logic_vector(C_DEPTH-1 downto 0);
19     signal din             : std_logic;
20     signal enable          : std_logic;
21
22     constant C_MAX_CNT : integer := 8*10**6;
23     signal cnt : integer range 0 to C_MAX_CNT-1;
24
25 begin
26
27     process (clk, reset)
28     begin
29         if reset = '1' then
30             cnt <= 0;
31         elsif clk'event and clk = '1' then
32             if cnt = C_MAX_CNT-1 then
33                 cnt <= 0;
34             else
35                 cnt <= cnt + 1;
36             end if;
37         end if;
38     end process;
39
40     enable <= '1' when cnt = C_MAX_CNT-1 else '0';
41
42     process (clk, reset)
43     begin
44         if reset = '1' then
45             shift_register <= (0 => '1', others => '0');
46         elsif clk'event and clk = '1' then
47             if enable = '1' then
48                 shift_register <= shift_register(C_DEPTH-2 downto 0) & din;
49             end if;
50         end if;
51     end process;
52
53     din  <= shift_register(C_TAP_1-1) xor shift_register(C_TAP_2-1);
54     dout <= shift_register(C_DEPTH-1);
55
56 end behavioral;
57
```