Universidad de Alcalá

Departamento de Electrónica

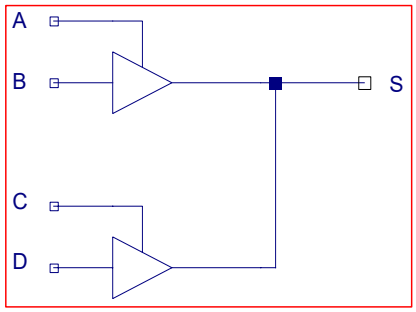
Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

Ejercicios_2.

Ejercicio 1.

Crear el código VHDL sintetizable que modele el circuito de la figura.




```
library ieee;
use ieee.std_logic_1164.all;
entity ejercicio_1 is
  port (
    A, B : in  std_logic;
    C, D : in  std_logic;
    S    : out std_logic);
end ejercicio_1;

architecture rtl of ejercicio_1 is

begin
  s <= b when a = '1' else 'Z';
  s <= d when c = '1' else 'Z';
end ;
```

Ejercicio 1.

```
library ieee;
use ieee.std_logic_1164.all;
entity ejercicio_1_tb is

end ejercicio_1_tb;
architecture sim of ejercicio_1_tb is


  signal A_i, B_i : std_logic := '0';
  signal C_i, D_i : std_logic := '0';
  signal S_i      : std_logic;

begin  -- sim

  DUT : entity work.ejercicio_1
    port map (
      A => A_i,
      B => B_i,
      C => C_i,
      D => D_i,
      S => S_i);
```

```
B_i <= not B_i after 10 ns;
D_i <= not D_i after 40 ns;
```

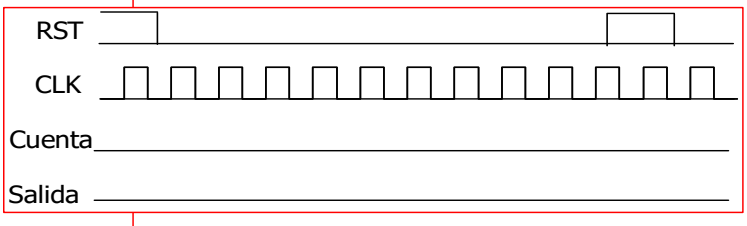
```
process
  begin  -- process
    A_i <= '1';
    C_i <= '0';
    wait for 200 ns;
    A_i <= '0';
    C_i <= '1';
    wait for 200 ns;
    A_i <= '0';
    C_i <= '0';
    wait for 200 ns;
    A_i <= '0';
    C_i <= '1';
    wait for 200 ns;
    A_i <= '1';
    C_i <= '1';
    wait for 200 ns;
    A_i <= '0';
    C_i <= '1';
    wait for 200 ns;
    report "fin de la simulacion" severity
failure;
  end process;
```

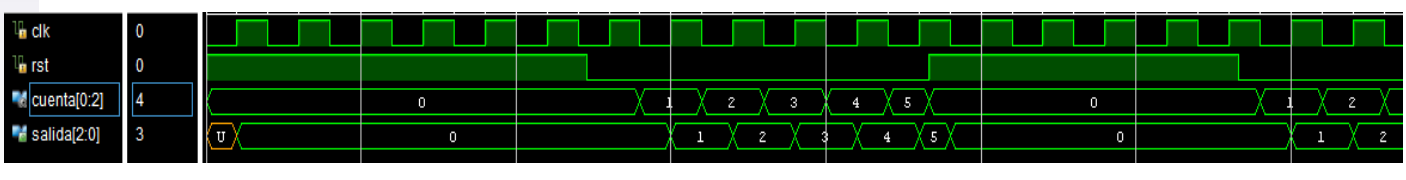

Ejercicio 2.

Para el modela de un contador en VHDL se ha utilizado el siguiente código.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_2 is
  port ( clk    : in  std_logic;
         rst    : in  std_logic;
         salida : out std_logic_vector(2 downto 0));
end ejercicio_2;

architecture arq of ejercicio_2 is
  signal cuenta :     unsigned(0 to 2);
begin
  process (clk, rst)
  begin
    if (rst = '1') then
      cuenta <= (others => '0');
    elsif (clk'event and clk = '0') then
      cuenta <= cuenta+1;
    end if;
    salida   <= std_logic_vector(cuenta);
  end process;
end arq;
```

1. Completar la siguiente gráfica





Ejercicio 3.

Para modelar un contador BCD cuya tabla de verdad es la mostrada en la fig1.1 se ha utilizado el código de la fig1.2. El contador dispone de una salida **FC** se pone a nivel alto cuando la cuenta alcanza el valor 9.

RST	CE	CLK	Q_{t+1}
1	X	X	0_x
0	1	↑	Q_t+1
0	0	X	Q_t

Indique justificadamente cuál o cuáles son los errores cometidos

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  entity cnt_BCD is
5    port (
6      CLK : in  std_logic;
7      RST : in  std_logic;
8      CE  : in  std_logic;
9      Q   : out std_logic_vector(3 downto 0);
10     FC  : out std_logic);
11 end entity;
12 architecture rtl of cnt_BCD is
13 begin
14   process(clk)
15   begin
16     if(CLK'event and CLK = '1') then
17       if RST = '1' then
18          Q    <= "0000";
19          FC   <= '0';
20       elsif CE = '1' then
21          Q    <= std_logic_vector(unsigned(Q)+1));
22         if Q = 9 then
23           Q   <= "0000";
24           FC <= '1';
25         else
26            FC  <= '0';
27         end if;
28       end if;
29     end if;
30   end process;
31 end rtl;
```


Ejercicio 3.

Proponer el código correcto para modelar el contador BCD del apartado anterior. El código debe ser sintetizable.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_3 is
  port (
    CLK       : in  std_logic;
    RST       : in  std_logic;
    CE        : in  std_logic;
    Q         : out std_logic_vector(3 downto 0);
    FC        : out std_logic);
end entity;
```

```
architecture rtl of ejercicio_3 is
  signal aux :     unsigned(3 downto 0);
begin
  process(clk)
  begin
    if(CLK'event and CLK = '1') then
      if RST = '1' then
        aux <= x"0";
      elsif CE = '1' then
        aux <= aux+1;
        if aux = 9 then
          aux  <= x"0";
        end if;
      end if;
    end if;
  end process;

  q  <= std_logic_vector(aux);
  FC <= '1'when aux = 9 else '0';

end rtl;
```

Ejercicio 4.

Dado el siguiente código VHDL

```
architecture rtl of c1 is
--
signal cnt,CEROS:  std_logic_vector (4 downto 0);
signal din : std_logic_vector(15 downto 0);
----
begin
  process (DIN,cnt)
    variable max : std_logic_vector (4 downto 0);
  begin
    cnt       <= (others => '0');
    max       := (others => '0');
    for j in DIN'range loop
      if DIN(j) = '0' then
        cnt   <= cnt+1;
        if max < cnt then
          max := cnt;
        end if;
      else
        cnt   := (others => '0');
      end if;
    end loop;  -- j
    CEROS <= (max);
  end process;
```

Indicar, razonadamente qué errores se cometen en el proceso anterior, así como los problemas que pueden presentar.

Ejercicio 5.

A un sistema digital le llega un dato ***A*** de 6 bits y proporciona un dato ***S*** que es igual a ***A*** siempre que este se corresponda con un código correcto, en caso contrario todos los bits de ***S*** permanecerán a 0. Se considera un dato correcto de ***A*** a aquel que tenga dos bits consecutivos a 1 y el resto a 0. Por ejemplo, los códigos 001100, 110000 se consideran códigos correctos, mientras que 010000 o 101100 son incorrectos.

Crear el código VHDL sintetizable que modele

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_4 is
  port(
    a : in  std_logic_vector(5 downto 0);
    s : out std_logic_vector(5 downto 0) );
end ejercicio_4;
architecture rtl of ejercicio_4 is
```

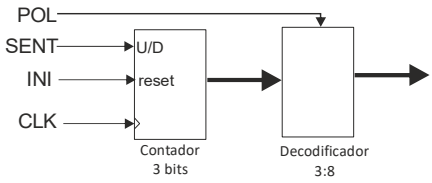
```
begin
  process (a)
    variable unos, ceros : unsigned(2 downto 0);
    variable consec      : std_logic;
  begin
    ceros         := (others => '0');
    unos          := (others => '0');
    consec        := '0';
    for i in a'range loop
      if a(i) = '1' then
        unos      := unos +1;
        if unos = 2 then
          consec := '1';
        end if;
      else
        ceros     := ceros+1;
        unos      := (others => '0');
      end if;
    end loop;  -- i
    if consec = '1' and ceros = 4 then
      s <= a;
    else
      s <= (others          => '0');
    end if;

  end process;
```

Ejercicio 6.

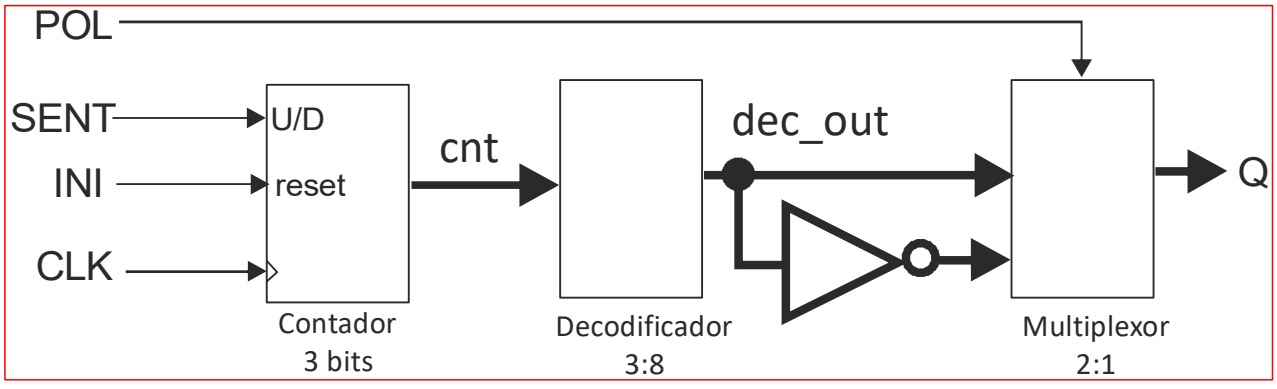
Un contador Johnson es aquel que genera una secuencia periódica en la que para cada valor, sólo hay activa una de las salidas. Por ejemplo, para 4 bits, y salidas activas a nivel **alto**, la secuencia seguida será: 0001, 0010, 0100, 1000,0001...

Crear el código VHDL sintetizable de un contador Johnson de 8 bits. El sistema dispone de una entrada POL que controla el valor con el que se activan las salidas: si POL=1 el nivel activo es el 1 y el 0 en caso contrario. Con otra entrada SENT se controla el sentido en el que se va activando las salidas: con SENT=1 se activan de izquierda a derecha y de derecha a izquierda si vale 0. Además, se dispone de una entrada de inicialización INI, que mientras se encuentre activada (nivel alto) la salida permanece en su valor inicial (00000001 ó 11111110, dependiendo del valor de POL).

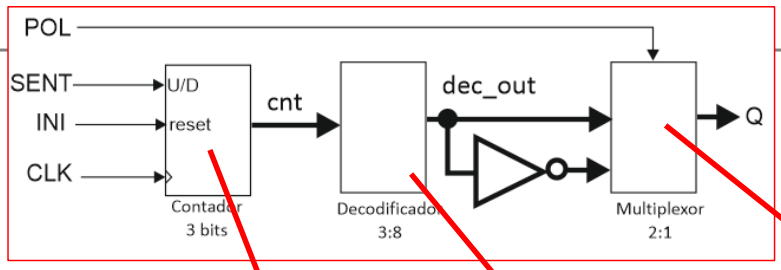
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_6 is
  port (
    CLK  : in  std_logic;
    INI  : in  std_logic;
    SENT : in  std_logic;
    POL  : in  std_logic;
    Q    : out std_logic_vector(7 downto 0));
end ejercicio_6;
```



Ejercicio 6.



Ejercicio 6.




```
process (dec_out, POL)
  begin
    if POL = '1' then
      Q <= dec_out;
    else
      Q <= not(dec_out);
    end if;
  end process;
```

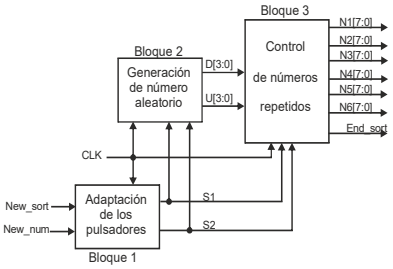
```
process (CLK, INI)
  begin
    if INI = '1' then
      cnt   <= ( others => '0');
    elsif CLK'event and CLK = '1' then
      if SENT = '1' then
        cnt <= cnt-1;
      else
        cnt <= cnt+1;
      end if;
    end if;
  end process;
```

```
process(cnt)
  begin
    dec_out        <= (others => '0');
    for i in 0 to 7 loop
      if(i = unsigned(cnt)) then
        dec_out(i) <= '1';
      end if;
    end loop;
  end process;
```


Ejercicio 7.

Se debe crear el código VHDL sintetizable que modele un sistema que genere de manera aleatoria los números de la lotería Primitiva. El resultado básico de este sorteo lo componen 6 números (***N1***, ***N2***, ***N3***, ***N4***, ***N5*** y ***N6***) comprendidos entre el 1 y el 49, ambos incluidos, en formato BCD de dos dígitos (decenas y unidades). La generación se consigue con un contador que trabaja con una frecuencia de reloj muy alta y capturando su valor mediante un pulsador. Al ser la frecuencia de reloj del contador muy alta es imposible predecir con la actuación sobre el pulsador el valor que tiene el contador.

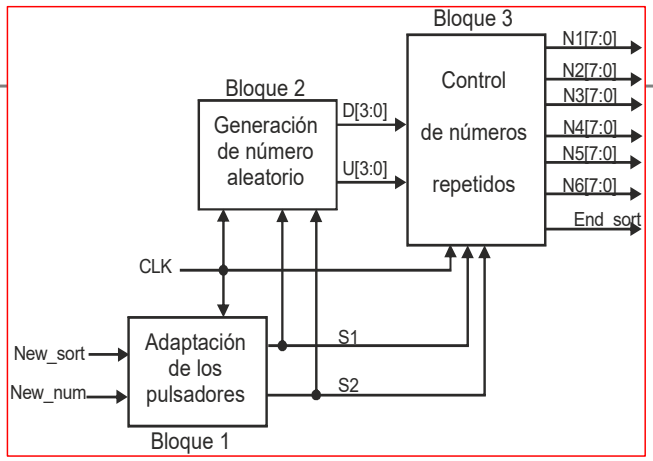
El sistema está controlado por dos pulsadores que generan sendas señales ***New_sort*** y A ***New_num*** las cuales, para simplificar el diseño, se supondrán libres de rebotes. En la figura siguiente se muestra un diagrama de bloques simplificado del sistema:



Donde:

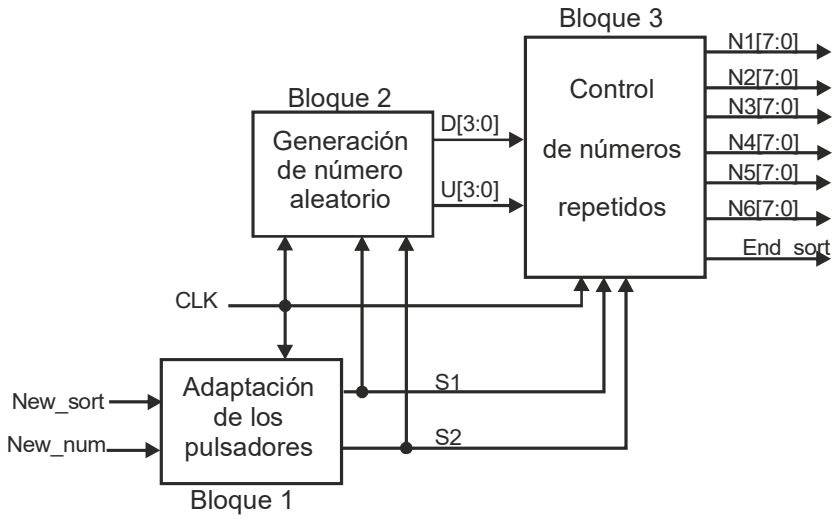
- Entrada ***New_sort***: Con un nivel alto se realiza el borrado de los números correspondientes a un sorteo. En consecuencia, mientras se mantenga a nivel bajo se realiza el proceso de generación los 6 números del sorteo. Para el nivel alto es indiferente el valor de los datos de salida (***N1***, ***N2***… ***N6***), mientras que la salida ***End_sort*** estará a nivel bajo.
- Entrada ***New_num***: Se utiliza para capturar un nuevo número. Este pasará a la salida correspondiente siempre y cuando no esté repetido. Una vez que se obtengan los 6 número del sorteo, esta entrada será ignorada.
- Las entradas ***New_sort*** y ***New_num*** permanecen a nivel alto o bajo durante un tiempo mucho mayor que el periodo de la señal ***CLK***.
- Las salidas **(N1, ... N6)**: representan los números del sorteo, los cuatro bits más significativos de corresponden con las decenas, mientras que los 4 menos significativos son las unidades.
- Salida ***End_sort***: indica, con un nivel alto durante un periodo de la señal ***CLK*** que ya te tiene el resultado del sorteo.
- El sistema dispone de una señal asíncrona ***RST***, no mostrada en el diagrama de bloques anterior, activa a nivel alto que se utiliza como señal de inicialización de todos los módulos secuenciales del diseño.

Ejercicio 7.



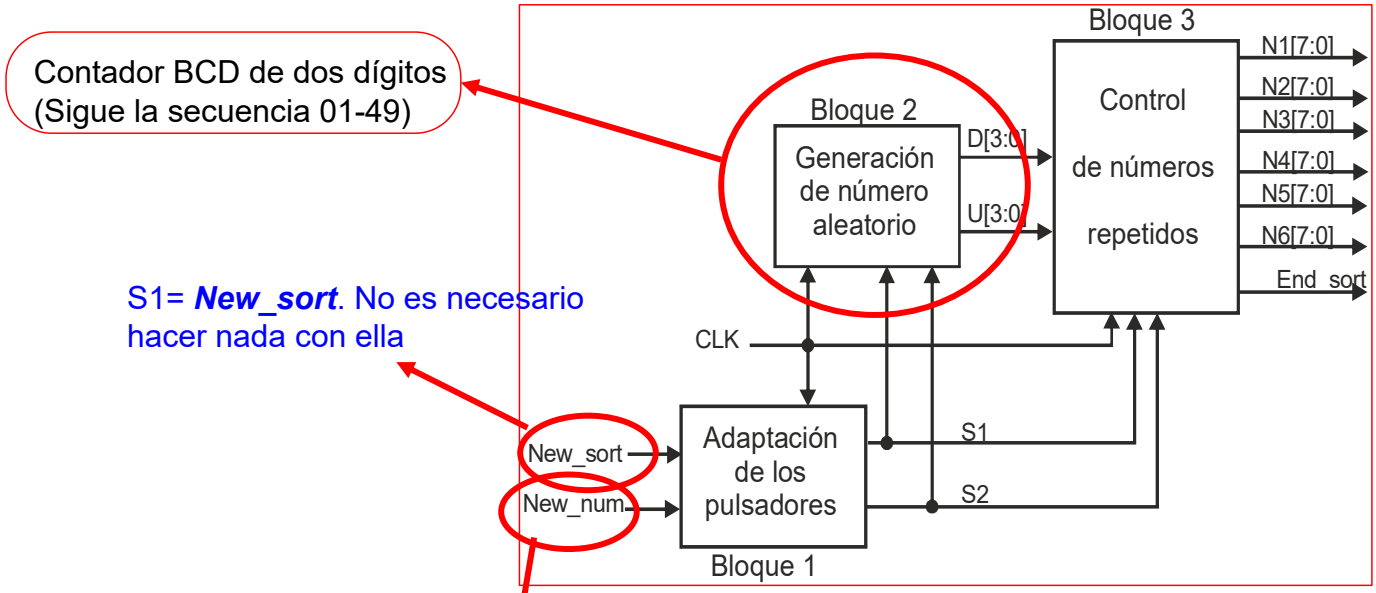

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_7 is
  port (
    CLK                         : in  std_logic;
    RST                         : in  std_logic;
    New_num                     : in  std_logic;
    New_sort                    : in  std_logic;
    N1, N2, N3                  : out std_logic_vector(7 downto 0);
    N4, N5, N6                  : out std_logic_vector(7 downto 0);
    End_sort                    : out std_logic);
end ejercicio_7;
```


Ejercicio 7.



- Bloque 1 se encarga de generar, a partir de las señales de los pulsadores, de generar las señales de control de los Bloques 2 y 3.
- Bloque 2: circuito que genera cada uno de los números aleatorios. Los números están codificados en BCD y constarán de dos dígitos: *D[3..0]* y *U[3..0]* para decenas y unidades respectivamente. Se corresponde con un contador BCD de dos dígitos.
- Bloque 3: circuito que controla que no haya números repetidos y almacena los números generados. El primer número se corresponde con ***N1***, el segundo con ***N2*** y así sucesivamente hasta ***N6***. Aquel número que esté repetido será desechado

Ejercicio 7.

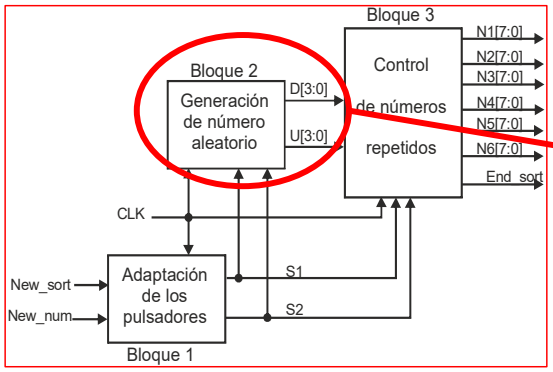


S1= ***New_sort***. No es necesario hacer nada con ella

S2= 1, durante un periodo de ***CLK*** cada vez que ***New_num*** se activa

Controla el Bloque 3

Ejercicio 7.

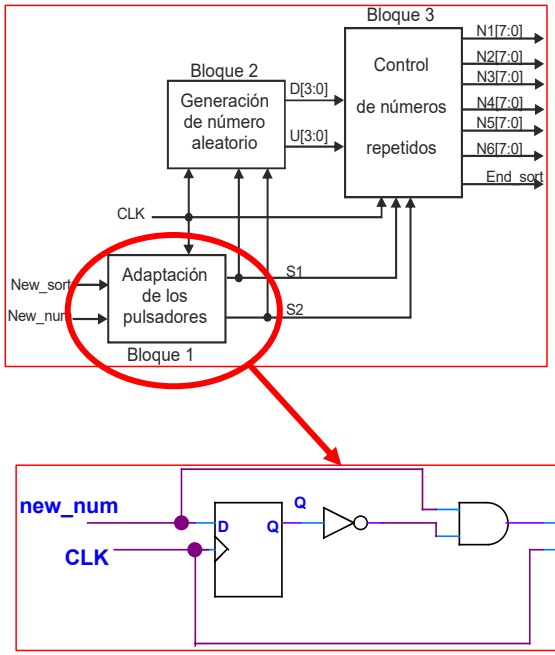



```
signal D,U:unsigned(3 downto 0);
signal DU :std_logic_vector(7 downto 0);
```

```
process (CLK, RST)
  begin
    if RST = '1' then
      D     <= (others => '0');
      U     <= (0       => '1', others => '0');
    elsif CLK'event and CLK = '1' then
      if New_sort = '1' then
        D   <= (others => '0');
        U   <= x"1";
      elsif U = 9 then
        U   <= x"1";
        if D = 4 then
          D <= (others => '0');
          U <=  x"1";
        else
          D <= D+1;
        end if;
      else
        U   <= U+1;
      end if;
    end if;
  end process;

  DU <= std_logic_vector(D&U);
```

Opcional

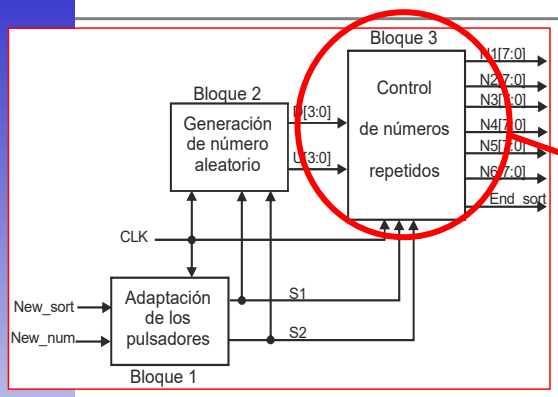
Ejercicio 7.




```
process(clk, rst)
  begin
    if rst = '1' then
      Q<= '0';
      S2<= '0';
    elsif clk'event and clk = '1' then
      Q<= New_num;
      S2<= New_num and not Q;
    end if;
  end process;
```

```
signal Q, S2:      std_logic;
```


7.




```
process (CLK, New_sort)
   variable igual : std_logic;
   variable num   : unsigned(2 downto 0);
 begin
   if RST = '1' then
     N_aux <= (others => (others => '0'));
     num   := (others => '0');
     End_sort   <= '0';
   elsif CLK'event and CLK = '1' then
     End_sort                          <= '0';
     if New_sort = '1' then
       N_aux <= (others => (others => '0'));
       num := (others => '0');
     else
       igual  := '0';
       if S2 = '1' and num < 6 then
         for j in 0 to 5 loop
           if N_aux(j) = DU then
             igual := '1';
           end if;
         end loop;
         if igual = '0' then
           N_aux(to_integer(num)) <= DU;
           num      := num+1;
         end if;
         if num = 6 then
           End_sort                      <= '1';
         end if;
       end if;
     end if;
   end if;
 end process;
```

```
N1 <= N_aux(0);
N2 <= N_aux(1);
N3 <= N_aux(2);
N4 <= N_aux(3);
N5 <= N_aux(4);
N6 <= N_aux(5);
```

```
type ar_2d is array (0 to 5) of std_logic_vector(7 downto 0);
  signal N_aux  :     ar_2d;
```

Ejercicio 8.

Un método iterativo para obtener la división de un dato ***A*** por otro ***B*** consiste en ir probando, uno a uno los bits del resultado y comenzando por el de mayor peso, si el bit es 1 ó 0. Así, por ejemplo, para realizar la división de un dato ***A*** de 8 bits 01111011_b (121_d), entre otro ***B*** de 4 bits 1000_b (8_d) se inicia poniendo el bit de mayor peso del resultado ***DIV*** a nivel alto (***DIV*** = 1000000_b -128_d). Si ***A***>= ***DIV****B*** significa que ese bit sebe ser un 1, 0 en caso contrario; en este caso (128*8=1024) debe ser 0. A continuación se examina si el siguiente bit es 1, para ello ***DIV*** =01000000_b (64_d), al realizar la comparación anterior (64*8=512) se obtiene que este bit también debe ser 0. A continuación se prueba con 00100000_b (32_d), obteniéndose que el segundo bit del resultado debe ser 0. Y así sucesivamente hasta obtener el bit de menor peso, Obteniéndose el resultado final ***DIV*** =00001111_b. Por su parte el resto será: 121-15*8= 1. Para la implementación del algoritmo se va utilizar un sistema secuencial de forma que cada valor intermedio se obtiene sincronizado con el flanco de subida de ***CLK***. El sistema dispone de una entrada ***D_OK*** que pasa nivel alto cuando hay un nuevo valor en ***A*** y ***B***, de forma que cuando pase a nivel bajo se inicia el proceso de cálculo. Los datos A y B tienen un tamaño de ***N*** y ***M*** bits, respectivamente. Una vez que éste finalice, la salida ***EC*** ***V*** y ***RESTO*** se mantiene el último de una señal asíncrona ***RST*** de i

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ejercicio_8 is
  generic (
    N      :      natural := 8;
    M      :      natural := 8);
  port (
    A      : in   std_logic_vector(N-1 downto 0);
    B      : in   std_logic_vector(M-1 downto 0);
    D_OK   : in   std_logic;
    CLK    : in   std_logic;
    RST    : in   std_logic;
    EC     : out  std_logic;
    RESTO  : out  std_logic_vector(     downto 0);
    DIV    : out  std_logic_vector(     downto 0));
end ejercicio_8;
```


Ejercicio 8.

A = 01111011_b (121_d),
B = 1000_b (8_d)

$\frac{A}{B} = \frac{121}{8} = 15{,}125$

DIV = 15_d (00001111_d),
RESTO = 1_d (0001_d)

1000_0000 (128) → (128*8=1024) 1024>121 → 0100_0000 (64) → (64*8=512) 512>121 → 0010_0000 (32) → (32*8=256) 256>121 → 0001_0000 (16) → (16*8=128) 128>121

0000_1000 (8) → (8*8=64) 64<121 → 0000_1100 (12) → (12*8=96) 96<121 → 0000_1110 (14) → (14*8=112) 112<121 → 0000_1111 (15) → (15*8=120) 120<121

0000_1111

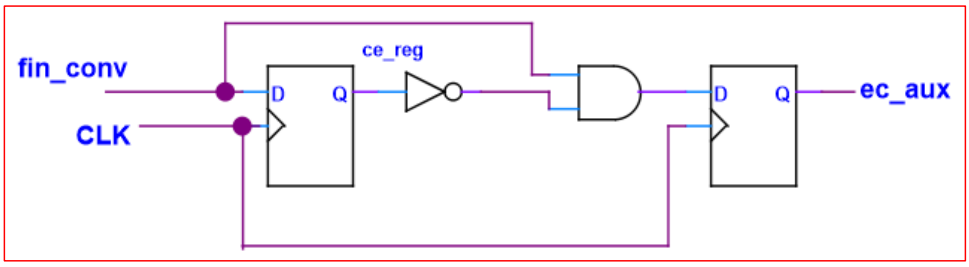
Ejercicio 8.

```
process (CLK, RST)
    variable q  : unsigned(N-1 downto 0);
  begin  -- process
    if RST = '1' then
      cnt      <= N-1;
      Q           := (others  => '0');
    elsif CLK'event and CLK = '1' then
      if D_OK = '1' then
        Q         := ( others => '0');
        cnt    <= N-1;
      else
        Q(cnt)    := '1';
        if unsigned(A) < Q*unsigned(B) then
          Q(cnt) := '0';
        end if;
        if cnt > 0 then
          cnt <= cnt-1;
        end if;
      end if;
    end if;
    div_aux    <= q;
  end process;
```

```
signal cnt :integer range 0 to N-1;
```

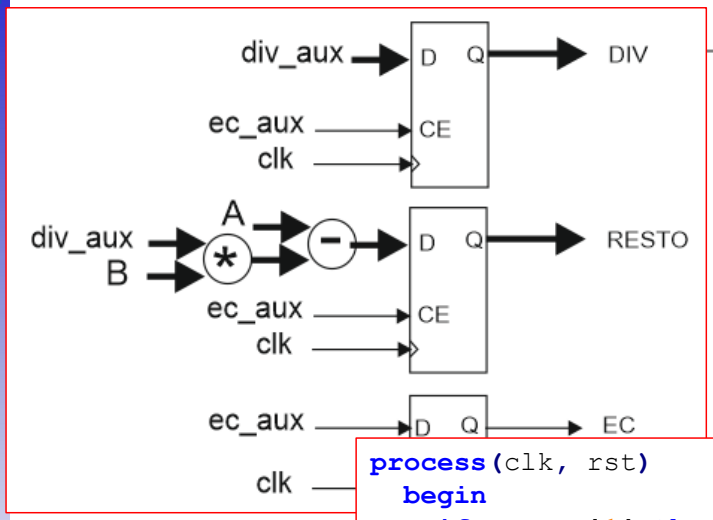
```
fin_conv <= '1' when cnt = 0 else '0';
```


Ejercicio 8.




```
process(clk, rst)
  begin
    if rst = '1' then
      ec_aux <= '0';
      ce_reg <= '0';
    elsif clk'event and clk = '1' then
      ce_reg <= fin_conv;
      ec_aux <= fin_conv and not ce_reg;
    end if;
  end process;
```

Ejercicio 8.




```
process(clk, rst)
  begin
    if rst = '1' then
      DIV         <= (others => '0');
      resto_aux   <= (others => '0');
      EC          <= '0';
    elsif clk'event and clk = '1' then
      EC          <= ec_aux;
      if ec_aux = '1' then
        DIV       <= std_logic_vector(div_aux);
        resto_aux <= unsigned(A)-(div_aux*unsigned(b));
      end if;
    end if;
  end process;
  RESTO <= std_logic_vector(resto_aux(RESTO'range));
```