ESTRUCTURA DE COMPUTADORES. UNIDAD 1
DISEÑO DIGITAL Y VHDL

1.1. Dadas las señales fuente y dest, ambas de tipo std_logic_vector(7 downto 0), escriba el código para que dest sea fuente desplazada aritméticamente una posición a la derecha.

Solución:

```
dest <= fuente(7) & fuente(7 downto 1);
```

1.2. Escriba el código equivalente al proceso adjunto, utilizando una sentencia concurrente de tipo *when – else*. La lista de sensibilidad no se indica, pero suponga que es la correcta.

```
process(  ...  )
begin
   if a < b then
      s <= b;
   elsif a < c then
      s <= c;
   else
      s <= a;
   end if;
end process;
```

Solución:

```
s <=   b when a < b else
       c when a < c else
       a;
```

1.3. Complete la lista de sensibilidad del proceso adjunto.

```
process(...)
begin
   if a = b then
      c <= d xor e;
   elsif f = '1' then
      g <= h;
   else
      i <= i + 4;
   end if;
end process;
```

Solución:

```
process( a, b, d, e, f, h, i )
```

1.4. Siendo *fuente* un *std_logic_vector(3 downto 0)* y *dest* un *std_logic_vector(6 downto 0)*, escriba el código para que *dest* sea igual a *fuente* multiplicado por 8 sin utilizar la multiplicación ni la suma.

Solución:

```
dest <= fuente & "000";
```

1.5. Siendo *clk, reset, d* y *q* señales de tipo *std_logic*, escriba el código para que *q* sea un biestable de tipo D con reset asíncrono activo a nivel bajo y activo por flanco de subida.

Solución:

```
process(clk, reset)
begin
    if reset = '0' then
        q <= '0';
    elsif clk = '1' and clk'event then -- también es válido rising_edge(clk)
        q <= d;
    end if;
end process;
```

1.6. Escriba una sentencia de tipo *assert* para comprobar en un test-bench que la señal s está a '1' o si no indicar "La señal s no vale 1" y parar la simulación.

Solución:

```
assert s = '1' report "La señal s no vale 1" severity error; -- también valdría severity failure
```

1.7. Escriba el código equivalente al indicado con una única sentencia de tipo *if*. Añada un proceso y su lista de sensibilidad si fuera necesario.

```
z <=  a when s = "00" else
      b when s = "01" else
      c;
```

Solución:

```
process(s, a, b, c)
begin
    if s = "00" then
        z <= a;
```

```
    elsif s = "01" then
        z <= b;
    else
        z <= c;
    end if;
end process;
```

1.8. Siendo *clk, reset, d* y *q* señales de tipo *std_logic*, escriba el código para que *q* sea un biestable de tipo D con reset síncrono activo a nivel bajo y activo por flanco de subida.

Solución:

```
process(clk)
begin
    if clk = '1' and clk'event then        -- también es válido rising_edge(clk)
        if reset = '0' then
            q <= '0';
        else
            q <= d;
        end if;
    end if;
end process;
```

1.9. Escriba una sentencia de tipo *assert* para comprobar en un testbench que la señal *s* es igual a '0' o si no sacar el mensaje "La señal s no vale 0" y parar la simulación.

Solución:

```
assert s = '0' report "La señal s no vale 0" severity error; -- también valdría severity failure
```

1.10. Escriba el código VHDL para generar un multiplexor cuya salida es la señal *z*, y cuyas entradas de datos son *a0*, *a1*, *a2* y *a3*, todas de tipo *std_logic_vector(31 downto 0)*. La señal de control es *sel*, de tipo *std_logic_vector(1 downto 0).* No hace falta que incluya la *entity* ni la *architecture*, sólo el código para la funcionalidad de la ALU (con process si es necesario).

Solución:

```
with sel select
    z <=    a0 when "00",
            a1 when "01",
            a2 when "10,
            a3 when others;
```

1.11. Dada la señal *dato4*, de tipo *std_logic_vector(3 downto 0)*, escriba el código VHDL para que *salida8*, de tipo *std_logic_vector(7 downto 0)*, sea *dato4* extendida en signo.

Solución:

```
salida8 <= dato4(3) & dato4(3) & dato4(3) & dato4(3) & dato4;
```

1.12. Escriba el código equivalente al indicado con una sentencia de tipo *case*. Añada un proceso y su lista de sensibilidad si fuera necesario.

```
with s select
      z <= a when "00",
           b when "01",
           c when others;
```

Solución:

```
process(s, a, b, c)
begin
    case s is
        when "00" => z <= a;
        when "01" => z <= b;
```

```
            when others => z <= c;
        end case;
    end process;
```

1.13. Siendo *clk, reset, d* y *q* señales de tipo *std_logic*, escriba el código para que *q* sea un biestable de tipo D con reset asíncrono activo a nivel alto y activo por flanco de bajada.

Solución:

```
process(clk, reset)
begin
    if reset = '1' then
        q <= '0';
    elsif clk = '0' and clk'event then          -- también es válido falling_edge(clk)
        q <= d;
    end if;
end process;
```

1.14. Escriba un proceso que genere una señal de reloj *clk* con período 20 ns (10 ns a nivel bajo y 10 ns a nivel alto), y que lo haga indefinidamente.

Solución:

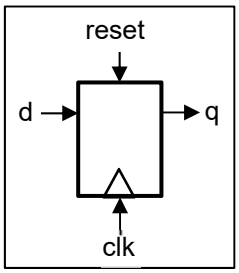
```
process
begin
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
end process;
```

1.15. Escriba el código VHDL para generar una pequeña ALU con señal de selección *s*, que si está a '0' genera la suma y si está a '1' genera la and. Las entradas de datos y la salida se llaman respectivamente *a*, *b* y *z*, y son todas de tipo *std_logic_vector(31 downto 0)*. No hace falta que incluya la *entity* ni la *architecture*, sólo el código para la funcionalidad de la ALU (con process si es necesario).

Solución:

```
z <= a + b when s = '0' else  a and b;
```

1.16. Utilizando VHDL, diseñe un proceso que, incluyendo la lista de sensibilidad oportuna, modele un flip-flop tipo D con entrada síncrona de Reset activo en alto y reloj activo por flanco de subida. Se adjunta un esquema de la entidad a modelar.



Solución:

```
process (clk)
begin
    if rising_edge (clk) then
        if reset = '1' then
            q <= '0' ;
        else q <= d ;
        end if;
    end if;
end process;
```

1.17. Escriba en la parte derecha el código equivalente al indicado con una sentencia de tipo *if*. Añada un proceso y su lista de sensibilidad si es necesario.

```
with s select
z <=   a when "00",
       b when "01",
       c when others;
```

Solución:

```
process(s, a, b, c)
begin
    if s=”00” then
        z <= a;
    elsif s=”01” then
        z <= b;
    else
        z <= c;
    end if;
end process;
```

1.18. Diseñe un multiplexor 4 a 1 con entradas de 8 bits de las siguientes cuatro formas:

- **a.** Con una sentencia ***if***.
- **b.** Con una sentencia ***case***.
- **c.** Con una sentencia ***when else***.
- **d.** Con una sentencia ***with select***.

Solución:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux4a1 is
      port ( D0, D1, D2. D3 : in std_logic_vector(7 downto 0);
          Sel: in std_logic_vector(1 downto 0);
          Y: out std_logic_vector(7 downto 0)       );
end Mux4a1;

architecture ArchIf of Mux4a1 is          -- ***Basada en la sentencia IF
begin
      process (Sel, D0, D1, D2, D3)       -- Por ser combinacional, en la lista de sensibilidad
      begin                               -- hay que poner todas las señales que intervienen
          if Sel = "00" then
              Y <= D0;
          elsif Sel = "01" then
              Y <= D1;
          elsif Sel = "10" then
              Y <= D2;
          else                            -- Utiliza else para no olvidar casos, equivale a Sel = "11"
              Y <= D3;
          end if;
      end process;
end ArchIf;

architecture ArchCase of Mux4a1 is        -- ***Basada en la sentencia CASE
begin
      process (Sel, D0, D1, D2, D3)
      begin
          case Sel is
              when "00" => Y <= D0;
              when "01" => Y <= D1;
              when "10" => Y <= D2;
              -- Se cierra con others para no olvidarnos casos
              when others => Y <= D3; -- "11"
          end case;
      end process;
end ArchCase;

architecture ArchWhen of Mux4a1 is        -- ***Basada en asignación condicional WHEN ELSE
begin
      Y <=  D0 when Sel = "00" else
            D1 when Sel = "01" else
            D2 when Sel = "10" else
            D3; -- Sel = "11"
end ArchWhen;

architecture ArchWith of Mux4a1 is        -- ***Basada en asignación condicional WITH-SELECT
begin
      with Sel select
          Y <=  D0 when "00",
                D1 when "01",
                D2 when "10",
                D3 when others; -- "11"
end ArchWith;
```

1.19. Diseñe un *testbench* para un multiplexor 4 a 1 con entradas de 8 bits del ejercicio anterior.

Solución:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux4a1Tb is
end Mux4a1Tb;

architecture Test of Mux4a1Tb is

        component Mux4a1                        -- Declaración de la entidad que se prueba
        port (D0, D1, D2, D3 : in std_logic_vector(7 downto 0);
                Sel : in std_logic_vector(1 downto 0);
                Y : out std_logic_vector(7 downto 0)      );
        end component;

                -- Declaración de señales, se podrían llamar igual que los puertos de Mux4a1.
                --  Se ponen nombres distintos para diferenciar que es puerto y que es señal.
        signal sD0, sD1, sD2, sD3, sY : std_logic_vector(7 downto 0);
        signal sSel : std_logic_vector(1 downto 0);

        constant CICLO : time := 10 ns;         -- Constante de tiempo

begin
        uut: Mux4a1 port map (                  -- Se instancia el módulo a probar (Unit Under Test)
                D0 => sD0,
                D1 => sD1,
                D2 => sD2,
                D3 => sD3,
                Sel => sSel,
                Y => sY         );

        ProcPrinc: process              -- Generación de estímulos y comprobación de resultados
        begin
                sD0 <= (others => '0');                         -- Valor asignado = 0
                sD1 <= "00000001";                              -- Valor asignado = 1
                sD2 <= X"02";                                   -- Valor asignado = 2
                sD3 <= (1 downto 0 => '1', others => '0');      -- Valor asignado = 3
                sSel <= "00";
                wait for CICLO;
                assert sY = sD0 report "Falla en el caso 00" severity failure;

                -- Señales que no cambian no hace falta volver a ponerlas porque conservan su valor.
                sSel <= "01";
                wait for CICLO;
                assert sY = sD1 report "Falla en el caso 01" severity failure;

                sSel <= "10";
                wait for CICLO;
                assert sY = sD2 report "Falla en el caso 10" severity failure;

                sSel <= "11";
                wait for CICLO;
                assert sY = sD3 report "Falla en el caso 11" severity failure;

                assert false
                        report "Si ha llegado aquí sin fallos previos, funciona  bien"
                                severity note;
                wait;                   -- Mata este proceso, y al ser el único finaliza la simulación
        end process;
end Test;
```

1.20. Diseñe un contador ascendente de 8 bits con reset asíncrono y carga en paralelo.

Solución:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Cont8Load is
   port (
        Clk : in std_logic;
        Reset : in std_logic;
        Load : in std_logic;
        Data : in std_logic_vector(7 downto 0);
        Q : out std_logic_vector(7 downto 0)
        );
end Cont8Load;

architecture Practica of Cont8Load is
        signal qAux : std_logic_vector (7 downto 0);          -- Señal auxiliar que almacena el valor,
                                                              -- no puede leerse la salida, Q
begin
    Q <= qAux;

    PrCont: process(Clk, Reset)
    begin
        if Reset = '1' then
            qAux <= (others => '0');
        elsif rising_edge(Clk) then
            if Load = '1' then
                qAux <= Data;
            else                                              -- funcionamiento normal
                qAux <= qAux + 1;
            end if;
        end if;
    end process;

end Practica;
```

1.21. Diseñe un contador ascendente de 8 bits con reset asíncrono y carga en paralelo que se elige entre cuatro posibles. Utilice para ello los módulos diseñados anteriormente (multiplexor y contador).

Solución:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Cont8Mux is
 port ( Clk, Reset, Load : in std_logic;
      Sel : in std_logic_vector(1 downto 0);
      D0, D1, D2, D3 : in std_logic_vector(7 downto 0);
      Q : out std_logic_vector(7 downto 0)          );
end Cont8Mux;

architecture Practica of Cont8Mux is
    component Mux4a1                                    -- Declaración del mux
    port ( D0, D1, D2, D3 : in std_logic_vector(7 downto 0);
        Sel : in std_logic_vector(1 downto 0);
        Y : out std_logic_vector(7 downto 0)        );
    end component;

    component Cont8Load                                 -- Declaración del contador
    port(   Clk : IN std_logic;
        Reset : IN std_logic;
        Load : IN std_logic;
        Data : IN std_logic_vector(7 downto 0);
        Q : OUT std_logic_vector(7 downto 0)      );
    end component;

                    -- Declaración de señales internas, el dato que sale del mux y va al contador
                    -- no es puerto, el resto son puertos y ya están declarados
    signal Data : std_logic_vector(7 downto 0);

begin
                    -- Instanciación (mapeo) de los componentes por nombre

    El_Mux: Mux4a1 port map (  D0 => D0, D1 => D1, D2 => D2, D3 => D3, Sel => Sel, Y => Data );

    El_Contador: Cont8Load port map (Clk => Clk, Reset => Reset, Load => Load,
                                     Data => Data, Q => Q );

end Practica;
```

1.22. Rellene las listas de sensibilidad de los siguientes tres procesos

process Clk	*process* A, B, C	*process* L.S. vacía
begin *if rising_edge(Clk) then* *if Rst='1' then* *Q <= '0';* *else* *Q <= D;* *end if;* *end if;* *end process;*	*begin* *if A<B then* *Z <= A;* *elsif A=C then* *Z <= B;* *else* *Z <= (others => '0');* *end if;* *end process;*	*begin* *Clk <= '0';* *wait for 10 ns;* *Clk <= '1';* *wait for 10 ns;* *end process;*

1.23. Complete la arquitectura de un multiplexor 4 a 1:

```
entity Mux4a1 is port(
        A0, A1, A2, A3 : in std_logic_vector(7 downto 0);
        Z : out std_logic_vector(7 downto 0);
        Ctrl : in std_logic_vector(1 downto 0));
end Mux4a1;
architecture Examen of Mux4a1 is
begin
```

Hay 4 posibilidades	
Z <= A0 when Ctrl = "00" else A1 when Ctrl = "01" else A2 when Ctrl = "10" else A3;	with Ctrl select Z <= A0 when "00", A1 when "01", A2 when "10", A3 when others;

process(A0,A1,A2,A3,Ctrl)
begin
case Ctrl is
when "00" => Z <= A0;
when "01" => Z <= A1;
when "10" => Z <= A2;
when others => Z <= A3;
end case;
end process;
	process(A0,A1,A2,A3,Ctrl)
	begin
	if Ctrl = "00" then
	Z <= A0;
	elsif Ctrl = "01" then
	Z <= A1;
	elsif Ctrl = "10" then
	Z <= A2;
	else
	Z <= A3;
	end if;
	end process;

```
end Examen;
```

1.24. Complete el siguiente bucle perteneciente a un testbench que prueba un contador descendente de 4 bits con salida llamada Q. El testbench debe generar una notificación si algún caso no es correcto. Considera que el contador inicialmente tiene el valor de 15 y que el testbench no debe comprobar desbordamientos.

Recuerde que para convertir un std_logic_vector a integer se puede usar la función *conv_integer(<señal std_logic_vector>)* y para convertir un integer a un std_logic_vector se puede usar la función *conv_std_logic_vector(<señal integer>,<num_bits>)*.

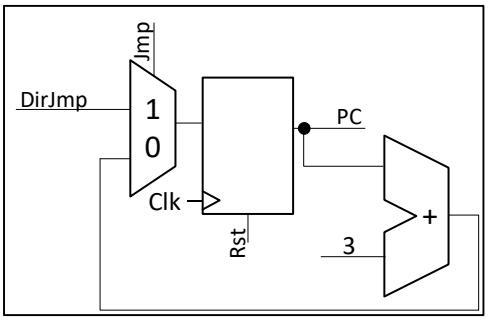
for i in 15 downto 0

assert i = conv_integer(Q) report “Error en caso “ & integer'image(i) severity error;
(No se valorará que esté indicado el valor de i)

```
    wait until Clk = '1';
    wait for 1 ns;
end loop;
```

ESTRUCTURA DE COMPUTADORES. UNIDAD 1
DISEÑO DIGITAL Y VHDL

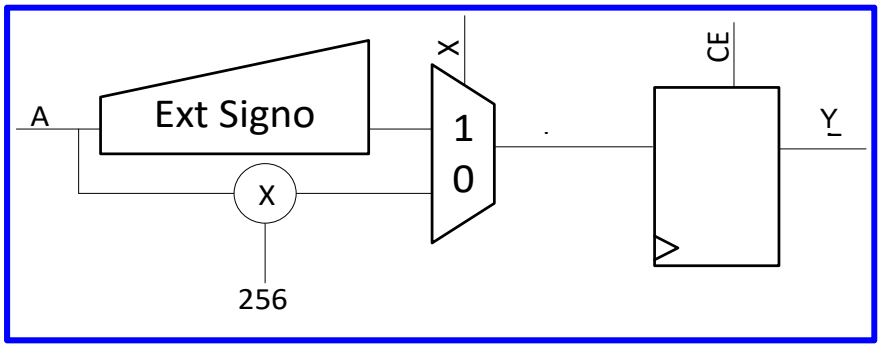
1.25. Se desea diseñar el módulo del PC (Program Counter) de un microprocesador no MIPS. Se sabe que cada instrucción ocupa 3 bytes y que se pueden realizar saltos a la dirección DirJump cuando la señal de control Jmp es igual a 1. El módulo tiene una señal de reset (Rst) asíncrona activa a nivel alto la cual pone el valor del PC a 0. Complete la arquitectura:




```
entity ModuloPC is port(
   Clk, Rst, Jmp : in std_logic;
   DirJmp : in std_logic_vector(15 downto 0);
  PC : out std_logic_vector(15 downto 0));
end ModuloPC;

architecture Problema of ModuloPC is
    signal PcAux : std_logic_vector(15 downto 0);

begin

    process(Clk, Rst)
    begin
        if Rst = '1' then
            PCAux <= (others => '0');
        elsif rising_edge(Clk) then
            if Jmp = '1' then
                PCAux <= DirJmp;
            else
                PCAux <= PCAux + 3;
            end if;
       end if;
    end process;

     PC <= PcAux;

end Problema;
```

1.26. Dibuje el circuito que sería sintetizado con el siguiente código VHDL. Todas las conexiones deben estar etiquetadas en el esquemático. Se sabe que A es una señal de tipo std_logic_vector(7 downto 0) Además, todos los componentes deben ser identificados correctamente, bien mediante un símbolo estándar o bien explicando su funcionamiento en texto auxiliar.

```
process(Clk)
begin
   if rising_edge(Clk) then
      if CE = '1' then
         if X = '1' then
            Y <= A(7) & A(7) & A(7) & A(7) & A(7) & A(7) & A(7) & A(7) & A;
        else
            Y <= A & "00000000"
        end if;
     end if;
  end if;
end process;
```



1.27. Dibuje el circuito que sería sintetizado con el siguiente código VHDL. Todas las conexiones deben estar etiquetadas en el esquemático. Además, todos los componentes deben ser identificados correctamente, bien mediante un símbolo estándar o bien explicando su funcionamiento en texto auxiliar.

Q1 <= A when Ctrl1 = '1' else
B when others;
Q2 <= C when Ctrl2 = '1' else
D when others;
Problema: Componente port map(PuertoA => Q1, PuertoB => Q2, Salida => Q3);