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*.

```
process(all)
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. 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”;
```

1.4. 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(all)
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;
```

1.5. 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
```

1.6. Escriba el código equivalente al indicado con una única sentencia de tipo *if*. Añada un proceso si fuera necesario.

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

Solución:

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

1.7. 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(all)
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;
```

1.8. 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;
```

1.9. 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;
```

1.10. Escriba el código equivalente al indicado con una sentencia de tipo *case*. Añada un proceso si fuera necesario.

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

Solución:

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

1.11. 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(all)
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;
```

1.12. 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:

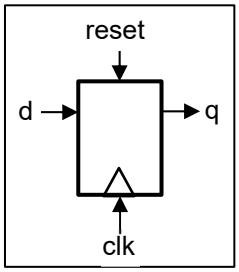
```
process
begin
      clk <= '0';
      wait for 10 ns;
      clk <= '1';
      wait for 10 ns;
end process;
```

1.13. 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;
```

1.14. Utilizando VHDL, diseñe un proceso que 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 (all)
begin
   if rising_edge (clk) then
      if reset = '1' then
         q <= '0' ;
      else q <= d ;
      end if;
   end if;
end process;
```

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

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

Solución:

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

1.16. 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;
use IEEE.numeric_std.all;


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

architecture ArchIf of Mux4a1 is          -- ***Basada en la sentencia IF
begin
      process (all)
      begin
          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 (all)
      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;
```

1.17. 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;
use IEEE.numeric_std.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 unsigned(7 downto 0);
		Sel : in std_logic_vector(1 downto 0);
		Y : out unsigned (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 : unsigned (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;
```

1.18. 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.numeric_std.all;

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

architecture Practica of Cont8Load is

begin

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

end Practica;
```

1.19. 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).

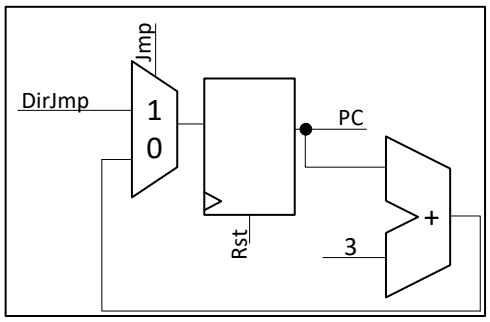
Solución:

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

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

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

1.20. 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.

```
for i in [ 15 downto 0 ] loop
    assert [ i = to_integer(Q)
                report "Error en caso " & to_string(i)
                    severity error; ]
    wait until Clk = '1';
    wait for 1 ns;
end loop;
```

1.21. 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 DirJmp 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:

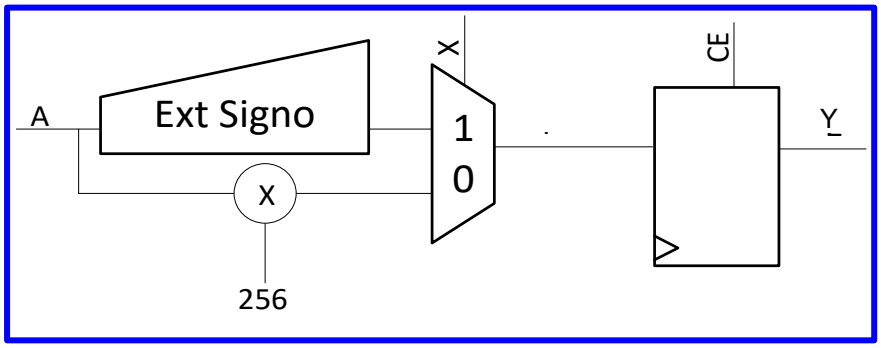



```
entity ModuloPC is port(
   Clk, Rst, Jmp : in std_logic;
   DirJmp : in unsigned(15 downto 0);
  PC : out unsigned(15 downto 0));
end ModuloPC;

architecture Problema of ModuloPC is
  begin

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

end Problema;
```

1.22. 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 signed(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(all)
begin
   if rising_edge(Clk) then
      if CE = '1' then
         if X = '1' then
            Y <= resize(A,16) ;
       else
            Y <= A & "00000000" ;
        end if;
     end if;
```

```
  end if;
end process;
```



1.23. 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);
```