Titulación: Grado en Ingeniería de Computadores
Asignatura: Tecnología de Computadores

Bloque 1: Introducción
Tema 3: Introducción a los lenguajes de descripción de hardware

Pablo Huerta Pellitero

ÍNDICE

- Bibliografía
- Herramientas de diseño de circuitos digitales
- Lenguajes de descripción de hardware
- El lenguaje VHDL

BIBILIOGRAFÍA

- F. Pardo, J.A Boluda, “VHDL, lenguaje para síntesis y modelado de circuitos”, editorial Ra-Ma
- Ll. Terés, Y. Torroja, S. Olcoz E. Villar, “VHDL, lenguaje estándar de diseño electrónico”, editorial MacGraw-Hill

HERRAMIENTAS DE DISEÑO DE CIRCUITOS DIGITALES

- Existen herramientas de diseño asistido por ordenador que facilitan el trabajo en distintas fases del flujo de diseño de circuitos digitales.
  - Simuladores lógicos.
  - Herramientas de síntesis.
  - Herramientas de verificación.
  - . . .
- En esta asignatura utilizaremos herramientas de simulación de circuitos digitales, que nos permitirán comprobar el correcto funcionamiento de los circuitos que diseñemos.
- Las herramientas de simulación necesitan que se les proporcione la información de cómo es el circuito en un formato apropiado:
  - Esquemático.
  - Netlist.
  - En algún lenguaje de descripción de hardware.

LENGUAJES DE DESCRIPCIÓN DE HARDWARE

- Actualmente, los lenguajes de descripción de hardware (HDLs) están sustituyendo casi por completo a los diseños con captura de esquemas debido a que:
  - Son estándares
  - Son reutilizables entre herramientas
  - Tienen las mismas funciones que los lenguajes de programación
  - Permiten verificar la corrección de los diseños con el propio lenguaje
- Además de cómo punto de entrada para las herramientas de simulación, los HDLs también se pueden usar como entrada para herramientas de síntesis si el circuito se describe de forma apropiada.
- Existen diversos HDLs (VHDL, Verilog, SystemC, ABEL, . . .), siendo los dos más populares VHDL y Verilog.

EL LENGUAJE VHDL

- VHDL: **V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC = Very High Speed Integrated Circuits
  - Lenguaje desarrollado a principios de los 80 para el DoD
    - Descripción y modelado de sistemas electrónicos digitales
    - Independiente de la tecnología de destino de la materialización
  - Estándar en 1987 – IEEE STd. 1076 – versiones VHDL-87 y VHDL-93
  - Lenguaje formal para la descripción de sistemas digitales a distintos niveles de abstracción independiente de la materialización
  - **Utilización:**
    - Descripción
    - Simulación
    - Documentación
  - **Características:**
    - Descripción a distintos niveles de abstracción
    - Simulación activada por eventos
    - Modular y jerárquico
    - Fuertemente tipado – descendiente directo de ADA

EL LENGUAJE VHDL

- La descripción de circuitos digitales en VHDL se puede hacer a distintos niveles de abstracción:
  - **Descripción estructural:** se describe la estructura de módulos, no el funcionamiento o el comportamiento.
  - **Descripción conductual:** se describe el comportamiento o la función que realiza el sistema y no su materialización estructural.
  - **Descripciones mixtas:** se describen comportamientos y estructuras.

- Tipos de datos:
  - Todas las señales internas y externas de un circuito descrito en VHDL tienen que pertenecer a algún tipo de datos.
  - Existen muchos tipos de datos en VHDL, pero por ahora sólo usaremos dos:
    - **std_logic**: representa 1 bit.
    - **std_logic_vector**: representa un vector de varios bits. El tamaño del vector se indica a la hora de declarar alguna señal de ese tipo.

EL LENGUAJE VHDL

- **Archivos VHDL**: los archivos que contienen código VHDL tienen una serie de características:
  - Son archivos de texto plano, que se pueden editar con cualquier editor como Bloc de Notas, NotePad++, Vi, Emacs, . . .
  - Tienen extensión .vhd o .vhdl
  - Dentro de un mismo archivo pueden describirse varios circuitos, o se puede emplear un archivo diferente para cada circuito que se quiera describir.
- **Estructura general de un archivo VHDL:**
  - Aunque hay muchas formas de organizar el código VHDL de los circuitos que se diseñan, nosotros vamos a tratar de utilizar siempre la siguiente estructura:

```
-- Librerías que se van a utilizar
. . .
. . .
-- Declaración de la entidad del circuito
. . .
. . .
-- Arquitectura del circuito
. . .
. . .
```

EL LENGUAJE VHDL: LIBRERÍAS

- **Las librerías o bibliotecas** incluyen tipos de datos, componentes ya hechos, funciones, etc, que podemos usar en nuestros diseños.
- Para usar una librería se utiliza la siguiente sintaxis:

```
library nombre_libreria;
use nombre_paquete_1;
use nombre_paquete_2;
...
```

  - La librería que utilizaremos habitualmente será la ***ieee,*** y dentro de ella el paquete ***ieee.std_logic_1164.all***
  - Si se necesita utilizar alguna operación aritmética (suma, resta, . . .) será también necesario utlizar los paquetes ***ieee.std_logic_arith.all*** y ***ieee.std_logic_unsigned.all***
  - Existe también una librería llamada **Work** que es en la que se van almacenando nuestros propios diseños.

EL LENGUAJE VHDL: ENTIDAD

- La entidad (***entity***) es la construcción del lenguaje VHDL que permite definir el interfaz que tiene un circuito, es decir que entradas y que salidas tiene el circuito.
- Las entradas y salidas de un circuito reciben en VHDL el nombre de *port*.

```
entity NombreDeLaEntidad is
    port (nombre_puerto_1: clase tipo;
          nombre_puerto_2: clase tipo;
          . . .
          nombre puerto _n: clase tipo );
end NombreDeLaEntidad;
```

La clase puede ser de entrada (***in***) o de salida (***out***). Hay otras clases, pero no las usaremos aún.

El tipo puede ser cualquier tipo de datos de VHDL. En nuestro caso sólo usaremos ***std_logic*** y ***std_logic_vector.***

EL LENGUAJE VHDL: ARQUITECTURA

- La entidad solamente define el interfaz del circuito. Para describir el funcionamiento se utiliza otra construcción del lenguaje que es la arquitectura (***architecture***). Una arquitectura por si sola no representa nada, tiene que ir asociada a una entidad, además una entidad puede tener asociadas varias arquitecturas que describen el circuito de distintas maneras o con distintos niveles de abstracción.
- La sintaxis básica de la arquitectura es la siguiente:

```
architecture NombreDeLaArquitectura of UnaEntidad is
-- Zona de declaraciones
begin
-- Cuerpo de la arquitectura
end NombreDeLaArquitectura;
```

- El tipo de sentencias que se pueden utilizar dentro de la zona de declaraciones y del cuerpo de la arquitectura es muy variado y lo iremos introduciendo con ejemplos en los temas siguientes.

EL LENGUAJE VHDL: SEÑALES

- **Concepto de señal:** las señales son un elemento del lenguaje que permite modelar nodos dentro un circuito (cables que unen partes internas de un circuito).
- Las señales se pueden leer y se pueden escribir, y tienen un comportamiento especial cuando se usan en determinados bloques que estudiaremos más adelante.
- Los puertos de una entidad se comportan como señales, pero con una excepción:
  - Los puertos de entrada sólo se pueden leer.
  - Los puertos de salida sólo se pueden escribir.
- Para añadir señales internas a una arquitectura hay que declararlas en la zona de declaraciones de la arquitectura, utilizando la siguiente sintaxis:

```
signal nombre_senyal is tipo;
```

EL LENGUAJE VHDL: ELEMENTOS SINTÁCTICOS

- Comentarios: cualquier línea que empieza por dos guiones. El compilador ignora las líneas que están comentadas.

```
-- Esto es un comentario
```

- Identificadores: se utilizan para dar nombre a módulos, señales, etc. No distinguen mayúsculas de minúsculas.
- Valores de un bit: se representan entre comillas simples.

```
‘1’ --esto es un bit con valor 1
‘0’ --esto es un bit con valor 0
```

- Cadenas de bits: se representan entre comillas dobles. Pueden llevar un prefijo que indica la base en que está representado. Si no se pone nada se supone que está en binario.

```
 “011011” --esto es una cadena binaria de bits
O“12307”  --esto es una cadena de bits en octal
X“12AB”  --esto es una cadena de bits en hexadecimal
```

EL LENGUAJE VHDL: OPERADORES

- Operadores aritméticos:
  - +, - : suma y resta.
  - *, / : multiplicación y división.
  - REM : resto de la división entera.
- Operadores relacionales: devuelven un valor booleano (verdadero o falso)
  - =, /= : igual y distinto.
  - >, >= : mayor y mayor o igual.
  - <, <= : menor y menor o igual.
- Operadores lógicos:
  - AND, OR, NOT, NAND, NOR, XOR
- Operador de concatenación:
  - & : concatena arrays de bits. EL array resultante es de tamaño igual a la suma de los tamaños de los arrays sobre los que opera.

EL LENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

- La asignación de valores a una señal o a un puerto de salida de una entidad se realiza mediante el operador de asignación concurrente: <= Existen tres formas de utilizarlo:

Asignación simple:

```
nombre_senyal <= valor;
```

Ejemplos:

```
a <= b + c;
a <= d AND e;
```

Asignación condicional:

```
nombre_senyal <= valor_1 when condicion_1 else
                                    valor_2 when
    condicion_2 else
                                        . . .
                          valor_n;
```

Ejemplo:

```
a <= not d when b > c else
          d when b < c else
          '0';
```

Asignación con selección:

```
with identificador select
nombre_senyal <= valor_1 when valor_id_1,
                 valor_2 when valor_id_2,
                 . . .
                 valor_n when others;
```

Ejemplo:

```
with A select
b <= "111" when "01",
     "000" when "10",
      not b when others;
```

EL LENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

- Se llama asignación concurrente por que todas las sentencias de ese tipo que hay dentro de una arquitectura son evaluadas por el simulador de forma concurrente (en paralelo)
- Ejemplos de uso del operador de asignación:

Puerta and de 2 entradas

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
  port(a, b: in std_logic;
       z: out std_logic);
end and2;

architecture arq1 of and2 is
begin
   z <= a and b;
end arq1;
```

Multiplexor de 2 a 1

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2a1 is
 port(x: in std_logic_vector(1 downto 0);
      sel: in std_logic;
      z: out std_logic);
end mux2a1;

architecture arq1 of mux2a1 is
begin
  z <= x(0) when sel = '0'
          else x(1);
end arq1;
```

EL LENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

- Ejemplos de uso del operador de asignación:

Multiplexor de 4 a 1

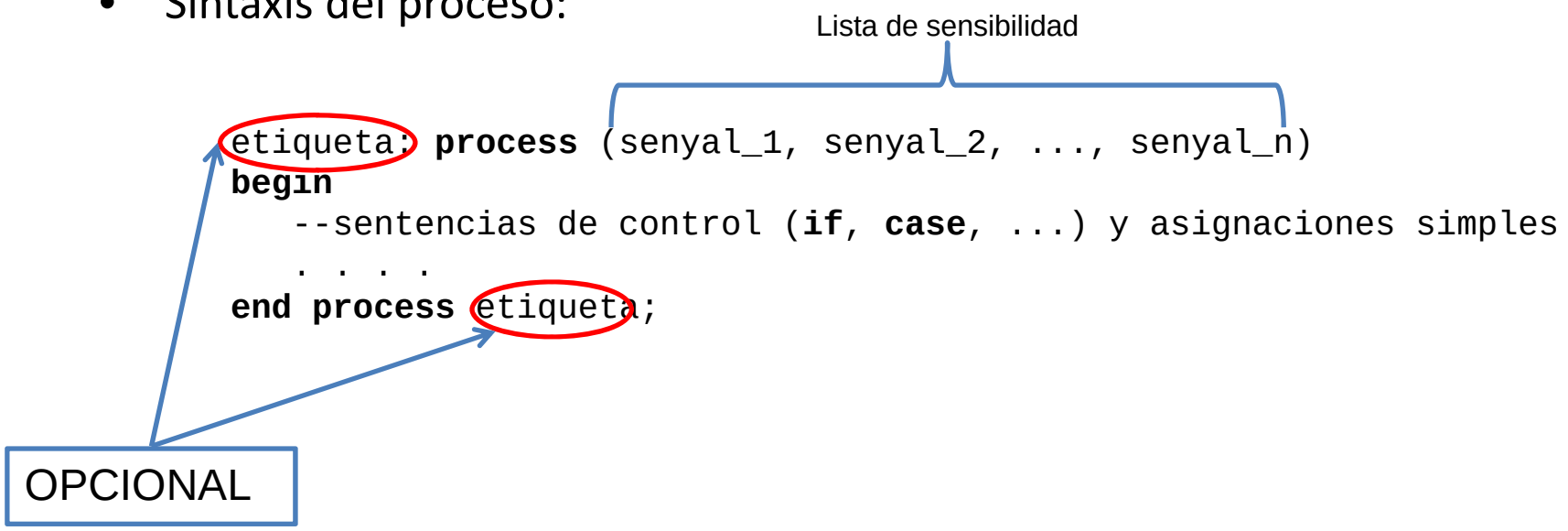
```
library ieee;
use ieee.std_logic_1164.all;

entity mux4a1 is
 port(x: in std_logic_vector(3 downto 0);
      sel: in std_logic_vector(1 downto 0);
      z: out std_logic);
end mux4a1;

architecture arq1 of mux4a1 is
begin
  with sel select z <= x(0) when "00",
                       x(1) when "01",
                       x(2) when "10",
                       x(3) when others;
end arq1;
```

EL LENGUAJE VHDL: PROCESOS

- Los procesos son un elemento del lenguaje VHDL que permite describir de forma más algorítmica circuitos digitales o partes de circuitos digitales.
- El simulador evaluará un proceso cuando se produzca un cambio en alguna de las señales de una lista que se conoce como "lista de sensibilidad". Un proceso es sensible a una señal si debe de ser evaluado cuando esa señal cambia.
- Sintaxis del proceso:



EL LENGUAJE VHDL: PROCESOS

- Las sentencias que hay dentro de un proceso son evaluadas por el simulador secuencialmente (una después de otra). Si existen varios procesos, el simulador los evalúa todos en paralelo.
- **Comportamiento de las señales:** cuando se da valor a una señal dentro de un proceso, este valor no se hace efectivo hasta que se termina de evaluar el proceso (se llega a **end process**):
  - Ejemplo: sean 'a', 'b' y 'c' 3 señales de tipo std_logic que valen en un momento dado '1', '1' y '0' respectivamente. Si la señal 'b' pasa a valer '0', ¿Qué sucede en el proceso ejemplo?:

```
ejemplo: process (b)
begin
        a <= b;
        c <= a;
end process ejemplo;
```

•Como b está en la lista de sensibilidad, se evaluará el proceso.
•El simulador apuntará que al salir del proceso 'a' tiene que valer '0' (lo que vale 'b').
•El simulador apuntará que al salir del proceso 'c' tiene que valer '1' (lo que valía 'a' al empezar el proceso).
•Cuando llega a **end process** el simulador hace efectivo el nuevo valor de 'a' y 'c', pasando a valer '0' y '1' respectivamente.

EL LENGUAJE VHDL: PROCESOS

- Dentro de los procesos se pueden utilizar una serie de sentencias de control que permiten controlar el flujo de ejecución del proceso. En este curso solo veremos la sentencia IF . . . THEN . . . ELSE y la sentencia CASE . . . WHEN
- IF . . . THEN . . . ELSE: permite en función de una o varias condiciones escoger que sentencias del proceso se ejecutarán. La parte ELSIF y ELSE es opcional.

```
if condicion then
...       -- sentencias
elsif otra_condicion then
...       -- sentencias
...
...
else
          --sentencias
end if;
```

EL LENGUAJE VHDL: PROCESOS

- CASE . . . WHEN : permite escoger entre varias sentencias dependiendo del valor que tome una señal o variable (en nuestro caso sólo trabajaremos con señales). Se deben cubrir todos los posibles valores de *senyal*.

```
case senyal is
        when valor_1 => ... –-sentencias
        when valor_2 => ... –-sentencias
        ...
        ...
        when others => ... --sentencias
end case;
```

OPCIONAL

EL LENGUAJE VHDL: PROCESOS

- Ejemplos: multiplexor de 2 a 1, y multiplexor de 4 a 1 de las transparencias 16 y 17, descritos mediante procesos.

Multiplexor de 2 a 1

```
architecture arq2 of mux2a1 is
begin
  process(x, sel)
  begin
     if sel = '0' then
        z <= x(0);
     else
        z <= x(1);
     end if;
  end process;
end arq2;
```

Multiplexor de 4 a 1

```
architecture arq2 of mux4a1 is
begin
  process(x, sel)
  begin
     case sel is
        when "00" => z <= x(0);
        when "01" => z <= x(1);
        when "10" => z <= x(2);
        when others => z <= x(3);
     end case;
  end process;
end arq2;
```

EL LENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

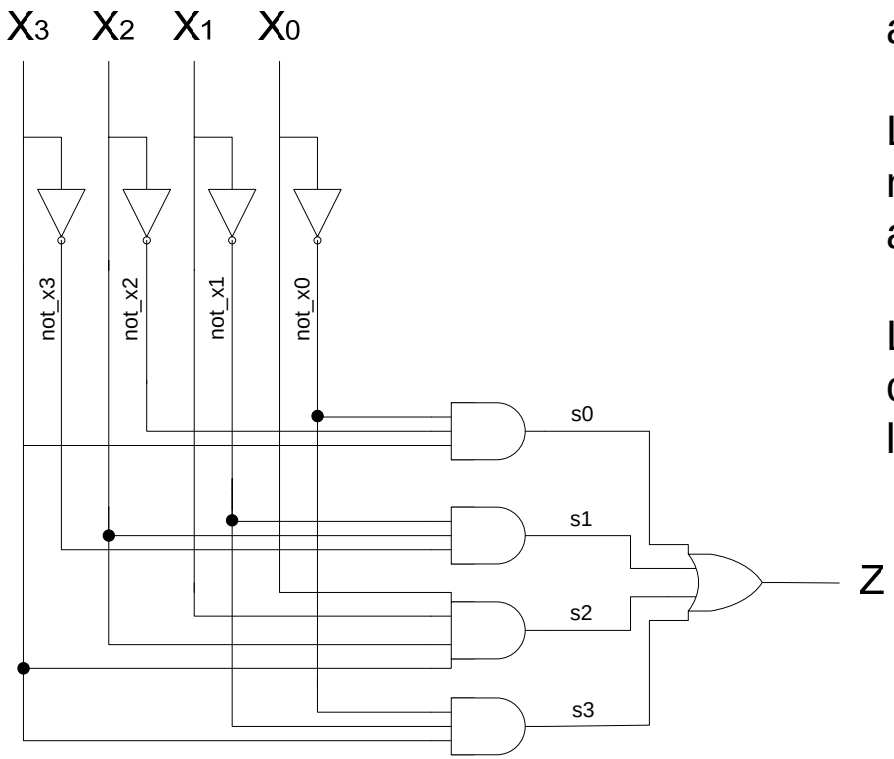
- Cuando decimos que un circuito está descrito de forma estructural queremos decir que se describe como un conjunto de **instancias** de varios módulos interconectadas mediante cables (**señales**).
- La forma de crear instancias de un componente es mediante la sentencia de instanciación, que tiene diversas sintaxis. La sintaxis habitual que utilizaremos en la asignatura es:

```
etiqueta: entity nombre_libreria.nombre_entidad(nombre_arquitectura)
port map(. . .);
```

- La lista que va después de ***port map*** indica que señales van conectadas a los distintos puertos que tiene el componente que se está instanciando. Existen dos formas diferentes de indicar esa lista de puertos:
  - Una lista ordenada de las señales que se desean conectar a los puertos de entrada y salida del componente.
  - Una lista *nombre_puerto => nombre_senyal*, que indica que señal va conectada a cada puerto.

EL LENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

- Ejemplo: utilizando los modelos de puertas lógicas del archivo *puertas_basicas.vhd* describir de forma estructural el siguiente circuito:



Se necesitarán una serie de señales auxiliares para unir las puertas.

Llamaremos not_x0, not_x1, not_x2 y not_x3 a las señales que están conectadas a la salida de cada uno de los inversores.

Llamaremos s0, s1, s2 y s3 a las señales conectadas a las salidas de cada una de las puertas **AND**.

EL LENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

- Ejemplo (continuación):

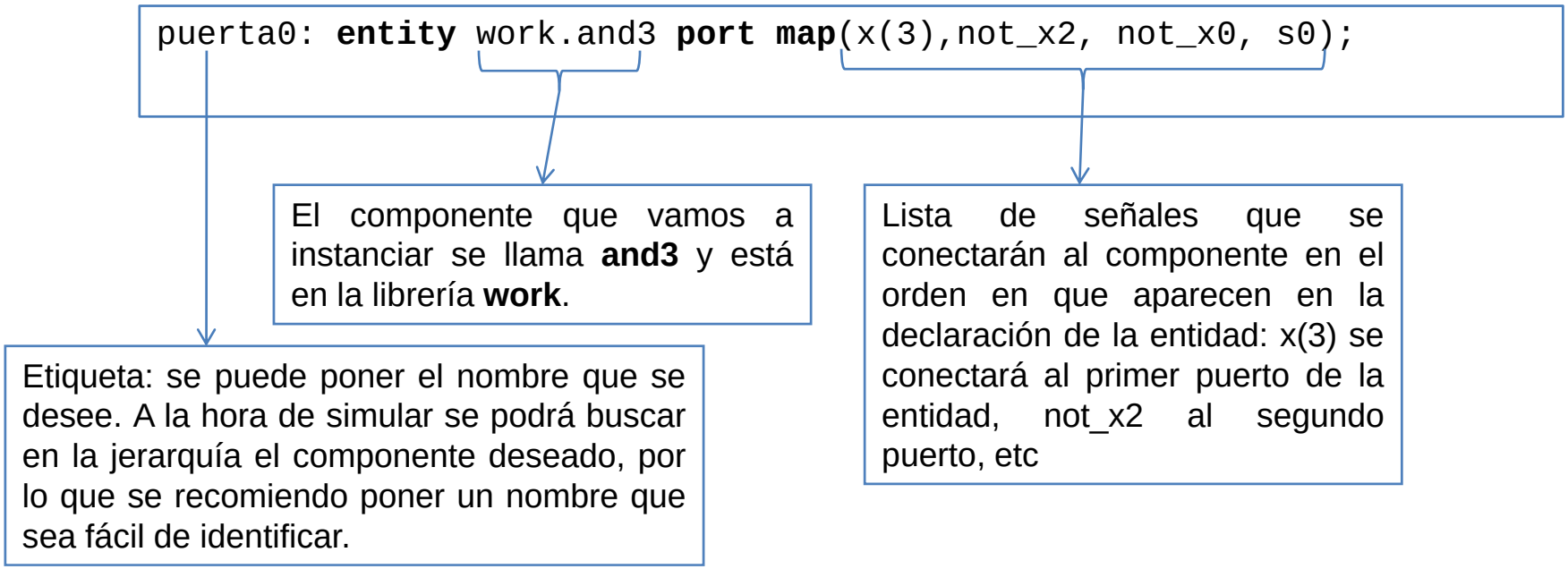
```
library ieee;
use ieee.std_logic_1164.all;

entity ejemplo_and_or_not is
  port(x: in std_logic_vector(3 downto 0);
       z: out std_logic);
end ejemplo_and_or_not;

architecture arq_1 of ejemplo_and_or_not is
signal s0, s1, s2, s3: std_logic;
signal not_x0, not_x1, not_x2, not_x3: std_logic;
begin
  inv_0: entity work.not1 port map(x(0), not_x0);
  inv_1: entity work.not1 port map(x(1), not_x1);
  inv_2: entity work.not1 port map(x(2), not_x2);
  inv_3: entity work.not1 port map(x(3), not_x3);
  puerta0: entity work.and3 port map(x(3),not_x2, not_x0, s0);
  puerta1: entity work.and3 port map(not_x3, x(2), not_x1, s1);
  puerta2: entity work.and4 port map(x(3), x(2), x(1), x(0), s2);
  puerta3: entity work.and3 port map(x(3), not_x1, not_x0, s3);
  puerta4: entity work.or4 port map(s0, s1, s2, s3, z);
end arq_1;
```

EL LENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

- Ejemplo (continuación): detalle de la sintaxis.



EL LENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

- Ejemplo (continuación): la lista de puertos se puede escribir de otra forma:

```
puerta0: entity work.and3 port map(a => x(3),
                                   b => not_x2,
                                   c => not_x0,
                                   z => s0);
```

Se indica por cada puerto, que señal irá conectada. No hace falta ponerlos en orden y se pueden dejar puertos sin conectar.

EL LENGUAJE VHDL: DESCRIPCIÓN CONCURRENTE

- Ejemplo (continuación): también se puede describir el mismo circuito de una forma más sencilla implementando la ecuación que describe el sistema mediante una asignación concurrente:

```
architecture arq_2 of ejemplo_and_or_not is
begin
  z <= ( x(3) and (not x(2)) and (not x(0)) ) or
        ( (not x(3)) and x(2) and (not x(1)) ) or
        ( x(3) and x(2) and x(1) and x(0) ) or
        ( x(3) and (not x(1)) and (not x(0)) );
end arq_2;
```

- ATENCIÓN: se debe tener cuidado con el orden de preferencia de los operadores AND, OR y NOT, y utilizar todos los paréntesis que sean necesarios para que la expresión sea la deseada. Si la expresión es muy larga se puede partir en varias líneas para que se lea con claridad.

EL LENGUAJE VHDL: DESCRIPCIONES MIXTAS

- A la hora de describir un circuito digital en VHDL se pueden utilizar simultáneamente todas las construcciones que hemos visto hasta ahora:
  - Asignaciones concurrentes.
  - Procesos.
  - Instancias de componentes.
- Todas estas construcciones son evaluadas en paralelo por el simulador.

EL LENGUAJE VHDL: ERRORES FRECUENTES

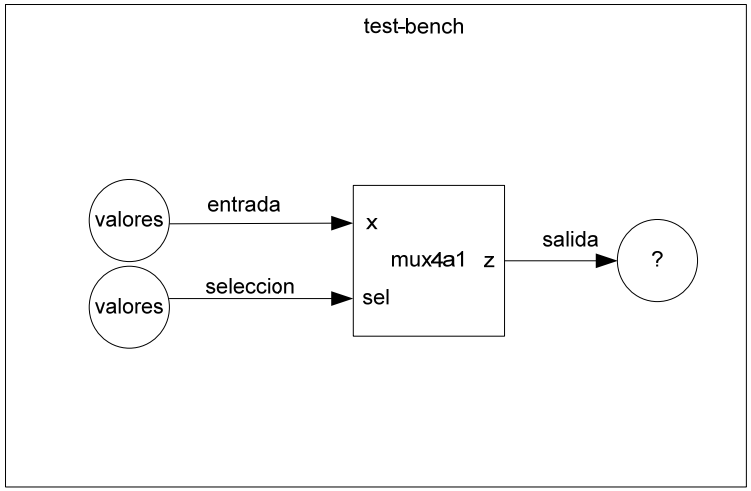
- Hay una serie de errores frecuentes que con la práctica se aprende a no cometer:
  - Las sentencias de asignación concurrente (***when else, with select when***) **NUNCA** pueden ir dentro de un **PROCESO**.
  - Las sentencias ***if then else***, y ***case when*** **SOLAMENTE** se pueden utilizar **DENTRO** de un **PROCESO**.
  - Una sentencia ***case when***, o una sentencia ***with select when*** **DEBEN** cubrir todos los casos posibles.
  - Una señal **NUNCA** debe ser escrita desde más de un proceso, asignación concurrente o instancia.
  - En la lista de sensibilidad de un proceso que describe lógica combinacional **DEBEN** aparecer **TODAS** las señales que se **LEEN** dentro del proceso.
  - En la lista de sensibilidad de un proceso que describe lógica secuencial **SOLAMENTE** debe aparecer la señal de **RELOJ.** Si hay **RESET**, **CLEAR** , o alguna otra señal **ASÍNCRONA TAMBIÉN** debe estar en la lista de sensibilidad.

EL LENGUAJE VHDL: TEST-BENCHS

- Hasta ahora hemos visto diversas maneras de describir circuitos digitales utilizando el lenguaje VHDL.
- Para comprobar que un circuito está bien diseñado es necesario simularlo para ver si se comporta como esperamos.
- Para simular circuito en VHDL se utilizan los *test-benches* o bancos de pruebas.
- Un *test-bench* es un circuito en el que se instancia el componente a simular y se le conectan las señales correspondientes a las entradas y a las salida.
- Mediante sentencias de asignación daremos valores apropiados a las señales de entrada, y el simulador se encargará de calcular las salidas del circuito para eso valores de entrada.

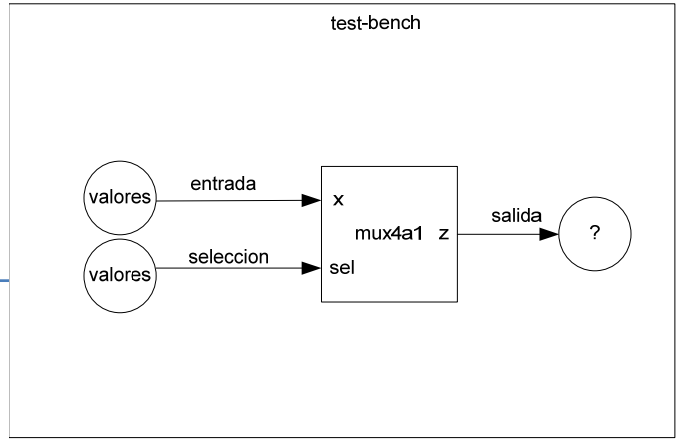
EL LENGUAJE VHDL: TEST-BENCHS

- Como el *test-bench* es un circuito, tendrá su entidad y su arquitectura.
- Como no tiene entradas ni salidas, será una entidad vacía.
- En la arquitectura del *test-bench* se declararán las señales que se conectarán al componente a simular, y se instanciará dicho componente.
- A las señales que son de entrada se les darán los valores apropiados para comprobar el correcto funcionamiento del componente que se está simulando.
- Ejemplo: *test_bench* del multiplexor de 4 a 1:



EL LENGUAJE VHDL: TEST-BENCHS

- Ejemplo (continuación):




```
library ieee;
use ieee.std_logic_1164.all;

entity test_mux4a1 is end test_mux4a1;
architecture test of test_mux4a1 is
signal entrada: std_logic_vector(3 downto 0) := "0000";
signal seleccion: std_logic_vector(1 downto 0);
signal salida: std_logic;
begin
  seleccion <= "00", "01" after 100 ns, "10" after 200 ns, "11" after
300 ns;
  entrada(0) <= not entrada(0) after 2 ns;
  entrada(1) <= not entrada(1) after 4 ns;
  entrada(2) <= not entrada(2) after 8 ns;
  entrada(3) <= not entrada(3) after 16 ns;
  inst_1: entity work.mux4a1 port map(entrada, seleccion, salida);
end test;
```

EL LENGUAJE VHDL: TEST-BENCHS

- Ejemplo (continuación): salida del simulador.