lación: Grado en Ingeniería de Computadores

Asignatura: Tecnología de Computadores

**ıe 1:** Introducción

**3:** Introducción a los lenguajes de descripción

rdware

Pablo Huerta Pellitero

CE

rafía

ientas de diseño de circuitos digitales

ajes de descripción de hardware

uaje VHDL

# _IOGRAFÍA

o, J.A Boluda, “VHDL, lenguaje para síntesis y modelado de
os”, editorial Ra-Ma

es, Y. Torroja, S. Olcoz E. Villar, “VHDL, lenguaje estándar de diseño
nico”, editorial MacGraw-Hill

# RAMIENTAS DE DISEÑO DE
# :UITOS DIGITALES

n herramientas de diseño asistido por ordenador que facilitan el
o en distintas fases del flujo de diseño de circuitos digitales.

- nuladores lógicos.
- rramientas de síntesis.
- rramientas de verificación.

a asignatura utilizaremos herramientas de simulación de circuitos
es, que nos permitirán comprobar el correcto funcionamiento de los
os que diseñemos.

erramientas de simulación necesitan que se les proporcione la
ación de cómo es el circuito en un formato apropiado:

- quemático.
- tlist.
- algún lenguaje de descripción de hardware.

# GUAJES DE DESCRIPCIÓN DE HARDWARE

mente, los lenguajes de descripción de hardware (HDLs) están yendo casi por completo a los diseños con captura de esquemas a que:

- estándares
- reutilizables entre herramientas
- nen las mismas funciones que los lenguajes de programación
- miten verificar la corrección de los diseños con el propio lenguaje

ás de cómo punto de entrada para las herramientas de simulación, Ls también se pueden usar como entrada para herramientas de s si el circuito se describe de forma apropiada.

diversos HDLs (VHDL, Verilog, SystemC, ABEL, . . .), siendo los dos opulares VHDL y Verilog.

# ENGUAJE VHDL

**V**HSIC **H**ardware **D**escription **L**anguage

- VHSIC = Very High Speed Integrated Circuits

nguaje 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

ándar en 1987 – IEEE STd. 1076 – versiones VHDL-87 y VHDL-93

nguaje formal para la descripción de sistemas digitales a distintos niveles de tracción independiente de la materialización

**lización:**

- Descripción
- Simulación
- Documentación

**racterísticas:**

- Descripción a distintos niveles de abstracción
- Simulación activada por eventos
- Modular y jerárquico
- Fuertemente tipado – descendiente directo de ADA

# ENGUAJE VHDL

cripción de circuitos digitales en VHDL se puede hacer a distintos
de abstracción:

- **scripción estructural:** se describe la estructura de módulos, no el cionamiento o el comportamiento.
- **scripción conductual:** se describe el comportamiento o la función que liza el sistema y no su materialización estructural.
- **scripciones mixtas:** se describen comportamientos y estructuras.

e datos:

las las señales internas y externas de un circuito descrito en VHDL tienen pertenecer a algún tipo de datos.

sten 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.

# ENGUAJE VHDL

**os VHDL**: los archivos que contienen código VHDL tienen una serie acterísticas:

archivos de texto plano, que se pueden editar con cualquier editor como c de Notas, NotePad++, Vi, Emacs, . . .

nen extensión .vhd o .vhdl

ntro de un mismo archivo pueden describirse varios circuitos, o se puede plear un archivo diferente para cada circuito que se quiera describir.

**tura general de un archivo VHDL:**

nque hay muchas formas de organizar el código VHDL de los circuitos que 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
. . .
. . .
```

# ENGUAJE VHDL: LIBRERÍAS

**brerías o bibliotecas** incluyen tipos de datos, componentes ya s, funciones, etc, que podemos usar en nuestros diseños.

sar una librería se utiliza la siguiente sintaxis:

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

librería que utilizaremos habitualmente será la ***ieee,*** y dentro de ella el quete ***ieee.std_logic_1164.all***

se necesita utilizar alguna operación aritmética (suma, resta, . . . ) será bién necesario utlizar los paquetes ***ieee.std_logic_arith.all*** y ***e.std_logic_unsigned.all***

ste también una librería llamada **Work** que es en la que se van acenando nuestros propios diseños.

# ENGUAJE VHDL: ENTIDAD

idad (***entity***) es la construcción del lenguaje VHDL que permite el interfaz que tiene un circuito, es decir que entradas y que salidas l circuito.

radas 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;
```

clase puede ser de entrada (***in***) o salida (***out***). Hay otras clases, pero las usaremos aún.

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

# ENGUAJE VHDL: ARQUITECTURA

idad solamente define el interfaz del circuito. Para describir el namiento se utiliza otra construcción del lenguaje que es la ectura (***architecture***). Una arquitectura por si sola no representa tiene que ir asociada a una entidad, además una entidad puede asociadas varias arquitecturas que describen el circuito de distintas as o con distintos niveles de abstracción.

axis básica de la arquitectura es la siguiente:

```
chitecture NombreDeLaArquitectura of UnaEntidad is
 Zona de declaraciones
gin
 Cuerpo de la arquitectura
d NombreDeLaArquitectura;
```

o de sentencias que se pueden utilizar dentro de la zona de aciones y del cuerpo de la arquitectura es muy variado y lo iremos uciendo con ejemplos en los temas siguientes.

# ENGUAJE VHDL: SEÑALES

**oto de señal:** las señales son un elemento del lenguaje que permite ar nodos dentro un circuito (cables que unen partes internas de un o).

eñales se pueden leer y se pueden escribir, y tienen un rtamiento especial cuando se usan en determinados bloques que aremos más adelante.

ertos de una entidad se comportan como señales, pero con una ión:

- puertos de entrada sólo se pueden leer.
- puertos de salida sólo se pueden escribir.

ñadir señales internas a una arquitectura hay que declararlas en la e declaraciones de la arquitectura, utilizando la siguiente sintaxis:

```
signal nombre_senyal is tipo;
```

# ENGUAJE VHDL: ELEMENTOS SINTÁCTICOS

ntarios: cualquier línea que empieza por dos guiones. El compilador las líneas que están comentadas.

```
o es un comentario
```

icadores: se utilizan para dar nombre a módulos, señales, etc. No uen mayúsculas de minúsculas.

s de un bit: se representan entre comillas simples.

```
esto es un bit con valor 1
esto es un bit con valor 0
```

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

```
1" --esto es una cadena binaria de bits
7"  --esto es una cadena de bits en octal
"  --esto es una cadena de bits en hexadecimal
```

# ENGUAJE VHDL: OPERADORES

dores aritméticos:

: suma y resta.

: multiplicación y división.

M : resto de la división entera.

dores relacionales: devuelven un valor booleano (verdadero o falso)

= : igual y distinto.

>= : mayor y mayor o igual.

<= : menor y menor o igual.

dores lógicos:

D, OR, NOT, NAND, NOR, XOR

dor de concatenación:

concatena arrays de bits. EL array resultante es de tamaño igual a la suma los tamaños de los arrays sobre los que opera.

# ENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

gnación de valores a una señal o a un puerto de salida de una d se realiza mediante el operador de asignación concurrente: <= tres formas de utilizarlo:

mple:

```
al <= valor;
```

;

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';
```

on selección:

```
cador select
 <= 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;
```

# ENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

ıa asignación concurrente por que todas las sentencias de ese tipo
y dentro de una arquitectura son evaluadas por el simulador de
concurrente (en paralelo)

os de uso del operador de asignación:

and de 2 entradas

```
ee;
td_logic_1164.all;

2 is
b: in std_logic;
out std_logic);


re arq1 of and2 is

and b;
```

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;
```

# ENGUAJE VHDL: ASIGNACIÓN CONCURRENTE

os de uso del operador de asignación:

Multiplexor de 4 a 1

```
ibrary ieee;
se ieee.std_logic_1164.all;

ntity mux4a1 is
ort(x: in std_logic_vector(3 downto 0);
     sel: in std_logic_vector(1 downto 0);
     z: out std_logic);
nd mux4a1;

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

# ENGUAJE VHDL: PROCESOS

ocesos son un elemento del lenguaje VHDL que permite describir de más algorítmica circuitos digitales o partes de circuitos digitales.

ulador evaluará un proceso cuando se produzca un cambio en de las señales de una lista que se conoce como “lista de lidad”. Un proceso es sensible a una señal si debe de ser evaluado o esa señal cambia.

s del proceso:

Lista de sensibilidad

```
queta: process (senyal_1, senyal_2, ..., senyal_n)
in
--sentencias de control (if, case, ...) y asignaciones simples
. . . .
 process etiqueta;
```

# ENGUAJE VHDL: PROCESOS

ntencias que hay dentro de un proceso son evaluadas por el dor secuencialmente (una después de otra). Si existen varios os, el simulador los evalúa todos en paralelo.

**rtamiento de las señales:** cuando se da valor a una señal dentro de ceso, este valor no se hace efectivo hasta que se termina de evaluar eso (se llega a **end process**):

mplo: sean ‘a’, ‘b’ y ‘c’ 3 señales de tipo std_logic que valen en un mento dado ‘1’, ‘1’ y ‘0’ respectivamente. Si la señal ‘b’ pasa a valer ‘0’, ué sucede en el proceso ejemplo?:

```
lo: process (b)

    a <= b;
    c <= a;
rocess 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.

# ENGUAJE VHDL: PROCESOS

de los procesos se pueden utilizar una serie de sentencias de que permiten controlar el flujo de ejecución del proceso. En este olo veremos la sentencia IF . . . THEN . . . ELSE y la sentencia CASE . . N

THEN . . . ELSE: permite en función de una o varias condiciones r que sentencias del proceso se ejecutarán. La parte ELSIF y ELSE es al.

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

# ENGUAJE VHDL: PROCESOS

. . WHEN : permite escoger entre varias sentencias dependiendo del que tome una señal o variable (en nuestro caso sólo trabajaremos ñales). 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

# ENGUAJE VHDL: PROCESOS

os: multiplexor de 2 a 1, y multiplexor de 4 a 1 de las transparencias
7, descritos mediante procesos.

ultiplexor de 2 a 1

```
cture arq2 of mux2a1 is

ss(x, sel)

 sel = '0' then
  z <= x(0);
se
  z <= x(1);
d if;
rocess;
2;
```

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;
```

# ENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

o decimos que un circuito está descrito de forma estructural nos decir que se describe como un conjunto de **instancias** de varios os interconectadas mediante cables (**señales**).

na de crear instancias de un componente es mediante la sentencia tanciación, que tiene diversas sintaxis. La sintaxis habitual que emos en la asignatura es:

```
ta: entity nombre_libreria.nombre_entidad(nombre_arquitectura)
ap(. . .);
```

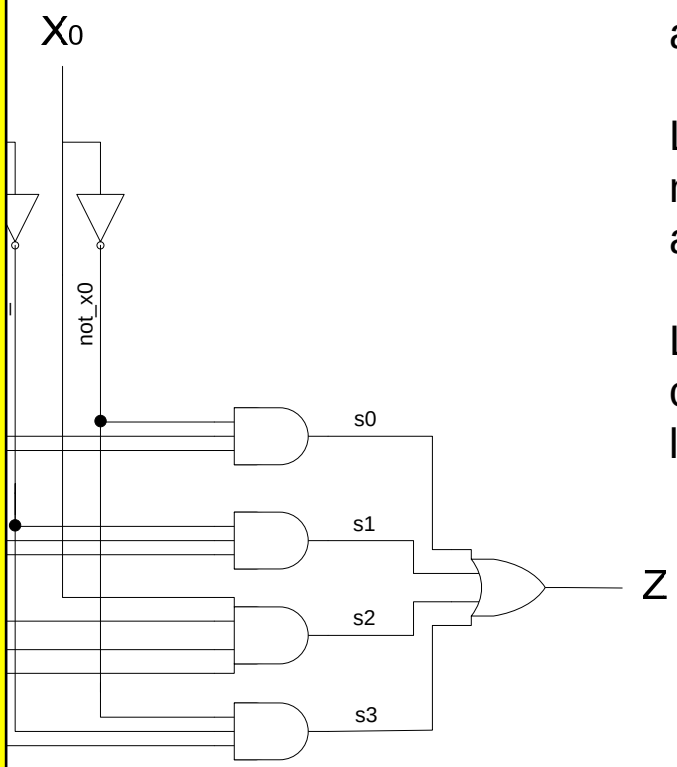
que va después de ***port map*** indica que señales van conectadas a tintos puertos que tiene el componente que se está instanciando. dos formas diferentes de indicar esa lista de puertos:

na lista ordenada de las señales que se desean conectar a los ertos de entrada y salida del componente.

na lista *nombre_puerto => nombre_senyal*, que indica que señal va nectada a cada puerto.

# ENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

o: utilizando los modelos de puertas lógicas del archivo *s_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**.

# ENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

o (continuación):

```
 ieee;
e.std_logic_1164.all;

ejemplo_and_or_not is
x: in std_logic_vector(3 downto 0);
z: out std_logic);
mplo_and_or_not;

cture arq_1 of ejemplo_and_or_not is
s0, s1, s2, s3: std_logic;
not_x0, not_x1, not_x2, not_x3: std_logic;

0: entity work.not1 port map(x(0), not_x0);
1: entity work.not1 port map(x(1), not_x1);
2: entity work.not1 port map(x(2), not_x2);
3: entity work.not1 port map(x(3), not_x3);
a0: entity work.and3 port map(x(3),not_x2, not_x0, s0);
a1: entity work.and3 port map(not_x3, x(2), not_x1, s1);
a2: entity work.and4 port map(x(3), x(2), x(1), x(0), s2);
a3: entity work.and3 port map(x(3), not_x1, not_x0, s3);
a4: entity work.or4 port map(s0, s1, s2, s3, z);
_1;
```

# ENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

o (continuación): detalle de la sintaxis.

```
): entity work.and3 port map(x(3),not_x2, not_x0, s0);
```

El componente que vamos a instanciar se llama **and3** y está en la librería **work**.

e poner el nombre que se de simular se podrá buscar componente deseado, por ndo poner un nombre que icar.

Lista de señales que se conectarán al componente en el orden en que aparecen en la declaración de la entidad: x(3) se conectará al primer puerto de la entidad, not_x2 al segundo puerto, etc

# ENGUAJE VHDL: DESCRIPCIÓN ESTRUCTURAL

o (continuación): la lista de puertos se puede escribir de otra

```
): 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.

# ENGUAJE VHDL: DESCRIPCIÓN CONCURRENTE

o (continuación): también se puede describir el mismo circuito de rma más sencilla implementando la ecuación que describe el a 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;
```

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

# ENGUAJE VHDL: DESCRIPCIONES MIXTAS

ora de describir un circuito digital en VHDL se pueden utilizar áneamente todas las construcciones que hemos visto hasta ahora:

- gnaciones concurrentes.
- cesos.
- tancias de componentes.

estas construcciones son evaluadas en paralelo por el simulador.

# ENGUAJE VHDL: ERRORES FRECUENTES

na serie de errores frecuentes que con la práctica se aprende a no er:

sentencias de asignación concurrente (***when else, with select when***) **NCA** pueden ir dentro de un **PROCESO**.

sentencias ***if then else***, y ***case when*** **SOLAMENTE** se pueden utilizar **NTRO** de un **PROCESO**.

a sentencia ***case when***, o una sentencia ***with select when*** **DEBEN** cubrir los los casos posibles.

a señal **NUNCA** debe ser escrita desde más de un proceso, asignación ncurrente o instancia.

la lista de sensibilidad de un proceso que describe lógica combinacional **BEN** aparecer **TODAS** las señales que se **LEEN** dentro del proceso.

la lista de sensibilidad de un proceso que describe lógica secuencial **LAMENTE** debe aparecer la señal de **RELOJ.** Si hay **RESET**, **CLEAR** , o alguna a señal **ASÍNCRONA TAMBIÉN** debe estar en la lista de sensibilidad.

# ENGUAJE VHDL: TEST-BENCHS

ahora hemos visto diversas maneras de describir circuitos digitales ndo el lenguaje VHDL.

omprobar que un circuito está bien diseñado es necesario simularlo er si se comporta como esperamos.

imular circuito en VHDL se utilizan los *test-benches* o bancos de as.

*t-bench* es un circuito en el que se instancia el componente a r y se le conectan las señales correspondientes a las entradas y a las

nte sentencias de asignación daremos valores apropiados a las s de entrada, y el simulador se encargará de calcular las salidas del o para eso valores de entrada.

# ENGUAJE VHDL: TEST-BENCHS

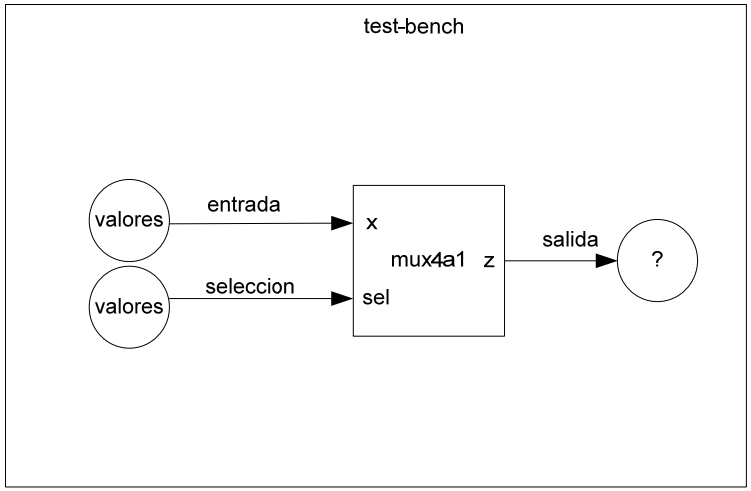
el *test-bench* es un circuito, tendrá su entidad y su arquitectura.

no tiene entradas ni salidas, será una entidad vacía.

arquitectura del *test-bench* se declararán las señales que se arán al componente a simular, y se instanciará dicho componente.

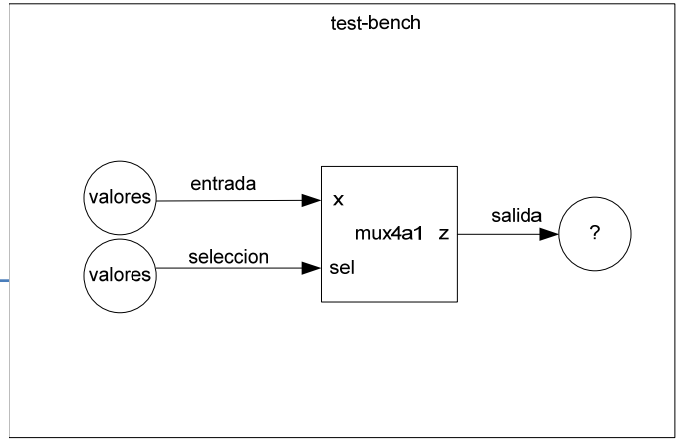
eñales que son de entrada se les darán los valores apropiados para obar el correcto funcionamiento del componente que se está ndo.

o: *test_bench* del multiplexor de 4 a 1:



# ENGUAJE VHDL: TEST-BENCHS

o (continuación):




```
e;
d_logic_1164.all;

_mux4a1 is end test_mux4a1;
e test of test_mux4a1 is
ada: std_logic_vector(3 downto 0) := "0000";
ccion: std_logic_vector(1 downto 0);
da: std_logic;

 <= "00", "01" after 100 ns, "10" after 200 ns, "11" after

) <= not entrada(0) after 2 ns;
) <= not entrada(1) after 4 ns;
) <= not entrada(2) after 8 ns;
) <= not entrada(3) after 16 ns;
ntity work.mux4a1 port map(entrada, seleccion, salida);
```

# ENGUAJE VHDL: TEST-BENCHS

o (continuación): salida del simulador.