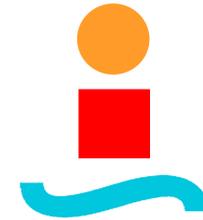




VHDL



Clase 1. Estructura de un diseño VHDL

**Introducción a los lenguajes HDL, ejemplo
básico y estructura de un diseño en VHDL**

Fernando Muñoz Chavero

Octubre de 2011

Clase 1:

Estructura de un diseño en VHDL

1.1 Introducción y ejemplo simple

1.2 La sección ENTITY

1.3 La sección LIBRARY

1.4 La sección ARCHITECTURE

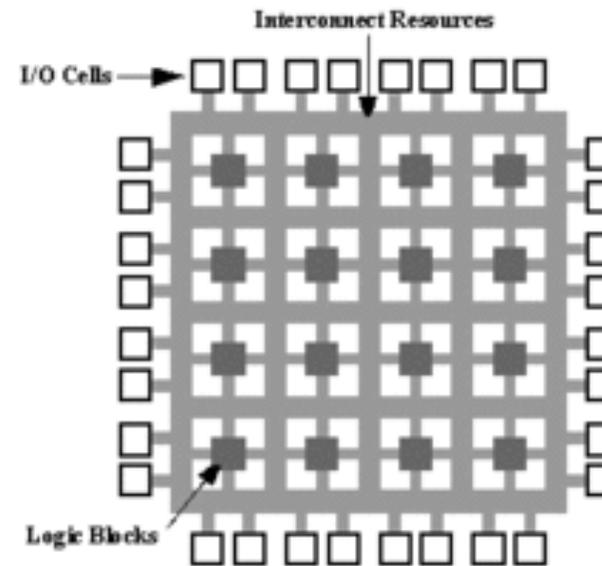
1.5 La sección CONFIGURATION

Bibliografía

- ◆ Douglas L. Perry. *VHDL*. McGraw-Hill, 2^a Edición. 1994.
- ◆ Apuntes de cátedra

¿Qué es una FPGA?

- ◆ FPGA = Field Programmable Gate Array.
 - Es un circuito integrado preparado para ser configurado por el cliente (diseñador).
 - Generalmente se configura utilizando un Leguaje de Descripción Hardware (HDL).



¿Porqué aprender a usar FPGAs?

- ◆ Prototipado rápido.
 - Time-to-market.
 - I+D.
 - Confidencialidad.



¿Porqué aprender a usar FPGAs?

- ◆ Rendimiento
 - Procesamiento en paralelo



JPMorgan Chase Spectacular (Time Square, NY)

¿Porqué aprender a usar FPGAs?

- ◆ Rendimiento → ASTRONOMÍA
 - Procesamiento en paralelo
- ◆ Proyecto SETI → BEE2 Systems



BEE2 de SETI (U. Berkeley)

- ◆ Primera FPGA:
 - ENTRADA: 16Gbps con ancho de banda (BW) de 800 MHz.
 - SALIDA: cuatro canales de 200MHz de BW.
 - PROCESADO: Filtrado polifásico.
- ◆ Cada uno de los cuatro canales procesado por otra FPGA
 - SALIDA: Espectrómetro con 256 Mcanales de 0.745Hz de BW.
 - OPERACIÓN: Filtrado, 32K FFT, cálculo de potencia (29.4 GMACs: billions of multiply-adds per second)
- ◆ El mismo sistema se realizó con DSP (C6415-7E and C6415T-1G de TI) y con procesadores (Pentium 4)
 - Mayor velocidad (throughput): 10-34 veces sobre DSP y 4-13 sobre MP Pentium.
 - Consumo de potencia: Un orden de magnitud mejor que DSP y dos órdenes de magnitud sobre MP Pentium.
 - Precio: 307% mejor que DSP y 500% mejor que MP.

¿Porqué aprender a usar FPGAs?

- ◆ Rendimiento → FÍSICA DE PARTICULAS
 - Procesamiento en paralelo
- ◆ Detector del Compact Muon Solenoid (CMS) del Large Hadron Collider (LHC) at CERN.

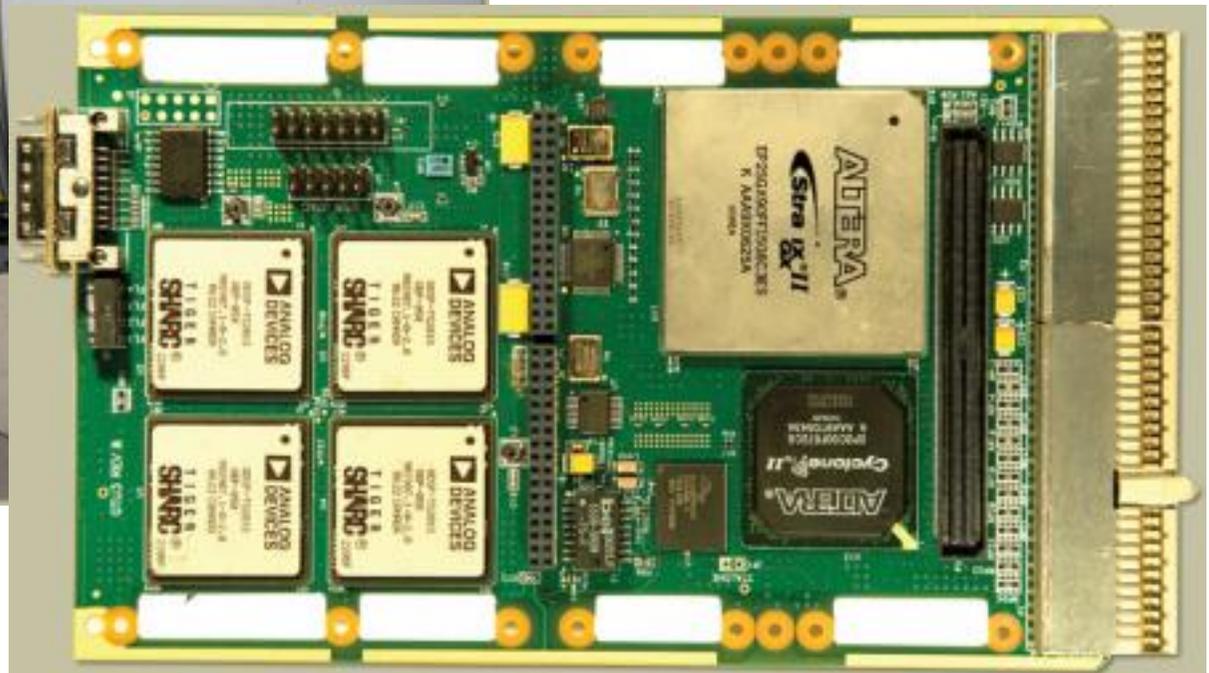


Experimento CMS del LHC (CERN)

- ◆ Utilización de FPGAs para reconocimiento de patrones.
- ◆ Detectores capturan impactos en una memoria analógica.
- ◆ Hay 9 millones de detectores leídos por 430 FED Front Ends Driver (tarjetas basadas en 14 (FPGAs Virtex de xilinx))
 - 3 FPGAs para recolección y distribución de datos.
 - 8 FPGAs para detección de patrones.
 - Una FPGA para transmitir los eventos interesantes detectados.
- ◆ Información de entrada de cada FED: 3.4 Gb/s
- ◆ Información de salida de cada FED: 400Mb/s

¿Porqué aprender a usar FPGAs?

- ◆ Procesamiento de señal en radares → Army's STARLite radar system (F18)



Empleos de VHDL | LinkedIn x

https://www.linkedin.com/jobs/search?keywords=VHDL&location=&trk=jobs_jserp_search_button_execute&locationId=

Fernando

¿Qué es LinkedIn? Únete hoy Inicia sesión

VHDL Ciudad o provincia Buscar

863 empleos de VHDL Cualquier momento

¿Te gusta esta búsqueda?
Recibe notificaciones cuando haya nuevos empleos que coincidan con tu búsqueda

Dirección de correo electrónico

Crear una alerta de empleo

Ubicación

- Austin, Texas (52)
- Sunnyvale, California (16)
- Baltimore, Maryland (16)
- Phoenix, Arizona (10)
- Melbourne, Florida (8)

Ver más

Empresa

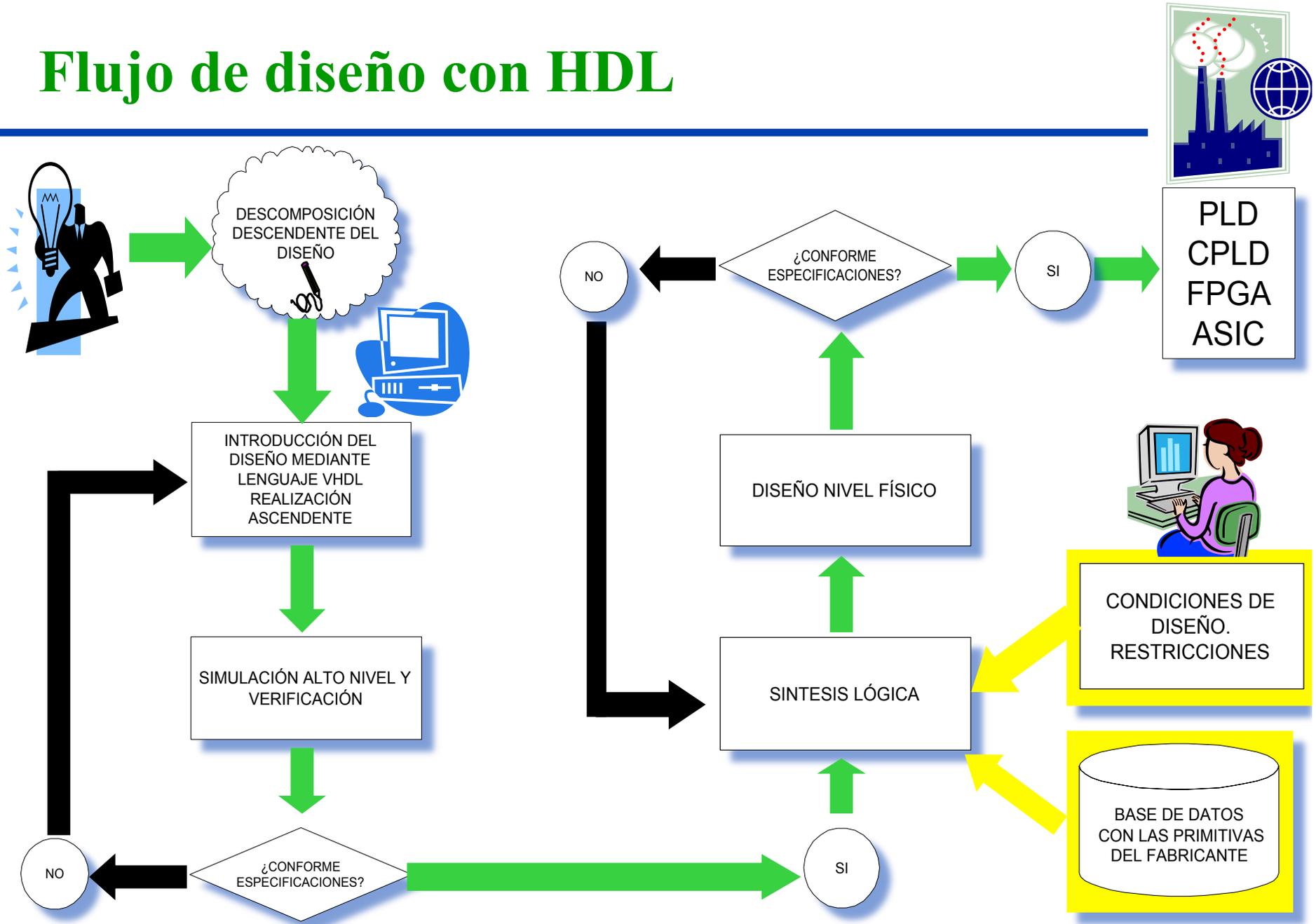
Verilog/VHDL Development Engineer
SeeScan
San Diego, California · 2 d
We have an exciting opportunity for someone who... of Verilog and/or VHDL (Verilog preferred...)

Engineer
Spherion
Melbourne, Florida · 3 d · [Solicita con tu perfil](#)
Seeking software and electrical engineers Design and implement analog/digital embedded electronic systems Understand communications, co and hardware and software requirements for embedded systems Document embedded software/hardware implementations Propose design products and conduct product development

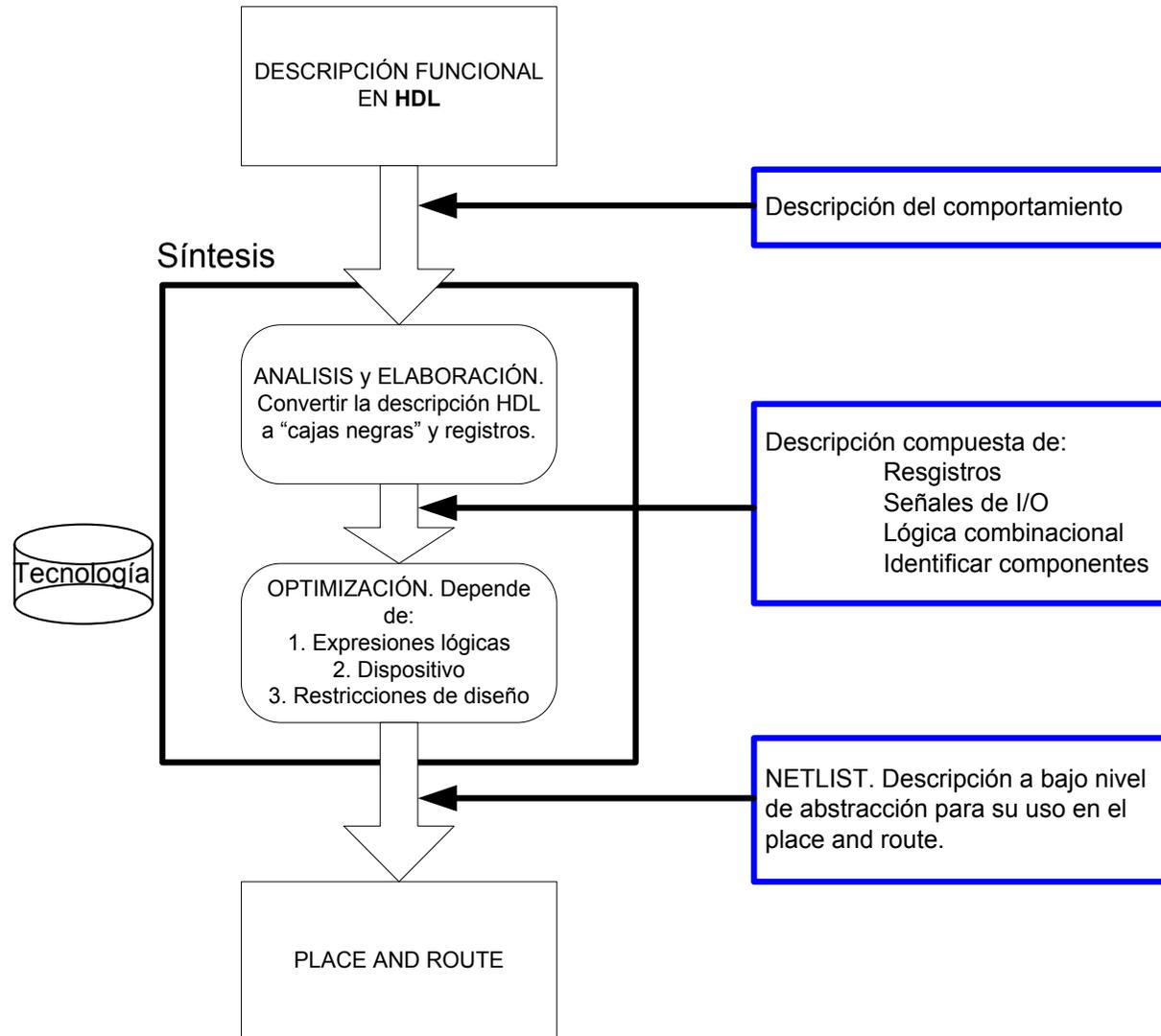
Characterization Product Engineer - Serdes
Microsemi Corporation
San Jose, California · 5 d
Verifying integrated circuit designs · Analyzing device behavior of FPGA/SOC products · Performing Failure Analysis and Yield Enhancement activities on FPGA/SOC products · Develop software for verifying and testing for FPGA/SOC products · Characterizing and evaluating FPGA/SOC products Validating on...

RF Engineer
Tektronix
Santa Clara, California · 22 d

Flujo de diseño con HDL

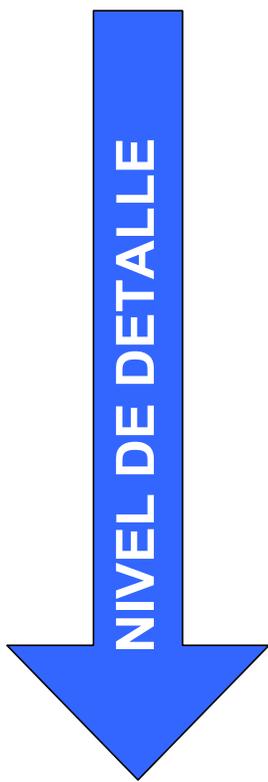


Síntesis de circuitos



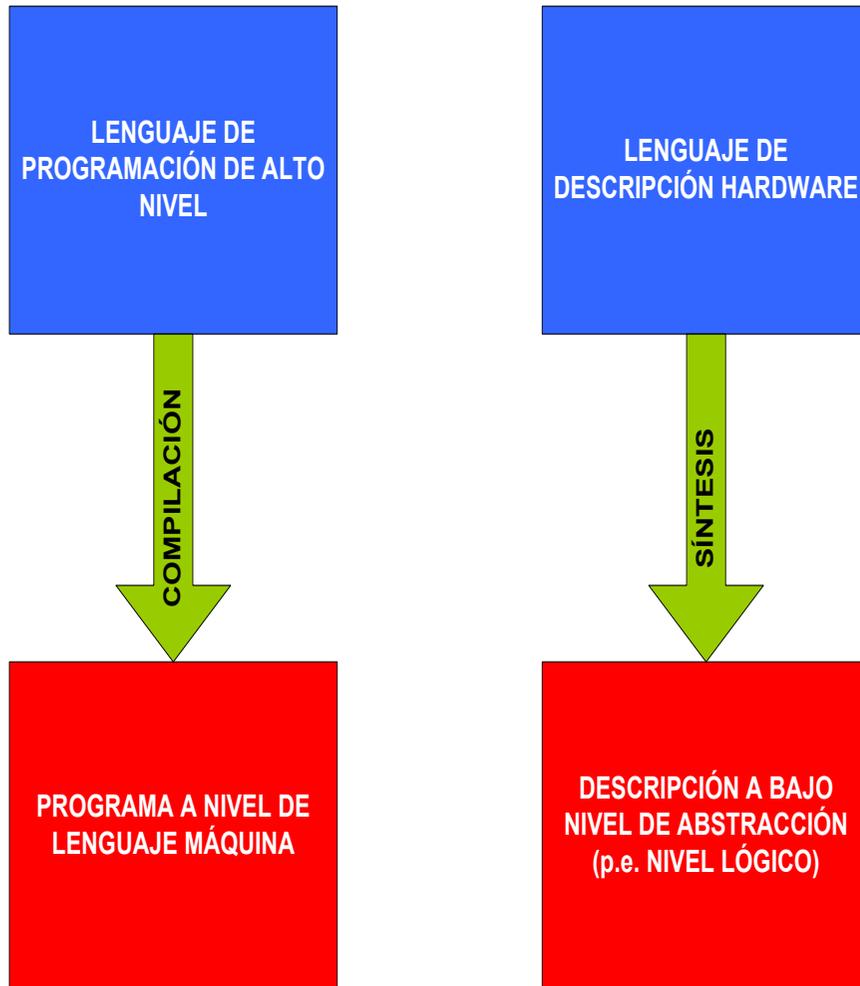
Niveles de abstracción en HDLs

Niveles de abstracción	Estilo descriptivo (HDL)	Tipo de datos
Nivel funcional (grandes bloques)	Algorítmico o descriptivo	Abstracto
Nivel de transferencia de registros (RTL)	Flujo de datos	Compuestos
Nivel de puerta lógica	En forma de esquema	Bit
Nivel de circuito	Interconexión de transistores. No es posible en HDL	Tensión e intensidad



NIVEL DE DETALLE

HDL: Programa o diseño



- ◆ La sintaxis es muy similar.
- ◆ En VHDL no se **programa**, se **describe**.
- ◆ Hay que pensar siempre que la descripción se corresponde con **circuitos** funcionando en paralelo

Ejemplo simple

□ Generalidades:

- VHDL no es sensible a mayúsculas y minúsculas.
 - `PEPE` es igual a `pEpE`
- Los comentarios son de línea precedidos por `--`
 - `-- esto es un comentario`

Ejemplo básico

□ El VHDL es complejo pero:

- El subconjunto necesario para síntesis es muy pequeño.
- Solo utilizaremos:

- Asignaciones:

```
estado_siguiete <= REPOSOS ;
```

- Comparaciones y operadores lógicos

= (igual), /= (no igual), > (mayor que), <= (menor o igual que), etc

And, xor, or, nand, nor, xnor, not, etc

- Sentencia if

```
If (estado_actual=REPOSOS) then ... End if;
```

- Sentencia case

Ejemplo básico

-- -- es comentario

--Zona de declaración de librerías

```
LIBRARY nombre_librería;  
USE librería.paquete_funciones.all;
```

*Declaración de librerías y
paquetes*

--Cabecera de la entidad

```
ENTITY nombre_entity IS  
  GENERIC(.....);  
  PORT(.....);  
END nombre_entity;
```

Definición de la entidad

--Cuerpo de la entidad

```
ARCHITECTURE nombre_architecture OF nombre_entity IS  
  --Declaración de componentes y señales  
  BEGIN  
    --Descripción de la funcionalidad  
  END nombre_architecture;
```

*Descripción de la
arquitectura*

--Enlace con las arquitecturas de otras entidades

```
CONFIGURATION nombre_configuracion OF nombre_entidad IS  
  FOR nombre_arquitectura  
  --Cuerpo de la configuración  
END nombre_configuracion;
```

*Selección de la
configuración*

Ejemplo básico

-- Multiplexor de dos entradas

```
entity mux21 is  
port (e1, e2: IN std_logic;  
       sel: IN std_logic;  
       o1: OUT std_logic);
```

```
end mux21;
```

```
architecture A of mux21 is
```

```
begin
```

```
  process (e1, e2, sel)
```

```
  begin
```

```
    if (sel='0') then
```

```
      o1 <= e1;
```

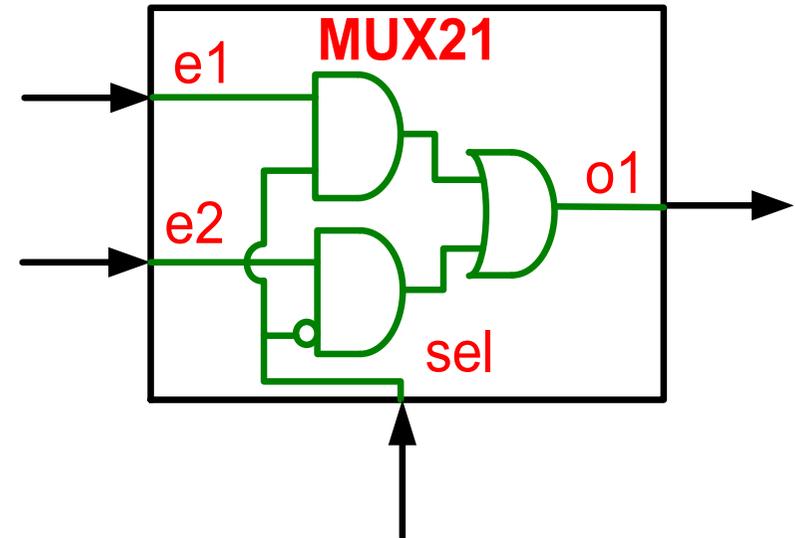
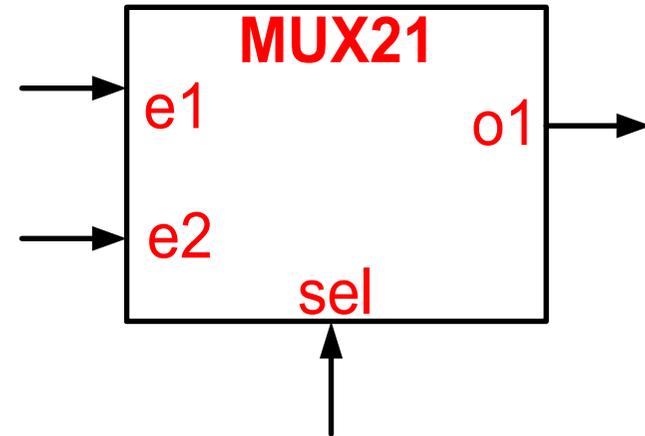
```
    else
```

```
      o1 <= e2;
```

```
    endif,
```

```
  end process;
```

```
end A;
```



Clase 1:

Estructura de un diseño en VHDL

1.1 Introducción y ejemplo simple

1.2 La sección *ENTITY*

1.3 La sección LIBRARY

1.4 La sección ARCHITECTURE

1.5 La sección CONFIGURATION

La sección ENTITY

ENTITY contador **IS**



GENERIC (
 N: integer := 10);



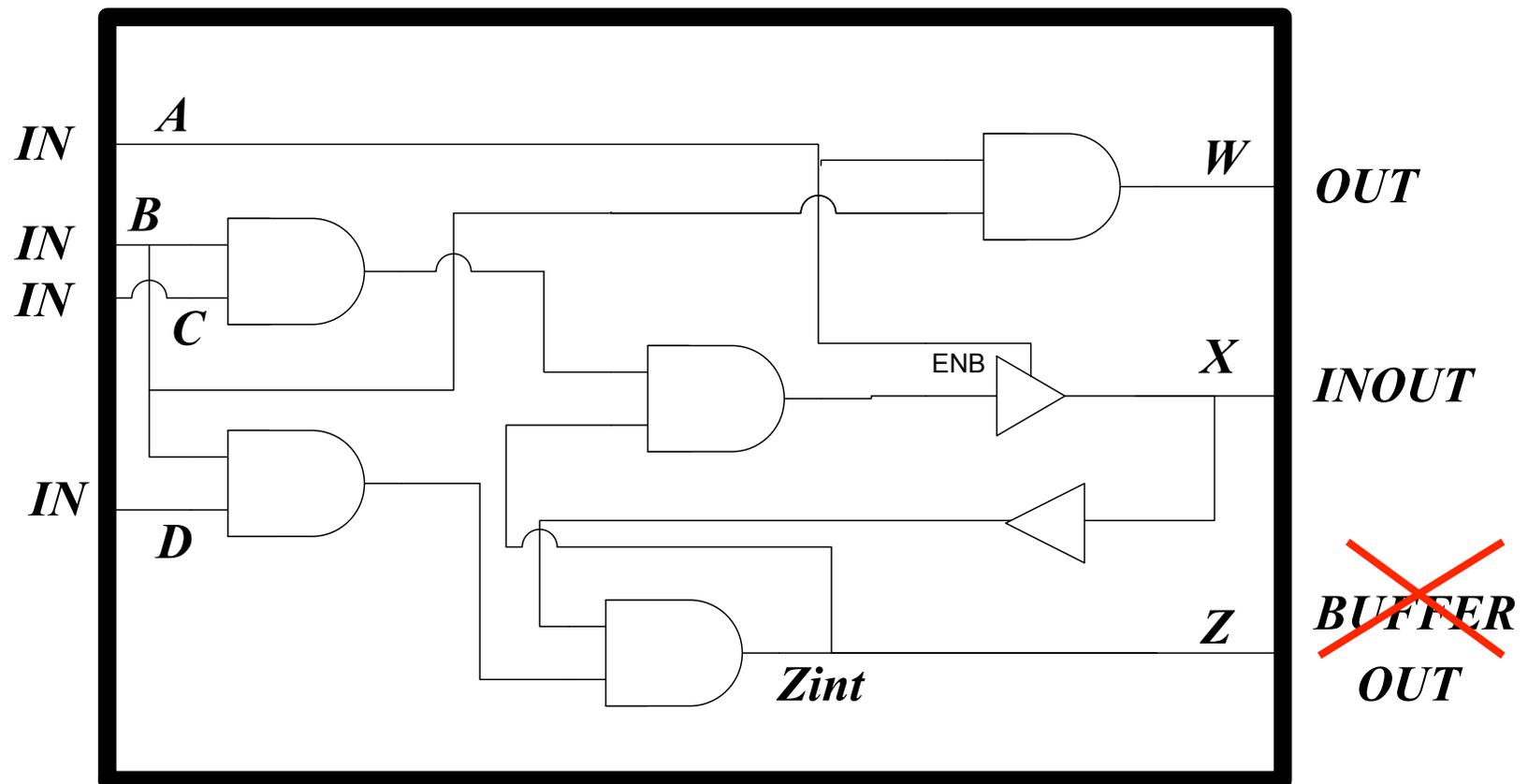
PORT (
 clk: **IN** STD_LOGIC;
 reset: **IN** STD_LOGIC;
 enable: **IN** STD_LOGIC;
 cuenta: **OUT** STD_LOGIC_VECTOR(N-1 **downto** 0)
);

La sección ENTITY. Declaración de puertos

- ◆ Un puerto debe tener:
 - Nombre
 - Dirección
 - » IN
 - » OUT
 - » INOUT
 - » BUFFER
 - Tipo de dato

La sección ENTITY. Declaración de puertos

◆ Ejercicio



```
Z <= Zint;
```

Clase 1:

Estructura de un diseño en VHDL

1.1 Introducción y ejemplo simple

1.2 La sección ENTITY

1.3 La sección LIBRARY

1.4 La sección ARCHITECTURE

1.5 La sección CONFIGURATION

Tipos de datos estándar

- En propio lenguaje VHDL define algunos tipos de datos estándar:
 - ↘ Todos los tipos de datos pueden **adoptar** unos **valores determinados**.
 - ↘ El usuario puede definir sus propios tipos.

boolean

false
true

real

1.02, 1.0e-10,
-37.4

bit

'0'
'1'

character

'2', 'A', 'r', ...

time

23ns, 2.2ps,
0fs

string

"cadena",
"1020",...

integer

1,256,0,-45,...

Tipos de datos estándar. Tipo *bit*

- Puede tomar **sólo** valores '0' y '1'.
- Para síntesis y simulación necesitamos otros posibles valores:
 - ↘ No inicializado
 - ↘ Alta impedancia
 - ↘ No definido
 - ↘ No importa
 - ↘ Valores débiles (pull-up).
- Para disponer de estos datos multievaluado necesitamos definir un nuevo *tipo de dato*.
- En la librería **IEEE 1164 (STD_LOGIC_1164)** se definen datos multievaluados.

IEEE Standard Logic 1164

- Las librerías en VHDL son conjuntos de:
 - ↘ Definiciones de tipos de datos.
 - ↘ Funciones aritméticas, de conversión y comparaciones.
- ⑤ Todos nuestros diseños comenzarán por:

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

- **Opcionalmente**

```
- use IEEE.numeric_std.all;
```

IEEE Standard Logic 1164

⑤ Paquete básico de la librería IEEE

⑤ **USE** IEEE.std_logic_1164.all

- std_logic
- std_logic_vector (MSB **downto** LSB)
- integer
- Lógica booleana (NOT, XOR, etc)

```
.....  
type STD_ULOGIC is (  
  `U ` ,    -- uninitialized  
  `X ` ,    -- strong 0 or 1 (= unknown)  
  `0 ` ,    -- strong 0  
  `1 ` ,    -- strong 1  
  `Z ` ,    -- high impedance  
  `W ` ,    -- weak 0 or 1 (= unknown)  
  `L ` ,    -- weak 0  
  `H ` ,    -- weak 1  
  - ` ,    -- don't care);  
.....
```

Dentro de la librería podemos encontrar la definición del tipo std_ulogic (std_logic es un caso particular de std_ulogic).

Operadores en VHDL y std_logic_vector_1164

Operador	Descripción	Tipo de dato de operandos	Tipo de datos de resultados
a ** b	Elevado a	Integer	Integer
a*b	multiplicación		
a/b	división		
a+b	Suma		
a-b	Resta		
a & b	Concatenación	1-D array	1-D array
a = b	igual	cualquiera	boolean
a /= b	distinto		
a < b	Menor que	Integer	boolean
a <= b	Menor o igual		
a > b	Mayor que		
a >= n	Mayor o igual		
not a	negación	Boolean, std_logic, std_logic_vector	El mismo tipo que el operando
a and b	and		
a or b	or		
a xor b	xor		

IEEE Standard Logic Numeric

```
-use IEEE.numeric_std.all;
```

- Introduce buses con significado numérico: **SIGNED** y **UNSIGNED**.
- Ejemplo: **"1001"**:
 - `STD_LOGIC_VECTOR(3 downto 0)`: Es simplemente grupo de bits sin significado numérico.
 - `SIGNED(3 downto 0)`: Un número en "complemento a 2" representa un **-7**
 - `UNSIGNED(3 downto 0)`: Un número sin signo, que representa un **9**.

IEEE Standard Logic Numeric: Sobrecarga de operadores

Operador	Descripción	Tipo de dato de operandos	Tipo de dato de resultado
a*b	Operadores aritméticos	Unsigned, natural, signed, integer	Unsigned, signed
a+b			
a-b			
a=b	Comparaciones	Unsigned, natural, signed, integer	boolean
a/=b			
a<b			
a<=b			
a>b			
a>=b			

Conversiones entre std_logic_vector y tipos numéricos

Tipo de dato origen (a)	Tipo de dato destino	Función de conversión
Unsigned, signed	Std_logic_vector	Std_logic_vector(a)
Signed, std_logic_vector	unsigned	Unsigned(a)
Unsigned, std_logic_vector	signed	Signed(a)
Unsigned, signed	integer	To_integer(a)
natural	unsigned	To_unsigned(a,size)
integer	signed	To_signed(a,size)

Ejemplo de asignaciones y operaciones aritméticas

```
signal std1: std_logic_vector(3 downto 0);  
Signal uns1: unsigned(3 downto 0);
```

```
uns1 <= std1;
```

ERROR: type mismatch

```
std1 <= uns1;
```

ERROR: type mismatch

```
uns1 <= 3;
```

ERROR: type mismatch

```
std1 <= 3;
```

ERROR: type mismatch

Ejemplo de asignaciones y operaciones aritméticas

```
signal std1: std_logic_vector(3 downto 0);  
Signal uns1: unsigned(3 downto 0);
```

```
uns1 <= unsigned(std1);
```

OK

```
std1 <= std_logic_vector(uns1);
```

OK

```
uns1 <= to_unsigned(3,4);
```

OK

```
std1 <= std_logic_vector(to_unsigned(3,4));
```

OK

Ejemplo de asignaciones y operaciones aritméticas

```
signal std1, std2, std3: std_logic_vector(3 downto 0);  
Signal uns1, uns2, uns3: unsigned(3 downto 0);
```

```
uns1 <= uns2+uns3;
```

OK. El mismo tipo

```
uns1 <= uns2+3;
```

OK: Sobrecarga de operadores

```
std1 <= std2+std3;
```

```
std1 <= std2+1;
```

ERROR: Para sumar hace falta un significado numérico

OK: Usamos funciones de conversión

```
std1 <= std_logic_vector(unsigned(std2)+unsigned(std3));
```

```
std1 <= std_logic_vector(unsigned(std2)+3);
```

Clase 1:

Estructura de un diseño en VHDL

1.1 Introducción y ejemplo simple

1.2 La sección ENTITY

1.3 La sección LIBRARY

1.4 La sección ARCHITECTURE

1.5 La sección CONFIGURATION

La sección ARCHITECTURE

```
architecture A of mux21 is
```

```
    Definiciones de señales
```

```
    Definiciones de componentes
```

```
Begin
```

```
    Órdenes concurrentes.
```

```
end A;
```

La sección ARCHITECTURE. Señal

- ◆ Una **señal** es un tipo de objeto que representa un cable, y por tanto interconecta componentes.
- ◆ Los **puertos** de la entidad son **señales** dentro de su arquitectura.
- ◆ Sintaxis:

```
signal nombre: tipo_de_señal;
```

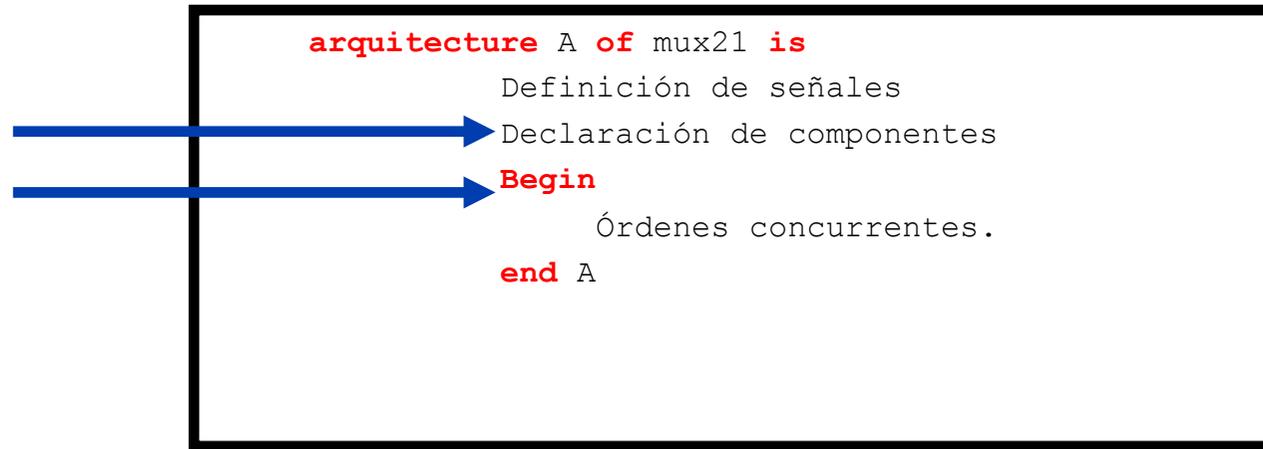
La sección ARCHITECTURE. Componente

- ◆ Componente: entidad de un orden inferior de jerarquía
- ◆ Sintáxis:

```
component nombre  
generic (  
    definición de generics);  
port (  
    definición de puertos);  
end component;
```

¡¡COPIA DE LA ENTITY!!

La sección ARCHITECTURE



- ◆ Una arquitectura siempre corresponde a una entidad asociada.
- ◆ Todas las órdenes dentro de la arquitectura se ejecutan de forma paralela (concurrente).
- ◆ Diferentes formas de escribir una arquitectura (Tema 3):
 - Describir una arquitectura en forma de interconexión de componentes
 - Describir el comportamiento
 - Mezclas

Clase 1:

Estructura de un diseño en VHDL

1.1 Introducción y ejemplo simple

1.2 La sección ENTITY

1.3 La sección LIBRARY

1.4 La sección ARCHITECTURE

1.5 La sección CONFIGURATION

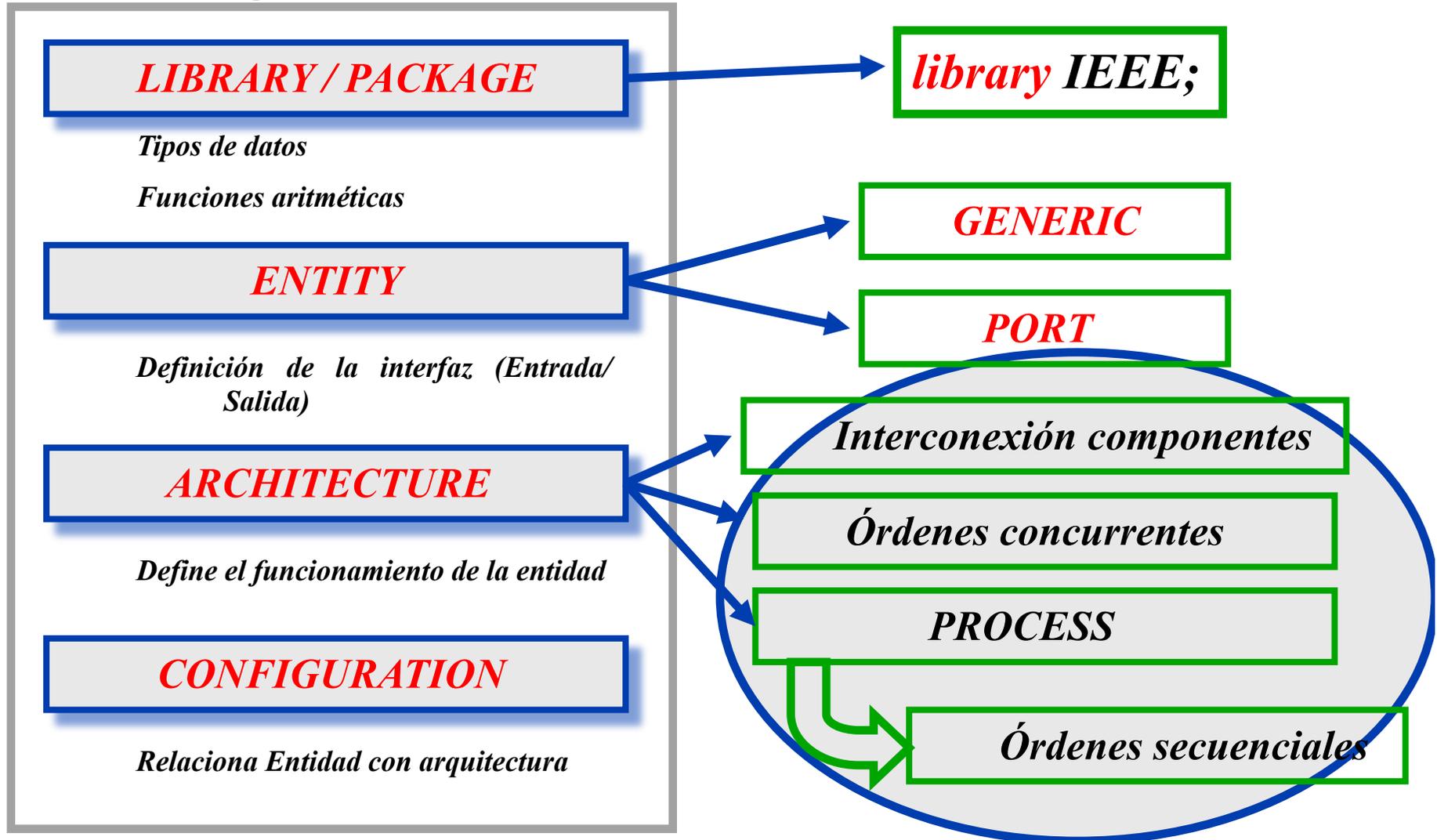
La sección CONFIGURATION

```
CONFIGURATION nombre_configuracion OF nombre_entidad IS  
  FOR nombre_arquitectura  
    FOR nombre_instancia : nombre_entidad  
      USE CONFIGURATION WORK.nombre_configuracion;  
    END FOR;  
    .....  
  END FOR;  
END nombre_configuracion;
```

- ◆ Mecanismo de asignación de una arquitectura a una entidad
- ◆ Una configuración está asignada a una entidad y una arquitectura
- ◆ Se pueden definir varias configuraciones diferentes con distintos nombres, de forma que en el nivel jerárquico superior se decida cual de ellas se utiliza.
- ◆ Normalmente solamente utilizado en software de simulación.

CONCLUSIONES

Estructura general VHDL



DEBERES

Escribir la descripción en VHDL, dejando en vacía la descripción de la arquitectura de los dos circuitos mostrados:

