TECNOLOGÍA DE COMPUTADORES

Tema 1

"Panorámica del lenguaje de descripción hardware VHDL" (1/2)

Agustín Álvarez Marquina

Introducción (I)

- **Causas del avance en el diseño de los sistemas digitales en los últimos años:**
  - Necesidad de reducir el tiempo de puesta en el mercado de un producto.
  - Abaratamiento en los costes de producción.
    - Tecnología.
    - Tiempo de diseño.

Introducción (II)

Jerarquía implícita en el diseño lógico:

- Un componente está compuesto de otros componentes de menos complejidad y así sucesivamente hasta llegar al elemento más básico considerado.
  - Ej. Transistor.
- Encapsulado.
  - Un CPU consta de diversas unidades funcionales: contador de programa (PC), pila, registros, ALU, etc.
  - Una ALU consta de un sumador, unidad lógica, acumulador, etc.
  - Un sumador consta de diversos bloques combinacionales. Ej. Multiplexor.
  - Un multiplexor consta de varias puertas lógicas. Ej. AND.

Introducción (III)

- **Ventajas de emplear herramientas de descripción hardware:**
  - Portabilidad de diseños entre diferentes herramientas.
  - Diseño independiente de la tecnología.

El lenguaje VHDL (I)

- **VHDL- VHSIC Hardware Description Language**
  - VHSIC- Very High Speed Integrated Circuits
- **Estándar IEEE 1076-1987.**
- **Desarrollado a partir del lenguaje ADA.**
- **Modelo de simulación por eventos.**
- **Metodología *top-down*.**

El lenguaje VHDL (II)

Historia:

- Agosto 1985. Aparece el primer borrador de VHDL presentado por Intermetrics, IBM y Texas Instruments.
- Diciembre 1987. Fue aprobado como estándar IEEE-1076-87 (VHDL-87).
  - Consolidación como herramienta de aplicación en la industria electrónica.
- Año 1993. Aprobación de una nueva versión del estándar con la incorporación de nuevas características y definición de nuevas librerías estándar.
  - VHDL-93.
- Año 2001. Última revisión (VHDL-2001).

Dominios de aplicación de VHDL (I)

- **Su principal dominio de aplicación es el modelado de dispositivos hardware.**
  - Objetivo: comprobación de su funcionalidad.
  - Es un lenguaje con una semántica orientada a la simulación.
- **Otros dominios:**
  - Detección de fallos.
  - Documentación.
  - **Simulación.**
  - **Síntesis** (generación automática de hardware).
  - Verificación formal.

Niveles de abstracción (I)

Grado de detalle

\- +

Niveles de abstracción	Ejemplo
Sistema	Descripción de un sistema y de sus prestaciones: especificaciones. Ejemplo: computadores, impresoras, unidades de disco.
Chip	Circuito que cumple con las especificaciones del sistema. Ejemplo: microprocesadores, memorias, puertos de E/S, etc.
Transferencia entre registros	Descripción de los flujos de datos. Ejemplo: conjunto de registros y buses de interconexión.
Lógica	Tabla de verdad y ecuaciones lógicas que describe un sistema. Implementación a través de bloques combinacionales y secuenciales.
Circuito eléctrico	Componentes eléctricos (transistores, resistencias, condensadores, etc.).
Físico	Descripción geométrica de elementos y compuesto químicos. Ejemplo: máscaras con metal, policristalino, área activa. En este nivel no puede expresarse comportamiento.

Niveles de abstracción (II)

- **Niveles de diseño cubiertos por VHDL:**
  - Sistema.
    - Solamente una pequeña parte.
  - Chip.
  - Transferencia entre registros.
  - Circuito lógico.
  - Circuito eléctrico.
    - No cubre la totalidad.

Niveles de abstracción (III)

- **Una de las características más destacadas de VHDL es que permite mezclar en una misma descripción diferentes niveles de abstracción.**

- **El tratamiento concurrente del lenguaje permite realizar descomposiciones jerárquicas.**
  - Modelos estructurales en los que los componentes pueden ser a su vez descritos a partir de otros modelos estructurales más sencillos.

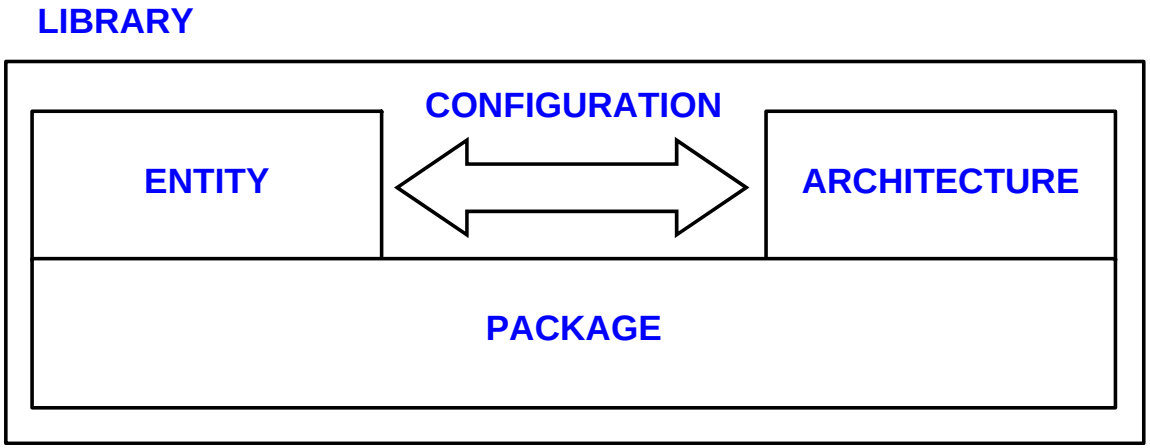
Jerarquías en VHDL (I)

- **Un diseño en VHDL consiste en una jerarquía de componentes compilados.**
  - Los componentes están implementados por medio de arquitecturas, siguiendo uno de los tres estilos de descripción:
    - Comportamiento.
    - Flujo de datos.
    - Estructural.
- **Una vez que los diseños están compilados, se ubican en una biblioteca y pueden utilizarse como componentes en los siguientes diseños.**

Jerarquías en VHDL (II)

- Los componentes tienen que ser declarados antes de realizar una instancia de ellos desde la biblioteca.

- Un diseño que contenga componentes requiere establecer una configuración que asocie entidades con su arquitectura correspondiente.



Entidades (I)

- **Definen el nombre de un componente y su interfaz de entrada-salida.**
  - Modelo de caja negra.
  - Un componente pude tener la complejidad que se desee.
    - Ejemplos: sumador, ALU, computador.
  - No pueden existir dos entidades con el mismo nombre.
  - La funcionalidad de la entidad se define a través de la arquitectura.
    - Una entidad puede tener asociada más de una arquitectura.

Entidades (II)

- **Formato de las entidades:**

```
ENTITY nombreEntidad IS
   [GENERICS]
   [PORTS]
   [declaraciones de constantes, variables, señales…]
   [BEGIN
      [sentencias…]
    END;]
END nombreEntidad;
```

- Las palabras reservadas aparecen en negrita y mayúsculas.
- Los corchetes indican sentencias opcionales.

Opciones de las entidades (I)

GENERIC(nombre: tipoDato**:=** [valorInicial]**);**

- **Pasa un dato específico a un componente.**
  - Muy útil para realizar código reusable.

```
ENTITY puerta_xor IS
   GENERIC(retardo: TIME:= 1NS);
   PORT(a,b:IN BIT; z:OUT BIT);
END puerta_xor;

ARCHITECTURE general OF puerta_xor IS
BEGIN
   z<= NOT (a XOR b) AFTER retardo;
END general;
```

Opciones de las entidades (II)

PORT(nombre**:** nombre_E/S tipoDato**);**

- **Con *nombre* nos referimos a una señal.**
  - Es conveniente que esté relacionado con la función que realizan, número de entradas, etc.
- **Con *nombre_E/S* nos referimos al modo que presenta la señal:**
  - **IN**: entrada.
  - **OUT**: salida.
  - **INOUT**: entrada y salida.
  - **BUFFER**: salida con realimentación interna.

Opciones de las entidades (III)

- **Con *tipoDato* nos referimos a uno de los tipos de datos predefinidos.**
  - Paquete estándar (por defecto):
    - **BOOLEAN**: *true*, *false*.
    - **CHARACTER**.
    - **TIME**.
    - **INTEGER**.
    - **REAL**.
    - **BIT**: *'0'*, *'1'*.
  - Otros paquetes (hay que declararlos previamente):

```
LIBRARY ieee
USE ieee.std_logic_1164.all;
```

Opciones de las entidades (IV)

- **Todas las entidades incluyen la sentencia *PORT* con la sola excepción de las entidades empleadas para test.**

```
ENTITY puerta_xor IS
   GENERIC(retardo: TIME:= 1NS);
   PORT(a,b:IN BIT; z:OUT BIT);
END puerta_xor;

ARCHITECTURE simple OF puerta_xor IS
BEGIN
   z<= NOT (a XOR b) AFTER retardo;
END simple;
```

Opciones de las entidades (V)

```
declaraciones de constantes, variables, señales…
  BEGIN
    [sentencias…]
  END;
```

- **Estas opciones se usan poco.**
  - ***Declaraciones***. Esta información se incluye normalmente en la arquitectura. Un uso consiste en definir aquí objetos comunes asociados a todas las arquitecturas.
  - ***Sentencias***. No pueden afectar a la funcionalidad del dispositivo (son pasivas).
    - Ejemplos: configuraciones prohibidas, violaciones de tiempo, numero de pines, etc.

Arquitecturas (I)

- Una arquitectura define la funcionalidad de la entidad a la que está asociada.

- Describen las operaciones que se efectúan sobre las entradas de la entidad y que determinan el valor de sus salidas en cada momento.

Arquitecturas (II)

Formato de las arquitecturas (II):

ARCHITECTURE nombreArquitectura **OF** entidadAsociada **IS**
    [constantes, variables, señales intermedias de interconexión,
        tipos, componentes…]
BEGIN
    [*PROCESS* sentencias secuenciales]
    [asignación concurrente de señales]
    [Llamada concurrente a un procedimiento]
    [Instancias de componentes]
    [*ASSERT*]
    [*GENERATE*]
    [*BLOCK*]
END nombreArquitectura**;**

Arquitecturas: estilos de descripción (I)

- **Independientemente del nivel de abstracción en el que nos encontremos la descripción del modelo se puede realizar siguiendo los siguientes estilos:**
  1. **Comportamiento** (Behavioral).
  2. **Flujo de datos** o **transferencia entre registros** (Data Flow).
  3. **Estructural** (Structural).
- **En VHDL un sistema puede mezclar diferentes estilos a la hora de describir sus diferentes componentes.**

Arquitecturas: estilos de descripción (II)

❶ Comportamiento (Behavioral).

- En este estilo de descripción se modela la funcionalidad del componente por medio de los recursos algorítmicos del lenguaje.
- Es decir, se describe el algoritmo que refleja el comportamiento de dicho componente.
- A veces, se llama también a este estilo algorítmico o secuencial.

Arquitecturas: estilos de descripción (III)

- **Flujo de datos o transferencia entre registros (Data Flow).**
  - En este estilo de descripción se especifican los flujos de datos del sistema y la interconexión entre sus componentes.
  - El proceso de descripción se realiza por medio de un de funciones lógicas, que se ejecutarán de forma concurrente.

Arquitecturas: estilos de descripción (IV)

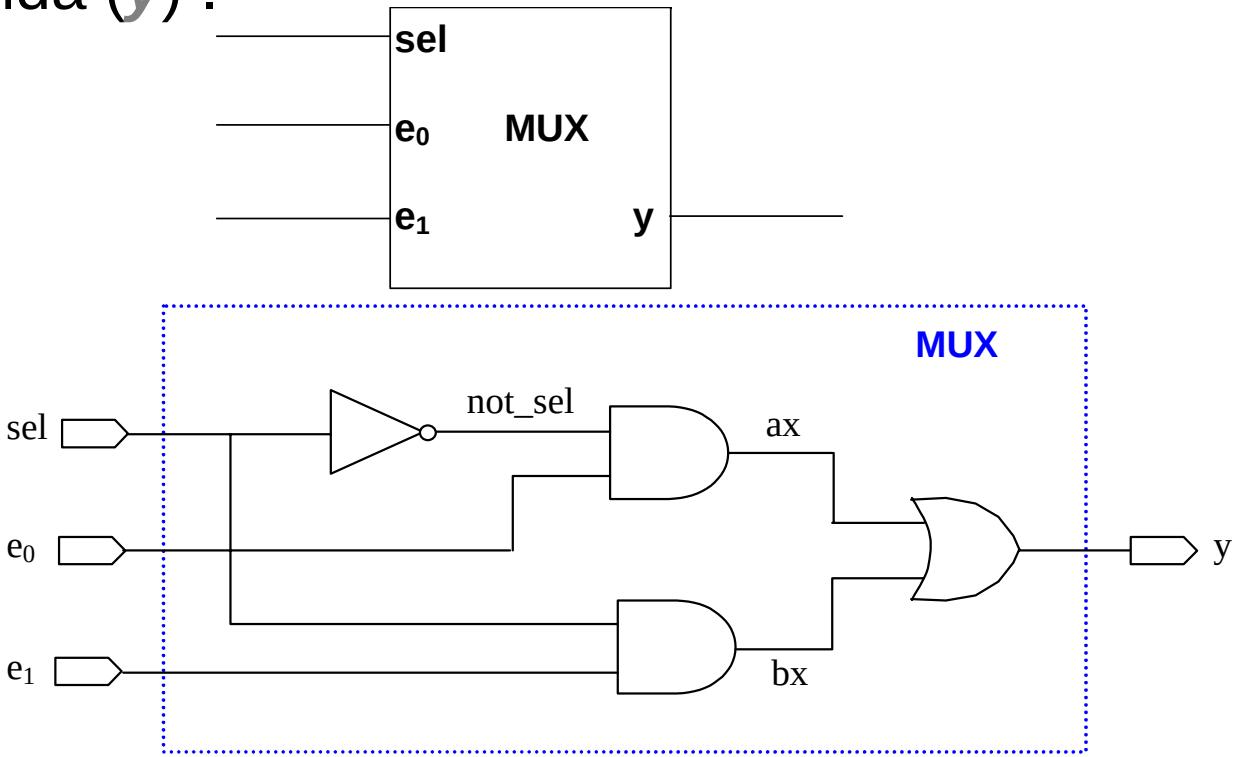
❸ Estructural (Structural).

- En este estilo de descripción se definen o instancian todas las partes del sistema y sus interconexiones.
- Resulta muy útil cuando se quiere aprovechar diseños compilados con anterioridad y que se encuentran almacenados en bibliotecas de componentes.

Ejemplo de estilos de descripción (I)

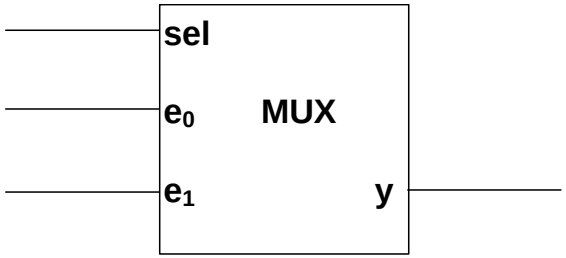
Ejemplo:

- Describir en VHDL un multiplexor de dos entradas de datos (e_0 y e_1), una señal de control (*sel*) y una línea de salida (*y*) .



Ejemplo de estilos de descripción (II)

- **Con independencia del estilo de descripción empleado, deben definirse las entradas y salidas del sistema.**
  - ➔ Declaración de la entidad (entity):




```
-- Los comentarios empiezan con dos guiones
ENTITY mux IS
PORT ( e0:  IN BIT;
       e1:  IN BIT;
       sel: IN BIT;
       y:   OUT BIT);
END mux;
```

Ejemplo de estilos de descripción (III)

- **Descripción de comportamiento o algorítmica:**

```
ARCHITECTURE comportamiento OF mux IS
BEGIN
     PROCESS(e0,e1,sel)
     BEGIN
          IF (sel='0') THEN
               y<= e0;
          ELSE
               y<= e1;
          END IF;
     END PROCESS;
END comportamiento;
```

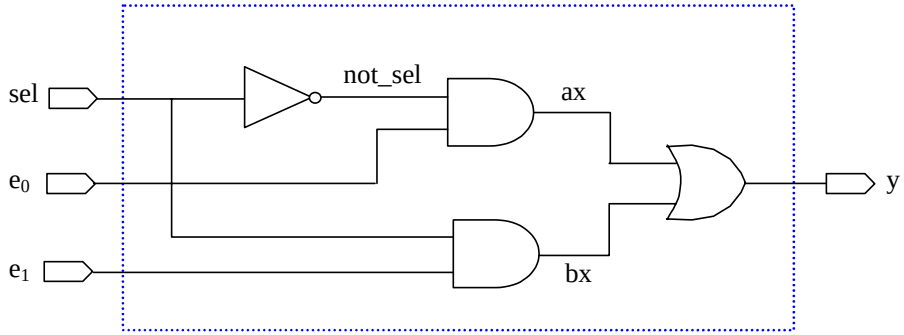
Ejemplo de estilos de descripción (IV)

Descripción de flujo de datos:

```
ARCHITECTURE flujo1 OF mux IS
BEGIN
  y<= (e0 AND (NOT sel)) OR (e1 AND sel);
END flujo1;
```

```
ARCHITECTURE flujo2 OF mux IS
     SIGNAL not_sel, ax, bx: BIT;
BEGIN
  not_sel<= NOT sel;
  ax<= e0 AND not_sel;
  bx<= e1 AND sel;
  y<= ax OR bx;
END flujo2;
```

- Dos soluciones diferentes con estilo de flujo de datos.



Ejemplo de estilos de descripción (V)

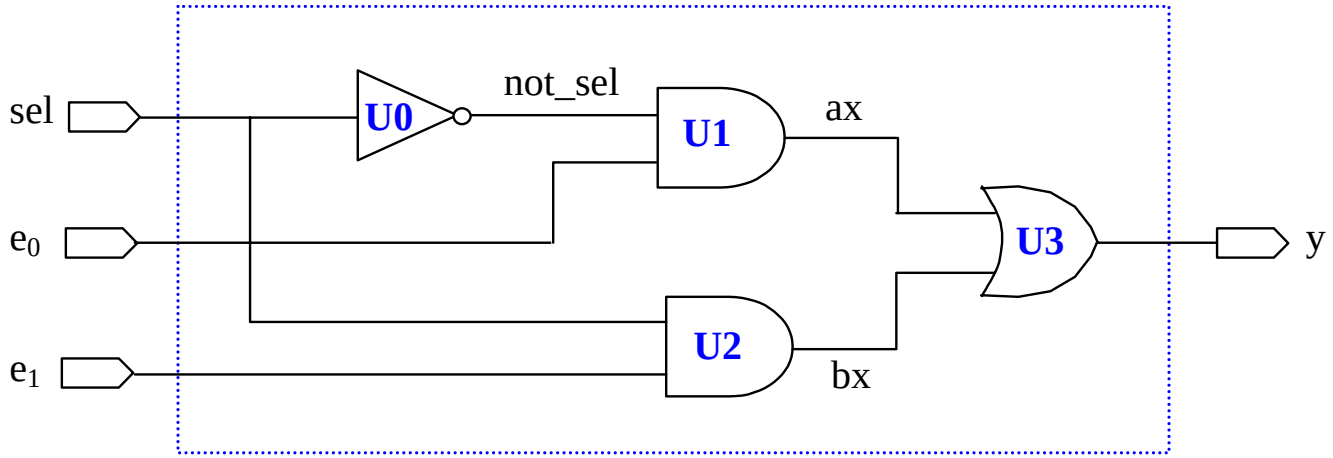
Descripción estructural:

```
ARCHITECTURE estructural OF mux IS
     COMPONENT inv
          PORT (e: IN BIT; y: OUT BIT);
     END COMPONENT;

     COMPONENT and2
          PORT (e1,e2: IN BIT; y: OUT BIT);
     END COMPONENT;

     COMPONENT or2
          PORT (e1,e2:IN BIT; y:OUT BIT);
     END COMPONENT;
```

•Sigue en la siguiente transparencia.

Ejemplo de estilos de descripción (VI)

```
    SIGNAL ax,bx,not_sel: BIT;

BEGIN
    U0: inv  PORT MAP(e=>sel, y=>not_sel);
    U1: and2 PORT MAP(e1=>e0, e2=>not_sel, y=>ax);
    U2: and2 PORT MAP(e1=>e1, e2=>sel, y=>bx);
    U3: or2  PORT MAP(e1=>ax, e2=>bx, y=>y);
END estructural;
```



Configuraciones (I)

- **Permiten asociar una arquitectura específica a una entidad.**

- **Pueden aparecer de forma implícita dentro de la propia arquitectura.**

```
FOR etiqueta: nombreComponente USE
  ENTITY WORK.nombreEntidad(nombreArquitectura);
```

- **Cuando no se especifica la configuración, las herramientas suelen tomar por defecto la última que se ha compilado.**

Configuraciones (II)

- **Formato de las configuraciones:**

```
CONFIGURATION nombreConfiguración OF nombreEntidad IS
    [USE nombreBiblioteca]
    FOR identificadorArquitectura USE
        FOR identificadorComponente1 USE
            ENTITY
                WORK.nombreEntidad_1(nombreArquitectura_1);
        END FOR;

        FOR identificadorComponente_N USE
            ENTITY
                WORK.nombreEntidad_N(nombreArquitectura_N);
        END FOR;
    END FOR;
END nombreConfiguración;
```

Configuraciones (III)

```
CONFIGURATION configuracionTestMiSistema OF
  entidadTestMiSistema IS
   FOR arquitecturaMiSistema
       FOR U0: reloj USE
          ENTITY WORK.reloj(arquitecturaReloj);
       END FOR;
       FOR U1: miSistema USE
          ENTITY WORK.miSistema(unaArquitectura);
       END FOR;
   END FOR;
END configuracionTestMiSistema;
```

Paquetes (I)

- **Un paquete es una colección de definiciones, tipos de datos, subprogramas, constantes, etc. que se agrupan y se hacen visibles a otros diseños.**
  - Declaración.
    - Es un “repositorio” para almacenar declaraciones que pueden ser accesibles globalmente a múltiples unidades de diseño.

```
PACKAGE nombrePaquete IS
    Declaración de subprogramas
    Componentes, tipos, subtipos, constantes, variables, USE
END nombrePaquete;
```

Paquetes (II)

- Body.
  - Si la declaración contiene subprogramas (funciones y procedimientos) han de especificarse aquí.

```
PACKAGE BODY nombrePaquete IS
    Declaración de subprogramas
    Cuerpo del subprograma
    Declaración tipos, subtipos, constantes
    USE
END nombrePaquete;
```

Paquetes (III)

```
-- Ejemplo de paquete para constante diferida
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;

-- Definición
PACKAGE masaAlimentacion IS
   CONSTANT gnd: STD_LOGIC_VECTOR(3 DOWNTO 0);
   CONSTANT vdd: STD_LOGIC_VECTOR(3 DOWNTO 0);
END masaAlimentacion;

-- Cuerpo del paquete
PACKAGE BODY masaAlimentacion IS
   CONSTANT gnd: STD_LOGIC_VECTOR:= “0000”;
   CONSTANT vdd: STD_LOGIC_VECTOR:= “1111”;
END masaAlimentacion;
```

Paquetes (IV)

```
-- Para utilizar elementos del paquete en otros
-- diseños
USE WORK.masaAlimentacion.all;
USE ieee.STD_LOGIC_1164.all;
```