LENGUAJES DE DESCRIPCIÓN HARDWARE

VHDL

VIOLETA HIDALGO IZQUIERDO- UNIVERSIDAD DE EXTREMADURA

Guía para comenzar a describir modelos de circuitos en lenguaje VHDL:

VHDL es cada vez más utilizado en la industria y el empeño por completar su conocimiento en profundidad es un estupendo remate en la formación profesional de los alumnos de Ingeniería.

El lenguaje VHDL nació para dar respuesta a numerosos problemas planteados en el desarrollo y documentación de hardware digital. La documentación requerida para describir un sistema electrónico puede ocupar miles de páginas y suele ser muy costoso remplazar la información contenida cuando la tecnología o las especificaciones cambian. Un lenguaje de descripción adecuado resuelve el problema ya que la "documentación" es ejecutable.

VHDL

Las siglas VHDL corresponden a **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage.

El primer borrador vio la luz en agosto de 1985 diseñado por Intermetrics, IBM y Texas Instruments e impulsado por el Departamento de Defensa de los Estados Unidos. En diciembre de 1987 fue aprobado como estándar del IEEE y posteriormente, en 1993, fue revisado y registrado como norma IEEE Std 1076-1993.

VHDL es un lenguaje de semántica orientada a la simulación y por ello su principal dominio de aplicación es el modelado de dispositivos hardware para comprobar su correcto funcionamiento.

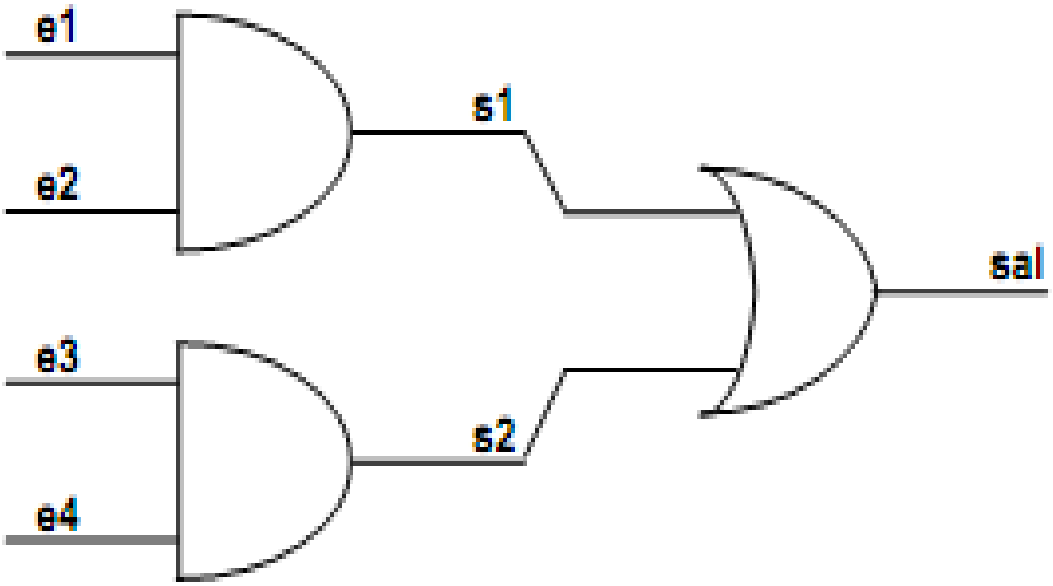
MODOS DE DESCRIPCIÓN DE CIRCUITOS LÓGICOS.

El lenguaje de descripción de hardware VHDL cuenta con diferentes modos de llevar a cabo la descripción.

Normalmente su aprendizaje se comienza desde la perspectiva del diseñador tradicional de hardware, utilizando las construcciones del lenguaje que permiten una descripción estructural desde las puertas lógicas hacia arriba.

Este enfoque resalta la correspondencia existente entre la realidad y el lenguaje pero oculta la verdadera potencia del modelo temporal soportado por el lenguaje. La descripción comportamental y la ejecución concurrente de procesos manifiestan la verdadera potencia del VHDL.

Seguidamente vamos a caracterizar cada uno de los modos de descripción basándonos en un ejemplo. Sea el circuito lógico de la figura siguiente:

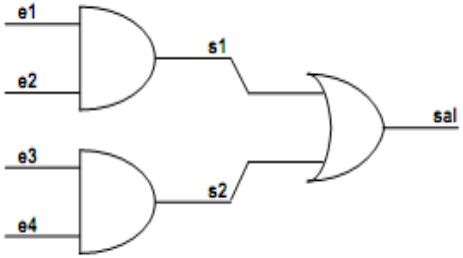


Circuito lógico

[1]

Podemos describirlo indicando la función que realiza, es decir, especificando su comportamiento:

sal = '1' si (e1 = e2 = '1') o si (e3 = e4 = '1');

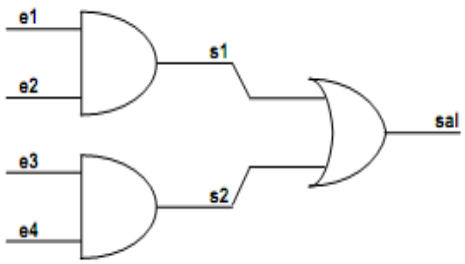


Circuito lógico

[2]

Podemos describirlo también especificando los componentes más básicos que lo forman y como se conectan entre si, es decir, podemos dar su estructura:

AND (e1, e2, s1);
AND (e3, e4, s2);
OR (s1, s2, sal);

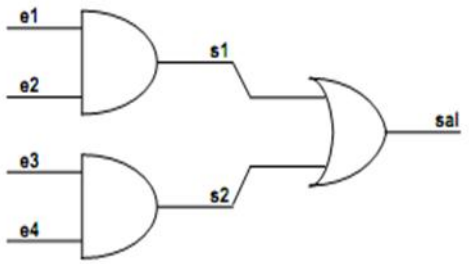


Circuito lógico

[3]

Finalmente, podemos proporcionar la relación entre unas señales y otras, es decir, indicar el flujo de señales:

```
sal = s1ORs2;
s1  = e1 AND e2;
s2  = e3 AND e4;
```



Circuito lógico

Distintos modos de descripción

Especificando su comportamiento:

sal = '1' si (e1 = e2 = '1') o si (e3 = e4 = '1'); [1]

Podemos dar su estructura:

```
AND (e1, e2, s1);
AND (e3, e4, s2);                [2]
OR (s1, s2, sal);
```

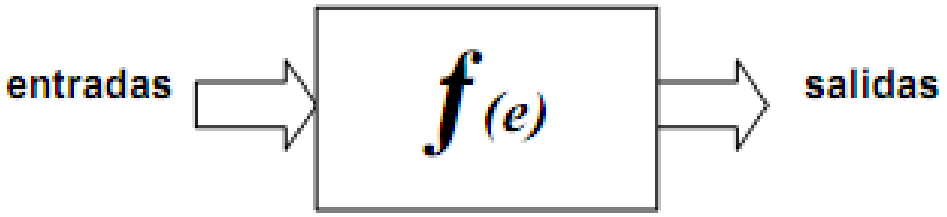
Indicando el flujo de señales:

```
sal = s1ORs2;
s1  = e1 AND e2;                 [3]
s2  = e3 AND e4;
```

El lenguaje de descripción hardware VHDL permite estos tres modos de descripción:

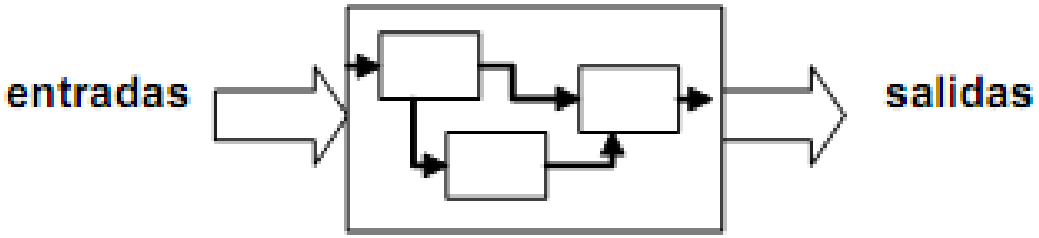
- comportamental (ecuación lógica 1)
- estructural (ecuación lógica 2)
- flujo de datos o RTL (ecuación lógica 3)

[1]: Modelado comportamental **lo importante** es la **función que relaciona la salida con la entrada**



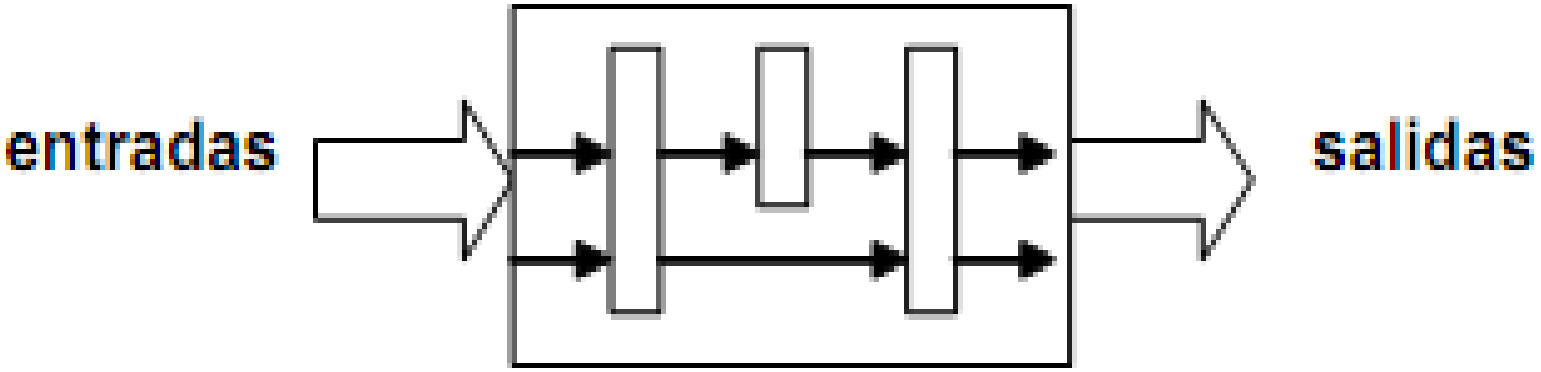
Esquema del modelado comportamental

[2]: En el modelado estructural se especifican los bloques que componen un circuito y sus interconexiones. Cada bloque integrante debe contar con su descripción previa de manera que se construye una jerarquía de descripciones donde las inferiores dan lugar a superiores más complejas y así sucesivamente.



Esquema del modelo estructural

[3]: En el modelado de flujo de datos o RTL (Register Transfer Logic) se declara la sucesión temporal con la que evolucionan las diferentes señales del modelo descrito.

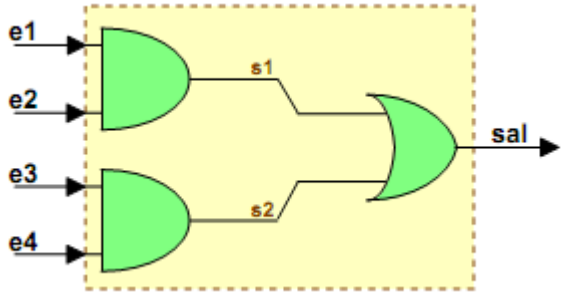


Modelo RTL

NOTA: Podemos modelar un circuito en cualquiera de los modos de descripción vistos.
Cada uno de los modos de descripción lleva asociada una sintaxis determinada que lo caracteriza.

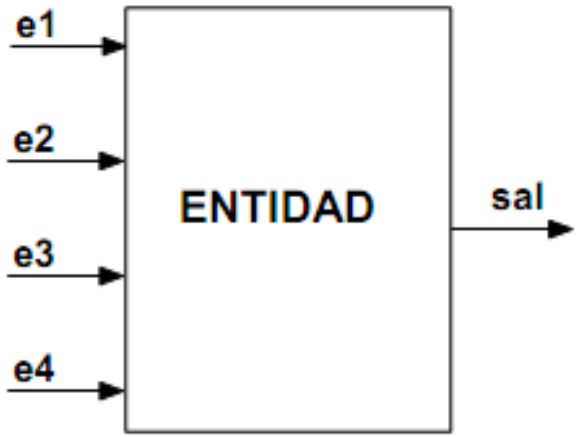
ENTIDAD Y ARQUITECTURA DE UN MODELO:

El circuito del ejemplo anterior puede ser visto como una caja negra que se relaciona con el exterior mediante un conjunto se señales, unas de entrada y otras de salida.



Circuito lógico (bloque encuadrado con una línea punteada que encierra los componentes) y su relación con el exterior

El bloque se conoce en VHDL como ENTIDAD y se declara en primer lugar. Una ENTIDAD indica las señales que entran al circuito y las que salen, es decir, declara la relación del circuito con el mundo exterior.



A cada ENTIDAD le corresponde al menos una descripción (comportamental, estructural o RTL) –aunque puede tener múltiples– que se especifica en su ARQUITECTURA.

Cada fichero fuente (modo texto) debe contener una declaración de entidad y, al menos, una arquitectura. Si modelamos más arquitecturas todas ellas pueden residir en el mismo fichero fuente (con extensión vhd para nuestro simulador.).

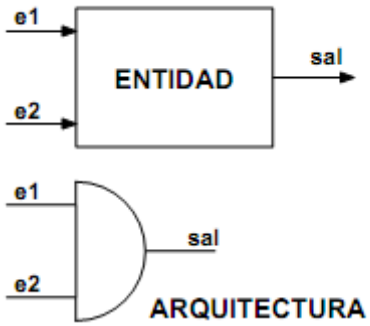
EJEMPLOS DE DESCRIPCIÓN COMPORTAMENTAL VHDL.

Vamos a empezar con una puerta AND de dos entradas sin retardo ye iremos introduciendo elementos nuevos del lenguaje que completen la descripción del circuito.

La descripción comportamental es la única disponible al más bajo nivel de modelado de circuitos digitales (puertas lógicas) ya que no es posible fundamentarla en bloques más básicos que ya serían los transistores

Modelo 1. Puerta AND sin retardo.

```
ENTITY and2 IS
     PORT (e1, e2: IN BIT; sal: OUT BIT);
END and2;
ARCHITECTURE comportamiento OF and2 IS
BEGIN
    PROCESS (e1, e2)
    BEGIN
        sal <= e1 AND e2;
    END PROCESS;
END comportamiento;
```

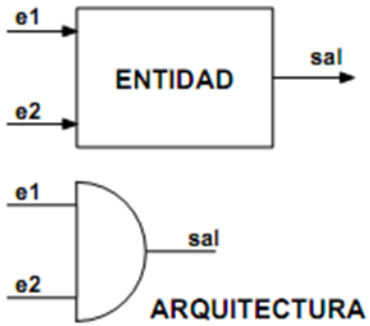


Elementos del modelo

VHDL

Es una descripción COMPORTAMENTAL donde la ARQUITECTURA declara simplemente la función que relaciona salida con entradas. Dicha arquitectura contiene un único proceso. El PROCESO cuenta con una sola sentencia de asignación de señal que corresponde a la ecuación lógica de la puerta AND. El proceso se despierta con la variación de las señales e1 o e2 enumeradas entre paréntesis.

```
ENTITY and2 IS
     PORT (e1, e2: IN BIT; sal: OUT BIT);
END and2;
ARCHITECTURE comportamiento OF and2 IS
BEGIN
    PROCESS (e1, e2)
    BEGIN
        sal <= e1 AND e2;
    END PROCESS;
END comportamiento;
```



Elementos del modelo

NOTA

Veremos la SINTAXIS DE VHDL, para más adelante continuar con los distintos tipos de descripción de circuitos

SINTAXIS DE VHDL: Elementos sintácticos

- **Comentarios**: Siguen a 2 guiones “--”
- **Identificadores**:
  - Identifican variables, señales, rutinas…
  - Mayúsculas y minúsculas: **no** se distinguen
  - Nombre compuesto incluyendo “_”
- **Números**:
  - Número: se considera que está en base 10
  - Cualquier otra base: base#número#
    - Ejemplo: 2#11000100# y 16#C4#
- **Caracteres**: Cualquier letra o carácter entre comillas simples: ‘c’
- **Cadenas**: Conjunto de caracteres entre comillas dobles: “hola”
- **Cadenas de bits**: Los tipos “bit” y “bit_vector” son en realidad de tipo carácter y matriz de caracteres, respectivamente. Se pueden definir números con estos tipos mediante la cadena de bits. Dependiendo de la base en que se especifique el número (B binario, O octal, X hexadecimal). Ejemplo: B”11101001”, O”126”, X”FE”.

SINTAXIS DE VHDL: Operadores y expresiones

- **Operadores varios**:
  - & - Concatenación: Concatena vectores. Dimensión del vector resultante = suma de las dimensiones de los vectores sobre los que opera: vector_largo<=x&y
- **Operadores aritméticos**:
  - ** - Exponencial: Elevar un nº a una potencia (entera): 4**2
  - ABS() – Valor absoluto
  - * - Multiplicación
  - / - División
  - MOD – Operación módulo: a MOD b a=b*n + a MOD b; (nЄZ)
    - Resultado: con valor absoluto menor que b y con el signo de b
  - REM – Resto de la división entera: a REM b a=[INT(a/b)]*b + a REM b
    - Resultado: con valor absoluto menor que b y con el signo de a
  - + Suma o signo positivo
  - - Resta o signo negativo

SINTAXIS DE VHDL: Operadores y expresiones

- **Operadores de desplazamiento**:
  - SLL, SRL – Desplazamiento lógico a izquierda y derecha
  - SLA, SRA – Desplazamiento aritmético a izquierda y derecha. Conserva el signo (el valor que tuviera el bit más significativo)
  - ROL, ROR – Rotación a izquierda y derecha
- **Operadores relacionales**:
  - Devuelven un valor booleano (true o false)
  - =, /= Igual a, distinto de
  - <, <=, >, >=
- **Operadores lógicos**:
  - NOT, AND, NAND, OR, NOR, XOR, XNOR

SINTAXIS DE VHDL: Operadores y expresiones

- **Precedencia de operadores**, de mayor a menor precedencia:

1. **, ABS, NOT
2. *, /, MOD, REM
3. +(signo), -(signo)
4. +, -, &
5. =, /=, <, <=, >, >=
6. AND, OR, NAND, NOR,XOR,XNOR

- Se admiten paréntesis para cambiar la precedencia normal

SINTAXIS DE VHDL: Tipos de datos

- **Tipos escalares**:
  - **Enteros**. Se definen mediante “RANGE”:
    - TYPE byte IS RANGE 0 TO 255;
    - TYPE indice IS RANGE 7 DOWNTO 1;
    - TYPE integer IS RANGE -2147483648 TO 2147483647; -- Predefinido
  - **Reales** (coma flotante). Se definen mediante “RANGE” con límites reales:
    - TYPE nivel IS RANGE 0.0 TO 5.0;
    - TYPE real IS RANGE -1.0E38 TO 1.0E38; --Predefinido
  - **Enumerados:** Pueden tomar cualquier valor especificado en un conjunto finito o lista:
    - TYPE nivel_logico IS (no_se, alto, bajo, Z);
    - TYPE bit IS ('0','1'); --Predefinido

SINTAXIS DE VHDL: Tipos de datos

- **Físicos**. Se corresponden con magnitudes físicas. Tienen un valor y unas unidades. Cualquier dato físico se escribe siempre con su valor seguido de la unidad: 10 mm, 1 µm, 23 ns.
  - TYPE longitud IS RANGE 0 TO 1.0E9

    UNITS

    µm;

    mm=1000 µm;

    m=1000 mm;

    inch=25.4 mm;

    END UNITS;
  - time: tipo predefinido en VHDL
    - Se usa para indicar retrasos
    - Tiene todos los submúltiplos, desde fs (femtosegundos) hasta hr (horas)

SINTAXIS DE VHDL: Tipos de datos

- **Tipos compuestos**:
  - **Matrices**: colección de elementos del mismo tipo a los que se accede mediante un índice. Monodimensionales (un índice) o multidimensionales (varios índices). El rango puede ser libre, especificándose más tarde en la declaración de cada dato.
  - Ejemplos:
    - TYPE word IS ARRAY(31 DOWNTO 0) OF bit;
    - TYPE vector_2 IS ARRAY(1 TO 4, 1 TO 4) OF real;
    - TYPE string IS ARRAY(positive RANGE <>) OF character; -- Predefinido
    - TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit; -- Predefinido
    - TYPE vector IS ARRAY (integer RANGE <>) OF real;
  - Se accede a los elementos de una matriz mediante su índice, siendo válido un rango: dato(3), datobyte <= datoword(2 TO 9)

SINTAXIS DE VHDL: Tipos de datos

- **Registros**: equivalente al tipo registro (record) de otros lenguajes:

  - TYPE trabajador IS

    RECORD

    nombre: string(1 TO 50);

    edad: integer;

    END RECORD;

  - Para referirse a un elemento dentro del registro se utiliza la misma nomenclatura que en Pascal, se usa un punto entre el nombre del registro y el nombre del campo: persona.nombre="Sara"

SINTAXIS DE VHDL: Tipos de datos

- **Subtipos de datos**: pueden definirse subtipos de datos que son restricciones o subconjuntos de tipos existentes:

  - Subtipos obtenidos a partir de la restricción de un tipo escalar a un rango:
    - SUBTYPE raro IS integer RANGE 4 TO 7;
    - SUBTYPE digitos IS character RANGE '0' TO '9';
    - SUBTYPE natural IS integer RANGE 0 TO entero_mayor; -- Predefinido
    - SUBTYPE positive IS integer RANGE 1 TO entero_mayor; -- Predefinido

  - Subtipos que restringen el rango de una matriz:
    - SUBTYPE id IS string(1 TO 20);
    - SUBTYPE word IS bit_vector(31 DOWNTO 0);

SINTAXIS DE VHDL: Declaración de objetos

- **Constantes**: se inicializan a un determinado valor que conservan inalterado (no puede cambiarse una vez inicializado):
  - CONSTANT e: real := 2.71828;
  - CONSTANT retraso: time := 10 ns;
  - CONSTANT tamano_max: natural; ⟶ Se permite declarar una constante sin especificar su valor siempre y cuando este valor sea declarado en otro sitio (se hace así para las declaraciones en paquetes que se verán más adelante)
- **Variables**: su valor puede ser alterado en cualquier instante. Se les puede asignar un valor inicial:
  - VARIABLE contador: natural := 0;
  - VARIABLE aux: bit_vector(31 DOWNTO 0);
  - Instrucción ALIAS: permite asignar un nombre adicional a un objeto previamente definido o ponerle nombre a una parte:
    - VARIABLE instruccion: bit_vector(31 DOWNTO 0);
    - ALIAS codigo_op: bit_vector(7 DOWNTO 0) IS instruccion(31 DOWNTO 24);

SINTAXIS DE VHDL: Declaración de objetos

- **Señales**: representan conexiones reales en el circuito.
  - Se declaran igual que las constantes y variables.
  - Pueden ser de varios tipos:
    - **normal**: si no se especifica el tipo.
    - **register**: debe especificarse con la palabra clave REGISTER.
    - **bus**: debe especificarse con la palabra clave BUS.
  - Al igual que en constantes y variables, a las señales también se les puede asignar un valor inicial.
  - Ejemplos:
    - SIGNAL selec: bit := '0';
    - SIGNAL datos: bit_vector(7 DOWNTO 0) BUS := B"00000000";

SINTAXIS DE VHDL: Declaración de objetos

- **Diferencias entre los distintos tipos de objetos**:
  - Constantes, variables y señales son cosas diferentes.
  - **Variables**: sólo tienen sentido dentro de un bloque (PROCESS) o un subprograma, es decir, en entornos de programación donde las sentencias son ejecutadas en serie. Las variables sólo se declaran en los procesos o subprogramas.
  - **Señales**:
    - Las señales pueden ser declaradas únicamente en las arquitecturas, paquetes (PACKAGE) o en los bloques concurrentes (BLOCK).
    - Las señales poseen un significado físico inmediato, y es el de representar las conexiones reales entre los elementos de un circuito. Las entradas y salidas de una entidad son consideradas señales.
    - Pueden usarse en cualquier parte del programa o descripción y se declaran siempre en la parte de arquitectura antes del BEGIN.
    - Esto indica que las señales son visibles por todos los procesos y bloques dentro de una arquitectura.
    - En un diseño las conexiones físicas entre unos elementos y otros son declaradas como señales.
    - Al asignar un nuevo valor a una señal, éste no es adoptado hasta el siguiente paso de simulación.

SINTAXIS DE VHDL: Declaración de objetos

- **Diferencias entre los distintos tipos de objetos**:
  - **Constantes**: las constantes pueden ser habitualmente declaradas en los mismos sitios que las variables y señales.

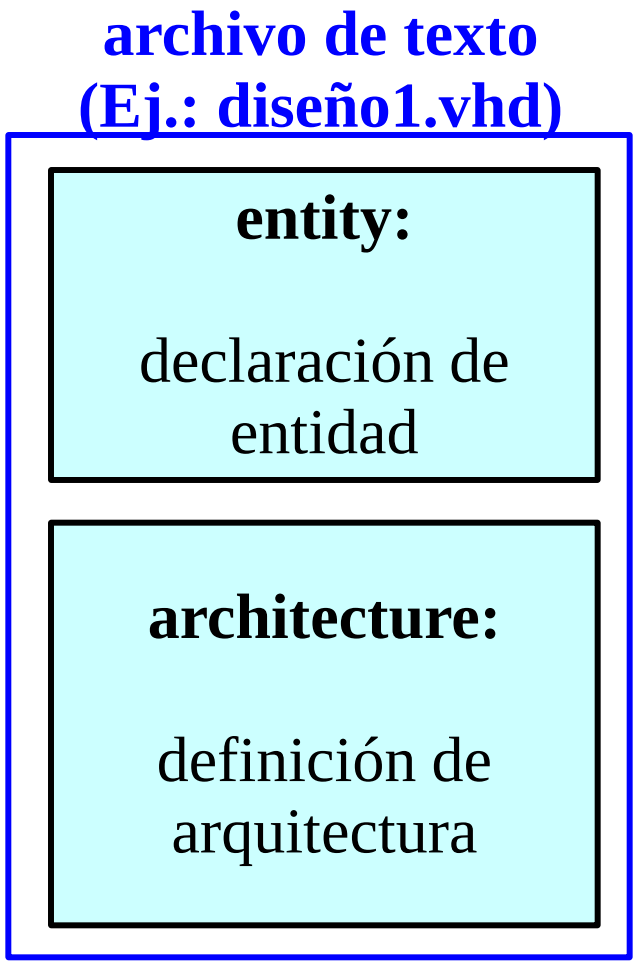
SINTAXIS DE VHDL: Atributos

- Los objetos en VHDL (constantes, variables, señales, tipos de datos, etc.) pueden tener información adicional llamada **atributos**.
- Se asocian a un objeto del lenguaje mediante la comilla simple: '
- Algunos atributos predefinidos:
  - Para un tipo escalar t (enteros, reales, físicos, enumerados):
    - t'left: límite izquierdo del rango del tipo t
    - t'right: límite derecho del rango del tipo t
    - t'low: límite inferior del rango del tipo t
    - t'high: límite superior del rango del tipo t
  - Para un tipo escalar t, x un miembro de este tipo y N un entero:
    - t'pos(x): posición de x dentro del tipo t
    - t'val(N): elemento N del tipo t
    - t'leftof(x): elemento que está a la izquierda de x en t
    - t'rightof(x): elemento que está a la derecha de x en t

SINTAXIS DE VHDL: Atributos

- Para un tipo u objeto de tipo matriz a, y N un entero desde 1 al número de dimensiones de la matriz (la dimensión se refiere al número de índices de la matriz), se pueden usar los siguientes atributos:
  - a'left(N): límite izquierdo del rango de la dimensión N de a
  - a'right(N): límite derecho del rango de la dimensión N de a
  - a'low(N): límite inferior del rango de la dimensión N de a
  - a'high(N): límite superior del rango de la dimensión N de a
- Para una señal s:
  - s'event: devuelve true si se ha producido un cambio en la señal s. Este atributo es muy usado.
  - s'stable(t): devuelve true si la señal s estuvo estable durante el último periodo de tiempo t.

Estructura de un archivo de programa VHDL



SINTAXIS DE VHDL: Declaración de entidad

- Una **entidad** define un módulo del circuito, especificando sus **entradas y salidas.**
- Es la estructura que permite implementar diseños jerárquicos en VHDL, ya que un diseño jerárquico es una colección de módulos interconectados entre sí. En VHDL estos módulos se definen mediante la palabra clave ENTITY.

- Sintaxis:

```
ENTITY nombre IS
   [GENERIC(lista de parámetros);]
   [PORT(lista de puertos);]
   [declaraciones]
[BEGIN
   sentencias]
END [ENTITY] [nombre];
```

Definir y declarar propiedades o constantes del módulo (opcional)

Definir las entradas y salidas del módulo (opcional)

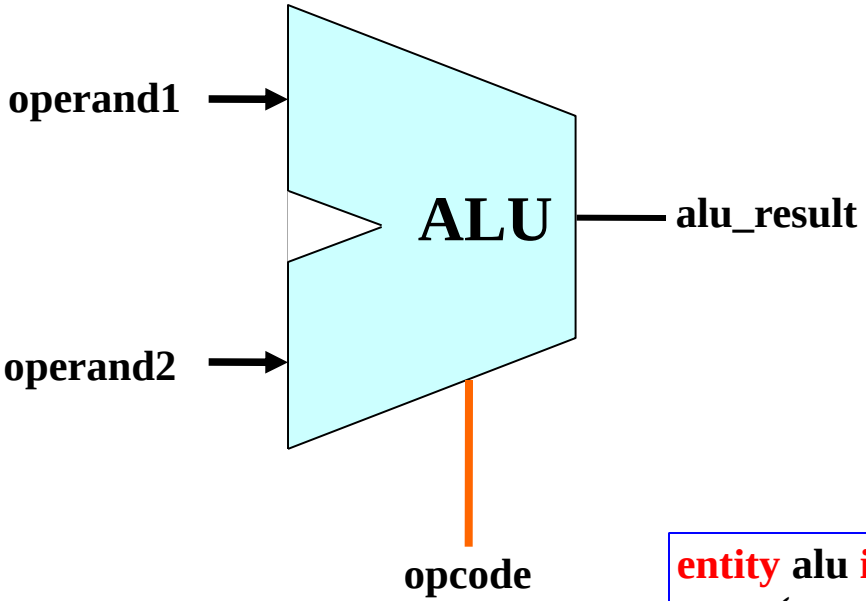
Declaraciones de constantes, etc. (opcional)

Se pueden incluir sentencias (se limitan a sentencias de indicación de errores o comprobación de alguna cosa, opcional)

SINTAXIS DE VHDL: Declaración de entidad

- PORT(lista de puertos): se definen las entradas y salidas del módulo que está siendo definido.
  - Sintaxis: nombre modo_de_conexión tipo_de_datos_de_la _línea
  - Modos de conexión:
    - IN: señal de entrada.
      - se pueden leer pero no puede asignárseles ningún valor, es decir, no se puede cambiar su valor en el programa ⇔ constantes.
    - OUT: señal de salida.
      - pueden cambiar y se les puede asignar valores, pero no pueden leerse, es decir, no pueden ser usadas como argumentos en la asignación de cualquier elemento del VHDL.
    - INOUT: señal tanto de entrada como de salida.
      - pueden ser usadas en el programa tanto para lectura como para escritura.
      - es preferible (salvo que sea imprescindible) usar señales separadas de entrada y de salida, en lugar de señales INOUT. Sólo se utilizará la clase INOUT cuando la señal sea realmente de entrada-salida, como en el bus de datos de un microprocesador.

Ejemplo: ALU



Sintaxis:

```
entity nombre_de_entidad is
  port (
nombre_de_senal: modo tipo_de_senal;
...
nombre_de_senal: modo tipo_de_senal);
end nombre_de_entidad;
```

```
entity alu is
 port(
        operand1:     in std_logic_vector(7 downto 0);
        operand2:     in std_logic_vector(7 downto 0);
        opcode:       in std_logic_vector(2 downto 0);
        alu_result:   out std_logic_vector(7 downto 0));
end alu;
```

SINTAXIS DE VHDL: Declaración de arquitectura

- En la arquitectura es donde se define el funcionamiento del módulo definido en la entidad.
- Una arquitectura siempre está referida a una entidad concreta, por lo que no tiene sentido hacer declaraciones de arquitectura sin especificar la entidad.
- Una misma entidad puede tener diferentes arquitecturas, pero es en el momento de la simulación o de la síntesis, cuando se especifica qué arquitectura concreta de la entidad se quiere simular o sintetizar.

- Sintaxis:

```
ARCHITECTURE nombre OF nombre_entidad IS
        [declaraciones]
BEGIN
        [instrucciones]
END [ARCHITECTURE] [nombre];
```

Parte declarativa: definición de los subprogramas (funciones, procedimientos, etc.), declaraciones de tipo, de constantes, de señales, de alias, de componentes, etc.

Instrucciones que definen el comportamiento, estructura y funcionalidad del circuito.

SINTAXIS DE VHDL: Declaración de arquitectura

- En la arquitectura es donde se define el funcionamiento del módulo definido en la entidad.
- Una arquitectura siempre está referida a una entidad concreta, por lo que no tiene sentido hacer declaraciones de arquitectura sin especificar la entidad.
- Una misma entidad puede tener diferentes arquitecturas, pero es en el momento de la simulación o de la síntesis, cuando se especifica qué arquitectura concreta de la entidad se quiere simular o sintetizar.

- Sintaxis:

```
ARCHITECTURE nombre OF nombre_entidad IS
        [declaraciones]
BEGIN
        [instrucciones]
END [ARCHITECTURE] [nombre];
```

Parte declarativa: definición de los subprogramas (funciones, procedimientos, etc.), declaraciones de tipo, de constantes, de señales, de alias, de componentes, etc.

Instrucciones que definen el comportamiento, estructura y funcionalidad del circuito.

- 21/11/2012

Estilos de descripción de circuitos

- Tres estilos: Uno ESTRUCTURAL y dos COMPORTAMENTAL.
- DESCRIPCIÓN COMPORTAMENTAL ALGORÍTMICA:
- ARCHITECTURE comportamental OF mux IS
- BEGIN
  - PROCESS (a, b, selec)
    - BEGIN
      - IF (selec='0') THEN
        - salida<=a;
      - ELSE
        - salida<=b;
      - ENDIF;
    - END PROCESS;
    - END comportamental;

- Las instrucciones de PROCESS se ejecutan secuencialmente cada vez que cambia alguna de las señales.
- En una descripción comportamental se indica el funcionamiento del circuito, no se indican ni sus componentes ni sus interconexiones.
  - PROCESS (a, b, selec)
    - BEGIN
      - IF (selec='0') THEN
        - salida<=a;
      - ELSE
        - salida<=b;
      - ENDIF;
    - END PROCESS;

- Se indica que si la señal selec es cero, entonces la salida es la entrada a, y si selec es uno, entonces la salida es la entrada b.

- No se indican ni los componentes ni sus interconexiones, sino simplemente lo que hace, es decir, su comportamiento o funcionamiento.

- Descripción concurrente. Permite la paralelización de las instrucciones. Y se encuentra más cercana a una descripción estructural del circuito.
- Ejemplo de descripción Flujo de datos o de Transferencia de Registros:
  - **ARCHITECTURE** flujo1 **OF** mux **IS**
  - **SIGNAL** noselec, ax, bx: bit;
  - **BEGIN**
    - noselec<=**NOT** selec;
    - ax<=a **AND** noselec;
    - bx<=b **AND** selec;
    - salida<=ax **OR** bx;
  - **END** flujo1;
- Todas las instr. se ejecutan cada vez que cambia alguna de las señales que intervienen en la asignación.

- **ARCHITECTURE** flujo2 **OF** mux **IS**
- **BEGIN**
- salida<=a **WHEN** selec=’0’ **ELSE** b;
- **END** flujo2;

- Es más sencillo sintetizar un circuito descrito al nivel de flujo de datos que otro a nivel más abstracto. La mayoría de estructuras de la desc. flujo de datos tiene correspondencia casi directa con su implementación hardware correspondiente.

Ejemplo de descripción RTL de un comparador de dos buses de 11 bits. El comparador tendrá 3 salidas activas a nivel más alto. Una se activará si a=b, otra si a>b, y la última si a<b.

```
ENTITY comp IS
PORT (a, b:IN bit_vector(10 DOWNTO 0);amayqb, aeqb, amenb: OUT bit);
END comp;
ARCHITECTURE flujo OF comp IS
BEGIN
    amayqb <='1' WHEN a>b ELSE '0';
    aeqb   <='1' WHEN a=b ELSE '0';
    amenqb<='1' WHEN a<b ELSE '0';
END flujo;
```

Estas instrucciones se ejecutan de forma concurrente.

- La instrucción básica de la ejecución concurrente es la asignación entre señales que viene gobernada por el operador <=.
- **Asignación condicional: WHEN..ELSE**

[id_instr:]

señal<= [opciones] {valor **WHEN** condición **ELSE**}

valor [**WHEN** condición];

- **Ejemplo:**

```
s<='1' WHEN a=b ELSE
     '0' WHEN a>b ELSE
     'X';
```

- **Asignación condicional: WHEN..ELSE**
- **Asignación con selección: WITH..SELECT..WHEN**

WITH expresión **SELECT**

señal <= [opciones] valor **WHEN** caso

{, valor **WHEN** caso};

- Ejemplo: Se utiliza la orden de selección para elegir el color en un semáforo ficticio:

```
WITH estado SELECT
    semaforo<=”rojo”        WHEN ”01”,
              “verde”       WHEN ”10”,
              “amarillo”    WHEN ”11”,
              “no funciona” WHEN OTHERS;
```

- BLOQUE CONCURRENTE: BLOCK
  - Sentencias de ejecución concurrente en bloque-
  - Diseños modulares.
  - Subdivisión del programa en una jerarquía de módulos.

id_block:

BLOCK [(expresión de vigilancia)] [**IS**]

[cabecera]

[declaraciones]

BEGIN

{sentencias concurrentes}

END BLOCK [id_block];

- La descripción estructural también se incluye dentro de un bloque de arquitectura, si bien la sintaxis interna es completamente diferente.

- Se interconectan los componentes y se evalúan inmediatamente con señales

- Se indica qué componentes son usados y de qué manera están interconectados

```
nux.vhd
*
 Arquitectura estructural de mux:
rchitecture estructural of mux is
                        component and2
                                port (e1, e2: in bit; y: out bit);
                        end component;
                        component or2
                                port (e1, e2: in bit; y: out bit);
                        end component;
                        component inv
                                port (e: in bit; y: out bit);
                        end component;
                        signal ax, bx, noselec: bit;
egin
        u0: inv port map (e=>selec, y =>noselec);
        u1: and2 port map (e1=>a, e2=>noselec, y=>ax);
        u2: and2 port map (e1=>b, e2=>selec, y=>bx);
        u3: or2 port map (e1=>ax, e2=>bx, y=>salida);
```

Estilos de descripción de circuitos. DESCRIPCIÓN ESTRUCTURAL

- En el cuerpo de la arquitectura se ponen los componentes y sus interconexiones.
- Para los componentes se utilizarán entidades definidas en una biblioteca,
- Para las conexiones se usarán señales declaradas al principio de la arquitectura.
- En el esquema se le han puesto nombres a las líneas de conexión internas al circuito.

- Estas líneas hay que declararlas como **SIGNAL** en el cuerpo de la arquitectura y delante del **BEGIN**.

```
nux.vhd
*
 Arquitectura estructural de mux:
rchitecture estructural of mux is
                component and2
                        port (e1, e2: in bit; y: out bit);
                end component;
                component or2
                        port (e1, e2: in bit; y: out bit);
                end component;
                component inv
                        port (e: in bit; y: out bit);
                end component;
                signal ax, bx, noselec: bit;
egin
        u0: inv port map (e=>selec, y =>noselec);
        u1: and2 port map (e1=>a, e2=>noselec, y=>ax);
        u2: and2 port map (e1=>b, e2=>selec, y=>bx);
```

- *Estas líneas hay que declararlas como **SIGNAL** en el cuerpo de la arquitectura y delante del **BEGIN**.*

- Una vez declaradas las señales que intervienen se procede a conectar entre sí las señales y entidades que representan componentes.
- Lo primero es identificar y poner cada componente, que es lo que se conoce como *replicación*, es decir, asignarle a cada componente concreto un símbolo.

- En este ejemplo se le ha llamado “u” a cada componente y se le ha añadido un número para distinguirlo.

- A continuación del nombre viene la entidad correspondiente al componente, que en este caso puede ser una puerta lógica and2, or2 o inv.

- Después se realizan las conexiones poniendo cada señal en su lugar correspondiente con las palabras **PORT MAP**.

- Así, los dos primeros argumentos en el caso de la puerta and2 son las entradas, y el último es la salida.

```
nux.vhd
*
 Arquitectura estructural de mux:
rchitecture estructural of mux is
                component and2
                        port (e1, e2: in bit; y: out bit);
                end component;
                component or2
                        port (e1, e2: in bit; y: out bit);
                end component;
                component inv
                        port (e: in bit; y: out bit);
                end component;
                signal ax, bx, noselec: bit;
egin
        u0: inv port map (e=>selec, y =>noselec);
        u1: and2 port map (e1=>a, e2=>noselec, y=>ax);
        u2: and2 port map (e1=>b, e2=>selec, y=>bx);
        u3: or2 port map (e1=>ax, e2=>bx, y=>salida);
```

BIBLIOGRAFÍA

- Pardo (2004). VHDL. Lenguaje para síntesis y modelado de circuitos.
- Ferrero (1999). Descripción de circuitos digitales mediante VHDL.
- http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/rubio_s_d/capitulo3.pdf