Estructuras de Computadores

Tutorial VHDL

Área de arquitectura y tecnología de los computadores.
Departamento de Tecnología de los computadores y de las comunicaciones.
Universidad de Extremadura.
Centro Universitario de Mérida.

Lenguajes de descripción de hardware (VHDL):

Los lenguajes de descripción de hardware vienen utilizándose desde los años 70 durante los ciclos de diseño de sistemas digitales asistidos por herramientas CAD. En los años 80 surgen el **VERILOG** y el **VHDL**, que logran imponerse como herramientas imprescindibles en el desarrollo de nuevos sistemas debido a la disponibilidad de herramientas HW y SW cada vez más potentes y a los avances en la fabricación de circuitos integrados.

VHDL: en 1987 **IBM y Texas Instruments** terminan la realización de VHDL que se establece como la norma IEEE-1076. Sus características principales son:

- **Descripción textual formalizada:** lenguaje de descripción de circuitos de forma adecuada para ser entendida tanto por humanos como por la máquina. Es formal (libre de ambigüedades) y normalizado (un único modelo del lenguaje, compatible con cualquier herramienta que respete la norma formal).

- **Amplio rango de capacidad descriptiva:** posibilitando la descripción del HW a diferentes niveles de abstracción.

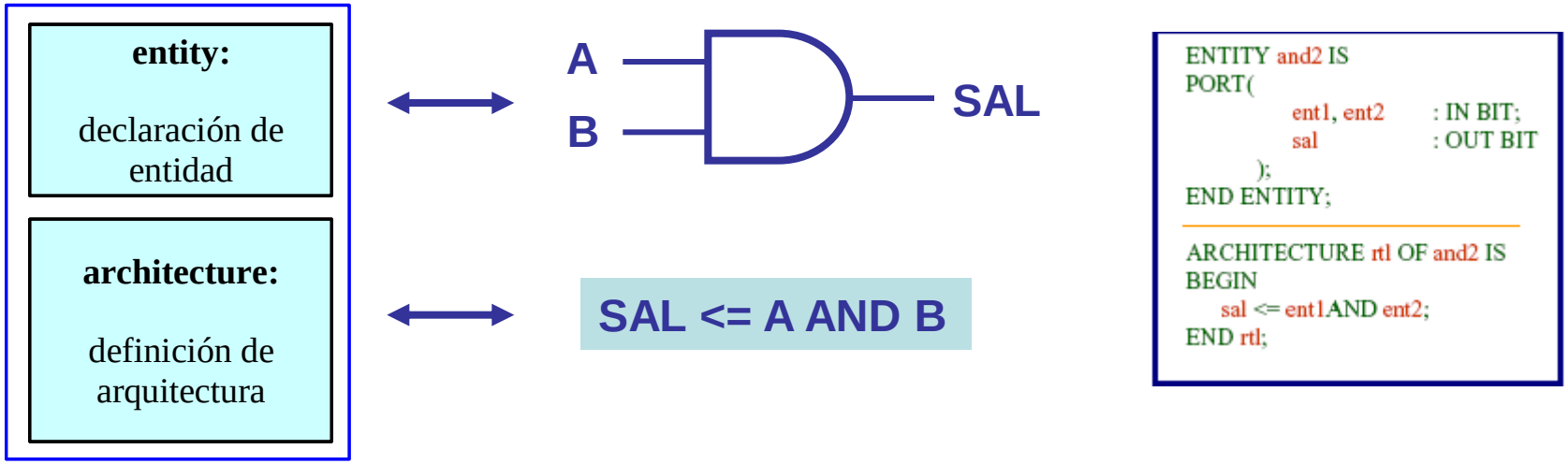
VHDL I: caracterización de circuitos.

Hay que entender los dos aspectos que lo caracterizan:

Un interfaz externo: una puerta and de dos entradas, por ejemplo, tiene tres terminales, dos entradas y una salida, y se diferencia de una puerta and de tres entradas en dos cosas: el número de terminales y el nombre **and** *de dos entradas* y **and** *de tres entradas*).

Un algoritmo de procesamiento de la información: cada disp. digital realiza una operación que le permite obtener ciertos niveles lógicos en sus terminales de salida a partir de los aplicados sobre los de entrada.

DECLARACIÓN DE ENTIDAD Y CUERPO DE ARQUITECTURA:



VHDL I: sintaxis básica de la declaración de entidad.

```
ENTITY {nombre del dispositivo} IS
PORT(
    {lista de puertos de entrada}        : IN     {tipo de dato};
    {lista de puertos bidireccionales} : INOUT  {tipo de dato};
    {lista de puertos de salida}         : OUT    {tipo de dato};
    {lista de puertos de salida}         : BUFFER {tipo de dato});
END ENTITY;
```

Para cada puerto (PORT) hay que declarar:

- **Su nombre:** etiquetas definibles por el usuario (no palabras reservadas).
- **Su dirección:** a partir de las características del terminal → entrada IN, salida OUT, bidireccionales INOUT.
- **Tipo de datos:** elección del tipo de datos de los objetos, muy importante pues determina el conjunto de valores que puede tomar el objeto.

VHDL I: cuerpos de arquitectura.

```
ARCHITECTURE {nombre_de_arquitectura} OF {nombre_de entidad} IS

        {zona de declaración}

        BEGIN

        {zona de descripción}

        END {nombre de arquitectura};
```

Las dos etiquetas que conforman la cabecera nombran a la propia arquitectura y a la entidad para la que se declara. Por lo demás, hay 2 zonas de inclusión de código:

- Entre la cabecera y el BEGIN: declaración de objetos necesarios para la descripción.
- Entre BEGIN y END {arquitectura}: donde se describe el funcionamiento.

En el ejemplo de la puerta AND, no se emplea ningún objeto, y la descripción del funcionamiento se realiza utilizando una sentencia concurrente de asignación.

VHDL I: simulación del modelo.

```
ENTITY test_bench_and2 IS
END test_bench;
ARCHITECTURE  test OF  test_bench_and2 IS
   signal  s_ent1, s_ent2, s_sal: BIT;
BEGIN
  s_ent1 <= '1' after 10 ns;
  s_ent2 <= '1' after 5 ns,
            '0' after 10 ns,
            '1' after 15 ns;
  dut: entity work.and2(rtl)
    port map (ent1=>s_ent1, ent2=>s_ent2, sal=>s_sal);
END  test  ;
```

La simulación VHDL se realiza conectando el modelo de pruebas a un conjunto de estímulos que permiten observar la respuesta del mismo. Para ello hay que construir un banco de pruebas. La figura anterior muestra un banco de pruebas para simular la puerta and. Consta de Declaración de Entidad y Cuerpo de Arquitectura, pero:

- La declaración de entidad no tiene puertos.
- El cuerpo de arquitectura no describe el comportamiento, sino que conecta el dispositivo a un conjunto de señales (estímulos y salidas)

VHDL II: unidades de diseño y librerías.

Un modelo VHDL se compone de un conjunto de unidades de diseño. Una unidad de diseño es la unidad mínima de código compilable por separado. Existen 5 tipos:

La declaración de Entidad: unidad primaria del lenguaje que identifica a los dispositivos, definiendo su interfaz (nombre de terminales y parámetros)

Cuerpo de la Arquitectura: unidad secundaria del lenguaje. Asociada a una determinada entidad, describe su funcionamiento lógico.

Declaración de Paquete: papel similar a las librerías en lenguajes de alto nivel.

Cuerpo del Paquete: unidad secundaria asociada a la declaración. Se emplea, si es necesario, para definir los elementos declarados en este.

Declaración de Configuración: unidad primaria que sirve para manejar el emplazamiento de componentes estructurales.

De todas ellas, las más importantes y las más usadas por vosotros, serán las 4 primeras, y dentro de esas 4, las primeras 2 son las más importantes.

VHDL II: unidades de diseño y librerías.

Las unidades de diseño VHDL se almacenan en librerías VHDL. Así, una librería VHDL es una colección de unidades de diseño (que quedan almacenadas en la librería al compilarse correctamente).

Importante: no confundir las librerías VHDL con las de un LAN. El equivalente a las librerías de un LAN son los Paquetes de VHDL.

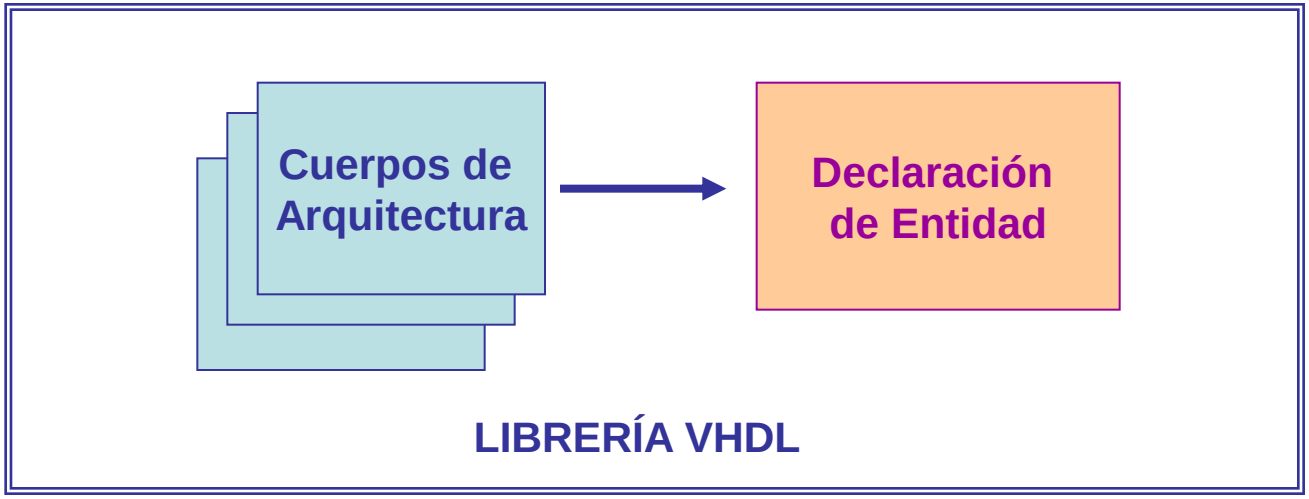
REGLAS:

- Una unidad primaria **(Entidad)** y cualquier unidad secundaria asociada a ésta **(Arquitectura)** deben almacenarse en la misma librería. Además es necesario compilar antes la primaria que la secundaria.
- Si una unidad de diseño tiene visibilidad sobre otra, ésta tiene que compilarse antes que la primera.
- No pueden existir dos unidades primarias (si secundarias) con el mismo nombre en una misma librería.

VHDL II: unidades de diseño y librerías.

+ REGLAS:

- Una declaración de Entidad puede tener asociados varios cuerpos de Arquitectura.
- Un Paquete puede estar formado únicamente por una Declaración de Paquete, esto es, el Cuerpo del Paquete puede no existir.
- Cada librería VHDL tiene un nombre lógico. Además, la librería de trabajo que se está utilizando en un determinado momento puede identificarse mediante la palabra Work.

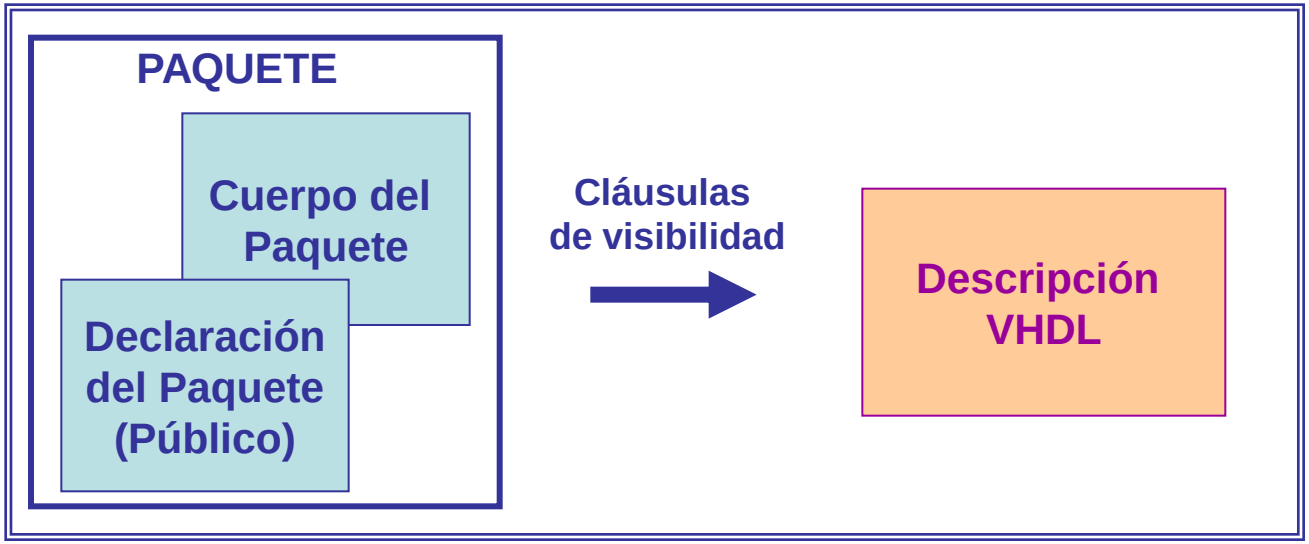


VHDL II: cláusulas de visibilidad.

Los modelos HW y el contenido de los Paquetes VHDL almacenados en una determinada librería pueden utilizarse en las unidades de diseño de otras librerías. Para ello, se emplean las cláusulas de visibilidad Library.:

LIBRARY libmodelos;
ARCHITECTURE rtl OF modelo1 IS…

Permite que las unidades de diseño almacenadas en la librería libmodelos puedan utilizarse en la arquitectura rtl de modelo1 (esta cláusula por sí sola no da acceso al contenido, solo permite nombrarlas).



VHDL II: cláusulas de visibilidad.

En el desarrollo de modelos VHDL, resulta muy frecuente el uso de paquetes de la librería IEEE. Dos de los paquetes de esa librería se llaman: std_logic_1164 y numeric_std. (os cansaréis de utilizarlos).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL, ieee.numeric_std.ALL;
```

ALL proporciona acceso a todo el contenido de dichos paquetes. Dentro de este paquete hay un tipo de datos que se llama std_ulogic. Si solo queremos acceder a dicho tipo de datos:

```
USE ieee.std_logic_1164.std_ulogic
```

EXCEPCIONES:

- La librería Work siempre es visible (**LIBRARY WORK** es inútil). Si quisieramos acceder a un paquete dentro de Work, si sería necesario hacer el **USE WORK.packwork.ALL.**
- La librería **STD** y el contenido del paquete **STANDARD** son siempre visibles.

VHDL III: 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”.

VHDL III: 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=(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

VHDL III: 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**

VHDL III: 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

VHDL III: Objetos y tipos de datos básicos: señales, variables y constantes.

Todos los objetos VHDL deben ser declarados (el lenguaje determina en que zonas) indicando la clase (constante, variable o señal), su tipo de datos, y si es preciso (constantes) su valor inicial.

Variables y constantes → similares a las de cualquier LAN.

Señales → empleadas para modelar nodos lógicos (muy parecidas a las variables). Se diferencian principalmemente en le mecanismo de actualización: inmediatamente después de una sentencia de asignación de valor a señal, el valor de la misma no cambia.

Ej: sean A y B dos señales, cuyos valores son '0' y '1' respectivamente, tras las sentencias:

```
A<=B after 5 ns; -- <= es el símbolo de asignación a señal.
B:=A;            -- <= símbolo de asignación a variable.
                 -- los guiones son el símbolo del comentario.
```

A y B valdrán '0'.

VHDL III: Objetos y tipos de datos básicos: declaración de objetos.

La fórmula de declaración de constantes, variables y señales es:

Tipo_de_Objeto Nombre_de_Objeto: Tipo_de_Datos := Valor_Inicial;

Por ejemplo:

```
SIGNAL nodo1: BIT:= '1';
VARIABLE var1: BIT;
CONSTANT pi: REAL:= 3.14159;
```

La inicialización es optativa en caso de señales y variables y obligatoria para las constantes.

Asignaciones de valor a señal: lo que hace es proyectar la transacción de un valor a la señal para un determinado instante de tiempo de simulación.

En el caso del ejemplo anterior, si el tiempo actual de simulación **T=10 ns**, A mantendrá el valor '**0'** hasta **T=15 ns**, instante en el que pasará a valer **'1'**.

VHDL III: Objetos y tipos de datos básicos: puertos de la Declaración de Entidad y Tipos de Datos.

Los puertos son señales direccionales. Debe respetarse siempre que:

- Los puertos de entrada solo pueden leerse.
- En los puertos de salida no puede escribirse.
- En los puertos bidireccionales se puede leer o escribir.

VHDL III: Tipos de Datos (indicamos los contenidos en el paquete std mediante (PREDEFINIDO))**.**

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)**
  - TYPE boolean IS (FALSE, TRUE); **(PREDEFINIDO)**
  - **…**

VHDL III: Tipos de Datos (indicamos los contenidos en el paquete std mediante (PREDEFINIDO))**.**

- **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 um, 23 ns.

  - **TYPE longitud IS RANGE 0 TO 1.0E9**

    UNITS

    um;

    mm=1000 um;

    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)

VHDL III: Tipos de Datos (indicamos los contenidos en el paquete std mediante (PREDEFINIDO))**.**

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)

VHDL III: Tipos de Datos (indicamos los contenidos en el paquete std mediante (PREDEFINIDO))**.**

- **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=“Gema"

VHDL III: Tipos de Datos (indicamos los contenidos en el paquete std mediante (PREDEFINIDO))**.**

- **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);

VHDL III: 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 de t en la posición N
    - 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

VHDL III: 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.

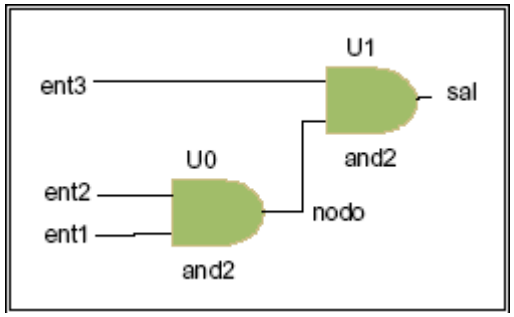
VHDL IV: Estilos de descripción:

- Los Cuerpos de Arquitectura sirven para describir el funcionamiento de los dispositivos HW. La descripción puede hacerse de 2 formas:
  - Mediante construcciones que representan un algoritmo de procesamiento.
  - Mediante la interconexión de otros dispositivos.

Por ejemplo:para describir una puerta and de 3 entradas, podemos abordar el proclema con diferentes estilos:

Podemos usar la sentencia: **sal <= ent1 AND ent2 AND ent3; (Estilo COMPORTAMENTAL)**

O podríamos basarnos en el diseño previo de una and de 2 entradas:



U0: ENTITY WORK.and2(RTL) PORT MAP(ent1, ent2, nodo)
U1: ENTITY WORK.and2(RTL) PORT MAP(nodo, ent3, sal)
(Estilo ESTRUCTURAL)

VHDL: Estilos de descripción:

- **ESTRUCTURAL:** una definición del funcionamiento que se escribe mediante la interconexión de otros dispositivos. Es la de menor grado de abstracción.

- **RTL (Register Transfer Level):** la siguiente en grado de abstracción. Se emplean construcciones muy simples (ecuaciones lógicas, operaciones aritméticas muy básicas), cercanas a la lógica descrita para la estructura HW.

- **COMPORTAMENTAL:** al que le corresponde el mayor grado de abstracción. El procesamiento realizado por el HW se describe mediante algoritmos complejos (similares a los emleados en los LAN's). Es lo más sencillo y rápido, pero poco realista.

En la práctica, la mayor parte del código VHDL escrito, se implementa mediante RTL o mediante ESTRUCTURAL (o mediante una mezcla de ambas).

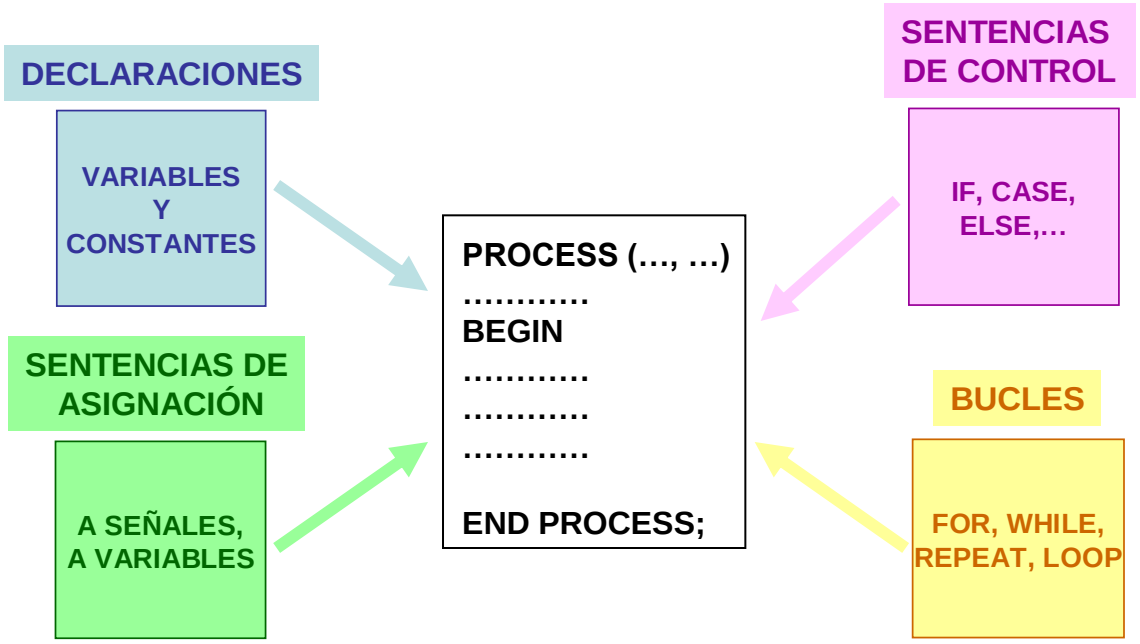
```
ARCHITECTURE {nombre_de_arquitectura} OF {nombre_de entidad} IS

        {zona de declaración}

        BEGIN

        {zona de descripción}

        END {nombre de arquitectura};
```

Generalmente, el nombre de la arquitectura se suele hacer coincidir con el nombre del estilo empleado para su descripción: **Estructural, RTL o Comportamental.**

VHDL: Procesos.

El aspecto de las zonas de declaración y descripción varía mucho dependiendo del estilo de descripción. Si se trata de RTL o comportamental, en la zona de declaración es frecuente encontrar declaraciones de señales, constantes y, a veces, tipos de datos mientras que el elemento descriptivo básico es **el proceso.**

Procesos: bloque de código que representa un algoritmo de procesamiento. El algoritmo se codifica dentro del proceso usando sentencias similares a las de los LAN, sentencias de asignación de valores a variables y a señales.



PROCESS (lista de señales)
{zona de declaración}
BEGIN

{algoritmo de procesamiento}

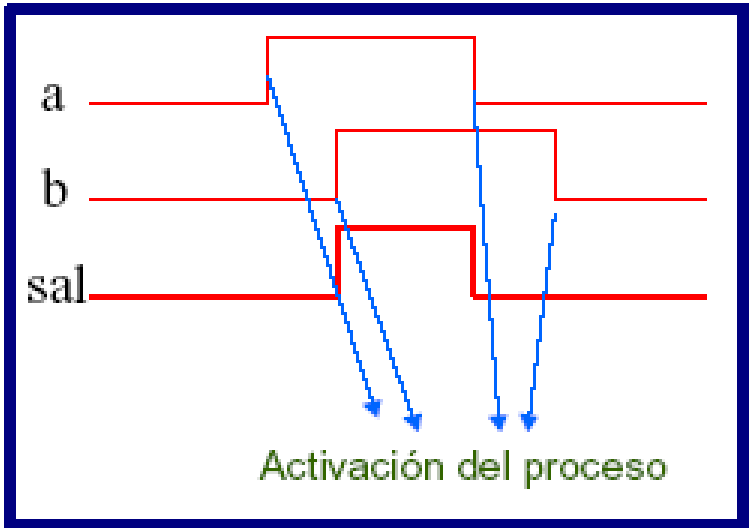
END PROCESS;

(lista de señales): opcional, puede sustituirse por **WAIT**, indicando la lista de señales que activarían la ejecución del proceso.

VHDL: Procesos, lista de sensibilización.

Ejemplo: proceso que modela una puerta and de dos entradas:

```
PROCESS(a, b)
      BEGIN
      IF a = '1' AND b = '1' THEN
        sal <= '1';
      ELSE
        sal <= '0';
      END IF;
END PROCESS;
```

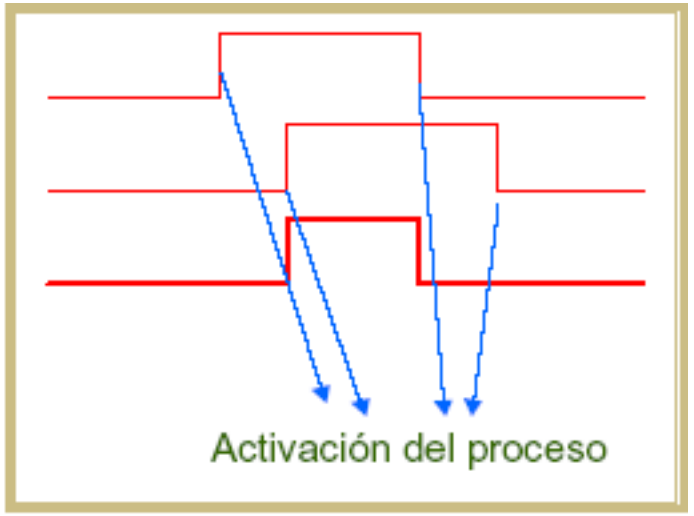


Con lista de sensibilización: el proceso es sensible a las señales a y b (equivalente a hacer WAIT A,B;). En la simulación, vemos como el proceso se ejecuta 4 veces, evaluando el IF en cada una de ellas, proyectando una asignación de valor '1' ó '0' para la salida. Al ejecutarse por 4ª vez, el proceso se suspende hasta una nueva alteración de las señales A o B.

VHDL: Procesos, sentencia WAIT.

Ejemplo: proceso que modela una puerta and de dos entradas:

```
PROCESS
      BEGIN
      WAIT ON a, b;
      IF a = '1' AND b = '1' THEN
        sal <= '1';
      ELSE
        sal <= '0';
      END IF;
END PROCESS;
```



Con sentencia WAIT: el proceso se activaría cada vez que hubiera un evento en A o en B, y discurrirá secuencialmente hasta volver a alcanzar la sentencia WAIT, donde se quedaría esperando un nuevo evento. Sería equivalente al ejemplo anterior si la sentencia WAIT estuviera al final del proceso.
WAIT se puede usar con condiciones lógicas: **WAIT UNTIL condición_lógica;** con retardos de tiempo: **WAIT FOR retardo;** y sola: **WAIT;** (el proceso no volvería a ejecutarse nunca).

VHDL: Instanciación de Componentes: descripción estructural.

Lo normal, al modelar circuitos con VHDL, es emplear una combinación de estilos: **RTL** para modelar en cada proceso la funcionalidad de un módulo y **estructural** para modular la topología del circuito. A medida que el circuito se complica, es posible que necesitemos muchos módulos, que por simplificación, se pueden **instanciar.**

Componente: zócalos para la inserción de una Declaración de Entidad y uno de sus Cuerpos de Arquitectura. Hay que declararlo en alguna zona visible al Cuerpo de Arquitectura que lo use. Cada sentencia de emplazamiento de un componente da lugar a una instancia.

Instancia: la instancia de un Componente emplaza a la Declaración de Entidad y Cuerpo de Arquitectura especificados en dicho Componente.

Entonces, para instanciar el proceso a seguir es:

- Declarar componentes para cada uno de los dispositivos que se desee utilizar.
- Especificar el dispositivo que se emplaza en cada instancia del componente.
- Colocar las instancias en el Cuerpo de Arquitectura.

VHDL: Instanciación de Componentes: descripción estructural.

Para simplificar, nosotros reduciremos el proceso de instanciación de componentes a los siguientes pasos:

Declarar el Componente: →

```
COMPONENT nombre IS
    PORT( lista_de_puertos);
END COMPONENT;
```

Emplazar una instancia: →

```
nombre_de_instancia: nombre_de_componente
PORT MAP( lista_de_conexión);
```

La lista de conexión establece la correspondencia entre puertos de componente y señales. Puede hacerse por nombre o por posición:

```
BEGIN

U0: and2
    PORT MAP (a =>ent1,
              b => ent2,
              sal => nodo);
U1: and2
    PORT MAP (a =>nodo,
              b => ent3,
              sal => sal);

END estruct;
```

```
BEGIN

U0: and2
    PORT MAP (ent1, ent2, nodo);
U1: and2
    PORT MAP (nodo, ent3, sal);

END estruct;
```

En la conexión por posición, cada señal queda conectada con el puerto que le corresponde en el orden de declaración; en la conexión por nombre con el que se indique explícitamente.

VHDL: Instanciación de Componentes: descripción estructural.

Reglas para la lista de conexiones:

- Un puerto de entrada solo puede conectarse a una señal cuyo valor pueda leerse.
- Un puerto de salida solo puede conectarse a una señal cuyo valor pueda escribirse.
- Si se desea dejar sin conectar algún puerto, puede hacerse mediante la palabra OPEN.
- Si se desea asociar un puerto con un valor determinado (en lugar de una señal), dependiendo de la herramienta de diseño empleada (posteriores a la norma 1993), puede hacerse:
  - Directamente sobre el puerto:

```
U0: inv PORT MAP('1', sal);
```

  - Empleando una señal auxiliar:

```
ent <= '1';
U0: inv PORT MAP(ent, sal);
```

VHDL: Instanciación de Componentes: descripción estructural.

```
ARCHITECTURE estructural_v2 OF acum4b IS

-- declaración de componentes:

COMPONENT restador IS
PORT(
a, b: IN UNSIGNED(3 DOWNTO 0);
sal: OUT SIGNED(4 DOWNTO 0)
   );
END COMPONENT;

COMPONENT calc_abs IS
PORT(
ent: IN SIGNED(4 DOWNTO 0);
sal: OUT UNSIGNED(3 DOWNTO 0)
   );
END COMPONENT;

COMPONENT sumador IS
PORT(
a: IN UNSIGNED(3 DOWNTO 0);
b: IN UNSIGNED(7 DOWNTO 0);
sal: BUFFER UNSIGNED(7 DOWNTO 0)
   );
END COMPONENT;

COMPONENT reg IS
PORT(
reset, clk: IN STD_LOGIC;
dato_ent: IN UNSIGNED(7 DOWNTO 0);
dato_sal: BUFFER UNSIGNED(7 DOWNTO 0)
   );
END COMPONENT;
```

```
-- configuracion de componentes:

 FOR U0: restador USE ENTITY WORK.restador(rtl);
 FOR U1: calc_abs USE ENTITY WORK.calc_abs(rtl);
 FOR U2: sumador  USE ENTITY WORK.sumador(rtl);
 FOR U3: reg      USE ENTITY WORK.reg(rtl);

-- DECLARACIÓN DE SEÑALES INTERNAS:

  SIGNAL nodo_int1: SIGNED(4 DOWNTO 0);
  SIGNAL nodo_int2: UNSIGNED(3 DOWNTO 0);
  SIGNAL nodo_int3: UNSIGNED(7 DOWNTO 0);

BEGIN

-- EMPLAZAMIENTO DE INSTANCIAS

U0: restador PORT MAP (dato_a, dato_b, nodo_int1);
U1: calc_abs PORT MAP (nodo_int1, nodo_int2);
U2: sumador  PORT MAP (nodo_int2, dato_sal, nodo_int3);
U3: reg         PORT MAP (reset, clk, nodo_int3, dato_sal);


END estructural_v2;
```

VHDL: Modelado Funcionamiento HW.

Los procesos modelan al HW emulando su actividad: en el transcurso del **tiempo de simulación:**

- El estado o salidas de un circuito se mantienen estables mientras las entradas no cambien → Los procesos no se ejecutan hasta que no se produzca un evento sobre sus señales (lista de sensibilización o sentencia WAIT).

- Al cambiar las entradas de un circuito, puede haber o no cambios en las salidas del circuito → Ejecución del proceso cuando hay eventos en las señales a las que es sensible durante el tiempo de simulación (los cambios pueden llevar asociados retardos).

¿Cuándo se materializa el tiempo de simulación? Cuando sometemos la simulación a **estímulos**. Los estímulos se suceden en el tiempo de simulación, pues son asignaciones de valor a señal controlados por retardos o por sentencias **WAIT FOR.**

VHDL: Modelado Funcionamiento HW.

Ciclo de simulación: parte del modelo de simulación que materializa la actualización de señales y ejecución de los procesos dentro del tiempo de simulación. Tiene 2 fases:

- **Primera parte:** comienza realizando las asignaciones proyectadas para el tiempo de simulación actual.

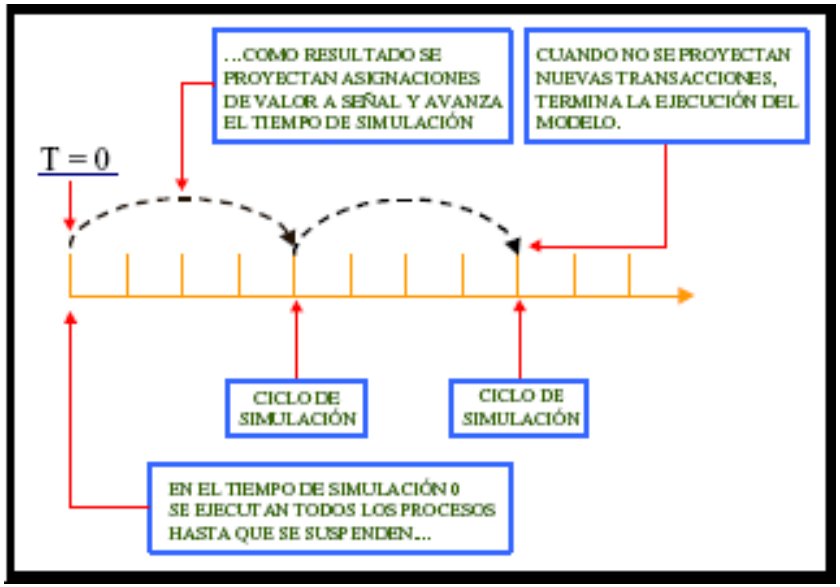
- **Segunda parte:** tras actualizar todas las señales, se ejecutan concurrentemente todos los procesos afectados por la actividad de las señales actualizadas en la primera parte del ciclo de simulación, o cuya ejecución quedó suspendida por algún WAIT FOR, hasta que se suspenda su ejecución.

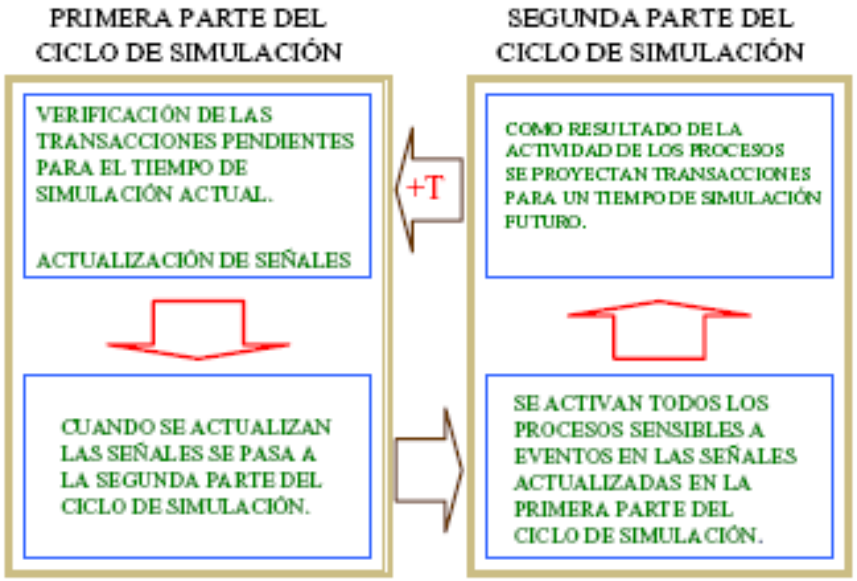
A partir de ese momento, el tiempo de simulación avanza por las proyecciones de asignación de valor a señal o sentencias WAIT FOR existentes en los procesos (propios del circuito y del generador de estímulos).

Al terminar un ciclo de simulación, el tiempo avanza hasta el instante en que encuentra de nuevo proyecciones de señal o sentencias WAIT FOR.

La simulación finaliza al alcanzar un tiempo máximo especificado.

VHDL: Modelado Funcionamiento HW.





VHDL: Sentencias dentro de los procesos.

De asignación de señales:

```
nombre_de_señal <= valor after tiempo;

nombre_de_señal <= valor after tiempo,
                   valor after tiempo,
                   . . .,
                   valor after tiempo;
```

De asignación de variables:

```
nombre_de_variable := valor;
```

Sentencia IF:

1.-
```
IF condición THEN
    SENTENCIAS
END IF;
```
2.-
```
IF condicion THEN
    SENTENCIAS
ELSE
    SENTENCIAS
END IF;
```
3.-
```
IF condición THEN
    SENTENCIAS
ELSIF condición THEN
    SENTENCIAS
ELSIF condición THEN
    SENTENCIAS
    . . .
END IF;
```

Sentencia CASE:

```
CASE expresión IS
    WHEN valor => SENTENCIAS
     WHEN valor1|valor2 => SENTENCIAS
    WHEN rango_de_valores => SENTENCIAS
    WHEN OTHERS => SENTENCIAS
END CASE;
```

Bucles:

```
FOR variable_del_bucle IN rango LOOP
    sentencias
END LOOP;

WHILE condicion LOOP
    sentencias
END LOOP;
```

VHDL: Funciones y Procedimientos.

VHDL proporciona dos tipos de subprogramas: Funciones y Procedimientos. Tienen funcionalidad limitada, pues no pueden trabajar con señales.

```
FUNCTION nombre_de_función (lista_de_parámetros) RETURN
tipo_de_datos;

PROCEDURE nombre_de_procedimiento(lista_de_parámetros);
```

Procesos Pasivos: sentencias ASSERT.

Son aquellos procesos en los que no se asigna valor a las señales. VHDL proporciona una por defecto, la sentencia ASSERT:

```
ASSERT expresión_booleana REPORT string
valor_de_tipo_SEVERITY_LEVEL;
```

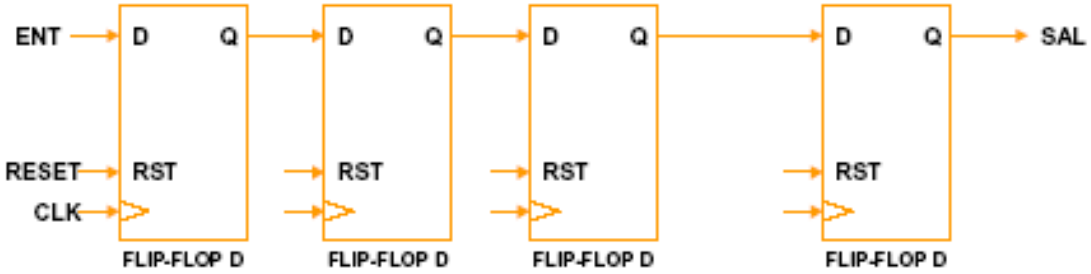
Cuando la condición booleana se evalúa a FALSE, se escribe el string en la consola del simulador. Si el valor de SEVERITY_LEVEL es ERROR o FAILURE, suele abortarse la simulación. Ej:

```
PROCESS(clk)
BEGIN
 IF clk'EVENT AND clk = '1' THEN
  ASSERT d'STABLE(tsu) REPORT ("violación de tiempo de set-up")
  SEVERITY WARNING;
 END IF;
END PROCESS;
```

VHDL: sentencia GENERATE.

Sentencia empleada para generar estructuras regulares de conexión, con un estilo compacto y parametrizable. La veremos con un ejemplo.

REGISTRO DE DESPLAZAMIENTO CON ESTILO ESTRUCTURAL:



Para su desarrollo, nos apoyaremos en el modelo de un flip-flop:

```
ENTITY ffd IS
PORT( clk, reset: IN BIT;
 D: IN BIT;
 Q: OUT BIT);
END ENTITY;

ARCHITECTURE rtl OF dff IS
BEGIN
 PROCESS(clk, reset)
 BEGIN
  IF reset = '1' THEN
   Q <= '0';
  ELSIF clk'EVENT AND clk = '1' THEN
   Q <= D;
  END IF;
 END PROCESS;
END RTL;
```

```
ENTITY reg_desp IS
GENERIC(N: IN NATURAL := 4);
port(clk, reset: IN bit;
    d_in: IN bit;
    d_out: OUT bit);
END ENTITY;


ARCHITECTURE estructural OF reg_desp IS

COMPONENT dff IS
 PORT(clk, reset: in BIT;
     d: in BIT;
     q: out BIT);
 END COMPONENT;
```

```
SIGNAL nodo_int: BIT_VECTOR(N-1 DOWNTO 0);

BEGIN
gen_reg:FOR I IN 0 TO N-1 GENERATE
 gen_izqda:IF I = 0 GENERATE
              U0: dff PORT MAP(clk, reset, d_in, nodo_int(0));
             END GENERATE;

 gen_dcha:IF I = N-1 GENERATE
              U0: dff PORT MAP(clk, reset, nodo_int(I-1), d_out);
             END GENERATE;

 gen_int:IF I /= 0 AND I /= (N - 1) GENERATE
            U0: dff PORT MAP(clk, reset, nodo_int(I -1), nodo_int(I));
           END GENERATE;

        END GENERATE;

END estructural;
```