TECNOLOGÍA DE COMPUTADORES

Tema 2

"Descripción de VHDL"

(2/2)

Agustín Álvarez Marquina

Tipos de sentencias en VHDL

Sentencias concurrentes

PROCESS

Asignación de señal

BLOCK

Llamadas a procedimientos

Llamadas a funciones

GENERATE

Instanciación de componentes

ASSERT

Sentencias secuenciales

WAIT

Asignación de señal

Asignación de variable

IF

CASE

LOOP

NEXT

EXIT

RETURN

NULL

ASSERT

Llamada a procedimientos

Llamada a funciones

Sentencias concurrentes (I)

Características:

- Se ejecutan de forma asincróna unas respecto de las otras en el mismo tiempo de simulación.
  - El orden en que se escriban es indiferente ya que no siguen un orden de ejecución predefinido.
    - No obstante conviene escribir el código en el orden que mejor se pueda entender y mejor documente el programa.

Sirven para especificar:

- Interconexiones entre componentes.
- Estructuras jerárquicas.
- Estructuras regulares.
- Transferencias entre registros.

Sentencia PROCESS (I)

- **Es concurrente, sin embargo, todas las sentencias contenidas en su parte descriptiva son de tipo secuencial.**
  - Las sentencias secuenciales se ejecutan en el mismo paso o ciclo de simulación.
- **Puede convivir con otras sentencias PROCESS o de cualquier otro tipo.**

Sentencia PROCESS (II)

- **Si en el dominio de una arquitectura hay más de una sentencia PROCESS, éstas se ejecutan concurrentemente: sin un orden prefijado.**
  - No hay una forma de indicar qué proceso se ejecutara primero y cuales después.
  - Existe un mecanismo para controlar el orden de ejecución declarando un proceso como POSTPONED.
    - Indica que la ejecución de ese proceso se realizara después del resto de los procesos no identificados de esta manera.

Sentencia PROCESS (III)

- **Sintaxis:**

[Etiqueta]: **PROCESS** [**(**lista de sensibilidad**)**] [**IS**]

Parte Declarativa

Subprogramas, tipos y subtipos de datos, constantes, variables, cláusula USE, ficheros, alias, atributos.

No pueden declararse señales, solamente elementos secuenciales.

BEGIN

Sentencias secuenciales.

WAIT

Asignación de señal.

Parte Descriptiva

END [**POSTPONED**] **PROCESS** [etiqueta]**;**

Asignación concurrente de señal (I)

- **Asigna valores a las señales.**
  - Es equivalente a una sentencia PROCESS.
- **En concurrente siempre que no se incluya dentro de un PROCESS.**
- **Sintaxis:**

  [etiqueta:] nombre**<=** [**TRANSPORT** | (**REJECT** tiempo) **INERTIAL** señal|] [**AFTER**] tiempo [**UNAFFECTED**]

  - **REJECT**: rechaza los pulsos menores de un determinado tiempo. Debe ir acompañado de INERTIAL.
  - **UNAFFECTED**: cuando ha de realizarse una asignación pero no se realiza ninguna acción.

Asignación concurrente de señal (II)

- **Ejemplos:**

```
y<= REJECT 10 ns INERTIAL a;
y<= a AFTER 10 ns;
y<= INERTIAL a AFTER 10 ns;
y<= REJECT 10 ns INERTIAL a AFTER 20 ns;
```

Asignación condicional de señal

- **Sintaxis:**

  señal<= valor1 **WHEN** condición **ELSE** valor2;

  señal<= valor1 **WHEN** condición1 **ELSE** valor2
  WHEN condicion2 [**ELSE** valor3];

- **Ejemplo:**

```
ARCHITECTURE flujo OF comparador IS
BEGIN
  igual<= '1' WHEN a=b ELSE '0';
  mayor<= '1' WHEN a>b ELSE '0';
  menor<= '1' WHEN a<b ELSE '0';
END flujo;
```

Asignación selectiva de señal

Sintaxis:

WITH selector **SELECT**

señal**<=** valor1 **WHEN** selección1,
señal**<=** valor2 **WHEN** selección2,
señal**<=** valorN **WHEN** selecciónN;

Ejemplo:

```
ARCHITECTURE flujo OF decodificador IS
   SIGNAL seleccion: BIT_VECTOR(1 DOWNTO 0);
BEGIN
   seleccion<= sel1 & sel0;
   WITH seleccion SELECT
        sal<= "0001" WHEN "00",
              "0010" WHEN "01",
              "0100" WHEN "10",
              "1000" WHEN "11",
              "0000" WHEN OTHERS;
END flujo;
```

Llamada concurrente a subprogramas (I)

- **Un procedimiento llamado concurrentemente se comporta exactamente como un PROCESS:**
  - Externamente la ejecución es concurrente e internamente secuencial.
- **Los argumentos del procedimiento desempeñan el papel de lista sensible.**
  - Si no hay argumentos en la lista sensible, ni tampoco una sentencia WAIT, habrá que especificarlo, ya que sino seria considerado un bucle infinito.
- **Sintaxis:**

  [etiqueta**:**] **POSTPONED** nombreProcedimiento**(**lista**);**

  [etiqueta**:**] nombreFuncion**(**lista de parametros**);**

Sentencia BLOCK (I)

- **Define una porción del diseño dentro de la parte descriptiva de una arquitectura.**
- **Los bloques pueden estar anidados jerárquicamente.**
- **Sintaxis:**

```
etiqueta: BLOCK [(expresion_de guarda)] [IS]
            [parte declarativa: GENERIC (GENERIC MAP),
            PORT (PORT MAP)]
          BEGIN
             sentencias concurrentes
          END BLOCK etiqueta;
```

Sentencia BLOCK (II)

- **Ejemplo.**

```
-- La etiqueta es obligatoria
rom: BLOCK
        PORT(direccion: IN bit_vector(15  DOWNTO 0);
             enable: IN bit;
             dato: OUT bit_vector(7  DOWNTO 0));

        PORT MAP(direccion=> ROM_dir, enable=> ROM_ena,
             dato=> ROM_dat);
     BEGIN
          ...
     END BLOCK rom;
```

Sentencia BLOCK (III)

- **Las ejecución de sentencias dentro de un BLOCK pueden estar supeditada al cumplimiento de una condición de vigilancia GUARDED.**

- **Solamente cuando se cumple la condición que acompaña al BLOCK, el resultado de la expresión que sigue al GUARDED se asigna.**
  - Si una asignación dentro de un BLOCK no tiene la palabra clave GUARDED se ejecutará aunque no se cumpla la condición de guarda.

Sentencia BLOCK (IV)

Sentencia BLOCK.

- Ejemplo:

```
ARCHITECTURE flujo_OF biestableD IS
BEGIN
    bD: BLOCK((clk='1' and (not clk'stable)
                or clear='0')
    BEGIN
       q<= GUARDED (d and clear) AFTER retardo;
    END BLOCK bD;
END flujo;
```

Instanciación de componentes (I)

- **Sintaxis:**

  etiqueta: nombreComponente **GENERIC MAP** (valores) **PORT MAP** (conexiones);

  - Ejemplo:

```
U1: biestableD GENERIC MAP(1 ns) PORT MAP (d,
  clk, clear, q);
```

  - Mismo ejemplo de otra manera:

```
U1: biestableD GENERIC MAP(retardo=> 1 ns) PORT
  MAP (d=>d, clk=>clk, clear=>clear, q=>q);
```

Instanciación de componentes (II)

- **Requiere, además, incluir la definición del componente en la parte declarativa de la arquitectura.**
- **Declaración del componente:**

  [etiqueta:] **COMPONENT** nombre
  GENERIC(locales**);**
  PORT(locales**);**
  END COMPONENT [etiqueta]**;**

  - Ejemplo:

```
COMPONENT biestableD
  GENERIC(retardo: TIME);
  PORT(d, clk, clear: IN BIT; q: OUT BIT);
END COMPONENT;
```

Instanciación de componentes (III)

Especificación de configuraciones.

- Asociación de entidades y arquitecturas a cada componente.
  - Si no se utiliza esta estructura se supone que la arquitectura asociada (y la entidad) es el última compilada.
  - También se pueden realizar estas asociaciones en las unidades de configuración (CONFIGURATION).
- Sintaxis:

  FOR etiqueta**:** nombreComponente **USE ENTITY**
  WORK.nombreEntidad**(**nombreArquitectura**);**
- Ejemplo:

```
FOR U0: reloj USE ENTITY WORK.reloj(reloj_arq);
```

Sentencia GENERATE (I)

Sentencia GENERATE.

- Equivale a un bucle hardware.
  - Se usa para replicar partes del modelo.
  - Es útil para describir series de componentes (registros, memorias).
- Sintaxis:

```
etiqueta: IF condición GENERATE
            [declaraciones]
          BEGIN
            [declaraciones]
          END GENERATE [etiqueta];
```

Sentencia GENERATE (II)

- Sintaxis (continuación):

etiqueta**: FOR** parametroRepetitivo **GENERATE**
[declaraciones]
BEGIN
[declaraciones]
END GENERATE [etiqueta]**;**

- Ejemplo 1:

```
G0: IF n<4 GENERATE
    U1: and2 PORT MAP(e1,e0,s)
END GENERATE;
```

Sentencia GENERATE (III)

- Ejemplo 2.

```
ARCHITECTURE arq0 OF registroParaleloParaleloCuatroBits IS
   COMPONENT biestableD
       PORT(d, clk, clear: IN BIT; q: OUT BIT);
   END COMPONENT;
   FOR ALL: biestableD USE ENTITY WORK.biestableD(bD_flanco_b);
BEGIN
   G1: FOR i IN 0 TO 3 GENERATE
      Ui: biestableD PORT MAP(entDatos(i),clk,
                              clear,salDatos(i));
   END GENERATE;
END registroParaleloParaleloCuatroBits;
```

Sentencias secuenciales (I)

- **Siempre están encapsuladas en procesos (*PROCESS*) y procedimientos (*PROCEDURE*).**
  - Especifican los algoritmos paso a paso. El orden de escritura determina el momento de la ejecución.
  - Similares a las de otros lenguajes de programación.

Sentencia WAIT (I)

- **Una sentencia PROCESS siempre tiene que contener en su parte descriptiva una sentencia WAIT.**
  - En caso contrario su ejecución se interpreta como un bucle infinito del que no se sale.

- **Sintaxis:**

  WAIT ON lista_señales;

  WAIT UNTIL condición;

  WAIT FOR tiempo;

Sentencia WAIT (II)

WAIT ON lista_señales.

- El proceso se activa cuando se produce un evento en alguna de las señales de la lista de sensibilidad.
- Es equivalente poner entre paréntesis las señales junto con la palabra PROCESS.
  - En el caso de utilizar esta última estructura no puede haber otro WAIT explícito en su parte descriptiva:

```
PROCESS(e1,e0)
BEGIN
   ...
   sentencias secuenciales
   ...
END PROCESS;
```

```
PROCESS
BEGIN
   ...
   sentencias secuenciales
   WAIT ON e1,e0;
END PROCESS;
```

Sentencia WAIT (III)

- **WAIT UNTIL condición;**
  - Ejemplo:

```
WAIT UNTIL contador >7;
```

- **WAIT FOR tiempo;**
  - Ejemplo:

```
WAIT FOR 5 ns;
```

- **Ejemplos combinados:**

```
WAIT ON interrupcion FOR 25 ns;
WAIT ON interrupcion UNTIL contador>7 FOR 25 ns;
```

Asignación secuencial de señal

- **No toman el valor que se les asigna de forma inmediata.**
- **No se actualizan sus drivers hasta que no se haya terminado de ejecutar completamente el PROCESS.**
- **Sintaxis:**

  [etiqueta:] nombre**<=** [**TRANSPORT** | (**REJECT** tiempo) **INERTIAL** señal|] [**AFTER**] tiempo

  - **REJECT**: rechaza los pulsos menores de un determinado tiempo. Debe ir acompañado de INERTIAL.
  - **UNAFFECTED**: cuando ha de realizarse una asignación pero no se realiza ninguna acción.

Asignación de variable

- **La sentencia de asignación de variable reemplaza el valor actual de la variable con el nuevo valor resultante de evaluar una expresión.**
  - La variable y el resultado de la expresión tienen que ser del mismo tipo.
- **Las variables toman el valor que se les asigna de forma inmediata.**
- **Sintaxis:**

  [etiqueta:] nombreVariable**:=** expresion;

Diferencias entre señal variable (I)

- **Ejemplo.**

```
ENTITY diferenciaSenalVariable IS
     PORT(a,b:IN integer; x,y: OUT integer);
END diferenciaSenalVariable;
```

```
TEST
...
      a<= 2 AFTER 1 ns;
      b<= 4 AFTER 1 ns;
...
```

Diferencias entre señal variable (II)

- **Ejemplo (continuación).**

```
ARCHITECTURE conSenal
OF diferenciaSenalVariable
IS
        SIGNAL c: integer;
BEGIN
        PROCESS(a,b,c)
        BEGIN
                c<= a;
                x<= c;
                c<= b;
                y<= c;
        END PROCESS;
END conSenal;
```

```
ARCHITECTURE conVariable
OF diferenciaSenalVariable
IS
BEGIN
        PROCESS(a,b)
        VARIABLE c: integer;
        BEGIN
                c:= a;
                x<= c;
                c:= b;
                y<= c;
        END PROCESS;
END conVariable;
```

Diferencias entre señal variable (III)

- **Ejemplo (continuación).**

ns	Delta	a	b	x	y
1	0	2	4	0	0
1	2	2	4	4	4

•Con señal

Conclusión:

c<= a;

es superflua

ns	Delta	a	b	x	y
1	0	2	4	0	0
1	1	2	4	2	4

•Con variable

Sentencia IF

- **Sintaxis:**

```
[etiqueta:] IF condición THEN
                sentencias secuenciales
            [ELSE
                sentencias secuenciales]
            END IF  [etiqueta];
```

```
[etiqueta:] IF condición THEN
                sentencias secuenciales
            ELSIF condición THEN
                    sentencias secuenciales
                [ELSE
                    sentencias secuenciales]
            END IF  [etiqueta];
```

Sentencia CASE

- **Sintaxis:**

```
[etiqueta:] CASE expresión IS
                WHEN elección1=>
                    secuencia de sentencias1;
                WHEN elección2=>
                    secuencia de sentencias2;
                WHEN OTHERS=>    -- Resto de casos
                    secuencia de sentenciasN;
    END CASE [etiqueta];
```

Sentencia LOOP

- **Sintaxis:**

  [etiqueta:] [forma de iteración FOR | WHILE] **LOOP**

  secuencias secuenciales

  END LOOP [etiqueta];

```
[etiqueta:] FOR identificador IN rango
LOOP
     secuencia de sentencias
END LOOP [etiqueta];
```

```
[etiqueta:] WHILE condición
LOOP
     secuencia de sentencias
END LOOP [etiqueta];
```

Sentencia NEXT

- **Detiene la ejecución de la iteración actual y pasa a la siguiente iteración.**
  - Solamente puede aparecer dentro de una sentencia LOOP.

- **Sintaxis:**

  [etiqueta:] **NEXT** [identificador bucle] [**WHEN** condición]**;**

- **Ejemplo:**

```
-- Interrumpe el FOR y sigue por el WHILE
termina: WHILE a< 100  LOOP
            --- sentencias
            sigue: FOR n IN 0 TO 100
                        --- sentencias
                        NEXT termina WHEN n=a;
                   END LOOP sigue;
         END LOOP termina;
```

Sentencia EXIT

- **Detiene la ejecución en ese instante y sale del bucle.**
  - Si hay varios bucles anidados, sale de donde se encuentre la instrucción o bien del bucle que se especifica en la etiqueta.
- **Solamente puede aparecer dentro de una sentencia LOOP.**
- **Sintaxis:**

[etiqueta:] **EXIT** [identificador bucle] [**WHEN** condición];

Sentencia NULL

- **No realiza función alguna. Pasa la ejecución a la siguiente sentencia secuencial.**
  - Útil en sentencias CASE cuando no se quiere realizar ninguna acción para alguna de las elecciones.

- **Sintaxis:**

  [etiqueta:] **NULL**;

Sentencia ASSERT (I)

- **Comprueba si una determinada condición booleana es cierta durante la simulación.**
  - Si no es cierta, se emite un mensaje y el tipo de error.
- **Esta sentencia es considerada secuencial si se encuentra dentro de una sentencia PROCESS o un PROCEDURE.**
  - Es útil para test.
- **Sintaxis:**

  [etiqueta**:**] **ASSERT** condición [**REPORT** expresión] [**SEVERITY** valor]**;**

Sentencia ASSERT (II)

- **Si REPORT esta presente tiene que incluirse una cláusula previa de definición de tipo de datos STRING con el mensaje que se desea dar.**

- **Si SEVERITY está presente tiene que especificarse el valor de tipo enumerado que aparece en el paquete STANDARD (NOTE, WARNING, ERROR, FAILURE).**

- **Ejemplo:**

```
ASSERT set= '1' AND reset= '1'
REPORT "¡¡set y reset activas al mismo tiempo!!"
SEVERITY ERROR;
```

Llamadas a procedimientos (I)

- **Es secuencial si es llamado por un PROCESS u otro PROCEDURE.**
  - Si es llamado por una ENTITY, ARCHITECTURE o BLOCK entonces es concurrente.

- **Sintaxis:**

  [etiqueta:] **PROCEDURE** nombre(listaParámetros);

Llamadas a procedimientos (II)

Formas de pasar argumentos:

- Con un ejemplo:

```
PROCEDURE limites(CONSTANT valores: IN vector_16;
VARIABLE min: OUT integer; VARIABLE max:OUT
integer:= 8);
```

① Posicional: tiene que escribirse en el mismo orden que se declaran:

```
limites(conjunto(0 TO 15),lim_inf,lim_sup);
```

② Asociación explícita: permite poner los parámetros en cualquier orden.

```
limites(min=>lim_inf,max=>lim_sup,valores=>
conjunto(0 TO 19));
```

Declaración de procedimientos (I)

- **Sintaxis:**

```
PROCEDURE nombre [listaParámetros: tipo[:=valor]] IS
    [parte declarativa]
BEGIN
    [sentencias secuenciales]
END [PROCEDURE] nombre;
```

  - Notas:
    - Los elementos que se declaran sólo son visibles en el cuerpo del procedimiento.
    - Similares a los de la sentencia PROCESS.
    - No se pueden declarar señales.
    - La lista de parámetros se declara de manera similar a como se declaraban los puertos en las entidades.

Declaración de procedimientos (II)

- **La lista de parámetros se declara de manera similar al PORT de una ENTITY:**
  1) Tipo de objeto (constante, variable, señal, fichero).
  2) Nombre del objeto.
  3) Modo del PORT (IN, OUT, INOUT).
  4) Tipo de dato.
  - Notas:
    - Las constantes pueden ser solo de modo IN.
    - Las variables pueden ser de modo IN, OUT y no necesitan ser declaradas.
    - Las señales pueden ser de modo IN, OUT, INOUT y necesitan ser declaradas.
      - Su uso como parámetros no es aconsejable, ya que puede dar lugar a confusión por la forma en que éstas actualizan su valor.

Declaración de procedimientos (III)

- **Ejemplo:**

```
PROCEDURE limites(CONSTANT valores: IN vector_16;
                  VARIABLE min, max: OUT integer) IS
   VARIABLE indice: integer;
BEGIN
   min:= valores(0);
   max:= valores(0);
   FOR indice IN 1 T0 15 LOOP
         IF min> valores(indice) THEN
              min:= valores(indice);
         END IF;
         IF max< valores(indice) THEN
              max:= valores(indice);
         END IF;
   END LOOP;
END limites;
```

Sentencias secuenciales (XXVIII)

- **Funciones.**
  - Siempre devuelven un valor.
  - Sus parámetros sólo pueden ser de modo IN.
  - Siempre se usan en expresiones.
  - El objeto por defecto es una CONSTANT, aunque también admiten señales, pero hay que definirlas mediante SIGNAL.
  - No pueden incluir sentencias WAIT.

Llamada a funciones

- **Es secuencial si es llamado por un PROCESS u otro PROCEDURE.**
  - Si es llamado por una ARCHITECTURE o BLOCK entonces es concurrente.

- **Sintaxis:**

  [Etiqueta:] nombreFunción(lista de parámetros);

  - Formas de pasar argumentos:
    - Posicional: los parámetros se escriben en el mismo orden que se declaran.

      ```
      temp_F<= Centigrados_A_Fahrenheit(temp_C);
      ```

    - Asociación implícita: permite poner los parámetros en cualquier orden.

      ```
      temp_F<= Centigrados_A_Fahrenheit(c=>temp_C);
      ```

Declaración de funciones (I)

- **Se realiza en la parte declarativa de la arquitectura (ARCHITECTURE), bloque (BLOCK) o paquete (PACKAGE).**
  - Hay que indicar el tipo de dato que devuelve.

- **Sintaxis:**

```
[PURE|IMPURE] FUNCTION nombre [(parámetros)]
RETURN tipoDatosResultado IS
    [declaraciones]
BEGIN
    [sentencias serie]          -- Debe incluir al menos un RETURN
END [FUNCTION] [nombre];
```

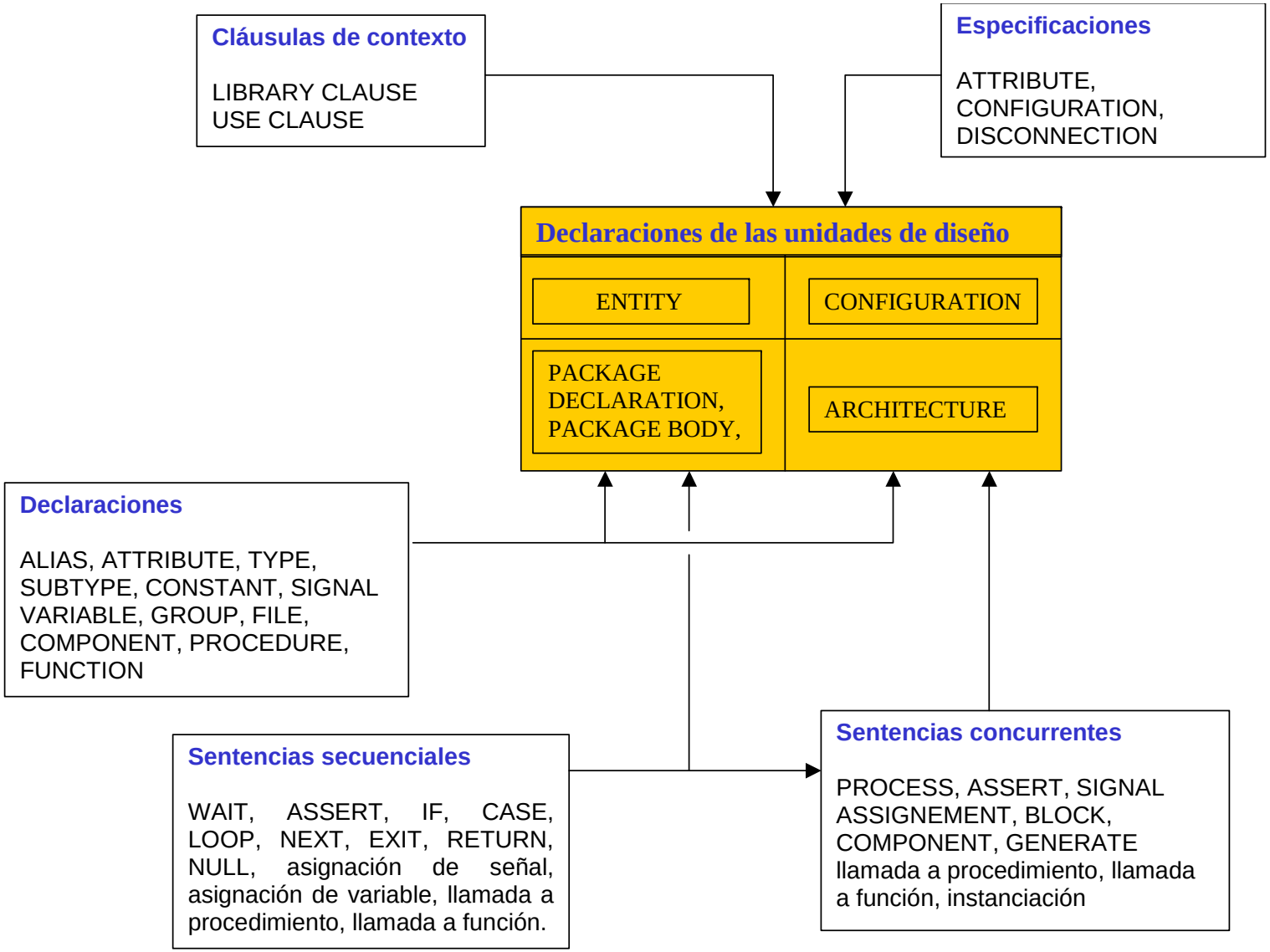
Declaración de funciones (II)

- PURE devuelve siempre el mismo valor para unos parámetros de entrada.
- IMPURE para unos mismos parámetros puede devolver valores diferentes.

Ejemplo anterior:

```
FUNCTION Centigrados_A_Fahrenhait(c: real) RETURN real IS
     VARIABLE f: real;
BEGIN
     f:= c* 9.0/5.0 + 32.0;
     RETURN(f);
END FUNCTION;
```

Resumen. Elementos de VHDL



Recursos VHDL y estilos de especificación

- **Especificación de comportamiento.**
  - PROCESS.
- **Especificación de flujo.**
  - Asignación concurrente de señal.
  - Asignación condicional de señal
  - Asignación selectiva de señal.
  - BLOCK.
- **Especificación estructural.**
  - Instanciación de componentes.
  - GENERATE.