Sesión 2: VHDL secuencial

Elementos léxicos de VHDL (1)

Delimitadores :

Simples: & ' ( ) *+ , - . / : ; < = >/

Compuestos: => ** := /= >= <= << >> <>

Identificadores:

identificador ::= letra { [subrayado] letra_o_digito}
letra_o_dígito ::= letra | dígito
letra ::= letra_mayúscula | letra_minúscula

En los identificadores es indiferente la caja (mayúscula o minúscula) de las letras, de manera que bit, Bit y BIT tienen el mismo significado.

Elementos léxicos de VHDL (2)

Números :

Base decimal:

número_decimal ::= entero [. entero] [exponente]
entero ::= dígito { [subrayado] dígito }
exponente ::= E [+] entero | E – entero

Ejemplos:

```
0          1          123_456_789        987E6     -- números enteros
0.0        0.5        2.718_28           12.4E-9   -- números reales
```

Base no decimal:

número_base ::= base # entero_base [. entero_base] # [exponente]
base ::= entero
entero_base ::= digito_extendido { [subrayado] digito_extendido }
digito_extendido ::= dígito | letra

Ejemplos:

```
2#1100_0100#          16#C4#    4#301#E1          -- el número entero 196
2#1.1111_1111_111#E+11          16#F.FF#E2        -- el número real 4095.0
```

Elementos léxicos de VHDL (3)

Caracteres:

Se representan con comillas simples sobre el carácter ASCII.

Ejemplos:

'A' '*' ''' ' '

Cadenas de caracteres (*strings*):

Se escriben colocando dobles comillas a la secuencia de caracteres.
*Se pueden utilizar como valor de un objeto de tipo array de caract*eres.

Ejemplos:

"Unacadena"
"" -- cadena vacía
"Una cadena en: ""Una cadena"". " -- cadena que contiene comillas

Elementos léxicos de VHDL (4)

Cadenas de bits

Especifican arrays de tipo bit ('0's y '1's):

cadena_de_bits ::= especificador_base "valor_bit"
especificador_base ::= B | O | X
valor_bit ::= digito_extendido { [subrayado] digito_extendido }

Ejemplos:

```
B"1010110"      -- la longitud es 7
O"126"          -- la longitud es 9, equivalente a B"001_010_110"
X"56"           -- la longitud es 8, equivalente a B"0101_0110"
```

Comentarios:

Comienzan con guiones (--) y se extienden hasta el final de la línea.

Objetos de datos VHDL (1)

Variables

Se utilizan de almacenamiento local en procesos y subprogramas .

Declaración:

VARIABLE nombres_de_variables : tipo [:= valor_inicial];

Ejemplos:

VARIABLE v1,v2 : bit := '1';

VARIABLE color_luz : tres_colores := rojo;

VARIABLE tab : tipo_tabla(0 TO 4) := (2,3,4,-2,0);

Objetos de datos VHDL (2)

Constantes

Son nombres asignados a valores específicos de un cierto tipo.

Declaración:

CONSTANT nombre_de_constante : tipo [:= valor];

Ejemplos:

```
CONSTANT  dirección : bit_vector := "00111101";

CONSTANT tabbit : bit_vector(1 TO 9) := (1 TO 3 =>'0', OTHER => '1');

CONSTANT tab : tipo_tabla(0 TO 4) := (2,3,4,-2,0);
```

Objetos de datos VHDL (3)

Señales

Son el medio de comunicación de datos dinámicos entre procesos.

Declaración:

SIGNAL nombres_de_señales : tipo [:= valor_inicial];

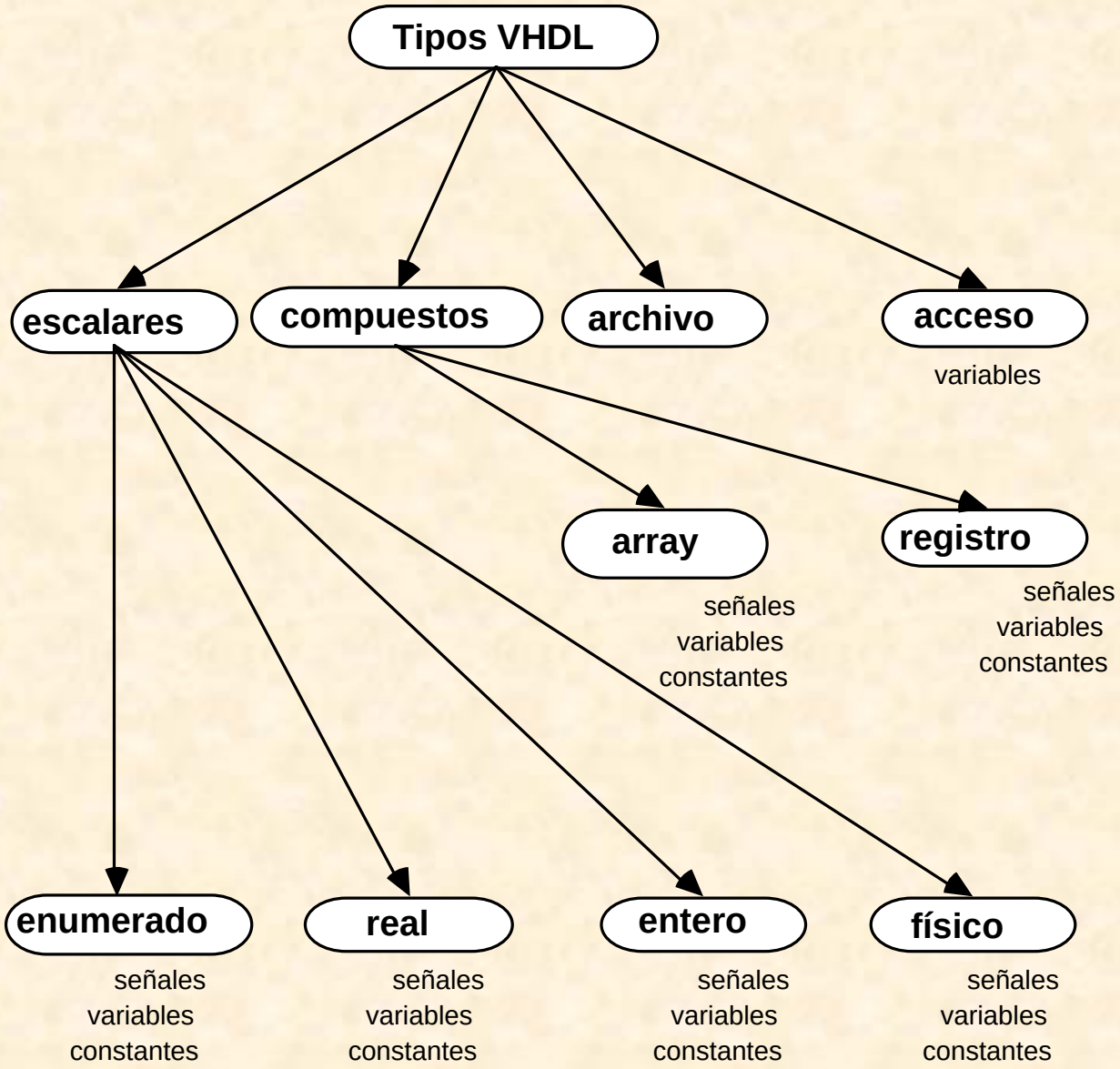
Ejemplos:

```
SIGNAL bus_direcciones : integer := 2**nb_bit - 1;

SIGNAL tabbit : bit_vector(1 TO 9) := (1 TO 3 =>'0', OTHER => '1');

SIGNAL tab : tipo_tabla(0 TO 4) := (2,3,4,-2,0);
```

Tipos de datos (1)



Tipos de datos (2)

Tipo entero (*integer*)

- *Se corresponde con los enteros matemáticos.*
- *Tiene asociado un conjunto de funciones matemáticas (suma, resta, etc.).*

Ejemplos:

type integer is range -2147483648 to 2147483647;
-- Definido en el paquete STANDARD y por tanto disponible directamente

TYPE byte_ent IS RANGE 0 TO 255;
TYPE palabra_entero_signo IS RANGE –32768 TO 32767;
TYPE indice_bit IS RANGE 31 DOWNTO 0;

Tipo real o de coma flotante (*real*)

- *Es una aproximación discreta a los números reales matemáticos.*
- *Se declara con la misma sintaxis que los enteros.*

Ejemplos:

type real is range -1.0E308 to 1.0E308;
-- Definido en el paquete STANDARD y por tanto disponible directamente

TYPE nivel_señal IS RANGE -10.00 TO +10.00;
TYPE probabilidad IS RANGE 0.0 TO 1.0;

Tipos de datos (3)

Tipo enumerado

- *Son conjuntos ordenados de identificadores o caracteres.*
- *Los elementos de un tipo enumerado deben ser distintos.*
- *Un mismo elemento puede aparecer en diferentes tipos enumerados.*

Ejemplos:

TYPE cuatro_estados IS ('X', '0', '1', 'Z');
TYPE color IS (rojo, amarillo, azul, verde, naranja);

Tipos enumerados definidos en el paquete *estandard* de VHDL:

```
TYPE severity_level IS (note, warning, error, failure);
TYPE boolean IS (false, true);
TYPE bit IS ('0', '1');
TYPE  character IS (…….
        DLE,    DC1,    DC2,    DC3,    DC4,    NAK,    SYN,    ETB,
        CAN, EM,SUB,    ESC,    FSP,    GSP,    RSP,    USP,
        ' ',    '!',    '"',    '#',    '$',    '%',    '&',    ''',
        '(',    ')',    '*',    '+',    ',',    '-',    '.',    '/',
        '0',    '1',    '2',    '3',    '4',    '5',    '6',    '7',
        ……..);
```

Tipos de datos (4)

Tipo físico

• *Es un tipo enumerado que se utiliza para representar cantidades físicas.*

Ejemplo:

```
TYPE corriente IS RANGE 0 to 10000000
        UNITS
                na;                     -- nano amperio
                ua      = 1000 na;      -- micro amperio
                ma      = 1000 ua;      -- mili amperio
                a       = 1000 ma;      -- amperio
        END UNITS;
```

Solo existe un tipo físico predefinido en el lenguaje (paquete STANDARD): TIME

```
TYPE TIME IS RANGE -2147483647 to 2147483647
        UNITS
                fs;                     -- femtosegundo
                ps      = 1000 fs;      -- picosegundo
                ns      = 1000 ps;      -- nanosegundo
                us      = 1000 ns;      -- microsegundo
                ms      = 1000 us;      -- milisegundo
                sec     = 1000 ms;      -- segundo
                min     = 60 sec;       -- minuto
                hr      = 60 min;       -- hora
        END UNITS;
```

Tipos de datos (5)

Tipo array

- *Es una colección indexada de elementos del mismo subtipo.*
- *Cada elemento del array puede ser accedido por uno (array unidimensional) o más (array multidimensional) índices.*
- *Un array se dice restringido cuando los límites del índice se establecen en la declaración del tipo. En caso contrario se dice no restringido, fijándose los límites posteriormente*

Ejemplos:

Array restringidos

```
TYPE palabra IS ARRAY (31 DOWNTO 0) OF bit;
TYPE transform IS ARRAY (1 TO 4, 1 TO 4) OF real;
TYPE banco_registros IS ARRAY (byte RANGE 0 TO 132) OF integer;
```

Array no restringido

```
TYPE vector IS ARRAY (integer RANGE <>) OF real;
```

Tipos array definidos en el paquete *estandard* de VHDL:

```
type string is array (positive range <>) of character;
type bit_vector is array (natural range <>) of bit;
```

Tipos de datos (6)

Acceso a los elementos de un array:

Ejemplo 1 (base escalar):

```
TYPE bus_de_datos IS ARRAY(0 TO 7) OF BIT;
VARIABLE  X : bus_de_datos;
VARIABLE  Y : BIT;
Y := X(0);
Y := X(15);
```

Ejemplo 2 (base otro array)

```
TYPE  memoria IS ARRAY (0 TO 7) OF bus_de_datos;
CONSTANT rom : memoria := (
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'),
                                    ('0','0','0','0','1','1','0','0'));
```

Cuarta palabra de la *rom* : X := rom(3);

Cuarto bits de esta palabra: Y := rom(3)(3);

Tipos de datos (7)

Subtipos

•*Se utilizan para definir subconjuntos de tipos ya declarados.*

• *Dos casos de subtipos:*

- *El primero puede restringir valores de un tipo escalar para que se ajusten a un rango especificado (una restricción de rango).*

  Ejemplo:
  SUBTYPE num_pines IS integer RANGE o TO 400;
  SUBTYPE dígitos IS character RANGE '0' TO '9';

- *El segundo un array no restringido especificando límites a los índices.*

  Ejemplo:
  SUBTYPE id IS string(1 TO 20);
  SUBTYPE palabra IS bit_vector(31 DOWNTO 0);

Subtipos de integer definidos en el paquete Standard:

subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;

Atributos

• Para cualquier **tipo o subtipo escalar *t*,** se pueden utilizar los siguientes atributos:

Atributo	Resultado
t'LEFT	Límite izquierdo de t
t'RIGHT	Límite derecho de t
t'LOW	Límite inferior de t
t'HIGH	Límite superior de t

Para un rango ascendente: t'LEFT = t'LOW, y t'RIGHT = t'HIGH.
Para un rango descendente: t'LEFT = t'HIGH, y t'RIGHT = t'LOW

• Para cualquier **tipo o subtipo discreto o físico *t***, *x* un miembro de *t*, y *n* un entero:

Atributo	Resultado
t'POS(x)	Número de posición de x en t
t'VAL(n)	Valor en la posición n de t
t'LEFTOF(x)	Valor en t de la posición a la izquierda de x
t'RIGHTOF(x)	Valor en t de la posición a la derecha de x
t'PRED(x)	Valor en t de la posición inmediata inferior a x
t'SUCC(x)	Valor en t de la posición inmediata superior de x

Para un rango ascendente: t'LEFTOF(x) = t'PRED(x), y t'RIGHTOF(x) = t'SUCC(x)
Para un rango descendente: t'LEFTOF(x) = t'SUCC(x), y t'RIGHTOF(x) = t'PRED(x)

Atributos

• Para cualquier **tipo u objeto array** ***a***, y siendo *n* un número entero entre 1 y el número de dimensiones de *a*, se puede utilizar los siguientes atributos:

Atributo	Resultado
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
a'RANGE(n)	Rango de la dimensión n de a
a'REVERSE_RANGE(n)	Inverso del rango de la dimensión n de a
a'LENGTH(n)	Longitud del rango de la dimensión n de a

Operadores predefinidos y expresiones

Grupo	Símbolo	Función
Aritmético (binarios)	**+** **-** ***** **/** **mod** **rem** ******	suma resta multiplicación división modulo (división entera) resto (división entera) exponenciación
Aritmético (monarios)	**+** **-** **abs**	signo más signo menos valor absoluto
Relacional	**=** **/=** **<** **>** **<=** **>=**	igual no igual menor que mayor que menor o igual que mayor o igual que
Lógico (binarias)	**and** **or** **nand** **nor** **xor**	'y' lógica 'o' lógica 'no-y' lógica 'no-o' lógica 'o-exclusiva' lógica
Lógico (monarios)	**not**	complemento
Concatenación	**&**	concatenación

Sentencias Secuenciales (1)

Sentencia de asignación de variable (:=)

• *Reemplaza el valor de una variable con un nuevo valor especificado.*

Sintaxis:

nombre_variable := expresión;

Ejemplos:

```
var := 0;

a := func(dato, 4);

varint := func(dato, 4) + 243 * func2(varint) - 2 ** dato;
```

Sentencias Secuenciales (2)

Sentencia IF

•*Seleccionan la ejecución de sentencias dependiendo de una o más condiciones.*

Sintaxis:

IF condición THEN sentencias_secuenciales
{ELSIF condición THEN sentencias_secuenciales}
[ELSE sentencias_secuenciales]
END IF;

Ejemplo:

```
ENTITY nand2 IS
  PORT(a,b : IN bitx01; c : OUT bitxo1);
END nand2;
ARCHITECTURE nand2 OF nand2 IS
BEGIN
  PROCESS
    VARIABLE temp : bitx01;
  BEGIN
    temp := a andx01 b;
    IF (temp = '1') THEN
      c <= temp AFTER 6 ns;
    ELSIF (temp = '0') THEN
      c <= temp AFTER 5 ns;
    ELSE
      c <= temp AFTER 6 ns;
    END IF;
    WAIT ON a,b;
  END PROCESS;
END nand2;
```

Sentencias Secuenciales (3)

Sentencia Case

Selecciona una secuencia de sentencias en función del valor de una expresión.

Sintaxis:

CASE expresión IS
alternativa {alternativa}
END CASE;
alternativa ::= WHEN elecciones => sentencias_secuenciales
sentencias_secuenciales ::= {sentencia_secuencial}
elecciones ::= elección{\| elección}
elección ::= expresión_simple \| rango_discreto \| nombre_simple \| OTHERS

Ejemplo:

```
ENTITY muxc IS
  PORT (i3,i2,i1,i0,x1,x0 : IN BIT; z : OUT BIT);
END muxc;

ARCHITECTURE muxc OF muxc IS
BEGIN
  PROCESS
    TYPE peso IS RANGE 0 TO 3;
    VARIABLE muxval : peso;
  BEGIN
    muxval := 0;
  IF (x0 = '1') THEN muxval := muxval + 1;
  END IF;
  IF (x1 = '1') THEN muxval := muxval + 2;
  END IF;

CASE muxval IS
      WHEN 0 =>
        z <= i0 AFTER 10 ns;
      WHEN 1 =>
        z <= i1 AFTER 10 ns;
      WHEN 2 =>
        z <= i2 AFTER 10 ns;
      WHEN 3 =>
        z <= i3 AFTER 10 ns;
    END CASE;
 WAIT ON  i3,i2,i1,i0,x1,x0;
END PROCESS;
```

Sentencias Secuenciales (4)

Sentencias Iterativas (LOOP, NEXT, EXIT)

- *Sirven para formar los bucles iterativos clásicos WHILE y FOR.*
- *NEXT: finaliza la ejecución de la iteración actual e inicia la siguiente; y*
- *EXIT: finaliza la iteración actual y el bucle.*

Sintaxis: ***[rotulo:] [esquema_de_iteración] LOOP***
sentencias_secuenciales
END LOOP [rotulo];
esquema_de_iteración ::= WHILE condición | FOR especificación_de_parámetros
especificación_de_parámetros ::= identificador IN rango_discreto
sentencia_next ::= NEXT [rótulo_del_loop] [WHEN condición];
sentencia_exit ::= EXIT [rótulo_del_loop] [WHEN condición];

Ejemplo:

```
ENTITY and_mask IS
  PORT(x,y,m : IN bit_vector (7 DOWNTO 0); z : OUT bit_vector (7 DOWNTO 0));
END and_mask;
ARCHITECTURE and_mask OF and_mask IS
BEGIN
  PROCESS
  BEGIN
   FOR i IN 0 TO 7 LOOP
    IF (m(i) = '0') THEN
      NEXT;
    ELSE
      z(i) <= x(i) AND y(i);
    END IF;
   END LOOP;
                WAIT ON x,y,m;
  END PROCESS;
END and_mask;
```

Sentencias Secuenciales (5)

Sentencia Null

- *Se utilizan para mostrar explícitamente que no se requiere ninguna acción.*
- *Su uso más frecuente es en sentencias CASE para las acciones por defecto.*

Sintaxis:

NULL

Ejemplo:

```
ENTITY ual IS
  PORT ( i1 , i2, op : IN integer; s : OUT integer);
END ual;
ARCHITECTURE comporta OF ual IS
BEGIN
  PROCESS
    VARIABLE result : integer;
  BEGIN
      CASE op IS
        WHEN 0 => result := i1 + i2;
        WHEN 1 => result := i1 - i2;
        WHEN 2 => result := i1 + 1;
        WHEN 3 => result := i2 + 1;
        WHEN 4 => result := i1 - 1;
        WHEN 5 => result := i2 - 1;
        WHEN 6 => result := i1;
        WHEN OTHERS => NULL;
      END CASE;
      s <= result;
     WAIT ON i1,i2, op;
  END PROCESS;
 END comporta;
```

Sentencias Secuenciales (6)

Sentencia Assert

• *Se utiliza para comprobar una* ***condición*** *e informar en caso que sea falsa.*

Sintaxis:

ASSERT condición [REPORT expresión] [SEVERITY expresión];

Ejemplo:

```
ENTITY tiempo_setup_ffd IS
 PORT(d,clk : IN BIT; q : OUT BIT);
END tiempo_setup_ffd;
ARCHITECTURE tiempo_setup_ffd OF tiempo_setup_ffd IS
BEGIN
  PROCESS     VARIABLE ultimo_cambio : TIME := 0 ns;
    VARIABLE ultimo_valor  : BIT := '0';
    VARIABLE ultimo_valor_clk : BIT := '0';
  BEGIN
    IF (ultimo_valor /= d) THEN
      ultimo_cambio := NOW;
      ultimo_valor := d;
    END IF;
    IF (ultimo_valor_clk /= clk) THEN
      ultimo_valor_clk := clk;
      IF (clk = '1') THEN
        ASSERT (NOW - ultimo_cambio >= 20 ns)
        REPORT "violacion del setup"
        SEVERITY WARNING;
      END IF;
    END IF;
                  WAIT ON d,clk;
  END PROCESS;
END tiempo_setup_ffd;
```

NOW

Función predefinida de VHDLque devuelve el tiempo de simulación

Sentencias Secuenciales (7)

Sentencia de asignación de señal (<=)

- *Es la encargada de planificar transacciones, esto es, pares (valor, tiempo), sobre el driver de una señal.*

Sintaxis:

señal <= [TRANSPORT] forma_de_onda;
forma_de_onda ::= elemento { , elemento}
elemento ::= expresión_valor [AFTER expresión_tiempo]| NULL [AFTER expresión_tiempo]

Ejemplo:

```
ENTITY linea_retardo IS
   PORT(a : IN BIT; b : OUT BIT);
END linea_retardo;
ARCHITECTURE única OF linea_retardo IS
BEGIN
        PROCESS
        b <= TRANSPORT a AFTER 20 ns;
                WAIT ON a;
        END PROCESS;
END única;
```

Sentencias Secuenciales (7)

Sentencia Wait

• *Suspende y activa los procesos para permitir la sincronización del intercambio de información en el dominio concurrente.*

Sintaxis:

WAIT[clausula_sensibilización][clausula_condición][clausula_temporización];
clausula_sensibilización ::= ON lista_sensibilización
lista_sensibilización ::= señal { , señal}
clausula_condición ::= UNTIL condición
condición ::= expresión_booleana
clausula_temporización ::= FOR expresión_tiempo

• *Cuando se ejecuta una sentencia WAIT dentro de un proceso, este suspende y capacita las clausulas de reactivación.*
• *El proceso queda suspendido hasta cumplir los requisitos de reactivación:* ***sensibilización***, ***condición*** *y* ***temporización****.*
• *La lista de* ***sensibilización*** *especifica un conjunto de señales a las que es sensible el proceso mientras está suspendido.*
• *Cuando ocurre un evento (cambio del valor de una señal) sobre alguna de estas señales, el proceso evalúa la condición.*
  • *Si el resultado es* ***true*** *o se ha omitido la condición, el proceso se activa a partir de la siguiente sentencia,*
  •*Si el resultado es* ***false*** *el proceso permanece suspendido.*
• *Si se ha omitido la clausula de sensibilización, el proceso es sensible a todas las señales de la expresión de la condición.*
• *Si se omite, y no existen clausulas de sensibilización o condición, el proceso suspende indefinidamente.*

Ejemplo:

```
ENTITY unidad IS
  PORT(e1, e2, cap : IN bit; s : OUT bit);
END unidad;
ARCHITECTURE unidad OF unidad IS
BEGIN
  PROCESS
  BEGIN
    WAIT ON e1,e2 UNTIL cap = '1' FOR 70 ns;
    s <= e1 AND e2;
  END PROCESS;
END unidad;
```

Subprogramas

Declaración de Subprogramas:

PROCEDURE identificador [parte_formal]
/FUNCTION designador [parte_formal] RETURN tipo
designador ::= identificador / operador
operador ::= literal_cadena
parte_formal ::= (especificación_parámetro { ; especificación_parámetro })
especificación_parámetro ::= lista_identificadores : modo tipo [:= expresión]
modo ::= [IN]/INOUT/OUT

Modo IN: *Permite solo la lectura*
Solo pueden ser objetos de la clase SIGNAL o CONSTANT, por defecto se supone la clase CONSTANT.

Modo OUT: *Permite solo la lectura*, *por defecto se supone la clase VARIABLE.*

Modo INOUT: *Permite tanto la lectura como la escritura.*
Los argumentos de una función solo tienen el modo IN, un procedimiento puede tener IN, INOUT y OUT.

Cuerpo de subprogramas:

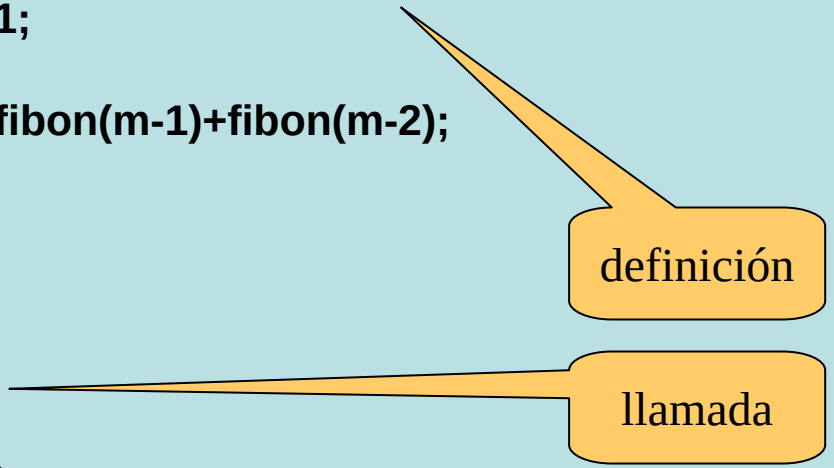
BEGIN
[sentencias_secuenciales]
RETURN expresion <------- sólo para funciones
[sentencias_secuenciales]
END designador;

Todos los subprogramas VHD pueden ser ***recursivos***.

Ejemplo de subprograma

```
ENTITY fibo IS
  PORT(n : IN natural; f : OUT natural);
END fibo;
ARCHITECTURE recursion OF fibo IS
BEGIN
  PROCESS
    FUNCTION fibon(m : natural) RETURN natural IS
    BEGIN
      IF m=0 or m=1 THEN
        RETURN 1;
      ELSE
        RETURN fibon(m-1)+fibon(m-2);
      END IF;
    END fibon;

  BEGIN
    f <= fibon(n);
    WAIT ON n;
  END PROCESS;
END recursion;
```



Sobrecarga(overloading)

Elemento sobrecargado: definido con más de un significado.

Significado concreto: determinado por el contexto local del elemento.

Se pueden sobrecargar: tipos enumerados, subprogramas y operadores.

Ejemplos de tipos enumerados:

```
TYPE bit IS ('0','1');
TYPE bit01x IS ('0','1',X);
TYPE bit 01z IS ('0','1', Z);
TYPE bit01xz IS ('0','1',X,Z);
```

Ejemplos de subprogramas:

```
TYPE  entero IS RANGE 0 TO 255;
TYPE vector IS ARRAY(0 TO 7) OF bit;
FUNCTION despla (a : entero) RETURN entero IS
BEGIN
        RETURN (a/2);
END despla;
FUNCTION despla (a : vector) RETURN vector IS
        VARIABLE result : vector;
BEGIN
        FOR i IN a'RANGE LOOP
                IF i = a'HIGH THEN result(i) := '0';
                        ELSE result(i) := a (i+1);
                EN IF;
        END LOOP;
        RETURN result;
END despla;
```

Practica 2: VHDL secuencial

Objetivos

1. Utilización de los objetos, tipos y construcciones secuenciales de VHDL.
2. Modelado del comportamiento de sistemas.

Práctica a realizar

Diseño de un operador matemático *con 2 entradas de datos de tipo* ***natural*** *y una entrada de operación tambien de tipo* ***natural****, que implemente al menos 3 operaciones matemáticas complejas, siendo una de ellas el* ***m.c.d.(entradas de datos).*** *La salida del resultado tendrá un retardo proporcional a la complejidad (número de operaciones aritméticas simples) de la operación compleja.*

Ejemplo*:* ***d = operador (a,b,c):***

d = m.c.d.(a, b)	***si c = 0 (utilizar el algoritmo de Euclides)***
d = m.c.m.(a, b)	*si c = 1*
d = a!	*si c = 2*
d = fibonachi(a)	*si c = 3 (último término de la serie de* ***a*** *elementos)*

Resultados a entregar

Documentación del programa en la que conste:

1. *Especificación* del sistema modelado.
2. *Listado* VHDL comentado del código del programa.
3. *Relación E/S* que muestre la respuesta del sistema frente a entradas significativas.

Algoritmo de Euclides para calcular el máximo común divisor de dos enteros a y b:

Si a >= b Se realizan las siguientes divisiones hasta que el resto de cero:

$a = b * q_1 + r_1$
$b = r_1 * q_2 + r_2$
$r_1 = r_2 * q_3 + r_3$
$r_2 = r_3 * q_4 + r_4$
.....
$r_n - 1 = r_n q_n + 1 + 0$

Siendo q_1, q_2, ... los cocientes y r_1, r_2, ... los restos.

m.c.d.(a, b) = r_n

Ejemplo: m.c.d.(200, 162)

200 = 162 x 1 + 38
162 = 38 x 4 + 10
38 = 10 x 3 + 8
10 = 8 x 1 + 2
8 = **2** x 4 + 0

m.c.d.(200, 162) = 2. (Como se han realizado 5 divisiones el retardo será de 5 *ns*)