Resumen VHDL'87

Elementos léxicos

Palabras reservadas

ABS	BUS	FUNCTION	NAND		
ACCESS	CASE	GENERATE	NEW	PROCESS	TRANSPORT
AFTER	COMPONENT	GENERIC	NEXT	RANGE	TYPE
ALIAS	CONFIGURATION	GUARDED	NOR	RECORD	UNITS
ALL	CONSTANT	IF	NULL	REGISTER	UNTIL
AND	DISCONNECT	IN	OF	REM	USE
ARCHITECTURE	DOWNTO	INOUT	ON	REPORT	VARIABLE
ARRAY	ELSE	IS	OPEN	RETURN	WAIT
ASSERT	ELSIF	LABEL	OR	SELECT	WHEN
ATTRIBUTE	END	LIBRARY	OTHERS	SEVERITY	WHILE
BEGIN	ENTITY	LINKAGE	OUT	SIGNAL	WITH
BLOCK	EXIT	LOOP	PACKAGE	SUBTYPE	XOR
BODY	FILE	MAP	PORT	THEN	
BUFFER	FOR	MOD	PROCEDURE	TO	

Identificadores

```
CUENTA    cuenta        x1          x_y          X22_34   siguiente_palabra
pp_34s    multiplexor   p500_a  ftp  a_v_u_i XyZ
```

Literales numéricos

```
0                        1                  123_456_789        987E6
0.0            0.5                 2.718_28          12.4E-9
2#1100_0100#             16#C4#      4#301#E1
2#1.1111_1111_111#E+11   16#F.FF#E2
```

Literales carácter

```
'A'      '*'      '''      ' '
```

Cadenas de caracteres

```
"Unacadena"
"Una cadena en: ""Una cadena"". "
```

Cadenas de bits

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

Declaraciones

Declaración de tipos enteros

```
TYPE byte_ent IS RANGE 0 TO 255;
TYPE palabra_entero_signo IS RANGE –32768 TO 32767;
TYPE índice_bit IS RANGE 31 DOWNTO 0;
TYPE contador IS RANGE 31 DOWNTO 0;
```

Declaración de tipos reales

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

Declaración de tipos enumerados

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

Declaración de tipos físicos

```
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;
```

Declaración de tipos *array*

```
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;
TYPE cuenta_color IS ARRAY (color RANGE <>) OF natural;
TYPE array_de_colores IS ARRAY (natural RANGE <>) OF color;
TYPE vector IS ARRAY (integer RANGE <>) OF real;
TYPE bus_de_datos IS ARRAY(0 TO 7) OF BIT;
TYPE memoria IS ARRAY (0 TO 7) OF bus_de_datos;
```

Declaración de tipos registro

```
TYPE instrucción IS
    RECORD
            código  : cod_op;
            fuente  : integer;
            destino : integer;
    END RECORD;

TYPE cod_op IS (sum , res, mul, div, sal);
```

Declaración de subtipos

```
SUBTYPE num_pines IS integer RANGE 0 TO 400;
SUBTYPE dígitos IS character RANGE '0' TO '9';
SUBTYPE id IS string(1 TO 20);
SUBTYPE palabra IS bit_vector(31 DOWNTO 0);
```

Declaración de variables

```
VARIABLE v1, v2 : bit := '1';
VARIABLE color_luz : tres_colores := rojo;
VARIABLE bus_direcciones : integer := 2**nb_bit - 1;
VARIABLE tab : tipo_tabla(0 TO 4) := (2,3,4,-2,0);
VARIABLE tabbit : bit_vector(1 TO 9) := (1 TO 3 =>'0', OTHERS => '1');
VARIABLE tabbit : bit_vector(1 TO 9) := ('0','0','0','1','1','1','1','1','1');
```

Declaración de constantes

```
CONSTANT  dirección : bit_vector := "00111101";
CONSTANT mascara : bit_vector(1 TO 3) := "1010";
CONSTANT tabbit : bit_vector(1 TO 9) := (1 TO 3 =>'0', OTHERS => '1');
CONSTANT tab : tipo_tabla(0 TO 4) := (2,3,4,-2,0);
CONSTANT dirección_max : integer := 2**nb_bit - 1;
CONSTANT retardo : TIME := param_tec(0.025 pF);
```

Declaración de señales

```
SIGNAL s1, s2 : bit := '1';
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);
```

Declaración de alias

```
SIGNAL  bus_instruc  :  bit_vector(31 DOWNTO  0)
ALIAS  código_oper  :  bit_vector(7 DOWNTO 0) IS  bus_instruc(31 DOWNTO 24);
ALIAS fuente  :  bit_vector(3 DOWNTO 0) IS bus_instruc(23 DOWNTO 20);
ALIAS destino  :  bit_vector(3 DOWNTO 0) IS bus_instruc(19 DOWNTO 16);
ALIAS dato  :  bit_vector(15 DOWNTO 0) IS bus_intruc(15 DOWNTO 0);
```

Declaración de componentes

```
COMPONENT puerta_y
        GENERIC(retardo : TIME);
        PORT(x,y : IN bit; z : OUT bit);
END COMPONENT;

COMPONENT puerta_o
        GENERIC(retardo : TIME);
        PORT(x,y : IN bit; z : OUT bit);
END COMPONENT;
```

Especificación de configuración

```
FOR ALL : puerta_y USE ENTITY WORK.and2(comportamiento);
FOR ALL : puerta_o USE ENTITY WORK.or2(comportamiento);

FOR puerta1, puerta2 : puerta_y USE ENTITY WORK.and2(comportamiento);
FOR puerta3 : puerta_o USE ENTITY WORK.or2(comportamiento);
portamiento;
```

Cláusula *library*

```
LIBRARY proyecto1;
LIBRARY IEEE;
```

Cláusula *use*

```
USE proyecto1.utilidad;
USE utilidad.fun;
USE proyecto1.utilidad.fun ;
USE IEEE.std_logic_1164.ALL;
USE STD.TEXTIO.ALL;
```

Atributos

Atributos asociados a tipos o subtipos escalares

Atributo	Resultado
t'LEFT	límite izquierdo del tipo enumerado t
t'RIGHT	límite derecho del tipo enumerado t
t'LOW	límite inferior del tipo enumerado t
t'HIGH	límite superior del tipo enumerado t

Atributos asociados a tipos o subtipos discretos o físicos

Atributo	Resultado
t'POS(x)	la posición de *x* en *t*
t'VAL(n)	el elemento que ocupa la posición *n* en *t*
t'LEFTOF(x)	el elemento que ocupa la posición a la izquierda de *x* en *t*
t'RIGHTOF(x)	el elemento que ocupa la posición a la derecha de *x* en *t*
t'PRED(x)	el elemento que ocupa la posición inmediata inferior a *x* en *t*
t'SUCC(x)	el elemento que ocupa la posición inmediata superiora *x* en *t*

Atributos asociados a *arrays* restringidos

Atributo	Resultado
a'LEFT(n)	el límite izquierdo del rango de la dimensión *n* de *a*
a'RIGHT(n)	el límite derecho del rango de la dimensión *n* de *a*
a'LOW(n)	el límite inferior del rango de la dimensión *n* de *a*
a'HIGH(n)	el límite superior del rango de la dimensión *n* de *a*
a'RANGE(n)	el rango de la dimensión *n* de *a*
a'REVERSE_RANGE(n)	el rango en orden invertido de la dimensión *n* de *a*
a'LENGTH(n)	la longitud del rango de la dimensión *n* de *a*

Atributos de señal

Atributo	Ejemplo
s'EVENT	IF (clk = '1') AND (clk'EVENT) THEN q <= d; END IF;
s'LAST_VALUE	IF (clk = '1') AND (clk'EVENT) AND (clk'LAST_VALUE = '0')
s'LAST_EVENT	ASSERT (d'LAST_EVENT >= 10 ns)
s'DELAYED[(tiempo)]	c <= a'DELAYED(7 ns) AND b'DELAYED(11 ns) AFTER 9 ns;
s'STABLE[(tiempo)]	b <= a'STABLE(10 ns);

Operadores

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

Sentencias Secuenciales

Sentencia de asignación de variable (:=)

```
var := 0;
a := func(dato, 4);
varint := func(dato, 4) + 243 * func2(varint) - 2 ** dato;
```

Sentencia condicional (if)

```
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;
```

Sentencia alternativa (case)

```
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;
```

Sentencia iterativa (for)

```
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;
```

Sentencia de aserto (assert)

```
ASSERT (NOW - ultimo_cambio >= 20 ns)
REPORT "violación del setup"
SEVERITY WARNING;
```

Sentencia de asignación de señal (<=)

```
b <= TRANSPORT a AFTER 20 ns;
```

Sentencia de espera (wait)

```
WAIT ON e1,e2 UNTIL cap = '1' FOR 70 ns;
```

Subprogramas

```
TYPE vec_ent IS ARRAY(0 TO 7) OF INTEGER;
PROCEDURE media (x : INOUT reg_pro) IS
      VARIABLE  total : INTEGER := 0;
      BEGIN
           FOR i IN 0 TO 7 LOOP
            total := total + x.vector(i);
          END LOOP;
          x.promed := total / 8;
END media;
```

```
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;
```

Sobrecarga

```
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);
```

```
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;
```

Sentencias concurrentes

Sentencia de asignación de señal concurrente condicional

```
y <= TRANSPORT '1' AFTER 10 ns WHEN (a = '1' AND b = '1')  ELSE  '0'  AFTER 10 ns;
z <= x AND y AFTER 5 ns;
z <= x AND y AFTER 5 ns;
```

Sentencia de asignación de señal concurrente seleccionada

```
WITH sel SELECT
    sal       <=      "00000001" AFTER 10 ns WHEN "000",
                      "00000010" AFTER 10 ns WHEN "001",
                      "00000100" AFTER 10 ns WHEN "010",
                      "00001000" AFTER 10 ns WHEN "011",
                      "00010000" AFTER 10 ns WHEN "100",
                      "00100000" AFTER 10 ns WHEN "101",
                      "0100000" AFTER 10 ns WHEN "110",
                      "10000000" AFTER 10 ns WHEN "111";
```

Instanciación de componentes

```
puerta1 : puerta_y GENERIC MAP(12 ns) PORT MAP(a, b,  s1);
puerta2 : puerta_y GENERIC MAP(14 ns) PORT MAP(c, d,  s2);
puerta3 : puerta_o GENERIC MAP(10 ns) PORT MAP(s1, s2, e);
```

Bloques guardados

```
ARCHITECTURE latch_guard OF latch IS
BEGIN
  G1 : BLOCK(clk = '1')
  BEGIN
    q   <= GUARDED d AFTER 5 ns;
   qb <= GUARDED NOT  d  AFTER 7 ns;
  END BLOCK G1;
END latch_guard;
```

Llamada a procedimiento concurrente

```
ENTITY sumador IS
      port(a,b,cin : IN bit; sum,cout : OUT bit);
END sumador;

ARCHITECTURE subprogs OF sumador IS

      PROCEDURE xor2(SIGNAL a,b : IN bit; SIGNAL z : OUT bit) IS
      BEGIN
            z <= a XOR b;
      END xor2;

      PROCEDURE or2(SIGNAL a,b : IN bit; SIGNAL z : OUT bit) IS
      BEGIN
            z <= a OR b;
      END or2;

      PROCEDURE and2(SIGNAL a,b : IN bit;SIGNAL z : OUT bit) IS
      BEGIN
            z <= a AND b;
      END and2;
```

```
      SIGNAL s1,s2,s3 : bit;

BEGIN
      p1: xor2(a,b,s1);
      p2: xor2(s1,cin,sum);
      p3: and2(s1,cin,s2);
      p4: and2(a,b,s3);
      p5: or2(s2,s3,cout);
END subprogs;
```

Sentencia de aserto concurrente

```
 ASSERT scolor = rojo
  REPORT "El color no es rojo, es " & colorcad(scolor)
  SEVERITY NOTE;

 ASSERT sa = '1'
  REPORT "sa /= '1', es " & bitcaracter(sa)
  SEVERITY NOTE;
```

Sentencia de generación

```
ARCHITECTURE generación OF vect_inv IS
   COMPONENT inv
      PORT(e : IN bit; s : OUT bit);
   END COMPONENT;
BEGIN
   gen : FOR I IN 0 TO 7 GENERATE
      inversor : inv PORT MAP (entradas(I), salidas(I));
   END GENERATE;
END generación;
```

```
ARCHITECTURE iterativa OF sumador_palabra IS

 COMPONENT medio_sumador
  PORT(i1, i2 : IN bit; ms, mc : OUT bit);
 END COMPONENT;
 COMPONENT sumador
  PORT(s1, s2 : IN bit; sce : IN bit; ss : OUT bit; scs : OUT bit);
 END COMPONENT;
 SIGNAL a : bit_vector(nbits-2 DOWNTO 0);
BEGIN
 suma : FOR i IN 0 TO nbits-1 GENERATE
  menos_sig : IF i = 0 GENERATE
   primer : medio_sumador PORT MAP(op1(0),op2(0),sum(0),a(0));
  END GENERATE menos_sig;
  media_sig : IF i>0 AND i<nbits-1 GENERATE
   medios : sumador PORT MAP(op1(i),op2(i),a(i-1),sum(i),a(i));
  END GENERATE media_sig;
  mas_sig : IF i = nbits-1 GENERATE
   ultimo : sumador PORT MAP(op1(i),op2(i),a(i-1),sum(i),arrastre);
  END GENERATE mas_sig;
 END GENERATE suma;
END iterativa;
```

Unidades de diseño

Declaración de entidad

```
-- Declaración de entidad sin puertos ni genéricos
ENTITY ent0 IS
END;

-- Declaración de entidad con puertos de entrada y salida
ENTITY ent1 IS
    PORT (s1 : IN BIT; s2 : OUT bit);
END ent1;

-- Declaración de entidad con genéricos y puertos
ENTITY ent2 IS
    GENERIC (n : IN POSITIVE);
    PORT (s1 : IN BIT; s2 : OUT BIT_VECTOR(1 TO n));
END ent2;

-- Declaración de entidad con genéricos, puertos, declaraciones (cláusula use, tipo y procedimiento),
-- y sentencias de entidad (proceso pasivo de aserto concurrente)
-- Se supone que se ha compilado previamente paquete_tiempo donde está definida la constante
-- retardo y el procedimiento proc_pasivo
ENTITY ent3 IS
    GENERIC ( n : IN POSITIVE);
    PORT (s1 : IN BIT; s2 : OUT BIT_VECTOR(1 TO n));
    USE WORK.paquete_tiempo.ALL; -- Donde está definida la constante retardo
    TYPE byte IS ARRAY (1 TO 8) OF BIT;
    PROCEDURE ini(SIGNAL s : INOUT byte) IS
    BEGIN
        s <= (OTHERS => '1') AFTER retardo;
    END ini;
BEGIN
    ASSERT s1'DELAYED'STABLE(5 ns)
        REPORT "error"
        SEVERITY ERROR;
    proc_pasivo(s1, retardo);
END ent3;
```

Arquitectura de entidad

```
ARCHITECTURE arq OF ent0 IS
BEGIN
END;

ARCHITECTURE arq2 OF ent1 IS
    CONSTANT retardo : TIME := 5 ns;
    SIGNAL s : bit;
BEGIN
    s2 <= fbit(s) AFTER 3 ns;
    s  <= s1 AFTER retardo;
END arq2;

ARCHITECTURE arq1 OF ent1 IS
    SIGNAL s : BIT := '1';
    PROCEDURE p(SIGNAL a : IN BIT; SIGNAL b : INOUT BIT) IS
    BEGIN
        b <= NOT b WHEN a = '1'  ELSE b;
    END p;
BEGIN
    p1 : p(s1, s);
    p2 : s2 <= fbit(s) AFTER 2 ns;
```

```
    p3 : PROCESS(s)
        VARIABLE v : BIT;
    BEGIN
        v := s;
    END PROCESS;
END arq1;

USE WORK.componentes.ALL; -- Para gen_reloj
ARCHITECTURE arq OF ent2 IS
    SIGNAL s : BIT_VECTOR(1 TO n);
    COMPONENT comp
        PORT(a : IN BIT; b : OUT BIT);
    END COMPONENT;
BEGIN
    s2 <= s AFTER 2 ns;
    c1 : FOR I IN 2 TO n GENERATE
        c : comp PORT MAP(s1, s(I));
    END GENERATE;
    c2 : gen_reloj POR MAP (s(s(1));
END arq;
```

Declaración de paquete

```
PACKAGE general IS
    TYPE bit01xz IS ('0','1','x','z');
    FUNCTION and01xz(x,y : bit01xz) return bit01xz;
    FUNCTION or01xz(x,y : bit01xz) return bit01xz;
    FUNCTION not01xz(x : bit01xz) return bit01xz;
    FUNCTION nand01xz(x,y : bit01xz) return bit01xz;
    FUNCTION nor01xz(x,y : bit01xz) return bit01xz;
    FUNCTION xor01xz(x,y : bit01xz) return bit01xz;
END general;
```

Cuerpo de paquete

```
PACKAGE BODY general IS
    FUNCTION and01xz(x,y : bit01xz) return bit01xz IS
    variable z : bit01xz;
BEGIN
     IF x = 'x' OR y = 'x' THEN
        z := 'x';
     ELSIF x = '1' and y = '1' THEN
        z := '1';
     ELSE
        z := '0';
     END IF;
    RETURN z;
END and01xz;

FUNCTION or01xz(x,y : bit01xz) return bit01xz IS
    variable z : bit01xz;
BEGIN
    IF      x = '1' OR y = '1' THEN   z := '1';
    ELSIF   x = '0' and y = '0' THEN  z := '0';
    ELSE    z := '1';
    END IF;
    RETURN z;
END or01xz;

FUNCTION not01xz(x : bit01xz) return bit01xz IS
    variable z : bit01xz;
BEGIN
```

```
    IF      x = 'x'   THEN   z := 'x';
    ELSIF   x = '1'           THEN   z := '0';
    ELSE    z := '1';
    END IF;
    RETURN z;
END not01xz;

FUNCTION nand01xz(x,y : bit01xz) return bit01xz IS
    variable z : bit01xz;
BEGIN
    z := not01xz(and01xz(x,y));
    RETURN z;
END nand01xz;

FUNCTION nor01xz(x,y : bit01xz) return bit01xz IS
     variable z : bit01xz;
BEGIN
    z := not01xz(or01xz(x,y));
    RETURN z;
END nor01xz;

FUNCTION xor01xz(x,y : bit01xz) return bit01xz IS
    variable z : bit01xz;
BEGIN
     z := or01xz(and01xz(not01xz(x),y),and01xz(x,not01xz(y)));
    RETURN z;
END xor01xz;
END general;
```

Declaración de configuración

```
CONFIGURATION configuracion_1 OF entidad_1 IS
    FOR estructural
        FOR componente1 : entidad_2
            FOR estructural
                FOR componente : entidad_3
                    USE ENTITY WORK.entidad_3(flujo_de_datos);
                END FOR;
            END FOR;
        FOR componente2 : entidad_4
            USE ENTITY WORK.entidad_4(flujo_de_datos);
        END FOR;
    END FOR;
END configuracion_1;
```

```
CONFIGURATION config1 OF dispositivo IS
  USE Work.componentes.all;
  FOR estructural
    FOR u1_and2: and2 USE
      ENTITY Work.and2(unica);
    END FOR;

    FOR u2_and2: and2 USE
      ENTITY Work.and2(unica);
    END FOR;

    FOR u3_xor2: xor2  USE
      ENTITY Work.xor2(unica);
    END FOR;
  END FOR;
```

```
END config1;
```

```
CONFIGURATION config2 OF dispositivo2 IS
  USE Work.componentes.all;
  FOR estructural
    FOR u1_dispositivo : dispositivo
     FOR estructural
       FOR u1_and2 : and2 USE
                ENTITY Work.and2(unica);
       END FOR;

       FOR u2_and2 : and2 USE
                ENTITY Work.and2(unica);
       END FOR;

       FOR u3_xor2 : xor2  USE
                ENTITY Work.xor2(unica);
       END FOR;
     END FOR;
   END FOR;
  END FOR;
END config2;
```