Sesión 5: Unidades de Diseño VHDL

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Unidades de Diseño

- Las *unidades de diseño* son las construcciones VHDL que se pueden analizar y compilar de forma independiente.
- Se guardan en una biblioteca de diseño (WORK por defecto).
- Una o más unidades de diseño en secuencia constituyen un *archivo de diseño*.
- Existen dos clases de unidades de diseño: las primarias, y las secundarias.

Las tres unidades de diseño primarias son:

declaración de entidad

declaración de paquete

declaración de configuración

Las dos unidades de diseño secundarias son:

cuerpo de arquitectura

cuerpo de paquete

Declaración de Entidad

Sintaxis:

```
ENTITY identificador IS
        cabecera
        parte_declarativa
[BEGIN
        {sentencia_entidad}]
END [ identificador] ;
```

```
cabecera ::=
        [genéricos_formales]
        [puertos_formales]
```

```
genéricos_formales ::= GENERIC (lista_genéricos);
```

```
lista_genéricos ::= genérico {, genérico}
```

```
genérico ::= [CONSTANT] identificadores : [IN] tipo [:= valor]
```

```
puertos_formales ::= PORT (lista_puertos);
```

```
lista_puertos ::= puerto {, puerto}
```

```
puerto ::= [SIGNAL] identificadores : [modo] tipo [bus] [:= valor]
```

```
modo ::= IN | OUT | INOUT
```

```
parte_declarativa ::= {elemento_declarativo_entidad}
```

```
elemento_declarativo_entidad ::=
  declaración_subprograma
  | cuerpo_subprograma
  | dclaración_tipo
  | declaración_subtipo
  | declaración_constante
  | declaración_señal
  | clausula_use

sentencia_entidad ::=
  sentencia_aserto_concurrente
  | llamada_proced._concurrente_pasivo
  | sentencia_proceso_pasivo
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

- En la parte declarativa de la entidad aparecen elementos utilizados exclusivamente en la entidad.
- Las sentencias concurrentes que se permiten en una declaración de entidad deben ser *pasivas*, es decir, no deben contener asignaciones de señales.

Ejemplos de declaración de entidad:

```
ENTITY ent IS
END;
```

```
ENTITY ent1 IS
   PORT (s1 : IN BIT; s2 : OUT bit);
END ent1;
```

```
ENTITY ent2 IS
   GENERIC (n : IN POSITIVE);
   PORT (s1 : IN BIT; s2 : OUT BIT_VECTOR(1 TO n));
END ent2;
```

```
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 : 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;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Arquitectura de Entidad

Sintaxis general de un cuerpo de arquitectura:

ARCHITECTURE identificador OF nombre_entidad IS
parte_declarativa_arquitectura
BEGIN
sentencias_arquitectura
END [identificador];

parte_declarativa_arquitectura ::= parte_declar._bloque

sentencias_arquitectura ::= {sentencia_concurrente}

sentencia_concurrente ::=
sentencia_bloque
| sentencia_instanciación_componentes

```
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;
```

Ejemplos:

```
ARCHITECTURE arq OF ent 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;
```

```
ARQITECTURE 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;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Paquetes

- Permiten la declaración de objetos, tipos y subprogramas con la posibilidad de referenciarlos desde muchos puntos fuera del paquete.
- Un paquete se compone de dos partes:

  la ***declaración*** del paquete, que define la parte pública o visible del mismo, y

  el ***cuerpo*** del paquete, que define la parte oculta, visible solo en el interior.

Declaración de Paquete

Sintaxis:

```
PACKAGE identificador IS
        parte_declarativa_paquete
END [nombre_simple_paquete];

parte_declarativa_paquete ::= {elemento_declarativo_paquete}

elemento_declarativo_paquete ::=
        declaración_subprograma
        | declaración_tipo
        | declaración_subtipo
        | declaración_constante
        | declaración_señal
        | clausula_use
        | declaración_componente
```

Ejemplo de 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 del Paquete

Sintaxis:

PACKAGE BODY nombre_simple_paquete IS
parte_declar._cuerpo_paquete
END [nombre_simple_paquete];
parte_declar._cuerpo_paquete ::=
{elem._declar._cuerpo_paquete}
Elem._declar._cuerpo_paquete ::=
declaración_subprograma
| cuepo_subprograma
| declaración_tipo
| declaración_subtipo
| declaración_constante
| declaración_alias
| clausula_use

Ejemplo:

```
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;
```

Ejemplo (continuación):

```
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;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Referenciación de los Elementos de un Paquete

1. Prefijando sus nombres con el nombre del paquete.

Ejemplo:

SIGNAL pc : general.bit01xz;

2. Utilizando la clausula USE en la región declarativa correspondiente.

Sintaxis general de la clausula USE:

USE nombre_seleccionado {, nombre_seleccionado} ;
nombre_seleccionado ::= prefijo.sufijo
prefijo ::= nombre
sufijo ::= identificador | ALL

Con el prefijo designamos el nombre de la biblioteca y con el sufijo el nombre del Paquete.

Si se van a utilizar todos los nombres del paquete, se puede utilizar el sufijo ALL.

Eemplo

Si en la biblioteca *proyecto1* existiese un paquete de nombre *utilidad*, podriamos referenciarlo con la siguiente declaración:

USE proyecto1.utilidad;

Si ahora queremos referenciar un objeto del paquete, por ejemplo, la función *fun*, lo haríamos con la siguiente declaración:

USE utilidad.fun;

El mismo resultado podemos obtenerlo utilizando la única declaración siguiente:

USE proyecto1.utilidad.fun;

La diferencia estriba en que en este último caso sólo hacemos visible la función *fun* del paquete utilidad, mientras que en el primero hacemos visible el paquete utilidad.

Si se van a utilizar todos los nombres del paquete, se puede utilizar el sufijo ALL.

USE general.ALL;

Declaración de Configuración

- Cuando se declara un componente en la arquitectura de una entidad lo que realmente se declara es una *plantilla* del componente que especifica los puertos y genéricos del mismo, pero nada se dice sobre su implementación.

- La declaración de componente se puede vincular a una entidad específica de una biblioteca por medio de la *especificación de configuración* que aparece dentro de la misma arquitectura.

- Un componente puede volver a vincularse a una nueva implementación cambiando la especificación de configuración y recompilando la arquitectura. Esta posibilidad es útil pero requiere la alteración del modelo y su recompilación.

- Para proporcionar mayor flexibilidad al problema de la configuración de un diseño, VHDL dispone de la unidad de biblioteca denominada *declaración de configuración*.

- Un mismo diseño puede disponer de varias declaraciones de configuración que vinculan sus componentes a diferentes entidades.

- De la misma forma, una determinada entidad puede formar parte de varias declaraciones de configuración correspondientes a varios diseños.

Sintáxis

```
CONFIGURATION  identificador OF  nombre_entidad IS
        {cláusula_use}
        configuración_bloque
END [identificador];

configuración_bloque ::=
        FOR especificación_bloque
                {cláusula_use}
                {elemento_configuración}
        END FOR;

especificación_bloque ::=
        nombre_arquitectura
        | rótulo_sentencia_bloque
        | rótulo_sentencia_generación
        [(especificación_índice)]

elemento_configuración ::=
        configuración_bloque
        | configuración_componente

configuración_componente ::=
        FOR especificación_componente
                [USE  indicación_vinculación;]
                [configuración_bloque]
        END FOR;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Posibilidad de las declaraciones de configuración para crear unidades de diseño conformadas con entidades y arquitectura previamente elaboradas y contenidas en bibliotecas de diseño.

La *entidad_1* tiene definidas tres arquitecturas:

arquitectura_1.1 de comportamiento
arquitectura_1.2 de flujo de datos
arquitectura_1.3 estructural.

El primer componente de *arquitectura_1.3* se puede instanciar con *entidad_2*, con un componente en su arquitectura que se puede instanciar con *entidad_3*.

Ejemplode declaración de configuración:

Declaración de *configuración_1* para *entidad_1:*

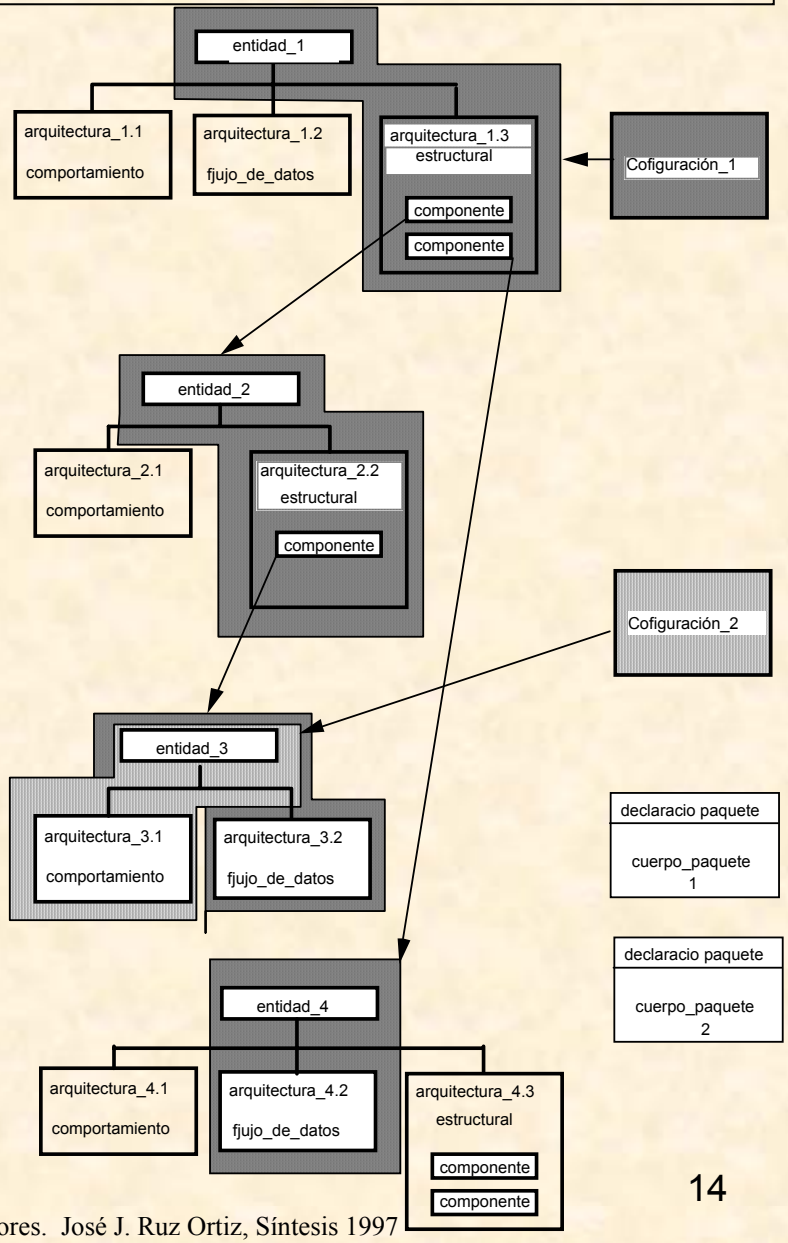
Asocia *arquitectura_1.3* con *entidad_1*

Instancia los componentes de dicha arquitectura a *entidad_2* y *entidad_4*

entidad_2 y *entidad_4* asociadas respectivamente A *arquitectura_2.2* y *arquitectura_4.2*.

El componente de *arquitectura_2.2* se instancia a *entidad_3* asociada a *arquitectura_3.2*.

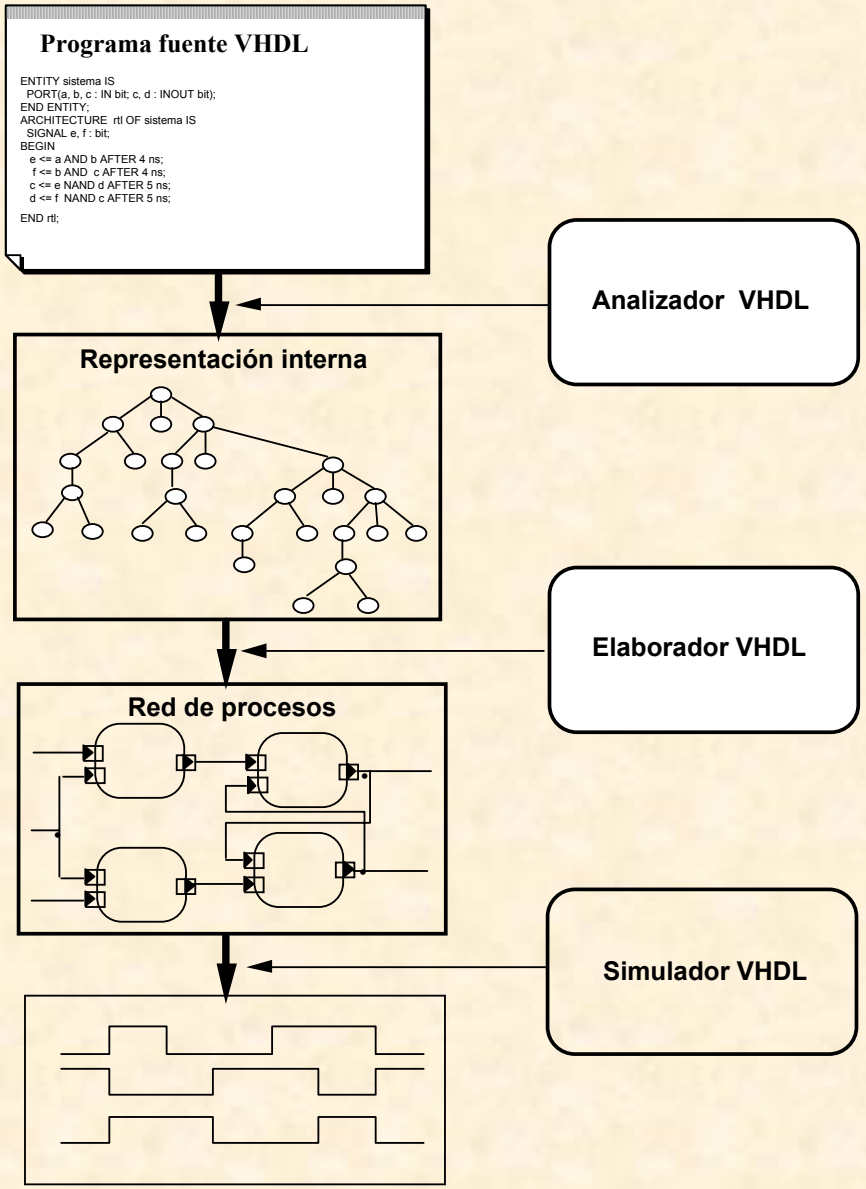
Se indica con sombreado la asociación entidad-arquitectura, y con flechas las instanciaciones de componentes y especificaciones de configuración.



Ejemplo de declaración de configuración (código):

```
CONFIGURATION configuracion_1 OF diseño 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;
```

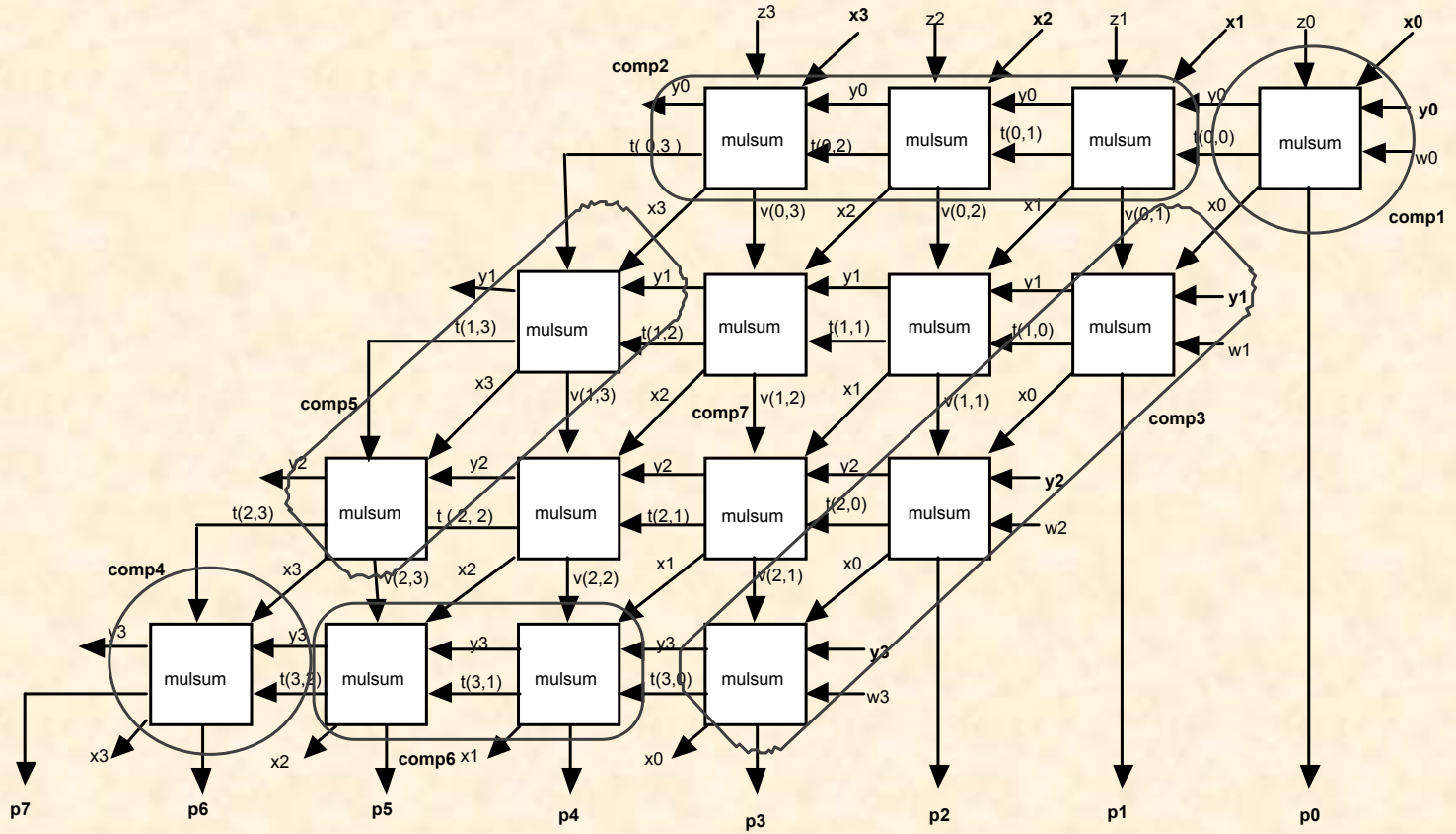
Fases en la ejecución de un programa VHDL



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Ejemplo completo: Multiplicador celular

```
                                  x3          x2          x1          x0
                                  y3          y2          y1          y0
                          ------------------------------------------------
                                  x3y0        x2y0        x1y0        x0y0
                      x3y1        x2y1        x1y1        x0y1
              x3y2    x2y2        x1y2        x0y2
x3y3          x3y2    x3y1        x3y0
-----------------------------------------------------------------------
p6            p5          p4          p3          p2          p1          p0
```



Declaración de entidad

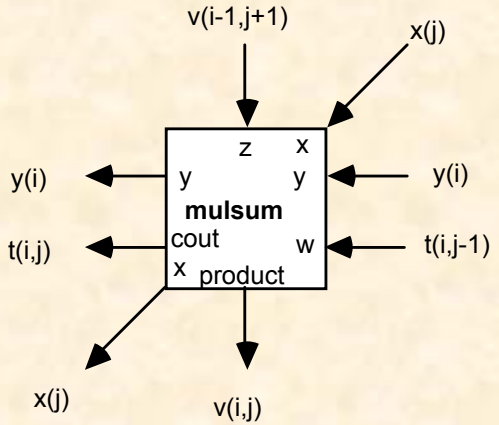
```
USE WORK.mul_pack.ALL;
ENTITY mul_n IS
  GENERIC(n : integer := 4);
  PORT(x,y,z,w : IN bit_vector(n-1 DOWNTO 0);
                    p : OUT bit_vector(((2*n)-1) DOWNTO 0));
END mul_n;
```

Diseño de la Celda Básica

$$cout = xyz + xyw + zw$$

$$product = xy \oplus z \oplus w$$

Entidad *mulsum*



Declaración de Entidad

```
ENTITY mulsum IS
  PORT(x, y, z, w : IN bit;  cout, product : OUT bit);
END mulsum;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Arquitectura de Comportamiento

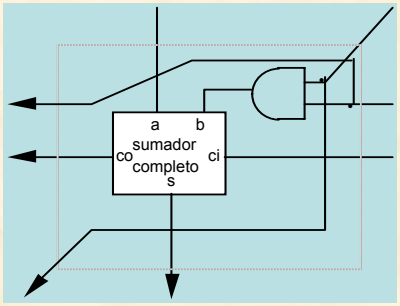
```
ARCHITECTURE comportamiento OF mulsum IS
BEGIN
  PROCESS(x, y, z, w)
    VARIABLE n : integer;
    CONSTANT producto : bit_vector(0 TO 3) := "0101";
    CONSTANT arrastre : bit_vector(0 to 3) := "0011";
  BEGIN
    n := 0;
    IF x = '1' AND y = '1' THEN n := n+1; END IF;
    IF z = '1' THEN n := n+1; END IF;
    IF w = '1' THEN n:= n+1; END IF;
    product <= producto(n) AFTER 10 ns;
    cout <= arrastre(n) AFTER 20 ns;
  END PROCESS;
END comportamiento;
```

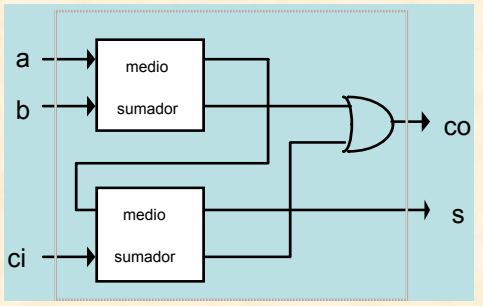
Arquitectura de Flujo de Datos

```
ARCHITECTURE flujo_de_datos OF mulsum IS
  SIGNAL s : bit;
BEGIN
  s <= (x AND y) XOR z;
  product <= s XOR w AFTER 10 ns;
  cout <= (x AND y AND z) OR (s AND w) AFTER 20 ns;
END flujo_de_datos;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Celda Básica: Arquitectura Estructural






```
USE work.ALL;
ARCHITECTURE estructural OF mulsum IS
        COMPONENT medio_sum
GENERIC(retardo_c, retardo_s : time);
PORT(i1, i2 : IN bit; carry, suma : OUT bit);
        END COMPONENT;
        COMPONENT and2_c
                GENERIC(retardo : time);
                PORT(i1, i2 : IN bit; o : OUT bit);
        END COMPONENT;
        COMPONENT or2_c
                GENERIC(retardo : time);
                PORT(i1, i2 : IN bit; o : OUT bit);
        END COMPONENT;
FOR ALL : medio_sum USE ENTITY WORK.medio_sumador(generica);
FOR ALL : and2_c USE ENTITY WORK.and2(generica);
FOR ALL : or2_c USE ENTITY WORK.or2(generica);
        SIGNAL a, b, c, d : bit;
BEGIN
        comp1 : and2_c GENERIC MAP(20 ns) PORT MAP(x, y, a);
        comp2 : medio_sum GENERIC MAP(10 ns, 20 ns) PORT MAP(a, z, b, c);
        comp3 : or2_c  GENERIC MAP(10 ns) PORT MAP(b, d, cout);
        comp4 : medio_sum GENERIC MAP(10 ns, 20 ns) PORT MAP(c, w, d, product);
END estructural;
```

Entidades Disponibles en la biblioteca WORK

```
ENTITY medio_sumador IS
  GENERIC(retardo_c, retardo_s : time);
  PORT(i1, i2 : IN bit; carry, suma : OUT bit);
END medio_sumador;

ARCHITECTURE generica OF medio_sumador IS
BEGIN
  carry <= i1 AND i2 AFTER retardo_c;
  suma <= i1 XOR i2 AFTER retardo_s;
END generica;
```

```
ENTITY and2 IS
  GENERIC(retardo : time);
  PORT(i1, i2 : IN bit; o : OUT bit);
END and2;

ARCHITECTURE generica OF and2 IS
BEGIN
  o <= i1 AND i2 AFTER retardo;
END generica;
```

```
ENTITY or2 IS
  GENERIC(retardo : time);
  PORT(i1, i2 : IN bit; o : OUT bit);
END or2;

ARCHITECTURE generica OF or2 IS
BEGIN
  o <= i1 OR i2 AFTER retardo;
END generica;
```

Arquitectura estructural

```
ARCHITECTURE estructural OF mul_n IS
  COMPONENT mulsum
    PORT(x,y,z,w : IN bit; cout, product : OUT bit);
  END COMPONENT;
  FOR ALL : mulsum USE ENTITY
       WORK.mulsum(comportamiento);
  SIGNAL t,v : bit_array(0 TO n-1, 0 TO n-1);
BEGIN
```

```
comp1 : mulsum
                PORT MAP(x(0),y(0),z(0),w(0),t(0,0),p(0));
fila1 : FOR j IN 1 TO n-1
GENERATE
comp2 : mulsum
                PORT MAP(x(j),y(0),z(j),t(0,j-1),t(0,j),v(0,j));
END GENERATE;
columna1 : FOR i IN 1 TO n-1
GENERATE
comp3 : mulsum
                PORT MAP(x(0),y(i),v(i-1,1),w(i),t(i,0),p(i));
END GENERATE;
comp4 : mulsum
   PORT MAP(x(n-1),y(n-1),t(n-2,n-1),t(n-1,n-2),p((2*n)-1),p((2*n)-2));
columnan : FOR i IN 1 TO n-2
GENERATE
comp5 : mulsum
   PORT MAP(x(n-1),y(i),t(i-1,n-1),t(i,n-2),t(i,n-1),v(i,n-1));
END GENERATE;
filan : FOR j IN 1 TO n-2
GENERATE
comp6 : mulsum
   PORT MAP(x(j),y(n-1),v(n-2,j+1),t(n-1,j-1),t(n-1,j),p(n+j-1));
END GENERATE;
columnaj : FOR i IN 1 TO n-2
GENERATE
filai :    FOR j IN 1 TO n-2
                GENERATE
                comp7 : mulsum
                   PORT MAP(x(j),y(i),v(i-1,j+1),t(i,j-1),t(i,j),v(i,j));
                END GENERATE;
END GENERATE;
END estructural;
```

Test para el caso de 8 bits

```
USE WORK.mul_pack.ALL;
ENTITY mul_8 IS
        PORT(a,b : INTEGER := 0);
END mul_8;
ARCHITECTURE esquema OF mul_8 IS
        COMPONENT mul_n
                GENERIC(n : integer);
                PORT(x,y,z,w : IN bit_vector(n-1 DOWNTO 0);
                        p : OUT bit_vector((2*n)-1 DOWNTO 0));
END COMPONENT;
SIGNAL x,y,z,w : bit_vector(7 DOWNTO 0); SIGNAL p : bit_vector(15 DOWNTO 0);
        SIGNAL ab, p_ent : INTEGER;
        FOR mult : mul_n USE ENTITY WORK.mul_n(estructural);
BEGIN
        mult : mul_n GENERIC MAP(8) PORT MAP (x,y,z,w,p);
        test : PROCESS
        BEGIN
                x <= int_b8(a);
                y <= int_b8(b);
                ab <= a*b;
                WAIT FOR 200 ns;
                p_ent <= b16_int(p);
                WAIT ON a,b;
        END PROCESS;
END esquema;
```

Paquete de tipos y funciones de conversión

```
PACKAGE mul_pack IS
  TYPE bit_array IS ARRAY(natural RANGE
                <>, natural RANGE <>) OF bit;
  SUBTYPE b8 IS bit_vector(7 DOWNTO 0);
  SUBTYPE b16 IS bit_vector(15 DOWNTO 0);
  FUNCTION b16_int(a : b16) RETURN INTEGER;
  FUNCTION int_b8(a : INTEGER) RETURN b8;
END mul_pack;
```

```
PACKAGE BODY mul_pack IS
FUNCTION b16_int  (a: b16) RETURN INTEGER is
  VARIABLE int_var : INTEGER := 0;
  BEGIN
   FOR i in 0 to 15 LOOP
    IF ( a(i) = '1') THEN
     int_var := int_var + (2**i);
    END IF;
   END LOOP;
 RETURN int_var;
END b16_int;
FUNCTION int_b8  (a: INTEGER) RETURN b8 is
  VARIABLE int_var : b8;
  VARIABLE temp1 : INTEGER := 0;
  VARIABLE temp2 : INTEGER := 0;
  BEGIN
   temp1 := a;
   FOR i in 7 downto 0 LOOP
     temp2 := temp1/(2**i);
     temp1 := temp1 mod (2**i);
     IF ( temp2 = 1 ) THEN
      int_var(i) := '1';
      ELSE
      int_var(i) := '0';
     END IF;
   END LOOP;
 RETURN int_var;
END int_b8;
END mul_pack;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

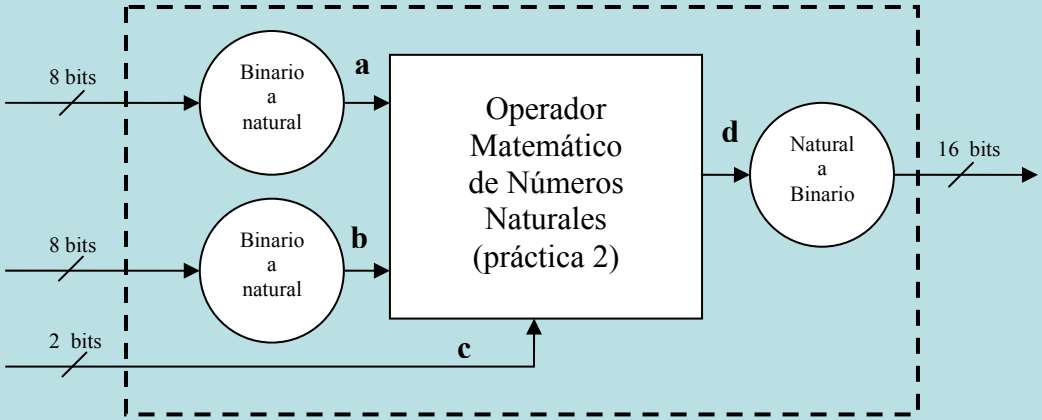
Practica 5: Unidades de diseño VHDL

Objetivos

1. Utilización de las unidades de diseño de VHDL, especialmente los paquetes.

Práctica a realizar

Diseño binario del operador matemático *de números naturales diseñado en la práctica 2 .*



Todos los tipos y funciones definidos en la práctica 2 y todos los necesarios para ésta, deberán definirse dentro de un paquete que se haga visible al principio de la declaración de entidad.

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.

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997