Sesión 4: Sentencias concurrentes y paquetes

Sentencias Concurrentes

- La principal sentencia concurrente de VHDL es el proceso (*process*), que define los límites de un dominio secuencial.
- Las restantes sentencias concurrentes son formas diferenciadas de procesos que proporcionan al lenguaje una mayor expresividad.
- Estas sentencias utilizan una sintaxis conveniente para:
  - Escribir programas con estilo de flujo de datos.
  - Escribir programas con estilo estructural.
  - Facilitar la descripción de arquitecturas regulares.
- Desde el punto de vista de la ingeniería del software, estas sentencias facilitan la reusabilidad en el dominio concurrente.

- Sentencias Concurrentes:
  - **Sentencia de Asignación de Señal Concurrente**
  - **Sentencia de Instanciación de Componentes**

Sentencia de Asignación de Señal Concurrente

- Es una forma compacta de describir un proceso que asigna a una señal una onda seleccionada por un conjunto de condiciones o por una expresión de control.
- Permite escribir programas VHDL con un estilo de *flujo de datos*, también denominado de *transferencia de registros* (RTL).

Sintaxis:

[rótulo :] asignación_señal_condicional
| [rótulo :] asignación_señal_seleccionada

asignación_señal_condicional ::=
señal <= opciones ondas_condicionales
ondas_condicionales ::= { onda WHEN condición ELSE } onda

asignación_señal_seleccionada ::=
WITH expresión SELECT
señal <= opciones ondas_seleccionadas

ondas_seleccionadas ::= { onda WHEN elecciones, } onda WHEN elecciones

opciones ::= [GUARDED] [TRANSPORT]

elecciones ::= elección | elección

onda ::= elemento_onda { , elemento_onda}

elemento_onda ::= expresión [AFTER expresión_tiempo]
| NULL [AFTER expresión_tiempo]

Asignación de señal condicional

- Opera de forma análoga a la sentencia IF en el dominio secuencial.
- La onda que se asigna a la señal se elige en base a un conjunto de condiciones booleanas que aparecen a la derecha del símbolo de la asignación.
- Las condiciones se comprueba hasta encontrar una con valor *true*, cuando ello ocurre, se asigna a la señal la onda asociada a esta condición.
- Esta sentencia es sensible a las señales que aparezcan en las expresiones de las ondas o en las propias condiciones.

Ejemplo:

```
ENTITY and2 IS
            PORT(a, b : IN bit; y : OUT bit);
END and2;
ARCHITECTURE condicional OF and2 IS
BEGIN
            y <= TRANSPORT '1' AFTER 10 ns WHEN (a = '1' AND b = '1')
                                          ELSE  '0'  AFTER 10 ns;
END condicional;
```

Equivalencia

• Una asignación de señal concurrente condicional es equivalente a un proceso que contiene la correspondiente asignación de señal secuencial condicionada por un *if-then-else* y con una lista de sensibilización formada por todas las señales que aparecen a la derecha del símbolo de asignación :

Asignación de señal concurrente condicional	**Proceso equivalente**
s <= onda_1 WHEN condición_1 ELSE onda_2 WHEN condición_2 ELSE .... onda_n;	PROCESS (señales a la derecha del símbolo <=) IF condición_1 THEN s <= onda_1; ELSIF condición_2 THEN s <= onda_2; ELSIF .... ELSE s <= onda_n; END PROCESS;

Asignación de señal concurrente no condicional

- Se puede considerar un caso particular de la asignación de señal concurrente condicional en la que a la derecha del símbolo de la asignación aparece tan solo la onda asignada.

Ejemplo:

```
ENTITY and2 IS
        PORT(x,y : IN bit; z : OUT bit);
END and2;
ARCHITECTURE uno OF and2 IS
BEGIN
        z <= x AND y AFTER 5 ns;
END uno;
```

Arquitectura equivalente:

```
ARCHITECTURE dos OF and2 IS
BEGIN
        PROCESS (x, y)
        BEGIN
        z <= x AND y AFTER 5 ns;
        END PROCESS;
END dos;
```

Asignación de señal seleccionada

- Se comporta de forma análoga a la sentencia CASE en el dominio secuencial.
- Se compone de una expresión y una onda asociada con cada valor.
- A la señal se le asigna la onda asociada al valor que toma la expresión.
- El proceso equivalente es sensible a todas las señales a la derecha del símbolo de la asignación.
- Es equivalente a un proceso que contiene las correspondientes asignaciones secuenciales seleccionadas por una sentencia *case* :

Asignación de señal concurrente seleccionada	**Sentencia de proceso equivalente**
WITH expresión SELECT s **<=** onda_1 WHEN elecciones_1, onda_2 WHEN elecciones_2, ... onda_n WHEN elecciones_n;	PROCESS (señales a la derecha del símbolo <=) CASE expresión IS WHEN elecciones_1 => s <= onda_1; WHEN elecciones_2 => s <= onda_2; ..... WHEN elecciones_n => s <= onda_n; END CASE; END PROCESS;

Ejemplo:

```
ENTITY descodificador IS
        PORT( sel : IN bit_vector(2 DOWNTO 0);
              sal : OUT bit_vector(7 DOWNTO 0));
END descodificador;


ARCHITECTURE seleccionada OF descodificador IS
BEGIN
        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",
                            "01000000" AFTER 10 ns WHEN "110",
                            "10000000" AFTER 10 ns WHEN "111";
END seleccionada;
```

Instanciación de Componentes

- Esta sentencia es el principal recurso disponible en VHDL para reutilizar entidades de diseño ya definidas.
- Permite la expresión de arquitecturas estructurales, es decir, arquitecturas que se definen por la declaración de unos componentes y su interconexión.
- Hay que utilizarla conjuntamente con otras dos sentencias declarativas:

  declaración de componentes**,** para nombrar las clases de componentes implicados en la instanciación,

  especificación de configuración**,** para designar la entidad-arquitectura (de las muchas posiblemente definidas) que se utilizará en cada ocurrencia de un componente de una clase dada.
- La declaración de las diferentes clases de componentes debe preceder a su instanciación.
- Por cada clase de componente debe existir una declaración en la que se especifique el nombre, y opcionalmente los *genéricos locales* y los *puertos locales*.
- Los genéricos locales también pueden aparecer en una declaración de entidad (aunque hasta el momento no los hemos utilizado en este curso) y sirven para pasar información estática.

Declaración de Componentes

Sintaxis:

```
COMPONENT  nombre_componente
        [genéricos_locales]
        [puertos_locales]
END COMPONENT;
```

Ejemplos:

```
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

- Especifica la configuración concreta de cada instancia de componente que interviene en una arquitectura estructural.

Sintaxis:

FOR especificación_componente
USE indicación_vinculación

especificación_componente ::=
lista_instanciación : nombre_componente

lista_instanciación ::=
rótulo_instanciación { , rótulo_instanciación }| OTHERS| ALL

indicación_vinculación ::=
aspecto_entidad [correspondencia_genérica] [correspondencia_puertos]

aspecto_entidad ::=
ENTITY nombre_entidad[(nombre_arquitectura)]
|CONFIGURATION nombre_configuración | OPEN

- La *lista de instanciación* puede explicitar cada uno de los *rótulos de instanciación* que aparecerán encabezando cada una de las *sentencias de instanciación de componentes*.
- Se puede explicitar solo parte de los rótulos y los restantes implícitos con la palabra *others*.
- Se puede dejarlos todos implícitos con la palabra reservada **ALL**.
- La *lista de instanciación* queda separada por dos puntos (:) del *nombre del componente* de la *declaración de componente* correspondiente.

Ejemplo

Componentes

```
ENTITY and2 IS
        GENERIC(retardo : TIME := 5 ns);
        PORT(x,y : IN bit; z : OUT bit);
END and2;

ARCHITECTURE comportamiento OF and2 IS
BEGIN
        z <= x AND y AFTER retardo;
END comportamiento;

ENTITY or2 IS
        GENERIC(retardo : TIME := 4 ns);
        PORT(x,y : IN bit; z : OUT bit);
END or2;

ARCHITECTURE comportamiento OF or2 IS
BEGIN
        z <= x OR y AFTER retardo;
END comportamiento;
```

- Hemos configurado todas las instancias de componentes de la clase *puertas_y* con la entidad *and2* y la arquitectura *comportamiento* que se suponen que ya residen en la biblioteca WORK.
- Las instancias de componentes de la clase *puerta_o* los hemos configurado con la entidad *or2* y la arquitectura *comportamiento*.

Especificación de Configuración(1)

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

Especificación de Configuración(2)

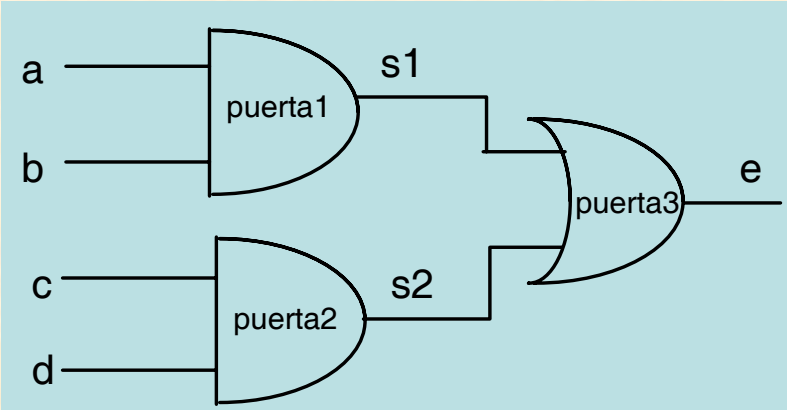
FOR puerta1, puerta2 : puerta_y USE ENTITY WORK.and2(comportamiento);
FOR puerta3 : puerta_o USE ENTITY WORK.or2(comportamiento);

Sentencia de Instanciación de Componentes

• Una sentencia de instanciación de componentes especifica el conexionado de un componente particular.

Sintaxis:

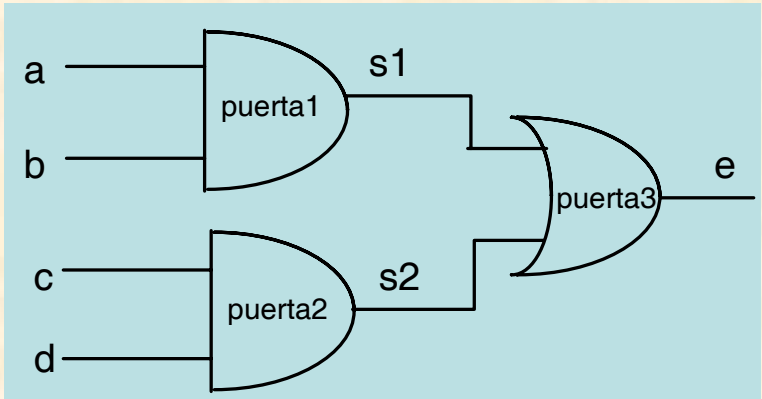
rotulo_instanciación :
nombre_componente
[correspondencia_genérica]
[correspondencia_puertos];




```
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);
```

Programa completo entidad-arquitectura

```
-- declaración de entidad
ENTITY simple IS
        PORT(a,b,c,d : IN bit; e : OUT bit);
END simple;
-- cuerpo de arquitectura
ARCHITECTURE estructural OF simple IS
        -- 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);
        -- declaración de señales de interconexión
        SIGNAL s1, s2 : bit;
BEGIN
        -- 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);
END estructural;
```



Proceso Equivalente de una Instanciación de Componentes

• Una sentencia de instanciación de componente es equivalente al proceso o conjunto de procesos del cuerpo de la arquitectura de la entidad componente.

Ejemplo, la arquitectura *estructural* de la entidad *simple* del anterior ejemplo es equivalente a la siguiente:

```
ARCHITECTURE estructural OF simple IS
   SIGNAL s1, s2 : bit;
BEGIN
     s1 <= a AND b AFTER 12 ns;
     s2 <= c AND d AFTER 14 ns;
     e <= s1 OR s2 AFTER 10 ns;
END estructural;
```

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;
```

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;

Estilos de descripción de un circuito lógico

- **Comportamiento**
- **Estructural**

Ejemplo.

Un sistema secuencial síncrono, que llamaremos *dos_con* (dos consecutivos) que produce salida '1' cuando aparecen dos o más valores iguales consecutivos en su entrada.

Diagramas de bloques y de estados:

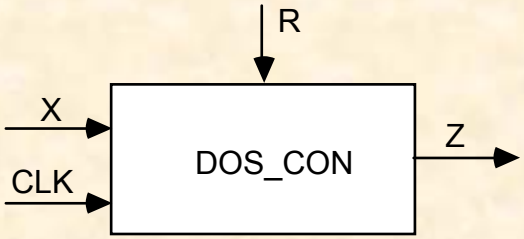


Diagrama de Bloques

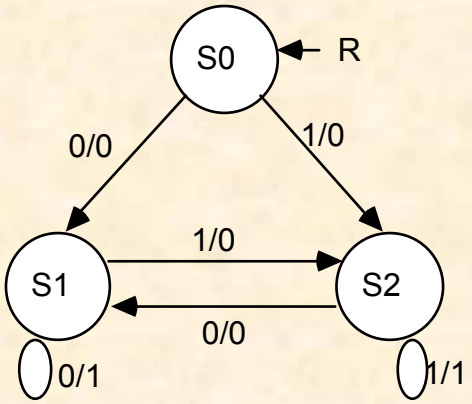


Diagrama de Estados

Diagrama de Bloques --> Declaración de entidad:

```
ENTITY dos_con IS
  PORT(clk, r, x : IN bit; z : OUT bit);
END dos_con;
```

Diagrama de estados --> lo traduciremos a diferentes arquitecturas.

Comportamiento

- Define la funcionalidad del dispositivo con un algoritmo secuencial, sin referencia alguna a su implementación.

- En nuestro caso escribiremos un algoritmo que sigue el comportamiento de su tabla de estados:

	X	
Estado	0	1
S0	S1/0	S2/0
S1	S1/1	S2/0
S2	S1/0	S2/1

```
ARCHITECTURE algorítmica OF dos_con IS
  TYPE estado IS (s0, s1, s2);
  SIGNAL q : estado := s0;
BEGIN
  PROCESS(r, x, clk, q)
  BEGIN
    IF (r'EVENT AND r = '0') THEN q <= s0;
    ELSIF (clk'EVENT AND clk = '1') THEN
     IF x = '0' THEN q <= s1; ELSE q <= s2; END IF;
    END IF;
    IF (q'EVENT OR x'EVENT) THEN
      IF (q = s1 AND x = '0') OR (q = s2 AND x = '1') THEN
        z <= '1';
      ELSE
        z <= '0';
      END IF;
    END IF;
  END PROCESS;
END algorítmica;
```

Estructural

- Una descripción estructural declara los componentes del dispositivo y expresa sus interconexiones.

- Esquemático de *dos_con* compuesto de biestables y puertas:

Y1

0	1	1	0
-	-	1	0

Y1 = x

Y0

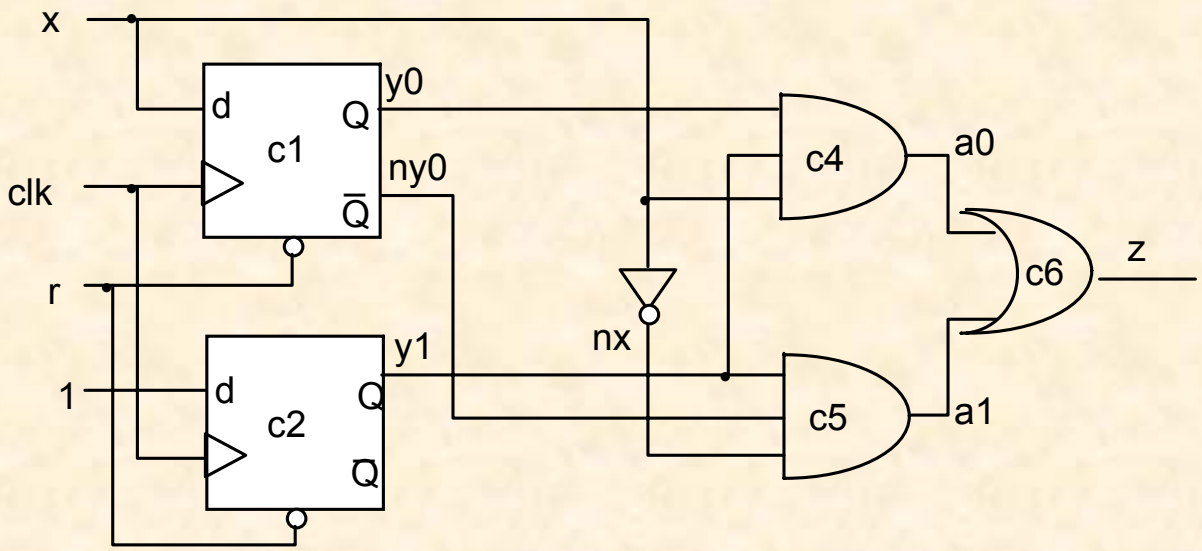
1	1	1	1
-	-	1	1

Y0 = 1

Z

0	0	0	1
-	-	1	0

$$Z = \overline{y_1} y_0 \overline{x} + y_1 \overline{y_0} x$$

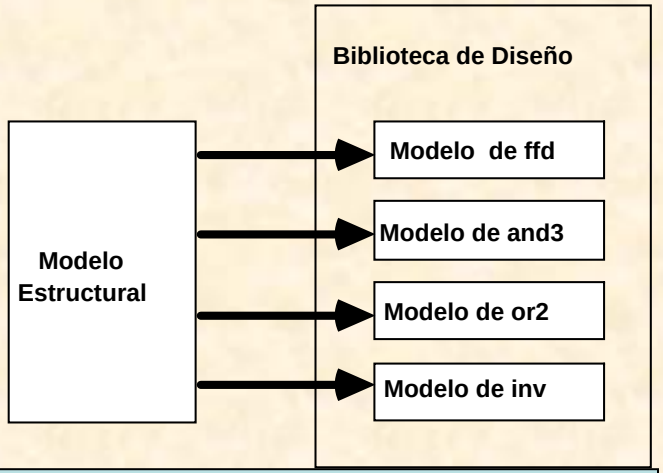


Modelo de flip-flop d (ffd)

```
ENTITY ffd IS
  PORT(clk, d, nclr : IN bit; q, qn : OUT bit);
END ffd;
ARCHITECTURE comportamiento OF ffd IS
BEGIN

PROCESS
       VARIABLE r : bit;
BEGIN
       WAIT UNTIL (((clk = '1') AND (clk'EVENT)) OR nclr = '0');
       IF nclr = '0'  THEN r := '0'; ELSE r:= d; END IF;
       q <= r;
      qn <= NOT r;
      END PROCESS;
END comportamiento;
```

Modelo de inversor (inv)

```
ENTITY inv IS
  PORT(e : IN bit; s : OUT bit);
END inv;
ARCHITECTURE comportamiento OF inv IS
BEGIN
  s <= NOT e;
END comportamiento;
```



Modelo de puerta and de tres entradas (and3)

```
ENTITY and3 IS
  PORT(e1, e2, e3 : IN bit; s : OUT bit);
END and3;
ARCHITECTURE comportamiento OF and3 IS
BEGIN
  s <= e1 AND e2 AND e3;
END comportamiento;
ortamiento;
```

Modelo de puerta or de dos entrdas (or2)

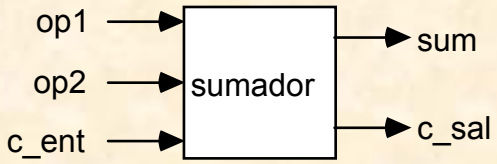
```
ENTITY or2 IS
  PORT(e1, e2 : IN bit; s : OUT bit);
END or2;
ARCHITECTURE comportamiento OF or2 IS
BEGIN
  s <= e1 OR e2;
END comportamiento;
```

Arquitectura Estructural

```
USE WORK.ALL;
ARCHITECTURE estructural OF dos_con IS

-- declaración de señales de interconexión
  SIGNAL y0, y1, a0, a1 : bit := '0';
  SIGNAL ny0, nx : bit := '1';
  SIGNAL uno : bit := '1';

-- declaración de componentes
  COMPONENT ff_d
    PORT(clk, d, nclr : IN bit; q, qn : OUT bit);
  END COMPONENT;
  COMPONENT invg
    PORT(e : IN bit; s : OUT bit);
  END COMPONENT;
  COMPONENT and3g
    PORT(e1, e2, e3 : IN bit; s : OUT bit);
  END COMPONENT;
  COMPONENT or2g
    PORT(e1, e2 : IN bit; s : OUT bit);
  END COMPONENT;

-- especificación de configuración
  FOR ALL : ff_d USE ENTITY ffd(comportamiento);
  FOR ALL : invg USE ENTITY inv(comportamiento);
  FOR ALL : and3g USE ENTITY and3(comportamiento);
  FOR ALL : or2g USE ENTITY or2(comportamiento);
```

```
-- instanciación de componentes

BEGIN
  c1 : ff_d   PORT MAP(clk, x, r, y0, ny0);
  c2 : ff_d   PORT MAP(clk, uno, r, y1, OPEN);
  c3 : invg   PORT MAP(x, nx);
  c4 : and3g  PORT MAP(x, y0, y1, a0);
  c5 : and3g  PORT MAP(ny0, y1, nx, a1);
  c6 : or2g   PORT MAP(a0, a1, z);
END estructural;
```

Un segundo ejemplo

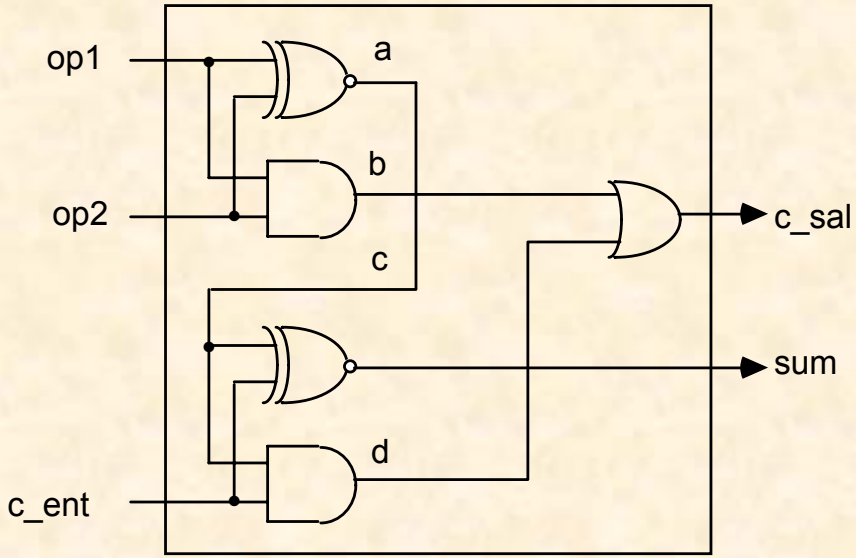



```
ENTITY sumador IS
  PORT(op1, op2 : IN bit;
       c_ent : IN bit;
       sum : OUT bit;
       c_sal : OUT bit);
END sumador;
```

Arquitectura de *comportamiento* :

```
ARCHITECTURE comportamiento OF sumador IS
  BEGIN

 PROCESS(op1, op2, c_ent)
FUNCTION bit_int (b : bit) RETURN INTEGER IS
  VARIABLE e : INTEGER;
  BEGIN
    IF b = '1' THEN e := 1; ELSE e := 0; END IF;
    RETURN e;
  END bit_int;
FUNCTION bool_bit (e : BOOLEAN) RETURN bit IS
  VARIABLE b : bit;
  BEGIN
    IF (e = TRUE) THEN b := '1'; ELSE b := '0'; END IF;
    RETURN b;
END bool_bit;
    VARIABLE res : INTEGER;
    BEGIN
      res := bit_int(op1) + bit_int(op2) + bit_int(c_ent);
      sum <= bool_bit((res MOD 2) = 1) AFTER 12 NS;
      c_sal <= bool_bit(res > 1) AFTER 10 NS;
    END PROCESS;
END comportamiento;
```

Arquitectura Estructural en Términos de Puertas




```
USE work.ALL;
ARCHITECTURE puertas OF sumador IS
  COMPONENT or2
    PORT(i1, i2 : IN bit; o : OUT bit);
  END COMPONENT;
  COMPONENT and2
    PORT(i1, i2 : IN bit; o : OUT bit);
  END COMPONENT;
  COMPONENT xor2
    PORT(i1, i2 : IN bit; o : OUT bit);
  END COMPONENT;
  SIGNAL a, b, c, d : bit;
BEGIN
  comp1 : xor2  PORT MAP(op1, op2, a);
  comp2 : and2  PORT MAP(op1, op2, b);
  comp3 : xor2  PORT MAP(a, c_ent, sum);
  comp4 : and2  PORT MAP(a, c_ent, d);
  comp5 : or2  PORT MAP(b, d, c_sal);
END puertas;
```

Practica 4 : Sentencias concurrentes y paquetes

Objetivos:

Diseño de pequeños sistemas digitales utilizando los estilos de descripción VHDL: comportamiento y estructural.

Práctica a realizar:

Diseña un sistema secuencial síncrono con al menos cinco estados utilizando los siguientes estilos de descripción en VHDL:

1. Comportamiento
2. Estructural

Nota: Las funciones y tipos necesarios deberán definirse dentro de un paquete.

Resultados a entregar:

Documentación del diseño incluyendo:

1. Especificación completa y precisa del sistema modelado.
2. Listado VHDL comentado de los programas correspondientes a cada uno de los estilos de descripción.
3. Tests de entrada/salida que muestren el correcto funcionamiento del sistema en cada uno de los niveles de descripción. Incluir diagramas de tiempo.