4.4. Métodos de diseño

4.4.1. Estilo estructural

Es la descripción estructural en la que se descompone en los componentes del sistema y se indican sus interconexiones. Cada subcircuito se denomina *component*. Un subcircuito se declara mediante la *declaración de componente*. Así, se especifica el nombre del subcircuito y los puertos de entrada y salida.

```
COMPONENT nombre_componente
PORT (port_list);
END nombre_componente;
```

Los componentes se declaran dentro de la Arquitectura antes de la sentencia BEGIN de la misma. La declaración es similar a la de una entidad, no se define el comportamiento, sólo los terminales externos.

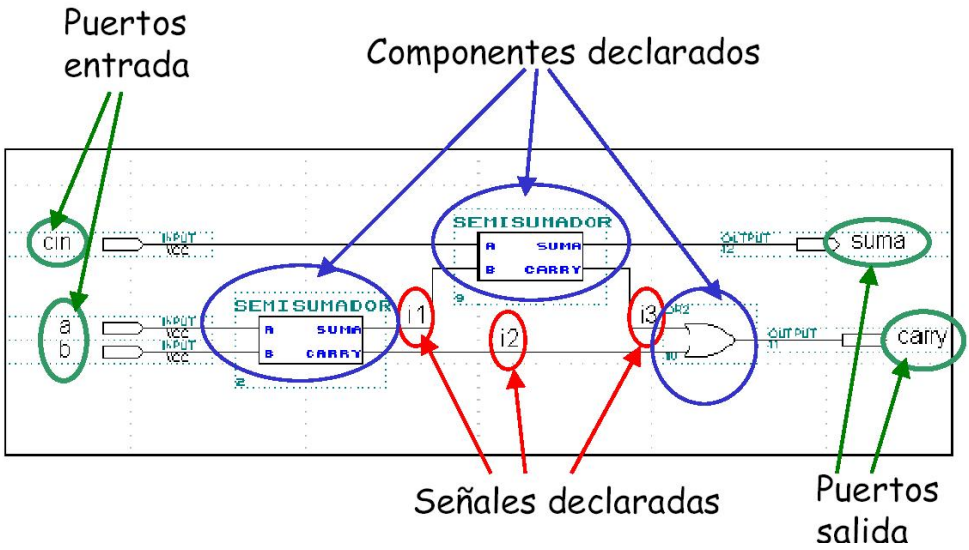
El comportamiento debe definirse fuera de la arquitectura donde se declara. Definiendo una Entidad de igual nombre que el Componente y con una arquitectura asociada al mismo.

Posteriormente, se pueden hacer llamadas al subcircuito sin necesidad de incluirlo en la descripción que se está realizando. Para ello, se utiliza:

```
etiqueta:nombre_componente PORT MAP (lista_asociación_ports);
```

El estilo estructural es fácilmente reconocible porque la operatividad del programa no se puede leer del código ya que está formado íntegramente por componentes y las señales que les unen a otros. Es decir, está formado por bloques o cajas negras a los cuales metemos información y sacamos las salidas, las cuales podrán o no ir a otros bloques. Para esto debemos conocer la operatividad de estos bloques.

Ejemplo: sumador de un bit




```
ENTITY sumador IS
PORT( a, b, cin : IN BIT;
suma, carry : OUT BIT);
END sumador;
ARCHITECTURE suma OF sumador IS
SIGNAL i1, i2, i3 : BIT;
-- Declaración del componente semisumador
COMPONENT semisumador
PORT (a, b : IN BIT; suma, carry : OUT BIT);
END COMPONENT;
-- Puerta or como componente
COMPONENT puerta_or
PORT (a, b : IN BIT; z : OUT BIT);
END COMPONENT;
BEGIN
u1 : semisumador PORT MAP (a, b, i1, i2);
u2 : semisumador PORT MAP (i1, cin, suma, i3);
u3 : puerta_or PORT MAP (i2, i3, carry);
END suma;
```

```
--Definición del componente semisumador
ENTITY semisumador IS
PORT (a, b : IN BIT;
suma, carry : OUT BIT);
END semisumador;
ARCHITECTURE semi OF semisumador IS
BEGIN
suma <= a XOR b;
carry <= a AND b;
END semi;
--Definición del componente puerta_or
ENTITY puerta_or IS
PORT (a, b : IN BIT;
z : OUT BIT);
END puerta_or;
ARCHITECTURE puerta_or OF puerta_or IS
BEGIN
z <= a OR b;
END puerta_or;
```

4.4.2. Estilo flujo de datos

Muestra la funcionalidad de un dispositivo mediante ecuaciones ejecutadas concurrentemente. En ellas se emplean asignaciones para las señales empleando operadores aritméticos y lógicos. Solo es recomendable usarlo para diseños tan sencillos. Como ejemplo ilustrativo se realiza la descripción del circuito de la Figura 1.

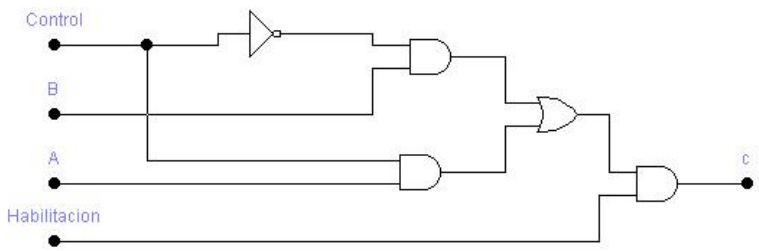


Figura 1

```
entity multi is port(
a,b     : in  bit;
control : in  bit;
enable  : in  bit;
c       : out bit
);
end multi;
architecture archmul of multi is
signal aux1,aux2,aux3: bit;
begin
aux1 <= b and (not(control));
aux2 <= a and control;
aux3 <= aux1 or aux2;
c    <= enable and aux3;
end archmul;
```

4.4.3. Estilo secuencial

Se emplean sentencias secuenciales y no concurrentes. Estas sentencias deben ir separadas de las concurrentes utilizando los Procesos (*Process*) y deben aparecen en el cuerpo de la arquitectura.

Las sentencias IF, CASE y LOOP pueden aparecer solamente en un Proceso.

La forma de un proceso es:

etiqueta_opcional : PROCESS (lista sensible _opcional o evento) declaraciones

BEGIN

sentencias secuenciales

END PROCESS etiqueta_opcional;

Por ejemplo:

```
Puerta_OR: PROCESS (a, b)
BEGIN
IF a='1’ or b=‘1’ THEN
z <= ‘1’;
ELSE
z<= ‘0’;
END IF;
END PROCESS;
```

La *lista sensible*, indica qué señales harán que se ejecuta el proceso, es decir, qué variable(s) debe(n) cambiar para que se ejecute el proceso. Las variables toman instantáneamente el valor especificado y sólo tienen sentido de existencia dentro de un proceso. Sin embargo, las señales, cambian su valor solamente al llegar al final del proceso.

Cuando un proceso no presenta una lista de sensibilidad, equivale a un bucle sin fin (el proceso se repite de forma continua). En ese caso, el proceso sólo se suspende cuando aparece una sentencia *Wait*.

```
PROCESS
BEGIN
IF (hora_alarma = hora_actual) THEN
sonido_alarma <= '1';
ELSE
sonido_alarma <= '0';
END IF;
WAIT ON hora_alarma, hora_actual;
END PROCESS;

PROCESS (hora_alarma, hora_actual)
BEGIN
IF (hora_alarma = hora_actual) THEN
sonido_alarma <= '1';
ELSE
sonido_alarma <= '0';
END IF;
END PROCESS;
```