

● ● ● ● ● ●

-
-
-
-
-
-

Asignación de una señal de forma condicional	91
Selección en la asignación de una señal	93

Práctica 2

Sumador de 4 bits y Sumador SIMD

En esta sesión se trata de consolidar los conocimientos adquiridos en la sesión anterior, a la vez que se presentan nuevas funcionalidades del programa LogicWorks y del lenguaje VHDL. Respecto a LogicWorks, una de ellas es encapsular diseños en módulos y otra es agrupar en el diseño varias señales individuales en un único trazo denominado bus. Respecto a VHDL, se describe cómo construir diseños jerárquicos y la especificación de buses. Estas funcionalidades son extremadamente útiles en la construcción de circuitos complejos.

En este sentido se construirá y simulará un **sumador de 4 bits con propagación serie del acarreo**, utilizando como módulo básico el **sumador de 1 bit (full adder)** diseñado en la sesión anterior. Así mismo, se añadirá a un sumador de 32 bits la capacidad de realizar en paralelo 4 operaciones de suma de tamaño de 1 byte (**sumador SIMD**).

Además, en un apéndice se describe la especificación en VHDL, utilizando constructores condicionales de asignación de señal, de componentes básicos en el diseño de circuitos combinatoriales como son el multiplexor, el decodificador y una puerta de tres estados.

Diseño de un sumador de 4 bits con propagación serie del acarreo

Para representar un número natural (a) se utiliza un vector de bits

$A = (a_{n-1}, \dots, a_1, a_0)$ que se interpreta de forma ponderada.

$$a = \sum_{i=0}^{n-1} a_i \times 2^i$$

donde a es el valor numérico que se calcula como la suma ponderada de los bits del vector de bits.

Expresiones algebraicas. Dados los vectores de bits (A, B) de entrada

$$A = (a_{n-1}, \dots, a_1, a_0) \quad B = (b_{n-1}, \dots, b_1, b_0)$$

que representan los números naturales a y b respectivamente, la operación suma se expresa como $s = (a + b) \bmod 2^n$. El resultado s se representa mediante un vector de bits $S = (s_{n-1}, \dots, s_1, s_0)$.

La suma de los vectores A y B se efectúa sumando bit a bit los vectores de bits y propagando el acarreo.

$$a_i + b_i + c_i = 2 \times c_{i+1} + s_i \quad 0 \leq i < n \quad c_0 = 0$$

Condición de irrepresentabilidad. Si la suma $a + b$ es mayor que $2^n - 1$ el resultado no se puede representar con n bits. La función de irrepresentabilidad es: $\text{Irre} = c_n$.

Modelo de comportamiento en VHDL

En esta sección, después de describir nuevas funcionalidades de VHDL, tales como la especificación de grupos de señales o buses, se describe un modelo de comportamiento para un sumador de 4 bits.

Tipos de datos en VHDL

Cada objeto VHDL debe tener asociado un tipo. La noción de tipo en VHDL es clave puesto que es un lenguaje fuertemente tipado que requiere que cada objeto sea de un cierto tipo. VHDL incluye varios tipos predefinidos y permite al usuario que defina sus propios tipos según los necesite.

Tipos de datos predefinidos en VHDL y tipos en la librería IEEE (objetos simples). Previamente ya hemos comentado el tipo `bit` predefinido en VHDL. De la librería “IEEE Standard 1164” (“package” `std_logic_1164.all`) hemos utilizado el tipo `std_logic`.

Tipos de datos predefinidos en VHDL y tipos en la librería IEEE (tipos de datos agregados). Previamente ya hemos comentado el tipo `bit_vector` predefinido en VHDL que es un vector de elementos tipo `bit`. En la librería “IEEE Standard 1164” se dispone del tipo `std_logic_vector` que es un vector de elementos de tipo `std_logic`.

Un vector de señales de un bit es de uso común en circuitos digitales. Por ejemplo, un bus de datos o de direcciones. Un vector es un conjunto de señales las cuales son todas del mismo tipo. Por ejemplo, una palabra es un vector de bits y memoria es un vector de palabras.

Ejemplo

```
signal bus: std_logic_vector (3 downto 0):= "0111";
```

Sintaxis

```
signal signal_name: std_logic_vector (n-1 downto 0) [ := initial value] ;  
signal signal_name: std_logic_vector (0 to n-1) [ := initial value] ;
```

Cada elemento de una señal de tipo `std_logic_vector` es un bit y los elementos se indexan de `n-1` a cero cuando se especifica `(n-1 downto 0)` y de 0 a `n-1` cuando se especifica `(0 to n-1)`. A la izquierda de "downto" o "to" se especifica el bit más significativo (MSB, Most Significant Bit) y a la derecha se especifica el bit menos significativo (LSB, Least Significant Bit). La especificación "downto" es apropiada para un sistema "little-endian". En este caso, los bits menos significativos se indexan con índices menores.

Un elemento de una señal de tipo `std_logic_vector` se especifica como `signal_name(indice)`. De forma semejante, se puede especificar un rango contiguo de bits de una señal de tipo `std_logic_vector`: `signal_name (msb downto lsb)`, donde `msb` es menor que el índice que referencia al bit más significativo y `lsb` es mayor que el índice que referencia al bit menos significativo. Esto es, se especifican los valores de los índices izquierdo y derecho de un objeto vector.

Ejemplo

```
signal bus: std_logic_vector (3 downto 0):= "0111";
```

Referencia al 2º elemento del vector de bits: `bus (2)`

Referencia a un rango contiguo de elementos del vector de bits: `bus (2 downto 1)`

Los siguientes ejemplos muestran dos formas de asignar el mismo valor a una señal de tipo `std_logic_vector`: elemento a elemento o todos los bits a la vez.

Ejemplo

```
signal bus: std_logic_vector (3 downto 0);
```

Elemento a elemento

```
d(3) <= '0';
```

```
d(2) <= '1';
```

```
d(1) <= '0';
```

```
d(0) <= '1';
```

Todos los bits a la vez

```
d <= "0101";
```

Ejemplo

```
signal bus: std_logic_vector (0 to 3);
```

Elemento a elemento

```
d(0) <= '1';
```

```
d(1) <= '0';
```

```
d(2) <= '1';
```

```
d(3) <= '0';
```

Todos los bits a la vez

```
d <= "1010";
```

En LogicWorks no se puede especificar un rango de bits en una señal cuando está en la izquierda de una asignación (`<=`).

Elementos léxicos en VHDL

Recordemos que la definición de valores en el tipo `std_logic` se ha efectuado mediante el tipo carácter. Puesto que el tipo carácter se utiliza en la definición, la forma de especificar los valores es parte de la definición y deben especificarse de la misma forma. Por ello deben especificarse con mayúsculas.

Caracteres. Un carácter se especifica entre comillas: Por ejemplo, los valores de tipo `std_logic` se especifican como: '0', '1', 'U', 'X', '-'.

Cadenas de caracteres. Una cadena de caracteres se especifica entre dobles comillas: Por ejemplo, el valor de una señal de 5 bits de tipo `std_logic_vector` se especifica como: "01UX-".

Los vectores de bits constantes se especifican como una cadena de caracteres.

Una cadena de bits representa una secuencia de bits. Para indicar que es una cadena de bits se utiliza el prefijo B. Las cadenas de caracteres también pueden interpretarse en hexadecimal y en octal. En estos dos últimos casos el prefijo es X y O respectivamente. Por ejemplo, en binario tendríamos B"101010", en hexadecimal X"B6" y en octal O"135".

Tengamos en cuenta que en hexadecimal un dígito representa 4 bits. En consecuencia el número B"101" es distinto del número X"5" puesto que en el primero sólo se utilizan 3 bits mientras que el segundo representa una secuencia de 4 bits. Casos similares deben tenerse en cuenta en octal.

Un vector en el que todos los bits tienen el mismo valor puede especificarse mediante la palabra clave "others" de la siguiente forma.

Ejemplo

```
signal bus: std_logic_vector (3 downto 0) ;  
bus <= (others => '0'); -- es lo mismo que bus <= "0000"
```

Operadores

Además de los operadores lógicos comentados en la sesión anterior se dispone de otros operadores.

Operadores aritméticos. +, - y * entre otros. El resultado que se obtiene es del mismo tipo que los operandos.

Para utilizar los operadores previos es necesario hacer visible el "package" `std_logic_arith.all` de la librería IEEE.

Otro operador permite concatenar vectores de bits.

Operador de concatenación ("Concatenate", &) de bits o vectores de bits. Junta en un vector de bits dos vectores de bits o un bit y un vector de bits.

Mediante este último operador se pueden especificar los valores de todos los bits de una señal a partir de señales que tienen menos bits o subconjuntos de bits de señales.

Ejemplo

```

signal bus: std_logic_vector (3 downto 0);
signal destino: std_logic_vector (4 downto 0);
destino <= bus(3) & bus (3 downto 1) & '1';

```

Modelo de comportamiento de un sumador de 4 bits

En la Figura 2.1 se muestra un esquema del circuito. Las señales A y B son grupos de señales de 4 bits. Esto es, los vectores de bits utilizados en las expresiones algebraicas descritas previamente. La señal de salida SUM también es un grupo de 4 señales.

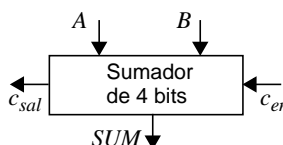


Figura 2.1 Módulo sumador de 4 bits.

Un sumador de 4 bits puede especificarse en VHDL utilizando un modelado de comportamiento y sentencias de asignación de señales concurrentes (Figura 2.2).

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity S4bits is
port (    A: in          std_logic_vector (3 downto 0) ;
          B: in          std_logic_vector (3 downto 0) ;
          cen: in        std_logic;
          SUM: out       std_logic_vector (3 downto 0) ;
          csal: out      std_logic);

end S4bits;

architecture behav of S4bits is
-- definición de una suma interna (SINTER) que tiene en cuenta el acarreo de salida
constant retardo: time := 60ns;

signal SINTER: std_logic_vector (4 downto 0);
begin
    SINTER <= ('0' & A) + ('0' & B) + ("0000" & cen) after retardo;
    csal <= SINTER(4);
    SUM <= SINTER (3 downto 0);

end behav;

```

Figura 2.2 Modelo de comportamiento en VHDL de un sumador de 4 bits.

Notemos que en la especificación del sumador de 4 bits se utiliza una señal interna con un bit más (mayor rango de valores) para disponer del acarreo de salida (Figura 2.2). La operación de suma se efectúa con vectores de 5 bits. Para ello se incrementa el número de bits con el que se representan los datos de entrada. En el caso de A y B se añade un bit, mientras que en el caso de c_{in} se añaden 4 bits a la izquierda. Como interpretamos los vectores de bits como números naturales, el valor de los bits añadidos es cero. En estas condiciones, el acarreo de salida es el bit más significativo de la señal SINTER y el valor de la suma está representado por los bits restantes.

En este ejemplo se especifica un retardo que es independiente del valor de los vectores de bits que se suman.

Esquema de un circuito sumador de 4 bits

En la Figura 2.3 se muestra el esquema de conexionado de 4 sumadores de 1 bit para construir un sumador de 4 bits con propagación serie del acarreo.

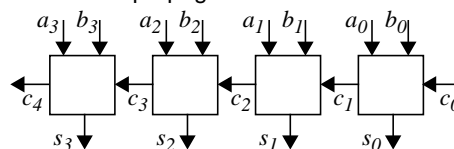


Figura 2.3 Sumador de 4 bits con propagación serie del acarreo.

Retardo del circuito. Es el tiempo, en el peor caso, que tarda en estabilizarse la última de las señales de salida, a partir del instante en que se han estabilizado todas las señales de entrada.

El esquema del sumador de 1 bit se muestra en la Figura 2.4. Supongamos que una puerta AND tiene un retardo de 10 ns, una puerta XOR tiene un retardo de 15 ns y una puerta OR-3 tiene un retardo de 15 ns.

En el esquema de la Figura 2.3, teniendo en cuenta el retardo de las puertas, la propagación del acarreo determina el retardo de la suma de los vectores de bit.

Se tardan 30 ns en calcular la señal s_0 y 25 ns en calcular la señal c_1 . En paralelo se ha calculado la señal $a_1 \oplus b_1$. Entonces, después de calcular c_1 el cálculo de la señal s_1 experimenta el retardo de una puerta XOR y el cálculo de la señal c_2 experimenta el retardo de una puerta AND y una puerta OR. Por tanto, el retardo de este sumador de 4 bits es $25 \times 4 = 100$ ns.

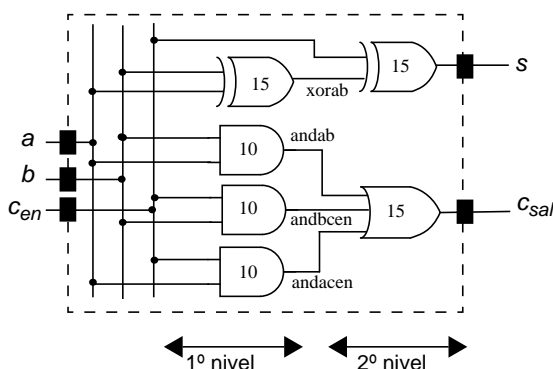


Figura 2.4 Sumador de 1 bit.

Diseño jerárquico o estructural en VHDL

A medida que los diseños son más complejos (incluyen más elementos) es necesario estructurar el diseño de forma modular. Ello ayuda a que el diseño se comprenda mejor, ya que se encapsulan detalles de implementación de algunas funcionalidades. Además, los módulos que implementan las funcionalidades pueden reutilizarse en otros diseños o refinar para el diseño que nos ocupa. Por otro lado, la verificación del correcto funcionamiento de un módulo concreto se puede efectuar de forma aislada e independiente de la interacción con otros módulos, lo cual facilita la tarea.

En el lenguaje VHDL se pueden describir modelos jerarquizados mediante lo que se denomina un modelado estructural.

Modelo estructural ("Structural model"). Se especifican los elementos o componentes que se utilizan y cómo se interconectan.

En la descripción de un modelo estructural, cada componente o elemento se ha definido previamente y puede haber sido descrito utilizando un modelo estructural o un modelo de flujo de datos (caso de modelo de comportamiento).

Una descripción estructural puede compararse a un diagrama esquemático como el de la Figura 2.3. Esto es, un esquema de bloques del circuito en el cual se describen los componentes y su interconexión. Por ello es útil partir del esquema de circuito para efectuar la descripción VHDL.

Recopilando, una descripción estructural se expresa en términos de subsistemas interconectados por señales. Cada subsistema puede a la vez estar expresado por una interconexión de subsistemas y así de forma sucesiva, hasta que se llega a un nivel de componentes “primitivos” descritos puramente en términos de comportamiento.

En VHDL la forma de describir un diseño estructural es:

- Especificación de la interfaz.
- Declaración de la lista de componentes que se van a utilizar.
- Declaración de las señales que definen los cables que interconectan los componentes. Esto es, las señales internas.
- Instanciación y etiquetado de cada instancia de un mismo componente de forma que cada instancia está unívocamente definida. Interconexión de las instancias.

Como ejemplo conductor utilizaremos el circuito de la Figura 2.3 que puede describirse utilizando como elementos sumadores de 1 bit.

El primer paso es independiente del tipo de modelado utilizado en VHDL para describir el funcionamiento de un módulo. La declaración de la interfaz se obtiene a partir de las señales de entrada y salida del módulo que se quiere modelar.

Declaración de componentes y señales internas

Antes de instanciar los componentes deben ser declarados en: a) la sección de declaraciones de la arquitectura, o b) en la declaración de “packages” que se utilizan en la cabecera del fichero. En VHDL esta declaración indica que los módulos están disponibles o accesibles para ser usados en el diseño que estamos especificando. Esta acción es la que describe el diseño de forma jerárquica.

Las señales que se van a utilizar para interconectar los módulos que se utilizan en el diseño (señales internas) se declaran dentro del cuerpo de la arquitectura, después de la palabra clave “architecture” y antes de la palabra clave “begin”.

Seguidamente se describe la declaración de componentes en un “package”. La opción de declaración de componentes en la arquitectura se describe en el Apéndice 2.1.

Declaración de los componentes en un “package”. Cuando el componente se declara en un “package” se utilizan las cláusulas “library” y “use” en la cabeza del fichero para indicar la librería y el “package”.

Sintaxis	Sintaxis
library logical-library-name;	library logical-library-name;
use logical-library-name. package.all;	use logical-library-name. package.my_component;

En la declaración de la parte derecha sólo es visible el componente que se quiere instanciar.

Instanciación de componentes e interconexión

El último paso al crear un modelo estructural es crear las instancias de los módulos que se utilizan en el diseño y describir su interconexión. En la interconexión se efectúa una asociación (mapeo) entre las señales de la interfaz de los módulos que se utilizan como componentes y las señales de entrada, salida e internas del diseño que nos ocupa. Estamos asociando señales en un nivel de la jerarquía del diseño con señales definidas en un nivel previo de la jerarquía.

La instanciación de componentes se efectúa después de la palabra clave “begin” en el cuerpo de la arquitectura.

```
begin
-- Component instantiation and connections
-- Concurrent Statements
end architecture_name;
```

Una sentencia de instanciación de componente es una sentencia concurrente. El orden en el cual se instancian los componentes no es importante desde el punto de vista de la ejecución (simulación) puesto que las sentencias son concurrentes y por tanto se ejecutan en paralelo. Esto es, el esquema de circuito que se describe mediante las sentencias es el mismo independientemente del orden en que se especifiquen.

Cada instancia de un módulo tiene un nombre que la identifica de forma unívoca y está seguida por el nombre del componente. El nombre del componente precede a la palabra clave “port map” a partir de la cual se especifica la asociación de señales entre niveles de la jerarquía. La sintaxis para la instanciación de un componente es la siguiente.

Sintaxis

```
instance_name: component_name
port map (port1 => signal1, port2 => signal2, . . ., portn => signaln);
```

En primer lugar se especifica el nombre de la instancia seguido de dos puntos y el nombre del componente y la palabra clave “port map”. El nombre de la instancia o etiqueta puede ser cualquier identificador legal y es el nombre de la instancia concreta.

El nombre del componente es el nombre declarado en el “package”. El nombre del puerto (“port1”) es el nombre del puerto en la especificación del componente. Señal (“signal1”) es el nombre de la señal a la cual se conecta el puerto que se ha especificado antes del símbolo =>. Notemos que se asocian de forma explícita los puertos y las señales mediante la asociación de nombres¹. Por tanto, aunque el nombre a la derecha e izquierda del símbolo => sea el mismo no existe ambigüedad.

1. La asociación también puede efectuarse de forma implícita, pero no lo utilizaremos.

Diseño estructural de un sumador de 4 bits

En la Figura 2.5 se muestra el esquema de circuito del sumador de 4 bits. Las entradas y salidas se especifican como buses de señales cuando es el caso.

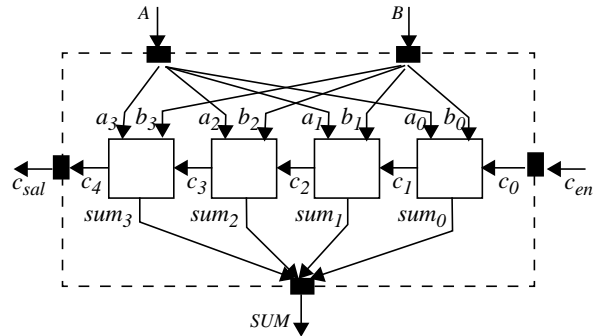


Figura 2.5 Esquema de un sumador de 4 bits construido con sumadores de 1 bit.

La declaración de la interfaz (“entity”) es la misma que en el caso de utilizar un modelo de comportamiento (Figura 2.2).

Además de otras librerías y “packages”, hay que indicar que la especificación del componente se encuentra en un “package” contenido en una librería. En nuestro caso la librería se denomina “Libs” y el “package” donde hemos almacenado el componente en la práctica anterior se denomina MiLib. A continuación se muestran dos posibilidades.

<pre>library IEEE; use IEEE.std_logic_1164.all; library Libs; use Libs.MiLib.all;</pre>	<pre>library IEEE; use IEEE.std_logic_1164.all; library Libs; use Libs.MiLib.S1bitDF;</pre>
---	---

La declaración de “library Libs” especifica la colección de “packages” que se tienen abiertos en LogicWorks. Esto es, las librerías (terminología en LogicWorks) accesibles en el panel de Librería ubicado en la parte derecha de la ventana de LogicWorks.

La declaración de señales y el resto del cuerpo de la arquitectura se muestra a continuación.

```
signal c1, c2, c3, c4: std_logic;
begin
  S1bit0: S1bitDF port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0));
  S1bit1: S1bitDF port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1));
  S1bit2: S1bitDF port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2));
  S1bit3: S1bitDF port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3));
  csal <= c4;
end structural;
```

Las señales internas definidas se corresponden con los nombres de la Figura 2.5.

En el Apéndice 2.1 se encuentra el listado completo.

Trabajo: Edite un fichero para modelar de forma estructural un sumador de 4 bits. Obtenga los componentes del sumador de 1 bit (Figura 1.24, página 20) de un “package”. Esto es, la librería creada en la práctica 1 (MiLib, elemento S1bitDF). Se puede crear y editar un fichero VHDL efectuando los pasos File -> New y en la ventana emergente se utiliza el deslizador vertical para poder seleccionar el rótulo “VHDL”. Posteriormente hay que seguir los pasos descritos en la práctica 1 para almacenar el fichero en disco. A continuación compile el fichero. Una vez no existan errores de compilación construya un elemento de librería para el sumador de 4 bits. Este elemento se almacenará en la librería que ha creado en la práctica 1. En la práctica 1 se ha descrito todo el proceso para crear un elemento y almacenarlo en una librería (página 23).

Herramienta de dibujo de esquemas

En esta sección utilizaremos el módulo encapsulado en la sección previa para crear un esquema de circuito.

Trabajo: Cree un fichero con la extensión .cct para incluir un esquema de circuito y ubique en el fichero el sumador de 4 bits que ha almacenado previamente en la librería.

En el sumador de 4 bits encapsulado observamos que la anchura y color de los puertos (patillas del encapsulado) de entrada y salida depende del número de señales que transportan.



Figura 2.6 Módulo sumador de 4 bits. Buses y señales de entrada y salida.

Buses. LogicWorks provee la funcionalidad de representar varias señales con un único trazo (cable) en el dibujo, mediante un elemento denominado bus.

Un bus se manipula, desde el punto de vista de trazado en el esquema de circuito, de igual forma que un cable que transporta sólo una señal.

Peine. El acceso individualizado a las señales que transporta el bus se efectúa mediante un elemento denominado peine (por la estructura del símbolo).

Para crear un peine que se conecte a los puertos de un módulo se efectúan los siguientes pasos:

- Selección del bus (componente señal en la patilla).
- Los siguientes pasos se muestran en la parte izquierda de la Figura 2.7. Entonces emerge la ventana que se muestra en la parte derecha de la Figura 2.7. En esta ventana se muestra el nombre de las señales que transporta el bus. Conteste afirmativamente y se visualiza un peine.

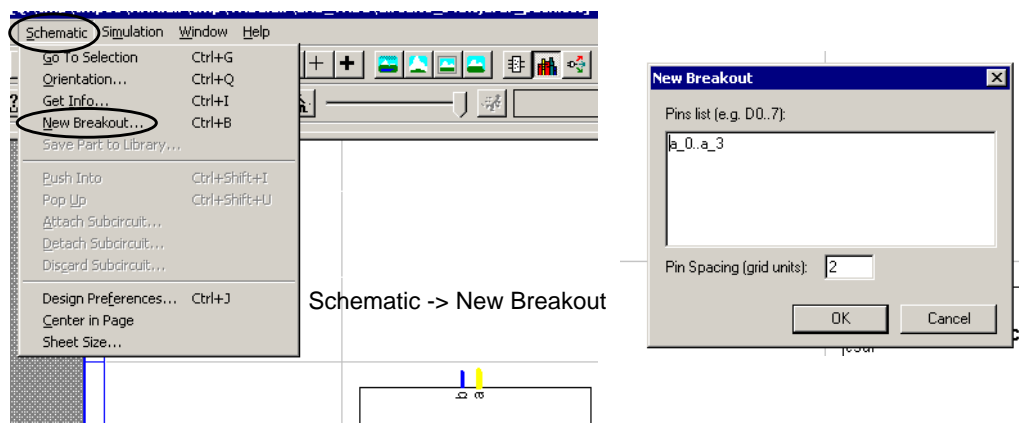


Figura 2.7 Creación de un peine. Visualización de las señales individuales que transporta el bus.

También puede solicitarse la creación del peine seleccionando el bus y pulsando el botón derecho del ratón. En la ventana que emerge se selecciona "Breakout" (Figura 2.8).

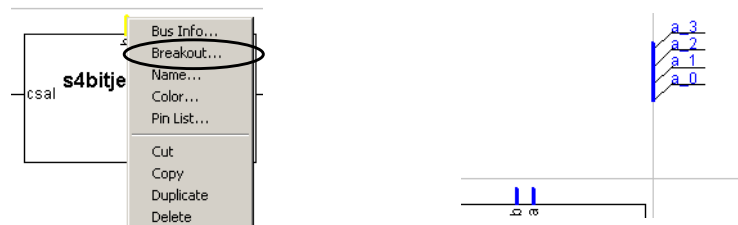


Figura 2.8 Creación de un peine y muestra de un peine.

El nombre de las señales se diferencia en el valor numérico del sufijo. El valor numérico sigue una secuencia creciente. La sintaxis es la siguiente: nombre_número. El texto es una secuencia de caracteres del alfabeto. En la Figura 2.8 la etiqueta es a y los valores numéricos son (0, 1, 2, 3).

- El siguiente paso es posicionar el peine en el circuito. Para ello se mueve el cursor a la posición deseada y se pulsa el botón izquierdo del ratón. Posteriormente se pulsa el botón derecho para que aparezca de nuevo el cursor.

- Conexión del peine con el bus correspondiente. La conexión se puede efectuar seleccionando el icono que se muestra en la parte izquierda de la Figura 2.9. Note que es igual al símbolo para dibujar cables que transportan sólo una señal, pero es más grueso. El bus se dibuja y manipula igual que un cable que transporta sólo una señal. El símbolo que se visualiza al dibujarlo se muestra en la parte derecha.



Figura 2.9 Icono para dibujar buses. Símbolo de un bus.

Nota: Seleccionando el trazo grueso del peine en un extremo, pulsando el botón izquierdo del ratón y moviendo el cursor se puede alargar el trozo para conectarlo donde interese. La creación de un ángulo de 90° sigue las mismas pautas que el trazado de cables.

Etiquetado de buses. Los buses se etiquetan de la misma forma que las señales, lo cual se ha explicado en la práctica 1 (página 34).

Trabajo: Cree los peines para acceder individualmente a las señales de las patillas (puertos) del módulo sumador de 4 bits y conéctelos a los puertos de entrada y salida. Además etiquete los buses y las señales del módulo.

Información sobre el bus

Señales que transporta un bus. Se selecciona el bus y se pulsa el botón derecho del ratón. Entonces aparece la ventana, que se muestra en la parte izquierda de la Figura 2.10, en la que se selecciona el rótulo "Bus Info". Posteriormente emerge una ventana que se muestra en el centro de la Figura 2.10. En esta ventana se visualizan las señales que transporta el bus.

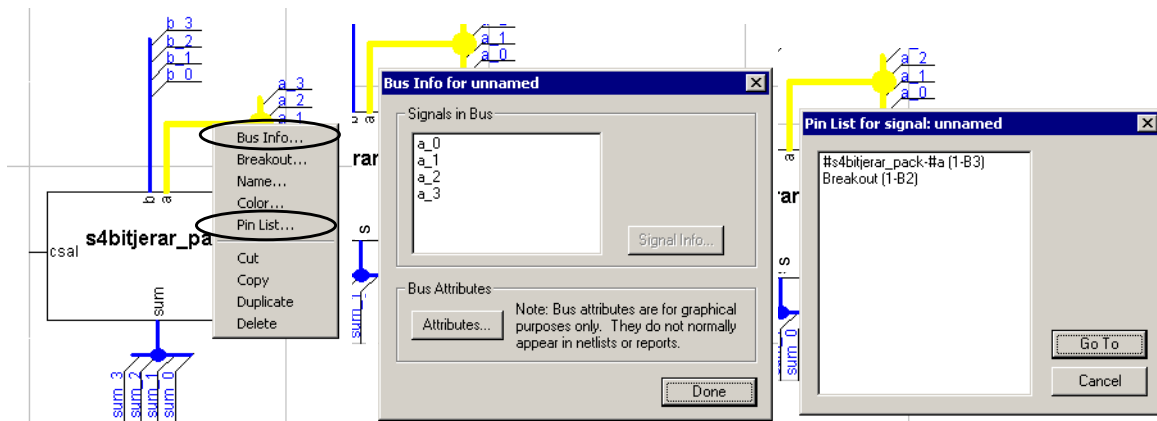


Figura 2.10 Observación de las señales que transporta un bus.

Lista de conexiones de un bus. Se efectúan los mismos pasos que en el caso previo pero se selecciona el rótulo "Pin List" (parte izquierda de la Figura 2.10). En la ventana emergente (parte derecha de la Figura 2.10) una línea especifica en primer lugar el nombre del elemento y al final de la línea, entre paréntesis, se indican las coordenadas del elemento en la ventana de diseño. La ventana de diseño está cuadrículada. Cada cuadrícula se identifica por dos coordenadas. La primera coordenada se refiere al eje vertical (valor numérico) y la segunda coordenada se refiere al eje horizontal (letra del alfabeto).

Señales de entrada y de salida en hexadecimal

Una vez diseñado el circuito hay que comprobar su funcionamiento asignando valores a sus entradas y observando los valores de las salidas. Cuando el número de señales de entrada y salida es grande, la utilización de elementos a nivel de bit es engorrosa. La librería *ac.clf* dispone de dos elementos que permiten activar y visualizar las señales en grupos de 4 bits.

- **Teclado hexadecimal (Hex Keyboard):** permite establecer valores entre 0 y F a un grupo de 4 señales de entrada de 1 bit mediante la pulsación del botón izquierdo

del ratón, después de posicionar el cursor sobre el símbolo numérico. La patilla más cercana al rótulo F es la de menor peso.

- **Visor hexadecimal (Hex Display):** efectúa las funciones de visualizador y conectado a un grupo de 4 salidas de 1 bit muestra el valor de las señales en una codificación hexadecimal. En el símbolo que representa al elemento existe un punto que indica que la salida más cercana es la de menor peso.

La Figura 2.11 muestra el sumador de 4 bits con los elementos teclado hexadecimal y visor hexadecimal conectados.

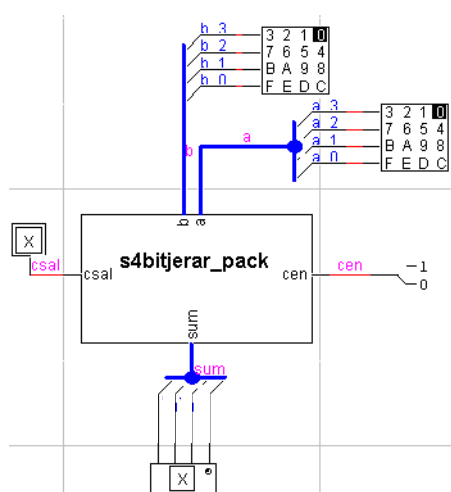


Figura 2.11 Sumador de 4 bits con dispositivos para establecer valores de entrada y observar la salida.

Trabajo: Añada los elementos que permiten establecer valores en los buses, en las señales de entrada y observar los valores de las salidas.

Comprobación del funcionamiento lógico

Trabajo: Utilizando los teclados, el interruptor y los visores compruebe el funcionamiento lógico del circuito para distintos valores de las entradas.

Fichero de comprobación. El funcionamiento lógico también puede comprobarse mediante un fichero de comprobación. El formato del fichero es el mismo que en la práctica anterior. La diferencia radica en que deben especificarse grupos de señales para

los puertos de entrada a y b en el sumador de 4 bits. La forma de efectuarlo es utilizar notación hexadecimal. Por ejemplo, el valor diez se representa como A. En la Figura 2.12 se muestra un fichero donde se han especificado varios vectores de comprobación para el sumador de 4 bits.

\$I a	\$I b	\$I cen	\$E sum	\$E csal
0	0	0	0	0
0	4	0	4	0
0	4	1	5	0
0	a	1	b	0
2	a	1	d	0
5	2	1	8	0
1	1	1	3	0

Figura 2.12 Muestra de vectores de comprobación para el sumador de 4 bits.

Medida del tiempo de retardo


En esta sección se muestran dos formas de medir el retardo. En primer lugar se muestra cómo medir el retardo utilizando un esquema de circuito y elementos de entrada y salida. Para ello hay que realizar un paso previo que es preparar el circuito. Posteriormente se efectúa la simulación temporal para medir el retardo. En segundo lugar se muestra cómo utilizar ficheros de estimulación para medir el retardo.

Inicialización del circuito

Para determinar experimentalmente el retardo del circuito es necesario seguir el proceso de simulación que se describe a continuación.

Preparación del circuito. Una forma de garantizar que una señal de entrada experimenta el retardo de propagación por los elementos del circuito, es establecer antes el valor indefinido en los cables del circuito.

Los pasos que deben efectuarse son los siguientes:

- 1 Selección del icono . Se pausa la simulación
- 2 Establecer el valor inicial en el componente señal de las patillas de los elementos que se utilizan para establecer valores lógicos en las entradas (teclado hexadecimal e interruptor).

Separe el teclado del peine. Seleccione la señal y pulse el botón derecho del ratón. En la ventana emergente seleccione el rótulo "Attributes" (parte izquierda de la Figura 2.13).

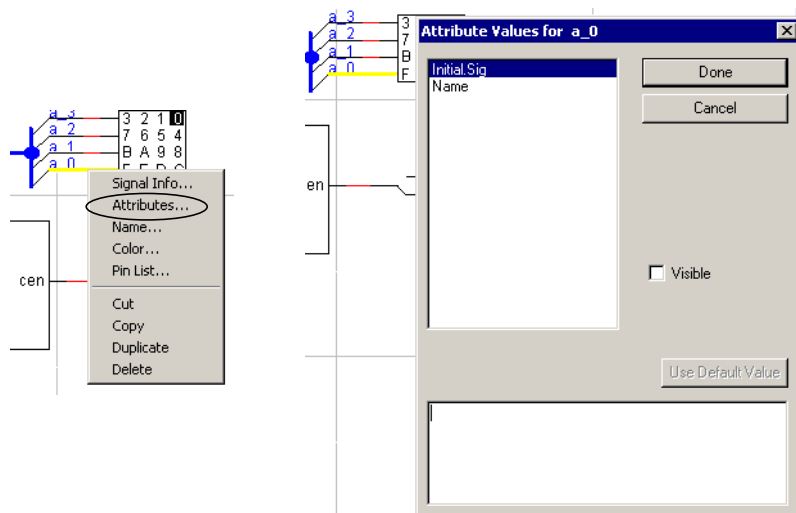


Figura 2.13 Pasos para establecer valores iniciales en la componente señal de una patilla.



Posteriormente se visualiza la ventana mostrada en la parte derecha de la Figura 2.13. Se selecciona el campo “Initial.Sig” y se establece el valor x (indefinido) posicionando el cursor en el marco inferior y utilizando el teclado.

Nota: Este paso sólo debe efectuarse en la primera simulación.



Trabajo: Establezca el valor indefinido en todas las señales que salen de los teclados hexadecimal y del interruptor.

Simulación temporal del circuito

Para determinar el retardo de un conjunto de valores de entrada deben efectuarse los siguientes pasos:

- 1 Parar el simulador pulsando en el icono .
- 2 Puesta a cero de la simulación mediante el icono .



- 3 Activación de la simulación mediante el icono  o  o el dial que permite establecer una velocidad de simulación.

La Figura 2.14 muestra la ventana de Tiempo después de efectuar los pasos previos.

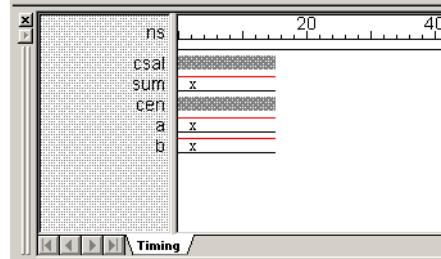





Figura 2.14 Propagación del valor indefinido hasta las salidas del sumador de 4 bits.

- 4 Cuando se estabilizan las señales de salida, la simulación se suspende ya que no se modifican los valores de salida y las entradas permanecen estables. Entonces se pausa el proceso de simulación mediante el icono .
- 5 Posiciónese en cada uno de los elementos de entrada de valores lógicos (teclado hexadecimal e interruptor). Una vez posicionado en uno de ellos, establezca un valor pulsando el botón izquierdo del ratón. El valor que se ha establecido no se observará hasta que se vuelva a activar la simulación. El valor que se establezca debe ser distinto del que había antes de la puesta a cero.
- 6 Active la simulación mediante el icono  o  o el dial que permite establecer una velocidad de simulación.

La Figura 2.15 muestra la ventana de Tiempo después de efectuar los pasos previos. En la Figura 2.16 se muestra la evolución de la ventana temporal en ASCII. La codificación de las filas de la Figura 2.16 se ha explicado en la práctica 1. En cuanto a las patillas que transportan varias señales (buses) el valor que se muestra está en hexadecimal. En la primera fila se observa la etiqueta del bus seguida, entre corchetes, en la segunda fila de las etiquetas de las señales individuales de menor a mayor ponderación.

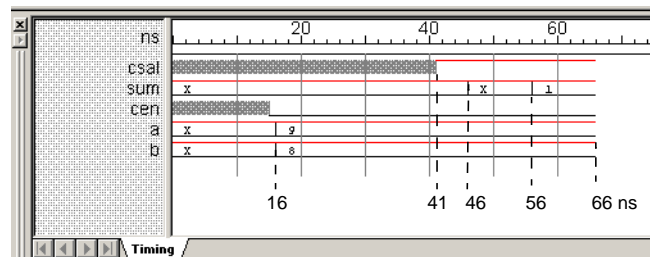


Figura 2.15 Simulación del sumador de 4 bits con valores de entrada: $a=9$, $b=8$ y $cen=0$.

Notemos que el instante de tiempo de inicio (Figura 2.15 y Figura 2.16) está retrasado 1 ns respecto al instante de tiempo final de la Figura 2.14. Este retardo es debido al elemento teclado hexadecimal y a que el simulador siempre establece 1 ns para propagar

el valor establecido en el teclado a la señal de las patillas del elemento. En la tabla de la Figura 2.16 se ha subrayado la fila correspondiente. Observemos también que la simulación prosigue una vez las señales de salida (csal y sum) son estables. Ello es debido a que alguna señal interna aún no está estabilizada.

A partir de los valores de la tabla de la Figura 2.16 se determina que el retardo para las entradas utilizadas es 40 ns (56 - 16).

\$T	\$D	\$O csal	\$O sum [sum_0 sum_1 sum_2 sum_3]	\$O cen	\$O a [a_0 a_1 a_2 a_3]	\$O b [b_0 b_1 b_2 b_3]
0	15NS	X	X	X	X	X
15NS	1NS	X	X	0	X	X
16NS	25NS	X	X	0	9	8
41NS	5NS	1	X	0	9	8
46NS	10NS	1	X	0	9	8
56NS	10NS	1	1	0	9	8
66NS	0	1	1	0	9	8

Figura 2.16 Evolución de la ventana temporal en ASCII.

Para observar, en la ventana de tiempo, las señales individualizadas que transporta un bus hay que desagregar las señales.

Desagregación de señales agrupadas en señales individuales. En la parte izquierda de la ventana de tiempo se encuentran las etiquetas. Seleccione la etiqueta del bus cuyas señales se quieren observar de forma individualizada y pulse el botón derecho del ratón. Seguidamente emerge la ventana que se muestra en la parte izquierda de la Figura 2.17. Seleccione el rótulo “Ungroup”.

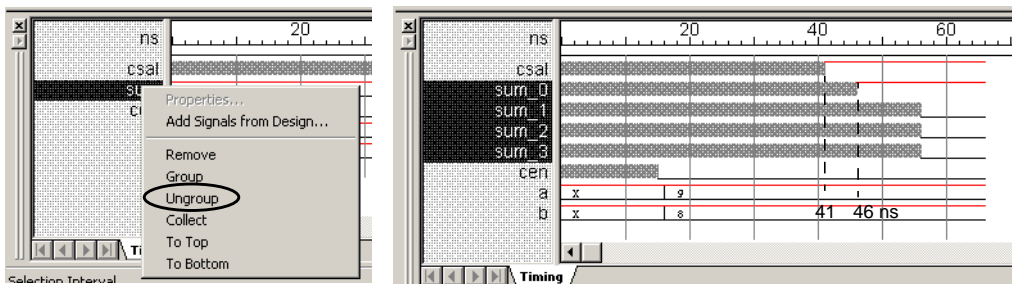


Figura 2.17 Orden para desagregar. Traza de la ventana temporal con las señales de sum desagregadas.

En la parte derecha de la Figura 2.17 se observan las señales de forma individualizada. Notemos que en la Figura 2.17 y también en la Figura 2.15 y en la tabla de la Figura 2.16 las señales de salida (csal y sum) son estables antes de que el simulador deje de simular. Ello es debido a que alguna señal interna no está estabilizada hasta que han transcurrido los 66 ns.

Por otro lado observemos que en $t = 46$ ns la señal sum_0 es estable mientras que el resto de señales del bus sum aún no lo son. Este comportamiento se observa de forma indirecta en la Figura 2.15 y en la tabla de la Figura 2.16. En la Figura 2.15 se observa que en $t = 46$ ns se vuelve a mostrar el valor 'X' para la señal suma. En la tabla de la Figura 2.16 se observa que la 5ª fila ($t = 46$ ns) es igual a la 4ª fila. Notemos también que la señal sum_0 se estabiliza después de 5 ns de que se haya estabilizado la señal csal.

Agrupación de un conjunto de señales. Se seleccionan todas las etiquetas de las señales que se quieren agrupar. Una vez se ha seleccionado la primera señal debe pulsarse la tecla "shift" para seguir seleccionando señales. Posteriormente se pulsa el botón derecho del ratón. En la ventana emergente se selecciona el rótulo "Group".

Una vez ha sido agrupado un conjunto de señales hay que determinar la ponderación de cada una de ellas en la señal agrupada. Ello determina el valor que se observa en el diagrama de tiempos.

Ponderación de las señales individuales en un bus. Se selecciona la señal de interés y se pulsa el botón derecho del ratón. En la ventana emergente se selecciona "Properties ..." (parte izquierda de la Figura 2.18). En la siguiente ventana emergente, en la parte inferior izquierda, se muestra la ponderación de las señales individuales (centro de la Figura 2.18). Seleccionando una de ellas se puede modificar la ponderación utilizando los rótulos "Promote" (incrementar la ponderación y "Demote" (reducir la ponderación). Si se quiere invertir la ponderación de todas las señales se selecciona "Reverse All".

La acción de establecer la ordenación en un conjunto de señales agrupadas es independiente de que previamente hayamos desagregado el mismo conjunto de señales.

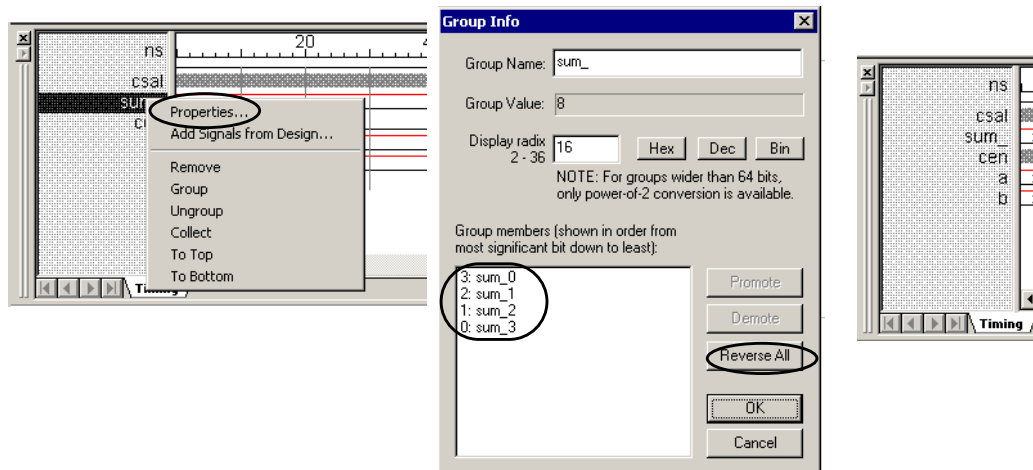


Figura 2.18 *Modificación de la ponderación de las señales individuales en la señal que las agrupa.*

Notemos que al agrupar las señales la etiqueta que se observa en la ventana de tiempos es “sum_”. Esta etiqueta se corresponde con el nombre de las señales individuales una vez se ha suprimido el sufijo numérico (parte derecha de la Figura 2.18).

Trabajo: Evalúe el retardo para varios vectores de entrada. Por ejemplo los de la Figura 2.12


Ficheros de estimulación

La medida del retardo para distintos conjuntos de valores se puede automatizar utilizando ficheros de estimulación. Un fichero de estimulación para medir retardos es similar a un fichero de comprobación, pero entre vectores de estímulos cuyo retardo se quiere medir, se inserta un vector de estímulos con las entradas indefinidas. El fichero de estimulación debe tener el mismo nombre que el del fichero que contiene el circuito a estimular y la extensión debe ser “.tsv”, como en un fichero de comprobación. En la Figura 2.19 se muestra un ejemplo. Este fichero se ha construido partiendo del fichero utilizado para la comprobación del funcionamiento lógico del sumador de 4 bits (Figura 2.12).

\$I a	\$I b	\$I cen
.X	.X	.X
0	0	0
.X	.X	.X
0	4	0
.X	.X	.X
0	4	1
.X	.X	.X
0	a	1
.X	.X	.X
2	a	1
.X	.X	.X
5	2	1
.X	.X	.X
1	1	1

Figura 2.19 *Fichero de estimulación para medir retardos.*

Una vez se ha creado y almacenado en disco el fichero de estimulación se siguen los siguientes pasos para efectuar la simulación.

- Se abre el fichero que contiene el circuito que se quiere estimular y se comprueba que en la parte correspondiente de la ventana de tiempos aparezcan las etiquetas de las señales que se quieren observar.
- Se para el simulador  .
- Para que el simulador retenga los resultados de la simulación hay que efectuar los

siguientes pasos. Seleccione la opción “Simulation->Simulation Options” (Figura 2.20). En la ventana emergente hay que marcar “Retain forever” (parte derecha de la Figura 2.20).

- Finalmente se activa la simulación y se observa el resultado en la ventana de Tiempo.

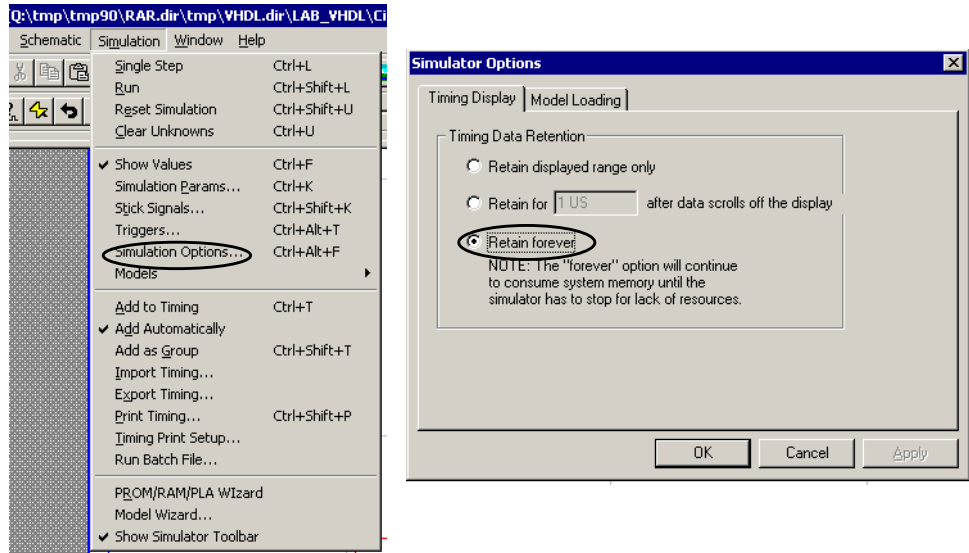


Figura 2.20 Pasos para almacenar la traza en la ventana de tiempos.

Por defecto, el simulador no estimula las entradas con un nuevo vector de comprobación hasta que todas las señales (salidas e internas) del caso previo son estables.

En la Figura 2.21 se muestra la traza de las señales cuando se utiliza como entrada el fichero de estímulos de la Figura 2.19. En la tabla de la Figura 2.22 se muestran sólo algunos vectores de comprobación de los incluidos en el fichero de estímulos.

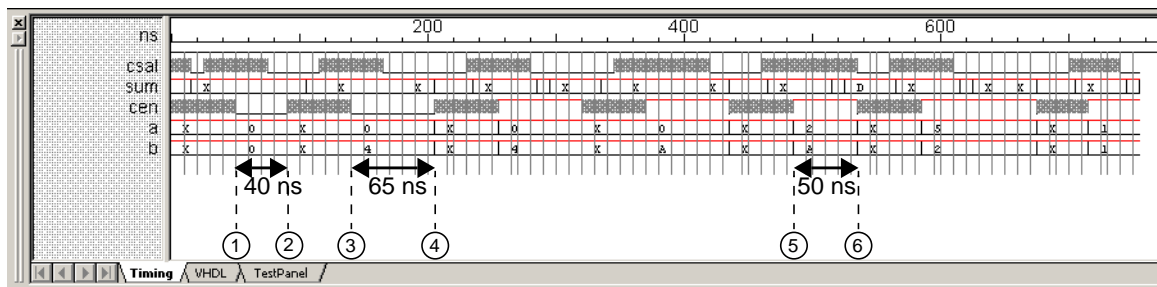


Figura 2.21 Traza en la ventana de tiempos después de utilizar un fichero de estímulos.

En la Figura 2.22 se observa que en $t = 50$ ns las entradas se estimulan con el vector ($cen = 0$, $a = 0$ y $b = 0$), interpretándose los últimos dos dígitos en hexadecimal (marca 1 en Figura 2.21). En $t = 90$ ns las salidas son estables ($csal = 0$ y $Sum = 0$, marca 2 en Figura 2.21). Entonces, el retardo es $90 - 50 = 40$ ns. Otro ejemplo es el vector de estímulos (0, 0, 4) el cual se aplica en $t = 140$ ns (marca 3 en Figura 2.21). Las salidas son estables en $t = 205$ ns (marca 4 en Figura 2.21). Entonces, el retardo son 65 ns. Finalmente comentamos el vector de estímulos (1, 2, A) que se aplica en $t = 485$ ns (marca 5 en Figura 2.21). Las salidas son estables en $t = 535$ ns (marca 6 en Figura 2.21). Por tanto, el retardo son 50 ns. En la Figura 2.21 se han identificado los 3 vectores de estímulos que se han comentado.

\$T	\$D	\$O csal	\$O sum_ [sum_0 sum_1 sum_2 sum_3]	\$O cen	\$O a [a_0 a_1 a_2 a_3]	\$O b [b_0 b_1 b_2 b_3]
50NS	25NS	X	X	0	0	0
75NS	5NS	0	X	0	0	0
80NS	10NS	0	X	0	0	0
90NS	15NS	0	0	X	X	X
140NS	25NS	X	X	0	0	4
165NS	5NS	0	X	0	0	4
170NS	10NS	0	X	0	0	4
180NS	25NS	0	X	0	0	4
205NS	15NS	0	4	X	X	X
485NS	30NS	X	X	1	2	A
515NS	10NS	X	X	1	2	A
525NS	10NS	X	D	1	2	A
535NS	15NS	0	D	X	X	X

Figura 2.22 Resultado en ASCII después de utilizar un fichero de estímulos.

Trabajo: Cree un fichero de estimulación y evalúe el retardo para varios vectores de entrada.

Programas de prueba en VHDL (“testbench”)

Un programa de verificación es un programa en VHDL diseñado específicamente para comprobar el funcionamiento de un circuito.

Cuando un programa de prueba se vaya a utilizar en un esquema de circuito, en la declaración “entity” se declaran como puertos de salida los puertos de entrada del circuito que se quiere comprobar (Figura 2.23).

En el cuerpo de la arquitectura del programa de prueba se especifica para cada puerto de salida un frente de onda. Esto es, la evolución de la señal en ese puerto a medida que transcurre el tiempo. Para ello se utilizan sentencias de asignación de señal concurrente.

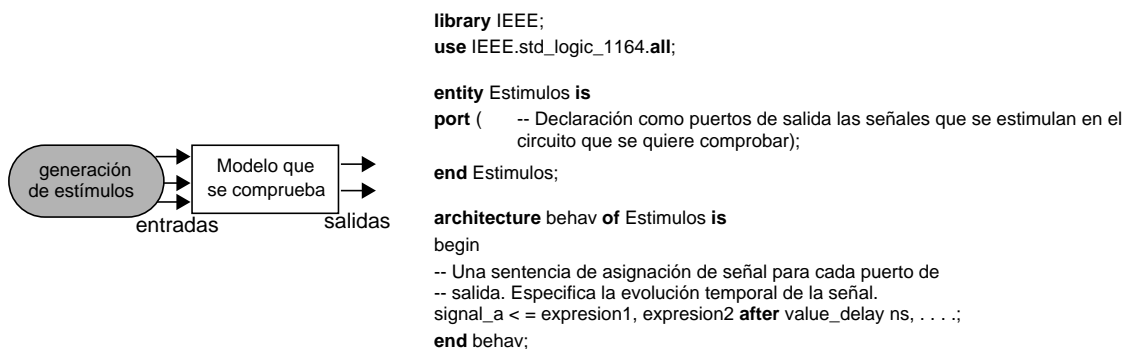


Figura 2.23 Esquema de circuito con un módulo para generar las señales de estímulo. Patrón de un fichero VHDL para la generación de señales de estímulo.

En la evolución temporal de las señales se efectúa un barrido de los valores que se quiere comprobar. El retardo que se especifica entre cambios en el valor de una señal debe ser igual o superior al retardo del circuito. En caso contrario, si se cambian las entradas antes de que las salidas sean estables, las salidas del circuito no serán correctas.

Frente de onda

En una sentencia de asignación de señal la expresión en la parte derecha se denomina elemento de frente de onda. Un elemento de frente de onda describe la asignación de un valor a una señal en un instante de tiempo determinado y tiene la siguiente sintaxis.

Ejemplo	Sintaxis
s <= a and b after 2 ns;	signal_name <= value expression [after time expresion] ;

Después del operador de asignación (<=) distinguimos una expresión que calcula un valor a la izquierda de la palabra clave “after” y una expresión de tiempo a la derecha de la misma palabra clave. La expresión de la izquierda evalúa el nuevo valor que se asigna a la señal y la evaluación de la expresión de tiempo determina el instante de tiempo relativo

en el cual la señal tomará el nuevo valor. El valor de la expresión de tiempo se suma al tiempo de simulación actual para determinar el instante de tiempo en que la señal recibirá el nuevo valor. Con respecto al tiempo de simulación actual, el par tiempo-valor representa el valor futuro de la señal y se denomina transacción.

En una asignación de señal podemos especificar varios elementos de frente de onda y como resultado varias transacciones. Las transacciones (elementos de frente de onda) se especifican en un orden de tiempo creciente. El frente de onda representa la sucesión de valores de una señal (física) en el transcurso del tiempo. Se están describiendo transacciones de una señal en instantes de tiempo futuro. La sintaxis es la siguiente.

Sintaxis

```
signal_name <= value expr1 [ after time expr1], value expr2 [ after time expr2], . . . ;
```

siendo $\text{expr1} < \text{expr2}$.

En el caso que nos ocupa “value exp” es un objeto constante y en cada transacción se produce un evento ya que el valor de la señal cambia.

Por ejemplo, el frente de onda de la parte superior de la Figura 2.24 se obtiene mediante la sentencia de asignación de señal mostrada en la parte inferior de la misma figura. Notemos que los tiempos son todos relativos al primer elemento del frente de onda.

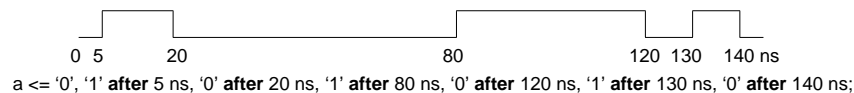


Figura 2.24 Especificación de un frente de onda.

Programa de prueba para el sumador de 4 bits

Para el sumador de 4 bits, en la Figura 2.25 se muestra un ejemplo de programa que permite verificar el funcionamiento para una secuencia de valores de entrada.

En la Figura 2.25 se especifican los valores en binario. Otra posibilidad es especificar los valores en hexadecimal. Por ejemplo, el frente de onda para la señal b puede especificarse de la siguiente forma.

```
b <= (others => 'x'), x"0" after 200 ns, (others => 'x') after 400 ns, x"4" after 600 ns, (others => 'x') after 800 ns, x"4" after 1000 ns, (others => 'x') after 1200 ns;
```

Después de compilar el fichero sin errores se crea un elemento de librería de la forma descrita en la práctica 1.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity S4bitJerar_pack_estimulos is
port (
    a: out      std_logic_vector (3 downto 0) ;
    b: out      std_logic_vector (3 downto 0) ;
    cen: out     std_logic );
end S4bitJerar_pack_estimulos;

architecture behav of S4bitJerar_pack_estimulos is
begin
    a <= "xxxx", "0000" after 200 ns, "xxxx" after 400 ns, "0000" after 600 ns, "xxxx" after 800 ns,
        "0000" after 1000 ns, "xxxx" after 1200 ns;
    b <= "xxxx", "0000" after 200 ns, "xxxx" after 400 ns, "0100" after 600 ns, "xxxx" after 800 ns,
        "0100" after 1000 ns, "xxxx" after 1200 ns;
    cen <= 'x', '0' after 200 ns, 'x' after 400 ns, '0' after 600 ns, 'x' after 800 ns, '1' after 1000 ns, 'x'
        after 1200 ns;
end behav;
```

Figura 2.25 Programa de generación de estímulos para el sumador de 4 bits.

Esquema de circuitos

Para comprobar el sumador de 4 bits se crea un esquema de circuito donde los elementos son: a) el elemento de librería que incluye el sumador de 4 bits y b) el elemento que incluye la generación de estímulos. En la Figura 2.26 se muestra el esquema del circuito resultante.

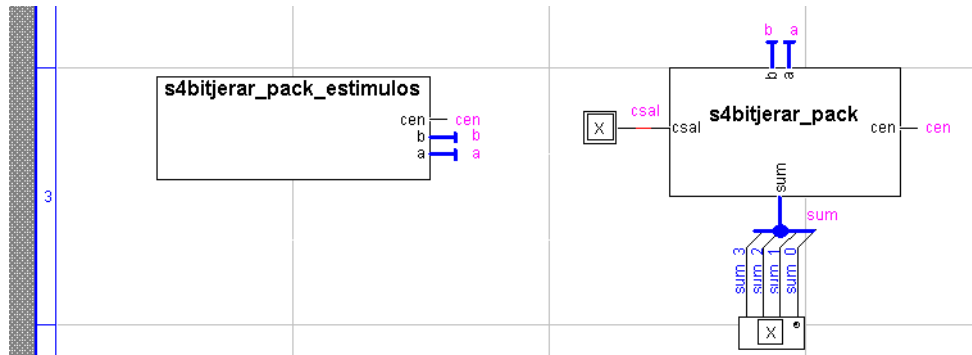




Figura 2.26 Esquema de circuito con un módulo para generar las señales de estimulación aplicadas al sumador de 4 bits.

En la Figura 2.26 se muestra otra posibilidad de LogicWorks para conectar dos señales. Dos señales están conectadas si se nombran de la misma forma. Esta posibilidad facilita que en el esquema de circuito no se crucen los cables, aunque en un circuito físico hay que conectar las señales mediante cables.

Simulación

Al etiquetar las señales las etiquetas se muestran en la parte asociada de la ventana de tiempos.

Los pasos para efectuar la simulación son los siguientes.

- Parar el simulador  .
- Limpieza de la ventana de tiempos.
- Indicación de que deben retenerse todos los resultados de salida.
- Establecer marcas en la ventana de tiempos.
- Activación de la simulación. Si se quiere simular paso a paso debe utilizarse “single step”  .

En la Figura 2.27 se muestra la evolución temporal de las señales en la ventana de tiempos. En la tabla de la Figura 2.28 se muestra parte de esta información en ASCII.

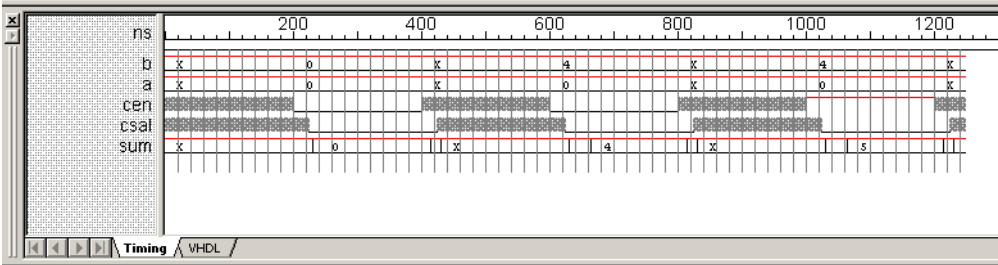


Figura 2.27 Traza en la ventana de tiempo del resultado de utilizar el módulo de estimulación en el sumador de 4 bits.

En $t = 200$ ns el circuito de prueba se estimula con el vector ($b = 0$, $a = 0$, $cen = 0$). La salida es estable en $t = 240$ ns. La salida perdura hasta que se inyectan nuevos estímulos al circuito de prueba. En $t = 400$ ns las señales de entradas al circuito de prueba toman el valor indefinido.

\$T	\$D	\$O b [b_0 b_1 b_2 b_3]	\$O a [a_0 a_1 a_2 a_3]	\$O cen	\$O csal	\$O sum [sum_0 sum_1 sum_2 sum_3]
0	200NS	X	X	X	X	X
200NS	25NS	0	0	0	X	X
225NS	5NS	0	0	0	0	X
230NS	10NS	0	0	0	0	X
240NS	160NS	0	0	0	0	0
400NS	15NS	X	X	X	0	0
415NS	10NS	X	X	X	0	X
425NS	5NS	X	X	X	X	X
430NS	170NS	X	X	X	X	X
600NS	25NS	4	0	0	X	X
625NS	5NS	4	0	0	0	X
630NS	10NS	4	0	0	0	X
640NS	25NS	4	0	0	0	X
665NS	135NS	4	0	0	0	4

Figura 2.28 Tabla en ASCII de la evolución temporal de las señales al utilizar el módulo de estimulación en el sumador de 4 bits.

En la Figura 2.29 se muestra un conjunto de sentencias de asignación de señal para comprobar el sumador de 4 bits. Debido a que el retardo que se especifica en dichas sentencias entre algunos cambios de valor en las señales es menor que el retardo de propagación del circuito, el funcionamiento es incorrecto. El resultado de la simulación con estos estímulos se muestra en la Figura 2.30. Notemos que la salida no es correcta.

```
a <= "xxxx", "0000" after 200 ns, "xxxx" after 230 ns, "0000" after 430 ns, "xxxx" after 460 ns, "0000"
after 660 ns, "xxxx" after 690 ns;
b <= "xxxx", "0000" after 200 ns, "xxxx" after 230 ns, "0100" after 430 ns, "xxxx" after 460 ns, "0100"
after 660 ns, "xxxx" after 690 ns;
cen <= 'x', '0' after 200 ns, 'x' after 230 ns, '0' after 430 ns, 'x' after 460 ns, '1' after 660 ns, 'x' after
690 ns;
```

Figura 2.29 Sentencias de asignación de señal que estimulan el circuito a una velocidad mayor que la permitida.

En la tabla de la Figura 2.31 se identifican los ficheros que se utilizarán en el diseño de la alu que efectúa operaciones SIMD. Los módulos `simd_ent.dvw` y `simd_sal.dvw` son los que deben implementarse de forma incremental en la práctica. Originalmente en estos ficheros se conectan las entradas a las salidas correspondientes. Esto es, se comportan como cortocircuitos.

Nombre	Señal de control	Descripción
alu.dvw	control(1) y control(0) control(1): operación con 32 bits (valor 0), operación SIMD (valor 1)	Especificación estructural de un circuito que realiza sumas y restas de vectores de bits.
sumador.dvw	control(0): suma (valor 0), resta (valor 1)	Especificación funcional del sumador algebraico SIMD.
simd_ent.dvw	control(1) y control(0)	Módulo que modifica las entradas de la alu antes de utilizar el módulo sumador.dvw cuando se realiza una operación SIMD.
simd_sal.dvw	control(1), control(0)	Módulo que modifica las salidas del módulo sumador.dvw cuando se realiza una operación SIMD.

Figura 2.31 Ficheros a utilizar en el diseño de la alu.

En la Figura 2.31 se muestra el interconexionado conceptual de los módulos en la alu. Las señales `a`, `b`, `a1`, `b1`, `s1`, `s2` y `s` representan vectores de bits de tamaño 32. La señal `control` representa un vector de 2 bits.

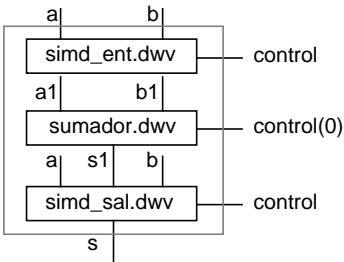


Figura 2.32 Módulos de la alu.

En la tabla de la Figura 2.31 se relacionan los ficheros de prueba para efectuar la comprobación de los diseños.

Nombre	Descripción
suma32.tsv	Entradas A, B y resultado de la suma como vectores de 32 bits.
resta32.tsv	Entradas A, B y resultado de la resta como vectores de 32 bits.
sumaSIMD.tsv	Entradas A, B y resultado de la suma en una operación SIMD.
restaSIMD.tsv	Entradas A, B y resultado de la resta en una operación SIMD.
todo.tsv	Entradas A, B y resultados de los cuatro tipos de operaciones.

Figura 2.33 Ficheros para comprobar los diseños.

Cada uno de los cuatro primeros ficheros contienen seis pares de vectores de entrada para realizar una comprobación rápida del funcionamiento. En el fichero todo.tsv se incluyen, para cada tipo de operación, cien pares de vectores de entrada y los correspondientes vectores de salida para realizar una comprobación más exhaustiva del funcionamiento.

En los diseños que se solicitan son necesarios multiplexores. Su especificación en VHDL, utilizando sentencias de asignación condicional concurrentes, se muestra en el Apéndice 2.2.

Los vectores de bits de entrada del sumador son de 32 bits ($a1$ y $b1$). Suponemos datos que se representan en un byte. Entonces, una entrada del sumador se crea concatenando 4 bytes. Los bytes que se quieren sumar o restar ocupan las mismas posiciones en los vectores de bits de las entradas del sumador.

Operación suma

En primer lugar nos centraremos en la operación suma. Los vectores de bits de entrada del sumador ($a1$, $b1$) deben modificarse para que no se propague el acarreo entre bytes al utilizar el sumador. Posteriormente hay que corregir el resultado del sumador ($s1$).

Anulación de la propagación del acarreo. Para que no se propague el acarreo de la suma de un par de bytes al siguiente par de bytes hay que sustituir por un cero el valor del bit más significativo de cada byte. La expresión analítica es la siguiente:

$$a1_{8 \times i + 7} = b1_{8 \times i + 7} = 0$$

donde $0 \leq i < 3$ indica el número de byte.

Corrección del resultado. Después de la suma el bit más significativo de cada byte del resultado ($s1$) tendrá por valor el acarreo de salida de la suma de los 7 bits menos significativos de los bytes correspondientes. El bit más significativo de cada byte del resultado ($s1$) se corrige efectuando la suma de los bits más significativos de los bytes correspondientes y la salida del sumador correspondiente al bit más significativo de la suma de los 7 bits menos significativos. La expresión analítica es la siguiente:

$$s21_{8 \times i + 7} = a_{8 \times i + 7} \oplus b_{8 \times i + 7} \oplus s1_{8 \times i + 7}$$

donde $0 \leq i < 3$ indica el número de byte y $s1_{8 \times i + 7}$ indica la salida del sumador, que es el acarreo de salida de los 7 bits menos significativos del byte i .

Todas las modificaciones que se solicitan deben mantener la funcionalidad de suma algebraica con 32 bits. Entonces, la salida de la alu es la siguiente:

$$s_{8 \times i + 7} = s1_{8 \times i + 7} \bullet \overline{\text{control}(1)} + s21_{8 \times i + 7} \bullet \text{control}(1)$$

En los diseños que se solicitan utilice el valor de los bits del bus control para establecer el valor cero o uno cuando sea necesario.

Operación resta

En segundo lugar nos centraremos en la operación resta. En una operación de resta se suma el minuendo, el sustraendo complementado y el valor uno. Tenga en cuenta que al indicar la operación de resta, el módulo sumador complementa el sustraendo y suma uno en la suma de los bits de menor peso (la operación es en 32 bits, `sumador.dvv`). En consecuencia los vectores de bits deben modificarse para que se genere un acarreo de entrada en la suma de los bits menos significativos de cada byte. Este acarreo se genera al operar con los bits más significativos de los dos bytes previos. Posteriormente hay que corregir el resultado.

Apéndice 2.1: Declaración de componentes

Declaración de los componentes en un “package”

<pre>library IEEE; use IEEE.std_logic_1164.all; library libs; use libs.MiLib.all; -- La especificación del componente en VHDL se obtiene de un package en una librería -- Los componentes se conectan mediante señales entity S4bitJerar_pack is port (A: in std_logic_vector (3 downto 0) ; B: in std_logic_vector (3 downto 0) ; cen: in std_logic; SUM: out std_logic_vector (3 downto 0) ; csal: out std_logic); end S4bitJerar_pack; architecture structural of S4bitJerar_pack is signal c1, c2, c3, c4: std_logic; begin S1bit0: S1bitDF port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0)); S1bit1: S1bitDF port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1)); S1bit2: S1bitDF port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2)); S1bit3: S1bitDF port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3)); csal <= c4; end structural;</pre>	<p>Librerías</p> <p>interfaz</p> <p>Listado de conexiones</p>
--	---

Declaración de componentes en la arquitectura

Los componentes se declaran dentro del cuerpo de la arquitectura, siguiendo a la declaración de ésta. Esto es, después de la palabra clave “architecture” y antes de la palabra clave “begin”.

La forma de efectuar la declaración de los módulos que se van a utilizar es describir su interface. En este caso se utiliza la palabra clave “component”. Seguidamente se especifican las señales de entrada y salida de la misma forma que se ha efectuado en la declaración de la interface (“entity”) al describir el módulo. La declaración finaliza con las palabras clave “end component”. En resumen, la declaración de un componente consiste del nombre del componente y de la interface (puertos), siendo la sintaxis la siguiente.

Sintaxis
<pre>component component_name [port (port_signal_name: mode type; port_signal_name: mode type; ... port_signal_name: mode type);] end component;</pre>

El nombre de un componente se refiere al nombre de una entidad definida explícitamente en un fichero VHDL. La lista de puertos de la interface especifica el nombre, el modo y tipo de cada puerto de forma idéntica a como se especifica en la declaración “entity”. En el fichero donde se declaran los componentes hay que indicar que el fichero que contiene la especificación del componente se almacena en el directorio de trabajo. Esta especificación se efectúa en la cabecera del fichero.

Sintaxis	Sintaxis
<pre>use work.all;</pre>	<pre>use work.my_component;</pre>

El nombre del fichero con extensión “.dwv” que contiene la descripción del componente que se quiere utilizar tiene que ser el mismo que el nombre especificado en “entity”. En la especificación mostrada a la izquierda se indica que son accesibles todas las definiciones existentes en el directorio de trabajo. En la especificación de la derecha se indica que sólo son accesibles las especificaciones incluidas en el fichero denominado my_component.dwv¹.

1. En LogicWorks en un fichero sólo puede haber una declaración de interface y la especificación de la arquitectura asociada.

<pre>library IEEE; use IEEE.std_logic_1164.all; use work.all; -- La especificación del componente en VHDL está en otro fichero del directorio -- Los componentes se conectan mediante señales entity S4bitJerar is port (A: in std_logic_vector (3 downto 0) ; B: in std_logic_vector (3 downto 0) ; cen: in std_logic; SUM: out std_logic_vector (3 downto 0) ; csal: out std_logic); end S4bitJerar; architecture structural of S4bitJerar is component S1bitDF port (x, y, cen : in std_logic; s, csal : out std_logic); end component; signal c1, c2, c3, c4: std_logic; begin S1bit0: S1bitDF port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0)); S1bit1: S1bitDF port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1)); S1bit2: S1bitDF port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2)); S1bit3: S1bitDF port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3)); csal <= c4; end structural;</pre>	Librerías
	interfaz
	Componentes
	Listado de conexiones

Apéndice 2.2: Multiplexor y decodificador

En este apéndice se describen los circuitos combinacionales multiplexor y decodificador, los cuales son elementos utilizados en un banco de registros. Además se describen otros constructores de VHDL que permiten describir modelos de comportamiento mediante sentencias de asignación condicional.

Multiplexor

Un multiplexor es un circuito que permite seleccionar una señal de salida de entre varias señales de entrada en función de una señal de entrada de control (señal de selección). La tabla de verdad de un multiplexor 4-1, selección de una salida de entre 4 entradas, se muestra en la parte izquierda de la Figura 2.34¹. Las señales de entrada se denominan D0, D1, D2, D3. Las señales de control son S1 y S0. En un caso general, el número de señales de control de 1 bit es n, siendo 2^n el número de señales de entradas entre las que se puede efectuar una selección. En la parte central de la Figura 2.34 se muestran unas expresiones lógicas que se obtienen de la tabla de verdad. En la parte derecha se muestra el símbolo utilizado para representar un multiplexor.

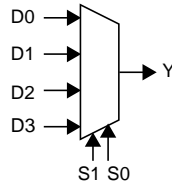
S1	S0	Y	Expresiones lógicas	Símbolo
0	0	D0		
0	1	D1		
1	0	D2		
1	1	D3	$Y = D0 \overline{S1} \overline{S0} + D1 \overline{S1} S0 + D2 S1 \overline{S0} + D3 S1 S0$	

Figura 2.34 Multiplexor: tabla de verdad, expresiones lógicas y símbolo en un esquema de circuito.

La señales de entrada de un multiplexor pueden ser grupos de bits (bus), en cuyo caso la selección es de todas las señales que transportan un bus.

En la Figura 2.35 se muestra un esquema con puertas lógicas de un multiplexor 4-1.

1. Notemos que es una tabla de verdad distinta de la convencional. En la columna de salida se especifica la entrada que se selecciona. Una tabla convencional tendría 6 variables (2 sel, 4 entradas) y para una combinación de las variables de selección tendríamos que, para las 2^4 combinaciones de las variables de entrada, sólo habría 2 salidas distintas que se corresponderían con los valores de una de las variable de entrada.

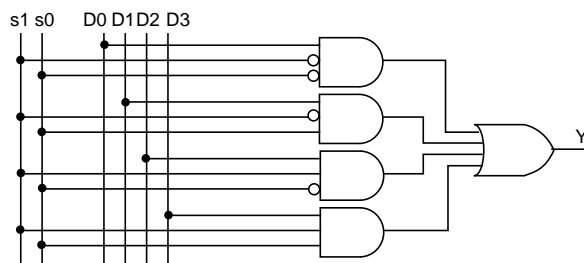


Figura 2.35 Esquema de puertas de un multiplexor. Los círculos en la entrada de una puerta indican señal negada.

Un multiplexor se puede utilizar para implementar funciones lógicas de forma sencilla. Una función de n variables de entrada tiene 2^n “minterm”¹ y cada “minterm” se corresponde con una de las 2^n entradas del multiplexor. Las n variables de entrada de la función lógica son las n líneas de selección del multiplexor. Una entrada del multiplexor es uno o cero dependiendo de si el correspondiente “minterm” de la función es uno o cero respectivamente.

Decodificador

Un decodificador es un bloque combinacional que convierte una forma de representación digital en otra. Usualmente, un decodificador tiene como entrada una representación compacta (codificada) y la convierte en una representación expandida (explícita).

Un decodificador binario de n entradas es un circuito combinacional que tiene n entradas y 2^n salidas. Dada una combinación de los valores de entrada sólo una de las salidas es 1 y las otras salidas son cero.

Un decodificador binario se utiliza siempre que un conjunto de valores ha sido codificado en binario y los valores tienen que distinguirse individualmente (decodificar). En general, un conjunto de 2^n elementos se puede codificar en n bits. Entonces, un decodificador de n bits se utiliza para identificar cuál de los elementos del conjunto ha sido codificado. Un ejemplo típico de utilización de un decodificador es en la decodificación de direcciones para acceder a posiciones de almacenamiento (memoria, banco de registros).

1. Recordemos que un “minterm” es un producto que contiene todas las variables utilizadas en la función lógica.

La tabla de verdad de un decodificador de 2 entradas se muestra en la parte izquierda de la Figura 2.36. Las señales de entrada se denominan S0, S1. Las señales de salida son D3, D2, D1 y D0. En la parte central de la Figura 2.36 se muestran unas expresiones lógicas que se obtienen de la tabla de verdad. En la parte derecha se muestra el símbolo utilizado para representar un decodificador.

S1	S0	D3, D2, D1, D0	Expresiones lógicas	Símbolo
0	0	0, 0, 0, 1	$D0 = \overline{S1} \overline{S0}$	
0	1	0, 0, 1, 0	$D1 = \overline{S1} S0$	
1	0	0, 1, 0, 0	$D2 = S1 \overline{S0}$	
1	1	1, 0, 0, 0	$D3 = S1 S0$	

Figura 2.36 Decodificador: expresiones lógicas y símbolo en un esquema de circuito.

En la Figura 2.37 se muestra el esquema con puertas lógicas de un decodificador de 2 entradas.

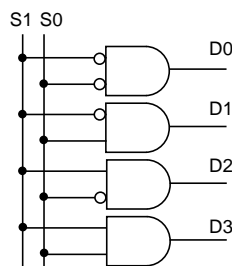


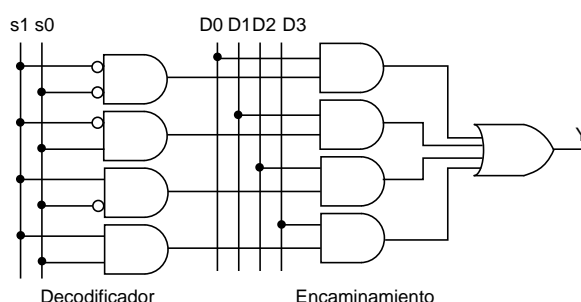
Figura 2.37 Esquema de puertas de un decodificador de dos entradas. Los círculos en la entrada de una puerta indican señal negada.

Un decodificador binario y una puerta OR permiten implementar cualquier expresión lógica. Esto es una consecuencia de la definición de un decodificador binario. Una salida del decodificador representa una de las posibles combinaciones de las señales de entrada ("minterm"), entonces, la implementación de suma de "minterms" se obtiene efectuando la OR de las salidas del decodificador que se corresponde con los "minterm" de la expresión lógica.

Decodificador - multiplexor

Un multiplexor es una extensión de un decodificador en el que una serie de entradas son decodificadas para obtener las señales de selección que permiten elegir una señal de un conjunto de señales de entrada.

En la Figura 2.38 se muestra un multiplexor donde se distingue un primer nivel de puertas que conforma un decodificador. La salida es un vector de bits donde sólo uno de los bits toma el valor uno. Los dos siguientes niveles de puertas encaminan a la salida la señal seleccionada en el decodificador.



En una puerta de tres estados, además de los 2 valores usuales 1 y 0, la salida puede tomar el valor Z (alta impedancia, no conectado). La puerta tiene dos entradas: dato de entrada y permiso ("enable"). La salida es igual a la entrada cuando la señal permiso es 1, mientras que la salida no está conectada si la señal permiso es cero (Figura 2.39).

permiso	sal	Símbolo
0	Z (alta impedancia)	
1	ent	

Las salidas de varias puertas de tres estados se pueden conectar juntas mientras sólo una de ellas tenga permiso en un momento dado. Por tanto se utilizan para conectar varios dispositivos al mismo bus.

Multiplexor. En la Figura 2.40 se muestra la utilización de puertas de tres estados para construir un multiplexor. El primer nivel de puertas es un decodificador. La salida de este decodificador determina cuál de las puertas de tres estados no está en alta impedancia.

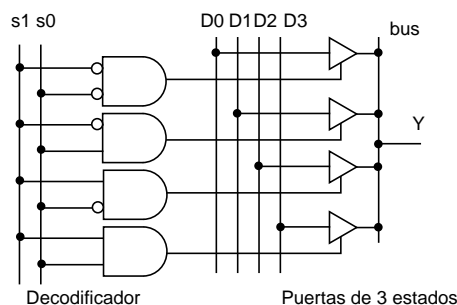


Figura 2.40 Multiplexor utilizando un decodificador y puertas de tres estados.

Constructores condicionales de asignación de señal en VHDL

Un circuito combinacional se puede describir a nivel de puertas y tendremos un modelo estructural. También se puede describir el comportamiento. Para ello en VHDL podemos utilizar sentencias de asignación de señal concurrentes. El modelo usualmente se denomina de flujo de datos. Recordemos que un modelo de flujo de datos describe un circuito en términos de su función y el flujo de datos a través del circuito.

En VHDL se pueden utilizar varios constructores para describir modelos de comportamiento. En esta sección nos centraremos en sentencias de asignación de señal concurrentes tanto simples como con capacidad para determinar una de varias posibilidades de asignación: a) sentencias de asignación de señal concurrentes, b) selección en la asignación de una señal (selected signal assignments) y c) asignación de una señal de forma condicional (conditional signal assignments).

Las sentencias de asignación de señal concurrentes se activan y ejecutan tan pronto se produce un evento en una de las señales. Esto es, una señal cambia de valor.

Sentencia de asignación de señal concurrente simple

En las prácticas previas ya se han visto varios ejemplos donde se utilizaban sentencias de asignación de señal concurrentes. En este apartado repasaremos la sentencia de asignación de señal básica. La sintaxis es la siguiente.

Ejemplo	Sintaxis
<code>s <= (x xor y) xor cen;</code>	<code>signal-name <= expresion;</code>

El valor evaluado en la expresión se transfiere a la señal. Tan pronto como se produce un evento (cambio en el valor) en una de las señales utilizadas en la expresión, la expresión se evalúa. El tipo de la señal a la que se asigna el valor debe ser el mismo que el tipo del valor que se evalúa en la expresión. Utilizando las expresiones lógicas que se obtienen de la tabla de verdad del multiplexor y del decodificador podemos escribir los códigos en VHDL que se muestran en la Figura 2.41.

Multiplexor	Decodificador
<pre> library IEEE; use IEEE.std_logic_1164.all; entity mux4 is port (d0, d1, d2, d3: in std_logic; s0, s1: in std_logic; y: out std_logic); end mux4; architecture behavB of mux4 is begin y <= (d0 and (not s1) and (not s0)) or (d1 and (not s1) and s0) or (d2 and s1 and (not s0)) or (d3 and s1 and s0); end behavB; </pre>	<pre> library IEEE; use IEEE.std_logic_1164.all; entity decodificador is port (sel0, sel1 : in std_logic; d0, d1, d2, d3: out std_logic); end decodificador; architecture behavS of decodificador is signal sel0_n, sel1_n: std_logic; begin sel0_n <= not sel0; sel1_n <= not sel1; d0 <= sel0_n and sel1_n d1 <= sel0 and sel1_n d2 <= sel0_n and sel1 d3 <= sel0 and sel1 end behavS; </pre>

Figura 2.41 Multiplexor y decodificador. Descripción mediante sentencias de asignación de señal simples.

Podemos obtener especificaciones más compactas del multiplexor si utilizamos conversión de tipos. En concreto convertimos un objeto de tipo `std_logic_vector` en un objeto de tipo `integer` para valores positivos (sin signo) utilizando la función `conv_integer` (Figura 2.42).

Multiplexor
<pre> library IEEE; use IEEE.std_logic_1164.all; entity mux4 is port (d: in std_logic_vector (3 downto 0); s: in std_logic_vector (1 downto 0); y: out std_logic); end mux4; architecture behav of mux4 is begin y <= d(conv_integer (s)); end behav; </pre>

Figura 2.42 Descripción VHDL de un multiplexor utilizando un índice para acceder a un elemento de un bus.

Asignación de una señal de forma condicional

Una sentencia de asignación concurrente de señal asocia una señal a una expresión. El término asignación de una señal de forma condicional se utiliza para describir sentencias donde una señal puede tener asociada más de una expresión. Cada una de las expresiones está asociada con una condición concreta.

La sintaxis de una sentencia de asignación de señal condicional es la siguiente.

Sintaxis
<pre>Target_signal <= expression when Boolean_expression else expression when Boolean_expression else ... expression;</pre>

Las condiciones se evalúan en secuencia y la señal recibe el valor de la primera expresión cuya condición lógica (Booleana) se cumple ("true"). Si no se cumple ninguna condición, la señal recibe el valor de la última expresión. Como las condiciones se evalúan en secuencia, si se cumple más de una condición, el valor que se asigna a la señal es el de la expresión asociada a la primera condición que se cumple. Una condición se basa en el estado de otras señales en el circuito.

Observemos que existe sólo un operador de asignación asociado con una sentencia de asignación condicional de señal.

La asignación de señal condicional se reevalúa tan pronto como cualquiera de las señales en las condiciones o expresiones cambia. El constructor "when-else" es útil para expresar funciones lógicas en forma de tabla de verdad.

Mediante operadores de relación se construyen expresiones cuyo resultado es de tipo boolean. El tipo entero y los tipos enumerados de boolean y caracteres están ordenados.

Operadores relacionales. "=" y "/=" entre otros. El primero es el operador igual y el segundo es el operador no igual o distinto. En particular, estos dos operadores se pueden utilizar para cualquier tipo de datos.

En la Figura 2.43 se muestran dos ejemplos de un multiplexor descrito con sentencias de asignación condicional de señales. La diferencia entre las dos descripciones reside en la utilización de señales binarias o de un bus en la selección de la entrada. Así mismo, en el caso de la descripción VHDL de la parte derecha, las señales de entrada son buses y el número de señales que transporta el bus se especifica de forma genérica. En la descripción se utiliza el operador relacional "=" para establecer condiciones. Observemos que en el caso de buses el valor se especifica utilizando dobles comillas como se ha comentado previamente.

Notemos que la especificación VHDL en los dos casos mostrados en la Figura 2.43 son una traducción directa de la tabla de verdad de un multiplexor. En el caso del código mostrado a la izquierda de la Figura 2.43 la condición “sel1=1 and sel0=1” no se explicita ya que si no se cumple ninguna condición se selecciona la última expresión.

Señales de 1 bit	Buses
<pre>entity mux is port (d0, d1, d2, d3: in std_logic; sel1, sel0 : in std_logic; Y: out std_logic); end mux; architecture behC of mux is begin Y <= d0 when sel1='0' and sel0='0' else d1 when sel1='0' and sel0='1' else d2 when sel1='1' and sel0='0' else d3; end behC;</pre>	<pre>entity muxN is generic (n: positive := 4); port (d0, d1, d2, d3: in std_logic_vector (n-1 downto 0); SEL: in std_logic_vector (1 downto 0); Y: out std_logic_vector (n-1 downto 0)); end muxN; architecture behCG of muxN is begin Y <= d0 when SEL = "00" else d1 when SEL = "01" else d2 when SEL = "10" else d3; end behCG;</pre>

Figura 2.43 Multiplexor. Descripción mediante asignación de señal de forma condicional.

En la parte izquierda de la Figura 2.44 se muestra la especificación VHDL de un decodificador de 3 entradas para 8 salidas. Notemos que igual que en el caso del multiplexor la especificación es una traducción directa de la tabla de verdad. En este caso, para tener en cuenta valores de las señales distintos de 1 y 0, se han especificado todas las condiciones de la tabla de verdad. La última expresión “xxxxxxx” será seleccionada cuando el valor de la señal de selección no es uno de los valores especificados previamente.

En la parte derecha de la Figura 2.44 se muestran la especificación VHDL de una puerta de tres estados. El número de señales del bus de entrada se especifica de forma genérica. Por ello, para especificar alta impedancia en el caso de que la señal permiso (“pe”) tome el valor cero se utiliza “others”.

Decodificador	Puerta de tres estados
<pre> library IEEE; use IEEE.std_logic_1164.all; entity decodificador is port (Sel : in std_logic_vector (2 downto 0); D: out std_logic_vector (7 downto 0)); end decodificador; architecture behC of decodificador is begin D <= "00000001" when Sel="000" else "00000010" when Sel="001" else "00000100" when Sel="010" else "00001000" when Sel="011" else "00010000" when Sel="100" else "00100000" when Sel="101" else "01000000" when Sel="110" else "10000000" when Sel="111" else "xxxxxxx"; end behC; </pre>	<pre> library IEEE; use IEEE.std_logic_1164.all; entity tresestados is generic (n : positive := 4); port (pe : in std_logic; en: in std_logic_vector (n-1 downto 0); sal: out std_logic_vector (n-1 downto 0)); end tresestados; architecture behavT of tresestados is begin sal <= en when (pe = '1') else (others => 'z'); end behavT; </pre>

Figura 2.44 Decodificador. Descripción mediante asignación de señal de forma condicional.

Selección en la asignación de una señal

Una asignación de señal con selección es similar a una selección condicional de la señal. Estas sentencias pueden tener sólo una señal a la que se asigna un valor y únicamente una expresión determina cuál de las elecciones es la asignada. La sintaxis es la siguiente.

Sintaxis
<pre> [label:] with choice_expression select target_name <= expression when choices, ... expression when choices; </pre>

La sentencia de asignación de forma selectiva se evalúa cada vez que existe un cambio en "choice_expression".

La señal recibirá el valor de la expresión cuya elección ("choices") está incluida en los valores de la expresión de elección ("choice_expression"), los cuales son constantes. La expresión seleccionada es la primera con una coincidencia en la elección. La elección puede ser una expresión fija (ej. 3) o un expresión que identifica un rango (ej. 3 to 12). Para una elección se siguen las siguientes reglas:

- Dos posibles elecciones tienen que tener una intersección nula.

- Todos los posibles valores de la expresión de elección deben estar presentes en el conjunto de elecciones a menos que la palabra clave “others” esté presente como elección.

Las elecciones pueden expresar un único valor, un rango o elecciones combinadas. Seguidamente se muestra un ejemplo. En la segunda elección se incluyen varios posibles valores. La elección “others” debe ser la última de las elecciones.

Ejemplo

```
target <= value1 when "000",
      value2 when "001" | "011" | "101",
      value3 when others;
```

En el ejemplo anterior la barra vertical “|” se utiliza como un carácter de selección en la sección de “choices” de una sentencia de selección.

En la Figura 2.45 se muestran dos ejemplos de un multiplexor descrito con sentencias de asignación condicional de señal. La diferencia entre las dos descripciones reside en la utilización de un bus o de valores codificados (enteros) en la selección de la entradas. Así mismo, en el caso de la descripción VHDL de la parte derecha, las señales de entrada son buses y el número de señales que transporta el bus y el rango de la codificación se especifican de forma genérica. Notemos que la especificación VHDL es una traducción directa de la tabla de verdad.

Selección mediante vector de bits	Buses. Selección mediante números codificados
<pre>library IEEE; use IEEE.std_logic_1164.all; entity mux is port (d0, d1 : in std_logic; d2, d3 : in std_logic; s : in std_logic_vector (1 downto 0); y: out std_logic); end mux; architecture behavS of mux is begin with s select y <= d0 when "00", d1 when "01", d2 when "10", d3 when "11"; end behavS;</pre>	<pre>library IEEE; use IEEE.std_logic_1164.all; entity muxN is generic (n: positive := 4; m: positive := 4); port (d0, d1: in std_logic_vector (m-1 downto 0); d2, d3: in std_logic_vector (m-1 downto 0); s : in integer range 0 to n-1; y: out std_logic_vector (m-1 downto 0)); end muxN; architecture behavSG of muxN is begin with s select y <= d0 when 0, d1 when 1, d2 when 2, d3 when 3; end behavSG;</pre>

Figura 2.45 Multiplexor. Descripción mediante selección en la asignación de una señal

En la parte izquierda de la Figura 2.46 se muestra la especificación de un decodificador de 3 entradas para 8 salidas. En esta especificación se tiene en cuenta el caso de que alguna de las señales del bus de selección no tome el valor 1 o 0.

En la parte derecha de la Figura 2.46 se muestran la especificación VHDL de una puerta de tres estados. El número de señales del bus de entrada se especifica de forma genérica. Por ello, para especificar alta impedancia, en el caso de que la señal permiso ("pe") tome el valor cero, se utiliza "others".

Bus de salida	Puerta de tres estados
<pre>library IEEE; use IEEE.std_logic_1164.all; entity decodificador is port (Sel : in std_logic_vector (2 downto 0); D: out std_logic_vector (7 downto 0)); end decodificador; architecture behS of decodificador is begin with Sel select D <= "00000001" when Sel="000", "00000010" when Sel="001", "00000100" when Sel="010", "00001000" when Sel="011", "00010000" when Sel="100", "00100000" when Sel="101", "01000000" when Sel="110", "10000000" when Sel="111" ; "00000000" when others ; end behS;</pre>	<pre>library IEEE; use IEEE.std_logic_1164.all; entity tresestados is generic (n : positive := 4); port (pe : in std_logic; en: in std_logic_vector (n-1 downto 0); sal: out std_logic_vector (n-1 downto 0)); end tresestados; architecture behavT of tresestados is begin with pe select sal <= en when '1', (others => 'z') when '0'; end behavT;</pre>

Figura 2.46 Decodificador. Descripción mediante selección en la asignación de una señal.

