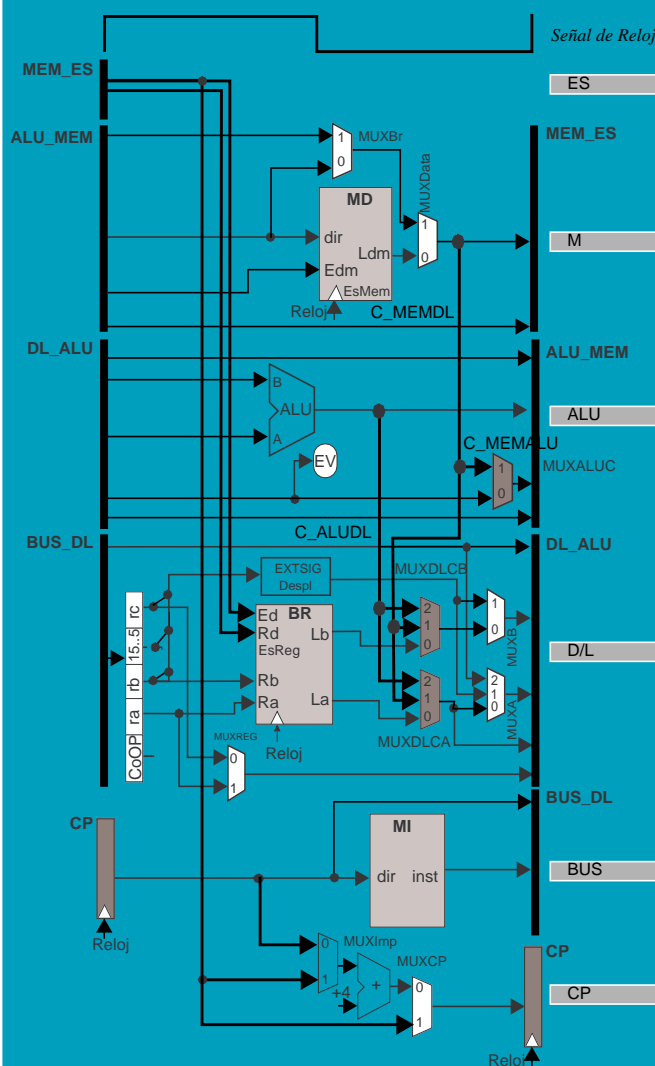


# Documentación de prácticas

## Arquitectura de Computadores II

### (MEI - ACAP)



```

process (D, clock)
begin
  -- clock rising edge
  if ( clock='1' and clock'event ) then
    Q <= D;
  end if;
end process;

```



E. Herrada  
J.M. Llabería  
A. Olivé

© Copyright 2018 los autores, Universidad Politécnica de Cataluña

# Contenido

.....

<b>Práctica 1</b>	<b>Sumador de 1 bit</b>	<b>1</b>
	Sumador de un bit	2
	Estructura básica de un modelo en el lenguaje VHDL	3
	Interface	4
	Arquitectura	5
	Paquetes y librerías	7
	Elementos léxicos de VHDL	7
	Operadores	8
	Utilización de Quartus	9
	Entorno	10
	Inicio de un proyecto	12
	Edición de ficheros vhd	16
	Compilación	19
	Activación del simulador Modelsim	22
	Preparación de la simulación	27
	Simulación	30
	Estimulación de las entradas	30
	Activación de la simulación	30
	Almacenamiento de la configuración de la simulación	33
	Comprobación del funcionamiento lógico	34
	Análisis de las señales en la ventana temporal	36
	Almacenamiento de resultados	38
	Modelo VHDL con retardo	39
	Objetos de datos en VHDL: Señales y constantes	40
	Especificación de retardos en VHDL	41
	Descripción en VHDL del sumador de 1bit con retardos	42
	Simulación lógica o funcional	43
	Simulación temporal del circuito para medir el retardo	43
	Utilización de la orden force	46





# Practica 1

## Sumador de 1 bit

.....

Los objetivos de esta sesión son por un lado familiarizarse con el entorno de trabajo de Altera Quartus y Modelsim y el aprendizaje de un subconjunto del lenguaje VHDL (VHSIC, "Very High Speed Integrated Circuits", **H**ardware **D**escription **L**anguage) que permite especificar circuitos digitales.

La herramienta Altera Quartus, a partir de ahora Quartus, se utiliza para especificar diseños y desencadenar la activación de otras herramientas. La herramienta Modelsim se utiliza para efectuar la simulación.

Por otro lado, otro de los objetivos es aprender una metodología de diseño, de simulación de circuitos y comprobación de un diseño de forma que se verifique el correcto funcionamiento del mismo. En este sentido se utilizará como ejemplo conductor un **sumador de 1 bit** o "**full adder (FA)**".

Es importante comprender, y saber utilizar, las funcionalidades de las herramientas Altera Quartus y Modelsim y la sintaxis y semántica del lenguaje VHDL, a medida que se describen en las distintas prácticas. En caso contrario, es bastante factible que en la misma práctica y en las siguientes prácticas tenga que derrochar energía innecesaria (asfixiado) para finalizarlas, debido a problemas relacionados con las herramientas o el lenguaje.

En particular, respecto al lenguaje VHDL, conocer su sintaxis no implica necesariamente saber diseñar con él. El lenguaje VHDL se utiliza para efectuar descripciones hardware. Mediante VHDL se pueden describir circuitos combinacionales y secuenciales<sup>1</sup>. Al utilizarlo hay que pensar en puertas y elementos de memorización, en lugar de en variables y funciones. Un circuito descrito en VHDL puede ser simulado utilizando una herramienta que reproduce el funcionamiento del circuito descrito. El funcionamiento de la herramienta de simulación (cómo se efectúa) es estandar y sigue la técnica de simulación mediante eventos discretos. Además, existen herramientas que traducen una descripción VHDL en un circuito con puertas y elementos de memorización.

1. Recordemos que un circuito es un conjunto de elementos que trabajan en paralelo. Por tanto, el lenguaje VHDL permite expresar paralelismo.

El proceso de traducción de VHDL a un circuito lógico se denomina síntesis. Una herramienta que efectúe esta traducción sólo contempla un subconjunto de las descripciones que se pueden expresar en VHDL (expresividad del lenguaje)<sup>2</sup>. Por tanto, debemos de tener en cuenta que especificar un circuito en VHDL no es semejante a escribir un programa en VHDL. Hay que recordar que al escribir el código se está especificando un circuito (hardware). En concreto, es necesario identificar de forma sencilla en el código los elementos que componen el circuito. Esto es, hay que ser capaces de extraer un esquema de circuito de una especificación en VHDL. También hay que ser capaz de traducir un circuito a una especificación VHDL<sup>3</sup>. En cualquier caso, las reglas o pautas de diseño software hay que utilizarlas al escribir código en VHDL.

Un simulador de VHDL (como Modelsim) es capaz de simular completamente la expresividad del lenguaje VHDL. Por tanto, las características del lenguaje no sintetizables se pueden utilizar en los programa de prueba de un circuito. Usualmente un programa de prueba no se sintetiza.

Por otro lado, lea detenidamente las explicaciones. Aunque pueden haber errores en el documento, bastantes de los problemas o dudas que se presentan son debidas a no haber leído alguno de los párrafos o la omisión de alguna palabra clave en alguno de ellos.

## Sumador de un bit

La expresión algebraica correspondiente a la suma de 3 bits con la misma ponderación es la siguiente:

$$x + y + c = 2 \times c_1 + s$$

El resultado son dos bits, el bit de menor ponderación se denomina suma (s) y el otro bit se denomina acarreo de salida ( $c_1$ ).

- 
2. El coste de traducir a hardware cualquier especificación en VHDL es muy costoso y probablemente no efectivo, ya que en bastantes ocasiones la síntesis puede que no sea de utilidad o interés. Por ejemplo, en programas de prueba.
  3. Sin utilizar sutilezas.

**Expresiones lógicas.** A partir de las entradas ( $x$ ,  $y$ ) y de un *acarreo* de entrada ( $c_{en}$ ) en la Figura 1.1 se muestran unas expresiones lógicas, que se obtienen de la tabla de verdad, para la suma ( $s$ ) y el *acarreo* de salida ( $c_{sal}$ ).

cen	x	y	csal	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

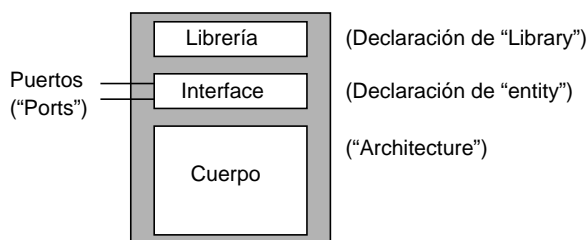
$$s = (x \oplus y) \oplus c_{en} \quad \oplus \text{ XOR}$$

$$c_{sal} = x \cdot y + c_{en} \cdot y + c_{en} \cdot x \quad + \text{ OR} \quad \cdot \text{ AND}$$

**Figura 1.1** Tabla de verdad y expresiones lógicas para un sumador de 1 bit.

## Estructura básica de un modelo en el lenguaje VHDL

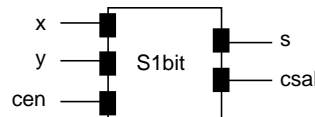
Cuando se diseña un sistema digital es una buena práctica distinguir la interface de la implementación (Figura 1.2). Por interface entendemos cómo se observa el sistema digital desde el exterior. Por implementación entendemos cómo se obtiene el comportamiento. Esto es, qué elementos se utilizan para obtener el comportamiento deseado. Además de los elementos anteriores se distingue la especificación de librerías.



**Figura 1.2** Descripción VHDL consistente de un interface y un cuerpo de la "architecture" e indicación de las librerías utilizadas.

## Interface

En VHDL una declaración “entity” da nombre al sistema o módulo que estamos diseñando y define su interface con el exterior en términos de una lista de puertos. Por ejemplo, una declaración “entity” es la equivalencia textual del símbolo de un sumador de 1 bit (Figura 1.3). El símbolo muestra la interface: señales de entrada (x, y, cen) y señales de salida (s, csal).



**Figura 1.3** Símbolo de un sumador de 1 bit.

En VHDL la interface se especifica de la siguiente forma.

Ejemplo	entity S1bit is			Sintaxis	entity NAME_OF_ENTITY is		
	port (	x: in	std_logic;		port (signal_names: mode		type;
		y: in	std_logic;		signal_names: mode		type;
		cen: in	std_logic;		...		
		s: out	std_logic;		signal_names: mode		type);
		csal: out	std_logic );		end [NAME_OF_ENTITY] ;		
		end S1bit;					

Se empieza con la palabra clave “entity”, seguido por el nombre del módulo y la palabra clave “is”. Seguidamente se declaran los puertos utilizando la palabra clave “port”. Una declaración “entity” siempre finaliza con la palabra clave “end”, opcionalmente ( [ ] ) seguida del nombre de la “entity”.

El nombre de la “entity” es un identificador seleccionado por el usuario. Los “signal\_names” consisten de una lista separada por punto y coma de uno o más identificadores que especifican las señales de la interface externa. Las señales (“signal\_names”) tienen un “mode” que es una palabra reservada para indicar la dirección de la señal: a) señal de entrada (“in”) y b) señal de salida (“out”). Posteriormente se relacionarán otras posibilidades. Los puertos se pueden enumerar en cualquier orden.

Un “signal\_name” es un objeto denominado “signal”. La implementación de un objeto “signal” es una secuencia de pares tiempo-valor. Con ello se describe el comportamiento de una señal transportada por un cable en un circuito digital. A una secuencia de pares tiempo-valor se la denomina frente de onda.

**Tipos (“type”).** Una señal tiene un tipo predefinido o definido por el usuario. Los tipos básicos en VHDL son “bit” y “bit\_vector”. Una señal de tipo “bit” puede tomar los valores 0 y 1. El tipo “bit\_vector” es un vector cuyos elementos son de tipo “bit”. Los tipos “bit” y “bit\_vector” no son suficientemente versátiles para especificar todas las posibilidades en una señal digital. Por ello se han definido otros tipos con un mayor rango de valores. En



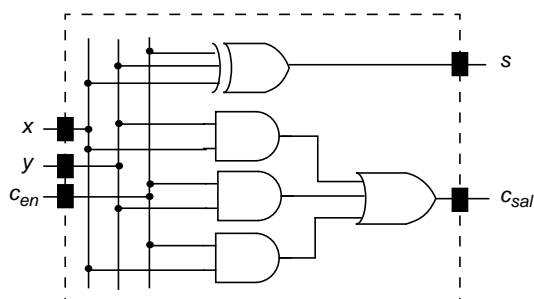
total se utilizan 9 posibilidades, estando entre ellas el valor no definido ("U"), alta impedancia ("Z"), no importa o no especificado ("don't care", "-"), el valor desconocido ("X") y codificaciones que indican también la fortaleza de la señal. Como ejemplo, el valor alta impedancia permite que una señal pueda ser activada desde múltiples fuentes.

La definición de tipo usualmente utilizada para señales de un bit es "std\_logic" que codifica los 9 valores descritos. Esta definición de tipo está incluida en el "package" std\_logic\_1164 de la librería IEEE. Posteriormente detallaremos otras definiciones de tipo usuales.

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. Posteriormente se detalla el tipo carácter.

## Arquitectura

La operación o función de una "entity" se describe en lo que se denomina cuerpo de la "architecture", el cual está constituido por una declaración y una descripción de la operación mediante sentencias de asignación de señales concurrentes. En la Figura 1.4 se muestra la descripción de un sumador de 1 bit con puertas lógicas que mimetiza las expresiones lógicas descritas previamente.



**Figura 1.4** Esquema de puertas de la operación suma de 1 bit.

En VHDL la función se especifica en el cuerpo de la arquitectura ("architecture") de la siguiente forma.

<b>Ejemplo</b>	<pre>architecture comportamiento of S1bit is begin     s &lt;= (x <b>xor</b> y) <b>xor</b> cen;     csal &lt;= (x <b>and</b> y) or (x <b>and</b> cen) or (y <b>and</b> cen); end comportamiento;</pre>	<pre>architecture architecture_name of NAME_OF_ENTITY is begin     -- Concurrent Statements end architecture_name;</pre>
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------

**Sintaxis**

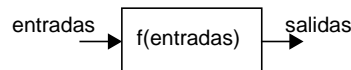
La línea cabecera del cuerpo de la arquitectura define el nombre de la arquitectura, por ejemplo “comportamiento”, y lo asocia con la declaración “entity”, S1bit. El nombre de “architecture” puede ser cualquier identificador que sea legal. El cuerpo principal de la arquitectura empieza con la palabra clave “begin” y finaliza con la palabra clave “end” seguida del nombre dado a la “architecture”. Pueden existir varios cuerpos de la arquitectura que se corresponden con una única definición de la interface (“entity”). Cada uno de los cuerpos describe alternativas para implementar la operación del circuito.

**Comentarios.** Una utilización apropiada de comentarios mejora la lectura y comprensión de cualquier código en VHDL. Un comentario empieza con dos guiones (--). El compilador VHDL ignora cualquier texto posterior a “--” hasta el final de la línea en la que aparecen los dos guiones. Un comentario que ocupa varias líneas necesita empezar cada línea con dos guiones.

Los puertos especificados en la interface son visibles en el cuerpo de la arquitectura. Los puertos se observan como terminales (pins) externos de un circuito. Ahora bien, desde el punto de vista interno del circuito diseñado, un terminal es un cable con una conexión externa que transporta una señal.

Para modelar el sumador de 1 bit utilizaremos un modelo que describe el comportamiento o la función (funcionalidad).

**Modelo de comportamiento (“Behavioral Model”).** Se especifican funciones que relacionan las salidas con las entradas (Figura 1.5).



**Figura 1.5** *Modelo de comportamiento.*

**Sentencia de asignación de señal.** En el cuerpo de la arquitectura se utilizan sentencias concurrentes, las cuales se denominan de asignación de señales. Estas sentencias especifican funciones que hacen uso de operadores lógicos. Los operadores lógicos permitidos son: and, or, nand, nor, xor, xnor, y not. Posteriormente se detallan otros tipos de operadores para otros tipos de señales. Como operador de asignación se utiliza el símbolo <=, que asigna el valor de la expresión que está a la derecha del símbolo a la señal que hay en la izquierda.

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

Las sentencias de asignación de señales se dice que son concurrentes porque pueden activarse y ser evaluadas a la vez. En este sentido, el orden con que se escriban no es importante, ya que no se evalúan secuencialmente. En el ejemplo del sumador las dos sentencias que se han explicitado se evalúan concurrentemente ( s <= . . . y csal <= . . . ).

Una sentencia de asignación de señal se ejecuta, esto es se evalúa la parte derecha y se asigna a la parte izquierda, cuando una o más señales en la parte derecha cambia su valor. Es decir, se produce un evento en una de las señales que se especifican en la parte derecha de la asignación. Adicionalmente se puede especificar un retardo asociado con el cambio, el cual detallaremos posteriormente cómo se especifica.

Recopilando, un sistema digital está básicamente conducido por datos y eventos, los cuales se producen en una señal y ello da lugar a que se produzcan eventos en otra señal. La ejecución de las sentencias está determinado por el flujo de valores en las señales. Por tanto, el orden en el cual se especifican las sentencias no es significativo.

## Paquetes y librerías

En VHDL un “package” (paquete o almacén) es un fichero que contiene declaraciones de objetos, tipos de datos, entre otros, utilizados de forma usual.

Una librería es un lugar donde se almacena información (directorio) que se referencia por un nombre lógico (camino en el árbol de directorios). En la librería se almacenan “packages”. Antes de utilizarse deben declararse utilizando la cláusula o palabra clave “library”.

Una vez se ha declarado una librería, las declaraciones contenidas en un “package” se hacen accesibles a un modelo VHDL mediante la palabra clave “use”.

En resumen, al principio de un fichero VHDL se utilizan las palabras clave “library” y “use” de la siguiente forma. La extensión “.all” indica que se utiliza todo el “package” y si sólo es un ítem se especifica el ítem (“my\_func”).

Ejemplo	Sintaxis
<b>library</b> ieee;	<b>library</b> logical-library-name-1, logical-library-name-2;
<b>use</b> ieee.std_logic_1164.all;	<b>use</b> logical-library-name-1.package1.all ;
	<b>use</b> logical-library-name2.package2.my_func;

Previamente hemos comentado que el tipo “std\_logic” está definido en el “package” std\_logic\_1164 perteneciente a la librería ieee. Por tanto, para utilizar el tipo “std\_logic” debemos de especificar la librería y el “package”. Esto es ieee.std\_logic\_1164.all.

Las librerías STD y WORK están implícitamente declaradas. La primera contiene “packages” estandar suministrados por el distribuidor. La segunda se refiere al directorio de trabajo, el cual se puede establecer dentro del entorno de simulación que se usa.

## Elementos léxicos de VHDL

Un identificador es un palabra definida por el usuario para nombrar objetos en modelos VHDL. Cuando se elige un identificador se deben tener en cuenta las siguientes reglas.

- Puede contener sólo caracteres alfanuméricos (A hasta Z, a hasta z, 0 - 9) y el carácter ( \_ ) guión bajo.
- El primer carácter debe ser una letra y el último carácter no puede ser un guión bajo.
- Un identificador no puede incluir dos guiones bajos consecutivos.
- En un identificador no se distingue entre mayúsculas o minúsculas.
- Un identificador puede ser de cualquier longitud.

Ciertos identificadores son utilizados por VHDL como palabras clave para constructores específicos y por ello no pueden utilizarse para identificar objetos o señales. Usualmente las palabras clave se muestran en negrita o en otro color.

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

**Espacios en banco.** VHDL no diferencia entre uno o varios espacios en blanco o tabulaciones.

**Sentencias en VHDL.** De forma similar a otros lenguajes de programación, cada sentencia VHDL finaliza con punto y coma.

**Paréntesis.** VHDL es laxo en los requisitos de uso de paréntesis. En cualquier caso, para mejorar la lectura y comprensión de un código, es una buena idea utilizar paréntesis, siempre que se crea necesario, con el objetivo de ayudar a entender el propósito de un código.

## Operadores

Los operadores lógicos están definidos para tipos "bit" y "std\_logic" y sus vectores. Posteriormente se detallarán otros tipos. Estos operadores se utilizan para definir expresiones lógicas o para efectuar operaciones bit a bit en vectores de bits. El resultado que se obtiene es del mismo tipo que los operandos.

**Operadores lógicos:** and, or, nand, nor, xor, xnor, y not.

Notemos que algún operador como “nand” o “nor” no son asociativos. En consecuencia deben utilizarse paréntesis para indicar el orden de evaluación. En la Figura 1.6 se muestran funciones, a nivel de puerta lógica, disponibles en VHDL.

Expresión lógica	Codificación VHDL	Operación
$Z1 = A1 \cdot B1$	$z1 \leq a1 \text{ AND } b1;$	AND
$Z2 = A2 + B2$	$z2 \leq a2 \text{ OR } b2;$	OR
$Z3 = A3 \oplus B3$	$z3 \leq a3 \text{ XOR } b3;$	OR exclusiva
$Z4 = \overline{A4}$	$z4 \leq \text{NOT } a4;$	negación
$Z5 = \overline{(A5 \cdot B5)}$	$z5 \leq a5 \text{ NAND } b5;$ $z5 \leq \text{NOT}(a5 \text{ AND } b5);$	NAND
$Z6 = \overline{(A6 + B6)}$	$z6 \leq a6 \text{ NOR } b6;$ $z6 \leq \text{NOT}(a6 \text{ OR } b6);$	NOR

**Figura 1.6** Funciones lógicas disponibles en VHDL.

## Utilización de Quartus

Para diseñar un circuito hay que crear un proyecto en el entorno Quartus. Utilizaremos cuatro directorios para organizar la información. El objetivo es que no se estremezclen los ficheros propios con los creados por las herramientas. Los ficheros propios también se separan para identificar fácilmente los ficheros que se corresponden con las descripción del hardware, de los ficheros que se utilizan para su comprobación.

- **CODIGO.** En este directorio se almacenan los ficheros \*.vhd que utilizamos en el diseño.
- **PRUEBAS.** En este directorio se almacenan los ficheros que utilizaremos para comprobar el funcionamiento del diseño con Modelsim.
- **RESULTADOS.** En este directorio se almacenan los ficheros resultados de la simulación con Modelsim.
- **QUARTUS.** En este directorio Quartus almacena los ficheros que crea. En particular se almacena la información que Quartus interpreta como proyecto. Posteriormente se detallan algunos de los ficheros.


Estos directorios deben crearse antes de iniciar un proyecto en Quartus.

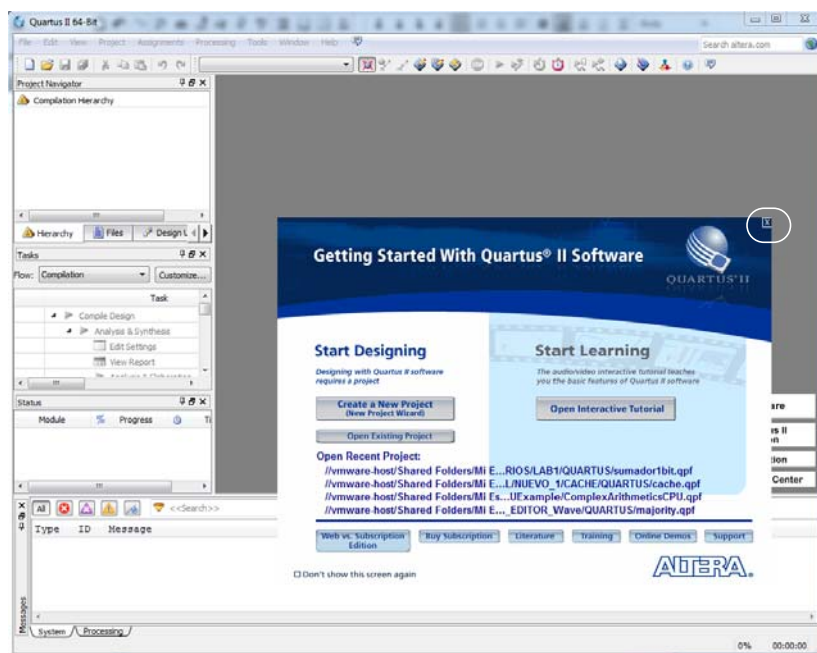
---

*Trabajo: Cree un directorio denominado LAB1 para realizar la práctica. Dentro de este directorio cree el directorio denominado FUNCIONAL y, dentro de este último, cree los cuatro directorios descritos previamente.*

---

## Entorno

Cuando se inicializa Quartus emerge una pantalla donde se muestran distintas opciones que no utilizaremos por ahora. Para seguir se posiciona el cursor  en el rótulo "x" y se pulsa el botón izquierdo del ratón (Figura 1.7).



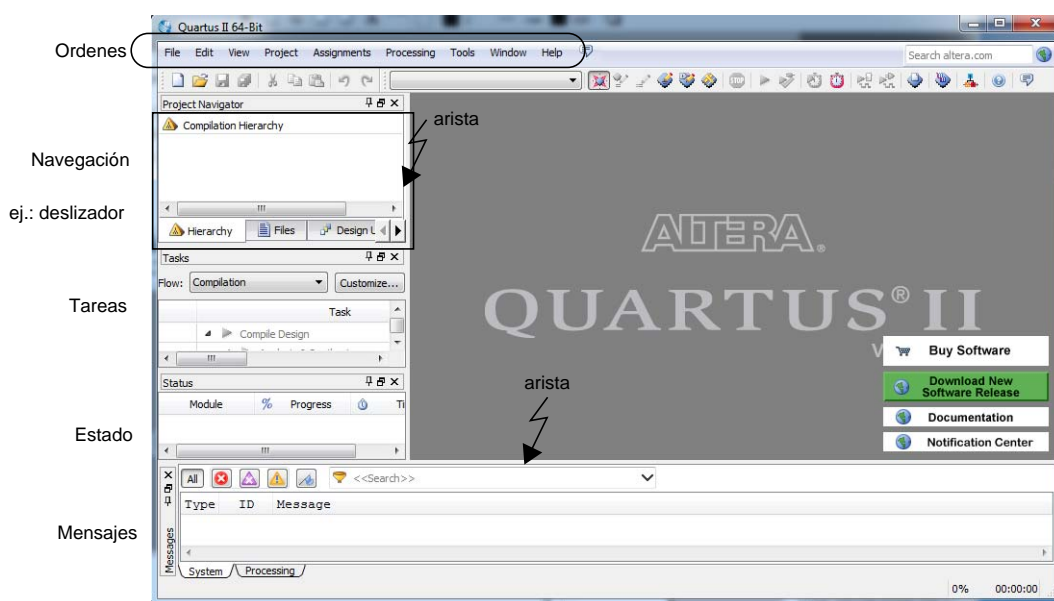
**Figura 1.7** Ventana de bienvenida.

Una vez inicializado Quartus, se observan varias ventanas en la pantalla, incluidas en el marco (ventana) que crea Quartus al inicializarse (Figura 1.8).

- La ventana de navegación. En esta ventana se distinguen varias pestañas que permiten visualizar la organización del diseño en ficheros, en entidades ("entity"), etc.
- La ventana de tareas y la ventana de estado. En esta ventana se muestran las posibles tareas que realiza Quartus y el estado de las mismas.
- La ventana de mensajes. En esta ventana Quartus visualiza mensajes relativos a las tareas que le han sido encomendadas. En esta ventana se distinguen varias pestañas que clasifican los mensajes.
- La paleta de órdenes que se pueden solicitar a Quartus. Por ejemplo, crear un proyecto, editar un fichero, compilar, activar el simulador, especificaciones de configuración.

Las ventanas visualizan sólo parte de la información. Para visualizar el resto se utiliza el deslizador (horizontal, vertical) de la ventana.

El tamaño (horizontal/vertical) de algunas ventanas se puede modificar posicionando el cursor en una arista de la ventana. El símbolo del cursor se modifica y en ese instante se puede redimensionar la ventana en la dirección deseada pulsando el botón izquierdo y moviendo el cursor.




**Figura 1.8** Ventanas de trabajo en Quartus.

Las operaciones que pueden realizarse con Quartus son accesibles mediante el conjunto del menú (rótulos) que se observa en la parte superior de la ventana (paleta de órdenes). En general el cursor se posiciona en un rótulo y al pulsar el botón izquierdo del ratón emerge una ventana. Posteriormente hay que seleccionar una orden.

En general, siempre que se utiliza el cursor para efectuar una selección hay que pulsar el botón izquierdo del ratón. En una paleta de órdenes o en rótulos emerge una ventana de órdenes. Por otro lado, cuando se efectúa la selección (botón izquierdo) dentro de una ventana, distinta de una paleta, el botón del ratón que se utiliza usualmente para que emerja la ventana, es el botón derecho. En consecuencia, cuando no se indique el botón utilizado, se entiende implícitamente que es el botón izquierdo o el derecho, teniendo en cuenta la descripción efectuada previamente.

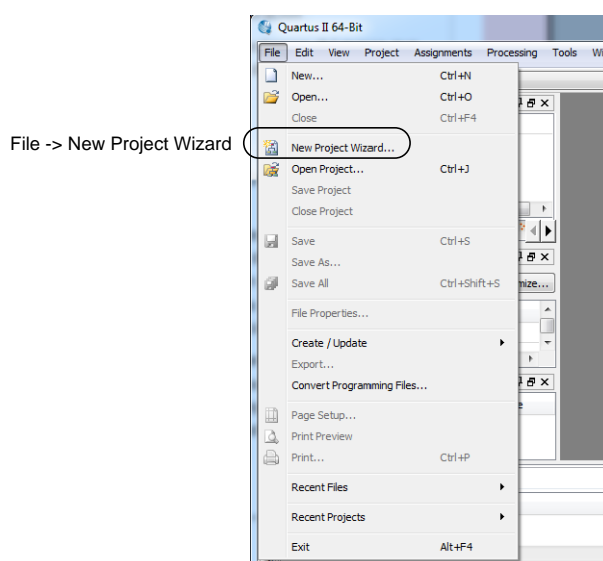
Quartus muestra nuevas ventanas cuando se solicita una operación.

En el marco (ventana) de Quartus se observan varias ventanas. Las ventanas posibles pueden observarse dando la orden “View -> Utility Windows” (paleta de órdenes). Para activar una ventana es suficiente posicionar el cursor encima del acrónimo correspondiente y pulsar el botón del ratón. Las ventanas que muestra Quartus pueden reposicionar o desligarse  del marco que crea Quartus al inicializarse. Las ventanas también se pueden desactivar (“x”).

## Inicio de un proyecto

En este apartado se muestra cómo utilizar la herramienta Quartus para crear un proyecto.

Para utilizar la herramienta debemos efectuar los siguientes pasos: a) posicionar el cursor en el rótulo “File” ubicado en la parte izquierda superior de la ventana de Quartus (Figura 1.9), b) pulsar el botón izquierdo del ratón y emerge una ventana con rótulos, c) liberar el botón, mover el cursor hasta el rótulo que especifica la orden deseada (en este caso “New Project Wizard”) y d) pulsar el botón izquierdo del ratón (selección de la orden).

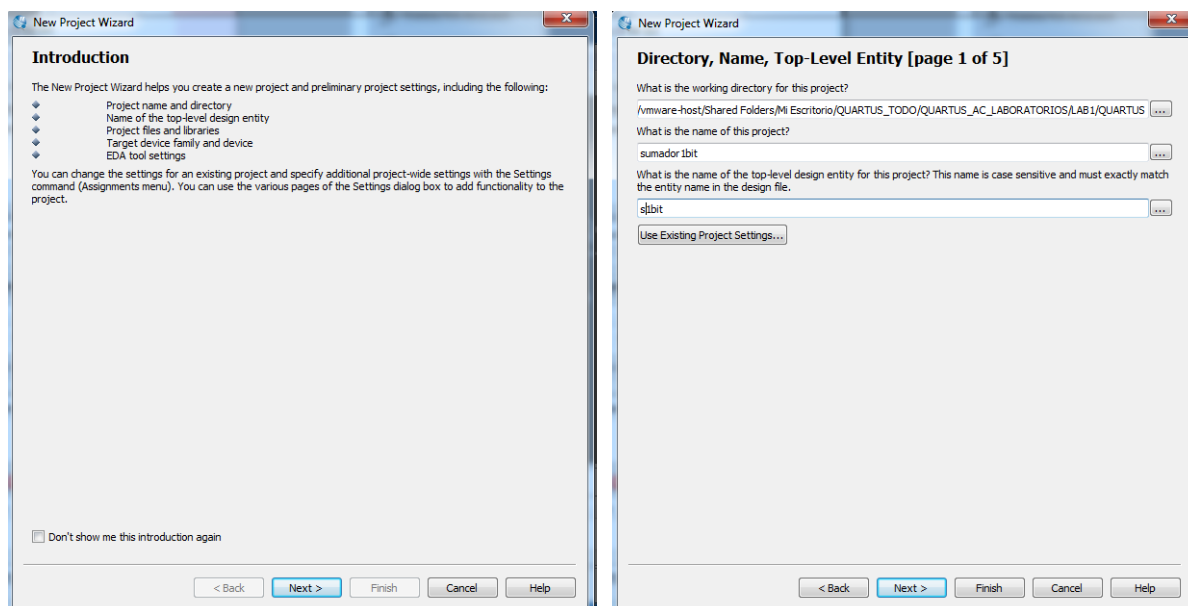


**Figura 1.9** Inicio de un proyecto.

En la parte izquierda de la Figura 1.9 se muestra una simbología para explicitar los pasos previos. Una flecha (->) indica que el rótulo está en otra ventana que se visualiza al pulsar el botón izquierdo del ratón en el rótulo (leyenda) previo. Al conjunto de pasos lo denominaremos dar la orden “último rótulo en el que se ha posicionando el cursor”. Finalmente, en la ventana emergente se selecciona “New Project Wizard”. Seguidamente



emerge una ventana que explica las funcionalidades que se permiten (Figura 1.10). En esta ventana y en las siguientes hay que pulsar con el ratón en el rótulo next para pasar a la siguiente ventana.



**Figura 1.10** Creación de un proyecto. Ventana de inicio y primera ventana para introducir información.

En la nueva ventana hay que especificar:

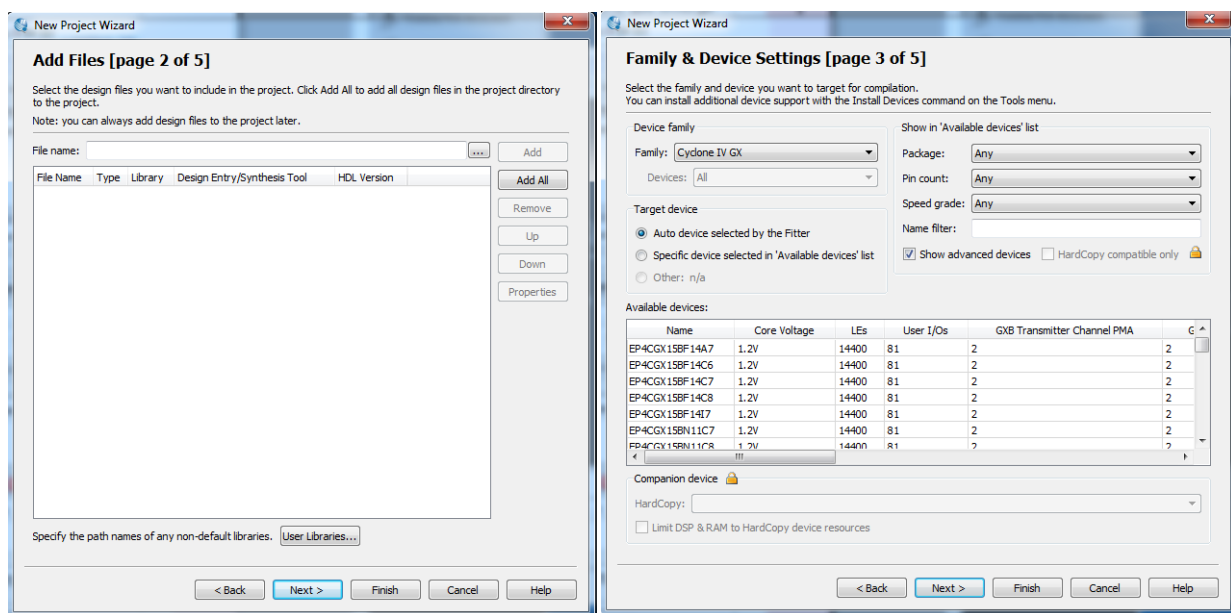
- Directorio de trabajo para el proyecto. Pulse el botón “...” y utilice la ventana emergente para recorrer el sistema de ficheros y ubicarse en el directorio que ha denominado QUARTUS.
- Nombre del proyecto. Para ello utilice el teclado<sup>4</sup>. Observe que el nombre se replica automáticamente en la siguiente entrada que hay que rellenar.
- Nombre de la entidad (“entity”) que es la más externa del proyecto. Este campo ha sido rellenado al nombrar el proyecto, pero puede modificarse<sup>5</sup>. Ahora bien, en cualquier caso, el nombre que se especifique debe utilizarse al declarar la “entity” más externa del proyecto.

4. En este proyecto se ha indicado “sumador1bit”.

5. En este proyecto se ha indicado "s1bit".

*Trabajo: Creación de un proyecto. Realice los pasos descritos hasta este punto y prosiga con los siguientes pasos que se describen para crear un proyecto. Utilice el mismo nombre (s1bit) para designar el proyecto y la entidad.*

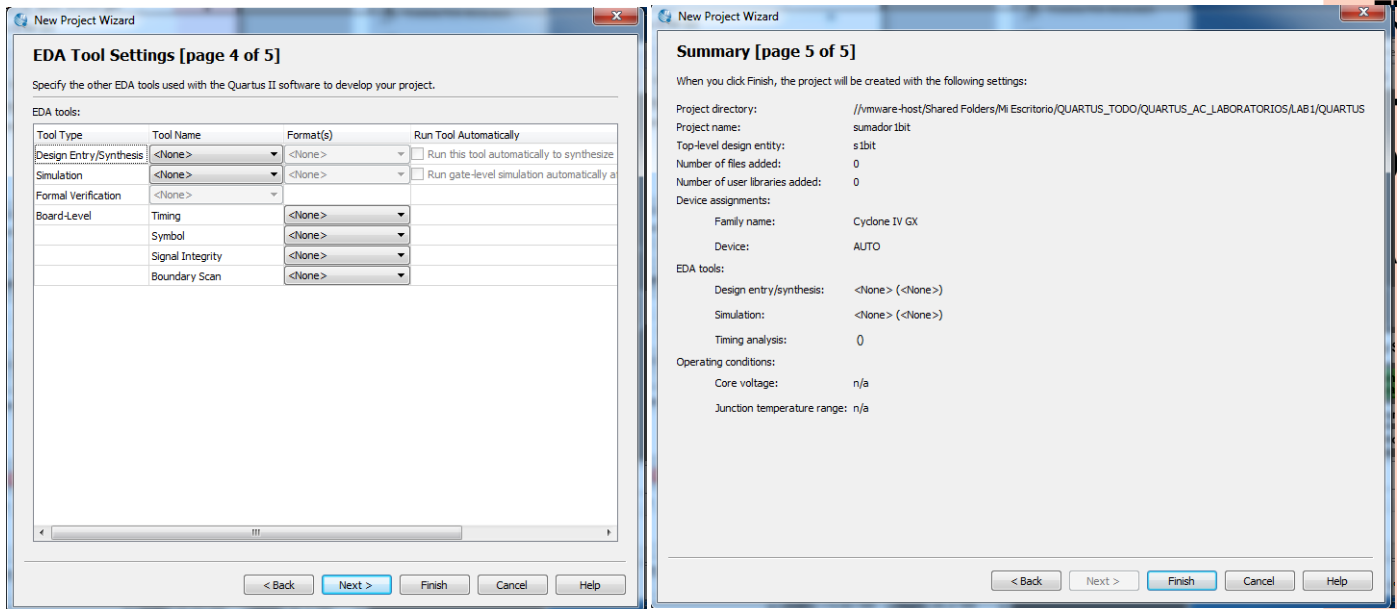
Una vez finalizada la especificación solicitada pulse “next” (Figura 1.11). En la ventana que emerge pulse también “next”, ya que no disponemos de los ficheros VHDL<sup>6</sup>. En las sucesivas ventanas que emergen al pulsar “next” se están seleccionando parámetros por defecto.



**Figura 1.11** Creación de un proyecto. Segunda y tercera ventana para introducir información.

En la quinta ventana pulse el rótulo “Finish” (Figura 1.12). En esta ventana se muestra un resumen de las selecciones efectuadas. Algunas de ellas pueden modificarse posteriormente.

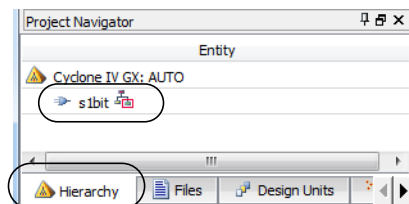
6. En el caso de disponer de los ficheros se pulsa en “...” con el ratón y se utiliza la ventana emergente para buscar en el sistema de ficheros los ficheros que se utilizan. Se selecciona el fichero y posteriormente se pulsa en el rótulo “Add”.



**Figura 1.12** Creación de un proyecto. Cuarta y quinta ventana para introducir información.

Al finalizar el proceso, en el directorio QUARTUS ha sido creado el directorio db y dos ficheros. Los ficheros son: nombre\_del\_proyecto.qpf (sumador1bit.qpf) y nombre\_de\_la\_entidad.qsf (s1bit.qsf). Los dos ficheros son ASCII. Esto es, pueden leerse utilizando un editor de texto. En ellos se encuentran detalladas las selecciones especificadas en el proceso.

Por otro lado, observe que en la pestaña "Hierarchy" de la ventana de navegación se muestra el nombre de la entidad.



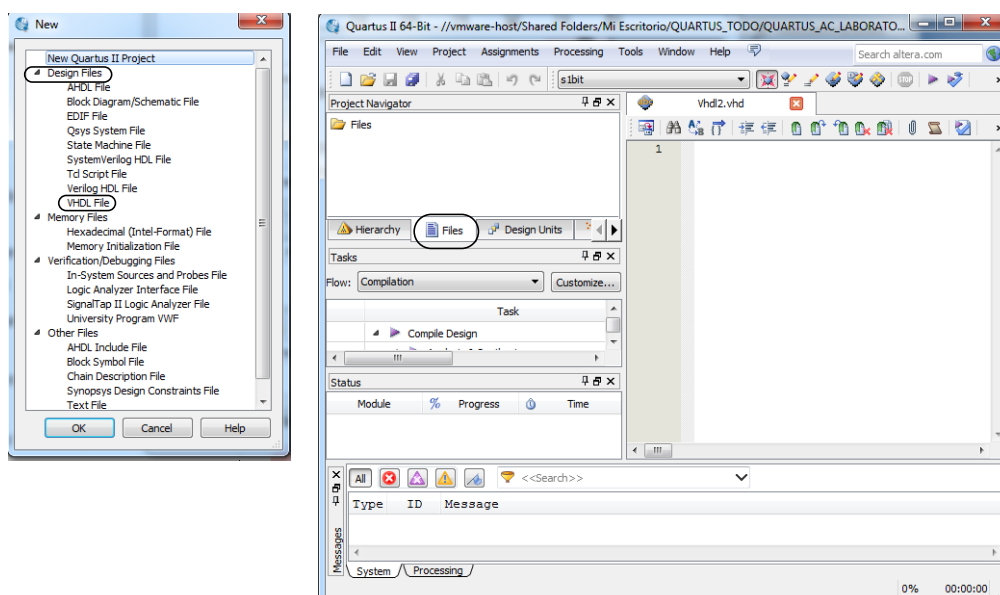
**Figura 1.13** Ventana de navegación. Pestaña Jerarquía.

Para almacenar un proyecto hay que dar la orden "File -> Save Project". Para cerrar un proyecto hay que dar la orden "File -> Close Project".

Una vez creado un proyecto, este puede abrirse, cuando está cerrado, seleccionando el fichero \*.qpf. Para ello hay que dar la orden “File -> Open Project” (ver Figura 1.9) y navegar en la ventana emergente hasta poder seleccionar el fichero. También puede comprobarse si está en la lista de proyectos abiertos recientemente “File -> Recent Projects” y seleccionarlo.

## Edición de ficheros vhdI

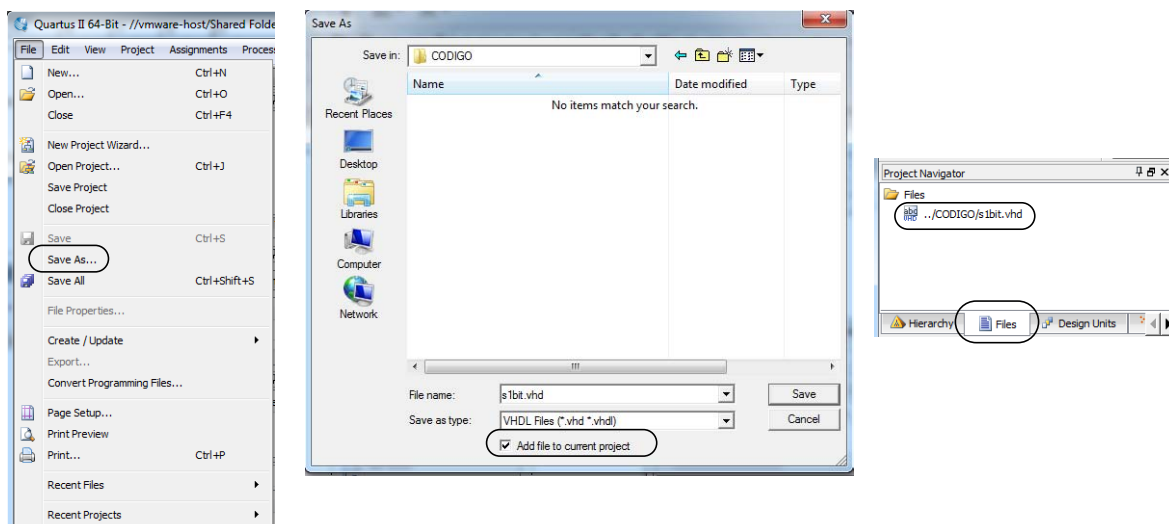
Los ficheros del proyecto se observan en la pestaña “Files” de la ventana de navegación<sup>7</sup>. Para añadir un fichero al proyecto hay que dar la orden “File -> New”. Una vez dada la orden, emerge una ventana donde se efectúa la selección “Design Files -> VHDL File” (Figura 1.14).



**Figura 1.14** Edición de un fichero VHDL.



Seguidamente hay que almacenar el fichero en el directorio CODIGO. Para ello, hay que dar la orden “File -> Save As”. En la pantalla emergente hay que posicionarse en el directorio CODIGO. Quartus propone un nombre para el fichero. Este nombre se corresponde con el nombre utilizado para denominar a la “entity” (Figura 1.15). Para que sea incluido en el proyecto hay que marcar “Add file to current project”.

7. Un fichero VHDL se puede editar utilizando cualquier editor textual. Posteriormente se puede añadir al proyecto cuando se crea el proyecto (página 2 al crear el proyecto) o posteriormente.



**Figura 1.15** Orden para almacenar un fichero en el proyecto. Observación del fichero en la pestaña “Files” de la ventana de navegación.

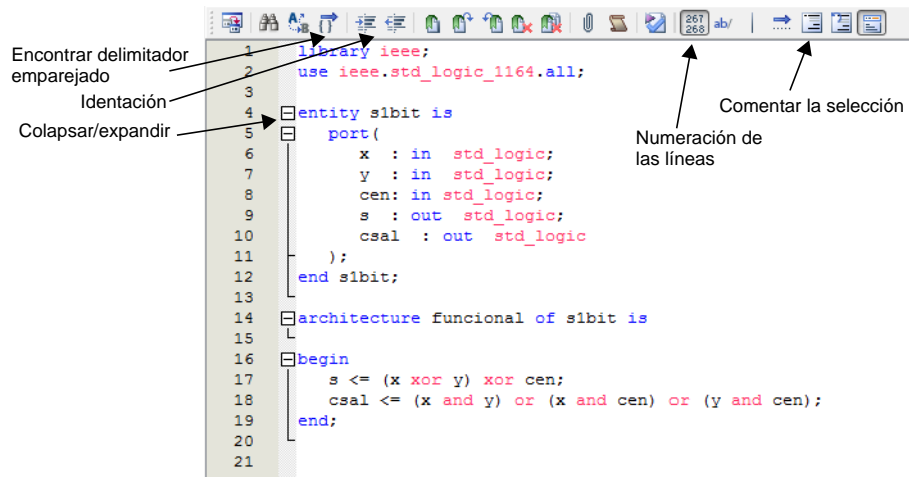
Una vez almacenado el fichero puede observarse en la pestaña “Files” de la ventana de navegación.

En la ventana que emerge, para efectuar la edición del fichero VHDL, hay una paleta de funcionalidades útiles para la edición. En particular el rótulo  permite desligar la ventana de edición del marco de Quartus. Posteriormente se puede utilizar el mismo rótulo  para asociar o ligar la ventana al marco de Quartus.

Para almacenar el fichero, una vez ha sido editado, hay que dar la orden “File -> Save”<sup>8</sup>.

8. Un fichero cerrado puede abrirse pulsando dos veces en el nombre del fichero que se muestra en la pestaña “Files” de la ventana de navegación. Si no está en la ventana debe de abrirse mediante la orden “File -> Open”. En la ventana que emerge hay una opción para incluir el fichero en el proyecto. Otra posibilidad para incluir ficheros en el proyecto es dar la orden “Assignments -> Setting”. En la pantalla emergente seleccionar “Files”. La ventana que se muestra es la de la Figura 1.15.

En la paleta asociada al editor existen otras funcionalidades que ayudan a la edición y al análisis del código (Figura 1.16).



**Figura 1.16** Funcionalidades del editor.

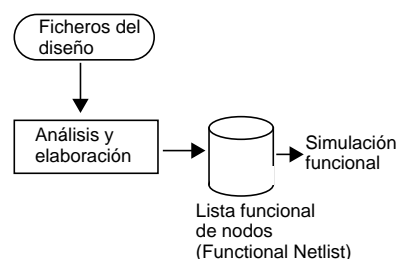
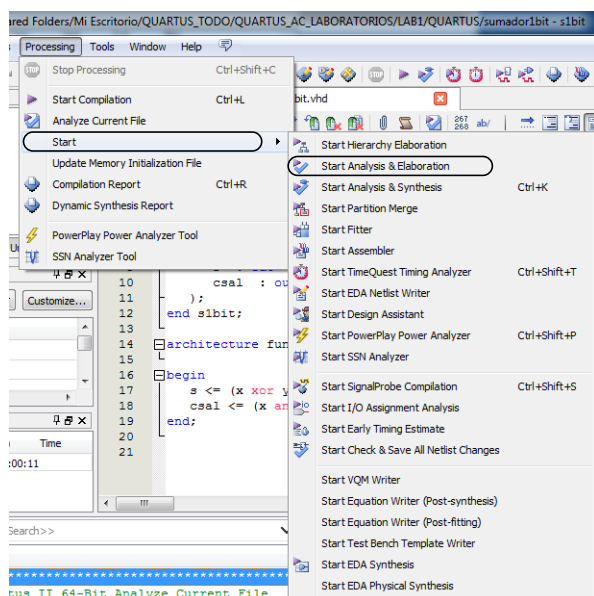
**Trabajo:** Efectúe los pasos descritos para la creación de un fichero. Incluya el código de la Figura 1.17 en el fichero.

<b>library</b> ieee;	Especificación de la librería que alberga las definiciones que deben utilizarse
<b>use</b> ieee.std_logic_1164.all;	
<b>entity</b> s1bit <b>is</b>	Declaración "entity". Nombre del módulo y descripción de la interface del módulo (entradas y salidas).
<b>port</b> ( x, y, : <b>in</b> std_logic;	
cen: <b>in</b> std_logic;	
s: <b>out</b> std_logic;	
csal: <b>out</b> std_logic );	
<b>end</b> s1bit;	
<b>architecture</b> funcional <b>of</b> s1bit <b>is</b>	Declaración "architecture" donde debe describirse el comportamiento del circuito.
<b>begin</b>	
s <= (x xor y) xor cen;	
csal <= (x and y) or (x and cen) or (y and cen);	
<b>end</b> funcional;	

**Figura 1.17** Descripción funcional en VHDL de un sumador de 1 bit.

## Compilación

Una vez se dispone de los ficheros que describen el diseño hay que compilar. Para ello hay que dar la orden “Processing -> Start -> Start Analysis&Elaboration” (Figura 1.18).

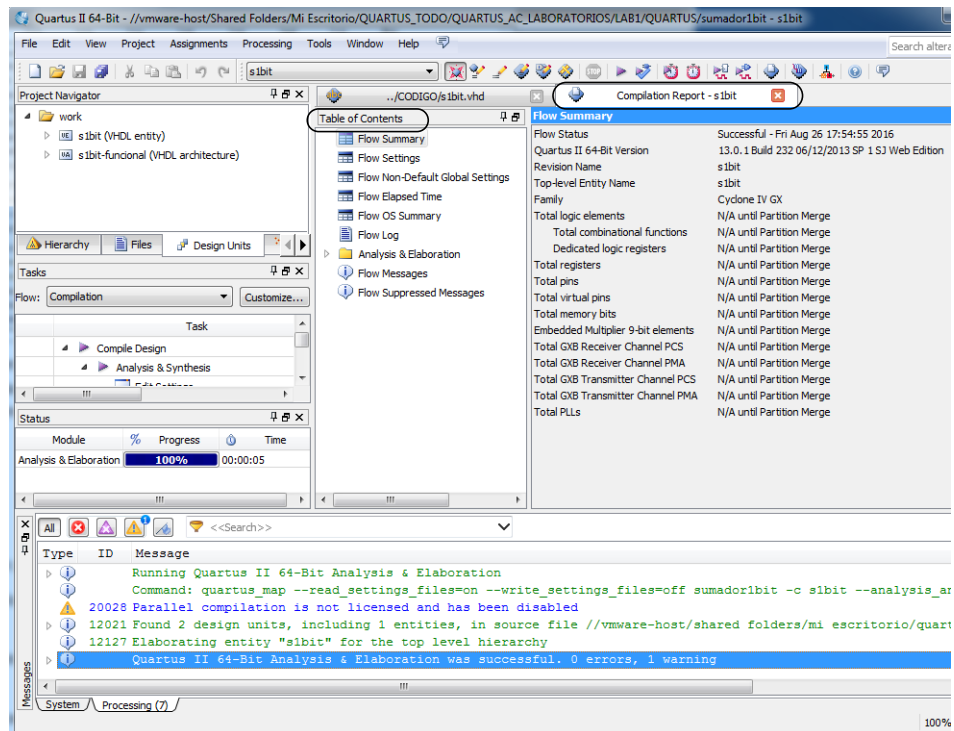


**Figura 1.18** Análisis y elaboración. Comprobación de la sintaxis y construcción de la jerarquía. Inicio de la síntesis.

Al final de la compilación emerge una ventana, denominada “Compilation Report”, con un resumen de la compilación (Figura 1.19). Así mismo, en la pestaña “Processing” de la ventana de mensajes, Quartus muestra el proceso de compilación. En particular indica si se han producido errores o si la compilación ha sido satisfactoria<sup>9</sup>. Durante el proceso de compilación, en la ventana de estado se observa la fracción que ya ha sido realizada.

En la ventana denominada “Table of Contents” se da información sobre el proceso de compilación. En particular, dentro de la carpeta “Analysis & Elaboration” se muestra información sobre la compilación demandada.

9. Si se produce un error, pulsando con el ratón, en la línea que lo identifica en la ventana de mensajes, se abre una ventana con el código VHDL mostrando la línea donde se detecta el error.



**Figura 1.19** Resumen de la compilación.

Después de una compilación satisfactoria, en la pestaña “Desing Units”, de la ventana de navegación, se observan las unidades creadas en el pseudodirectorio work. Este directorio no se observa en el sistema de ficheros. Una de las unidades es la declaración “entity” y la otra unidad es la declaración “architecture”. Pulsando en la flecha que hay en la izquierda se puede observar el fichero donde están declaradas.

Durante el proceso de compilación Quartus ha creado dos nuevos directorios en el directorio QUARTUS: incremental\_db y output\_files. Adicionalmente ha añadido ficheros en el directorio db. En el directorio output\_files están los ficheros (\*.rpt, \*.summary) que contienen la información identificada en la pestaña “Table of Contents” asociada a la ventana resumen de compilación (Figura 1.20).

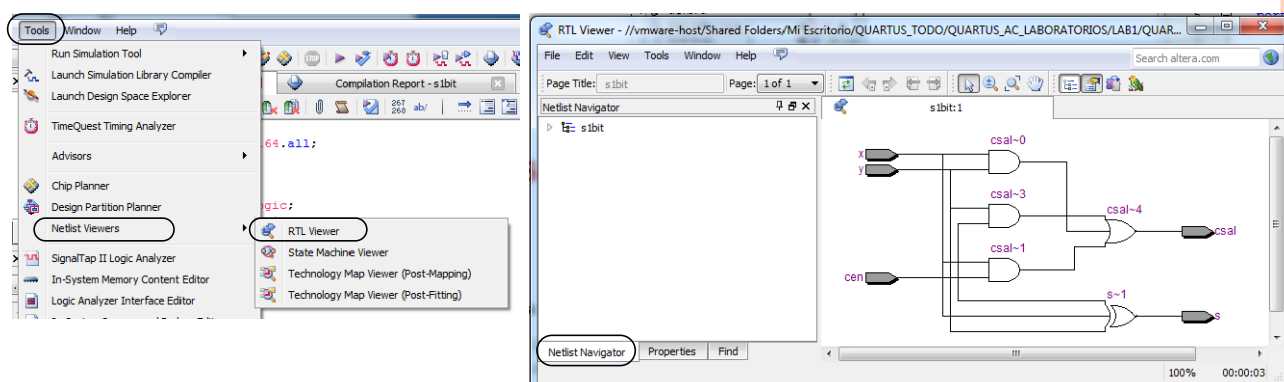
QUARTUS	directorio
incremental_db	directorio
output files	directorio
*.rpt	ficheros
*.summary	ficheros

**Figura 1.20** Información en el directorio QUARTUS al compilar un proyecto.



*Trabajo: Compile el proyecto.*

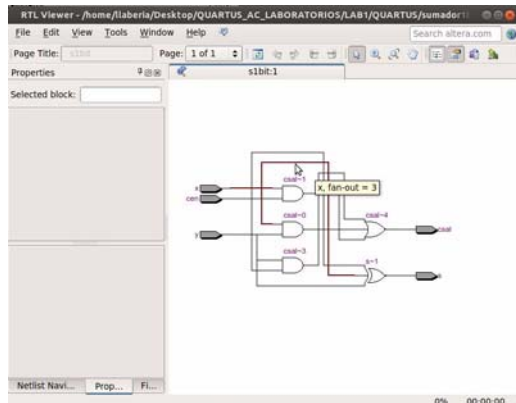
**Elaboración RTL.** Quartus efectúa una elaboración RTL (Register Transfer Level) de la descripción del diseño funcional en VHDL. La elaboración RTL muestra una visión a nivel de diagrama de bloques del circuito: puertas, registros, bloques funcionales, etc. El esquema de circuito que se muestra es el generado después del análisis y pasos iniciales en el proceso de síntesis. Para observarla hay que dar la orden “Tools -> Netlist viewers -> RTL Viewer”. En la pantalla emergente se observa la implementación en puertas lógicas de la especificación efectuada del sumador de 1 bit (Figura 1.21)<sup>10</sup>. Este esquema de circuito se puede imprimir en un fichero PDF dando la orden “File -> Print” o se puede exportar en forma JPG dando la orden “File -> Export”.



**Figura 1.21** Elaboración RTL del diseño efectuado.

La elaboración RTL es útil como ayuda para la detección de errores. Permite observar, en una fase inicial del diseño, la estructura del circuito que se está diseñando. Los nombres de las señales en los cables de interconexión se puede observar posicionando el curso encima de la interconexión (Figura 1.22, hay que esperar algún segundo). La elaboración RTL incluye sólo los bloques del circuito que están alimentados por entradas válidas y producen salidas que se conectan a otros bloques. En consecuencia, si se espera observar un bloque y no es así, hay que analizar la especificación VHDL.

10. Pulsando en el símbolo que hay en el lado izquierdo de “s1bit”, en la ventana de navegación por la lista de nodos (“Netlist Navigator”), se despliega una clasificación de los elementos del diseño. Pulsando en el símbolo que hay en el lado izquierdo de cada uno de ellos se observan los elementos concretos. Pulsando cualquiera de ellos quedan resaltados en el esquema del circuito. De forma similar, al pulsar un elemento en el esquema de circuito se resalta su descripción en la ventana Netlist Navigator”.

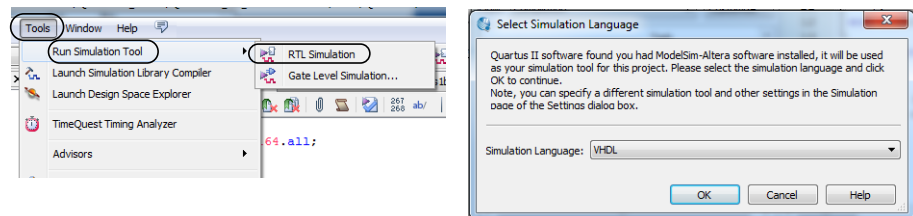


**Figura 1.22** Observación del nombre de una señal.

*Trabajo: Relacione la descripción funcional efectuada en VHDL con la implementación RTL.*

## Activación del simulador Modelsim

La simulación que se efectúa se denomina RTL (Register Transfer Level). Para efectuarla hay que dar la orden “Tools -> Run Simulation Tool -> RTL Simulation” (Figura 1.23).



**Figura 1.23** Activación de la simulación RTL.

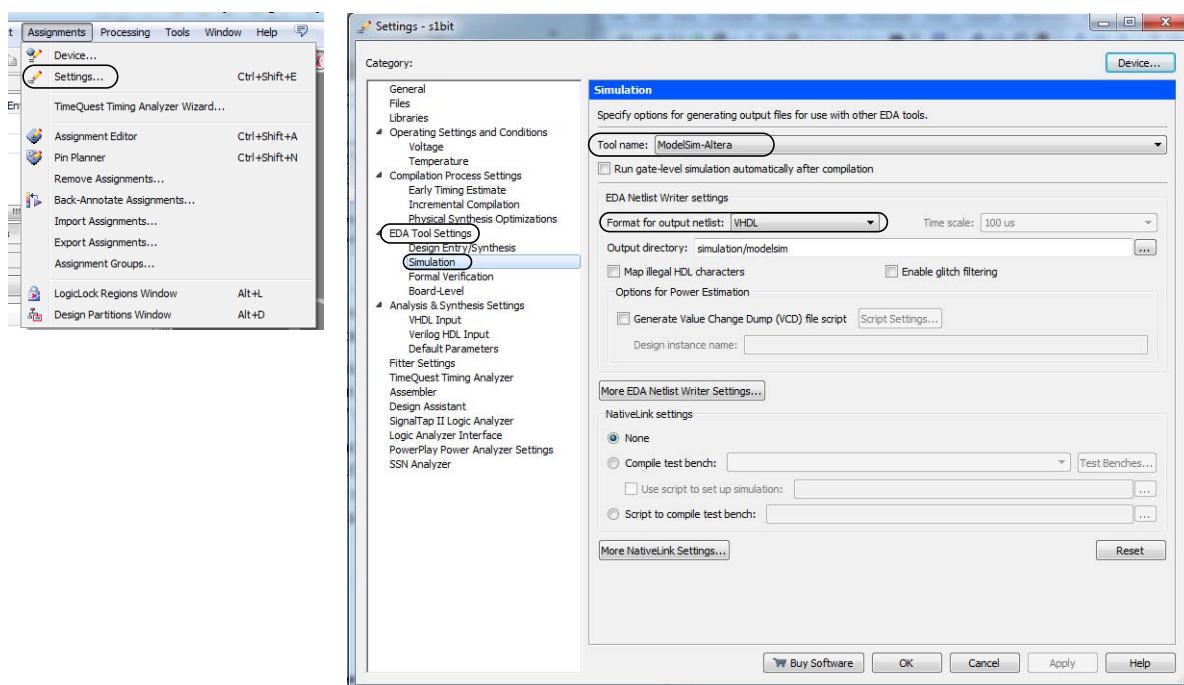
*Trabajo: Efectúe los pasos descritos para iniciar la simulación y los siguientes pasos que se describen.*

Cuando no ha sido especificado el simulador al crear el proyecto, Quartus efectúa una propuesta: utilizar el simulador Modelsim. Hay que responder responder OK (Figura 1.23, parte derecha). Si este no es el caso hay que indicarle el simulador. Para ello dar la orden “Assignments -> Setting” (Figura 1.24).

En la pantalla emergente seleccione “Simulation” y establezca las selecciones que se muestran en la Figura 1.24.

Quartus utiliza la funcionalidad que denomina “NativeLink” para activar el simulador Modelsim. Los mensajes correspondientes a la activación se muestran en la pestaña “System” de la ventana de mensajes de Quartus.

En la activación de Modelsim, Quartus crea el fichero s1bit\_nativelink\_simulation.rpt en el directorio QUARTUS. Así mismo, en este mismo directorio crea el directorio “simulation”, el cual contiene un directorio denominado “modelsim”. En este directorio hay otro directorio denominado “rtl\_work” que contiene los directorios y ficheros utilizados para la simulación.



**Figura 1.24** Indicación del simulador y del tipo de ficheros.

Adicionalmente, en el directorio “modelsim” están almacenados los ficheros: modelsim.ini, msim\_transcript y s1bit\_run\_msim\_rtl\_vhdl.do (Figura 1.25). El primer fichero especifica los parámetros para inicializar el simulador Modelsim. El último de los ficheros es el utilizado por Modelsim para iniciar la simulación. Finalmente el fichero msim\_transcript almacena la información que se muestra en la ventana “Transcript”, que emerge al ser activado el simulador Modelsim. Este fichero almacena todas las órdenes que han sido dadas en el entorno del simulador Modelsim durante la simulación.

QUARTUS	directorio
s1bit_nativelink_simulation.rpt	fichero
Simulation	directorio
modelsim	directorio
rtl_work	directorio
modelsim.ini	fichero
msim_transcript	fichero
s1bit_run_msim_rtl_vhdl.do	fichero

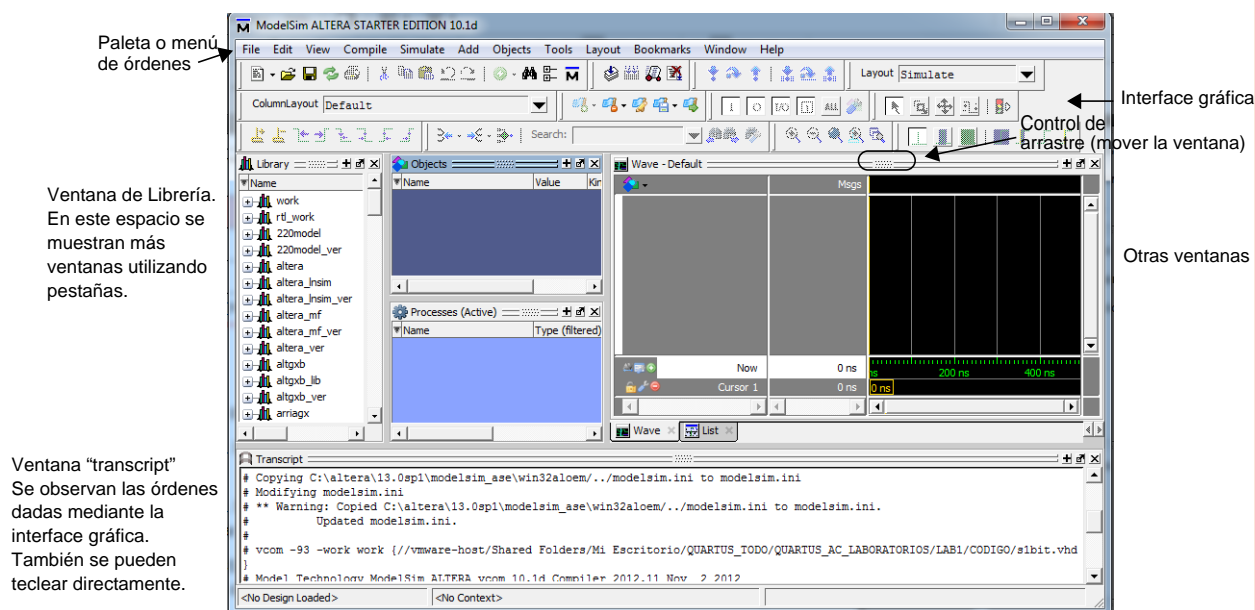
**Figura 1.25** Información añadida en el directorio *QUARTUS* al compilar el diseño.

En la Figura 1.26 se muestran las órdenes que ha ejecutado Modelsim después de ser activado por Quartus (contenido del fichero `s1bit_run_msim_rtl_vhdl.do`). El directorio “work” es el directorio de trabajo por defecto al simular un diseño en VHDL. Una de las órdenes asocia el directorio work al directorio `rtl_work`.

Ordenes	Comentario
<code>vlib rtl_work</code>	Crea el directorio <code>rtl_work</code>
<code>vmap work rtl_work</code>	asocia el directorio work al directorio <code>rtl_work</code>
<code>vcom -93 -work work {/vmware-host/. . . /LAB1/CODIGO/s1bit.vhd}</code>	compila los ficheros vhdl del diseño

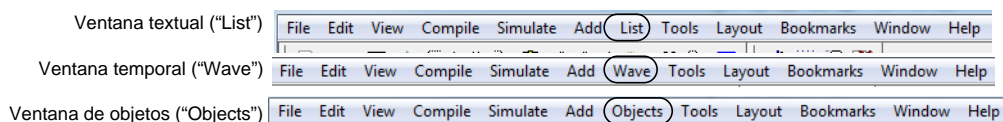
**Figura 1.26** Ordenes para iniciar una simulación en Modelsim.

En el marco (ventana) del simulador Modelsim se observan varias ventanas (Figura 1.27). Las ventanas que se observan están identificadas (✓) en la ventana emergente cuando se pulsa en el rótulo “View” de la paleta de órdenes. Para activar una ventana es suficiente posicionar el cursor encima del acrónimo correspondiente y pulsar el botón del ratón.




**Figura 1.27** Marco de Modelsim.

La paleta de órdenes se modifica en función de la ventana seleccionada (Figura 1.28).

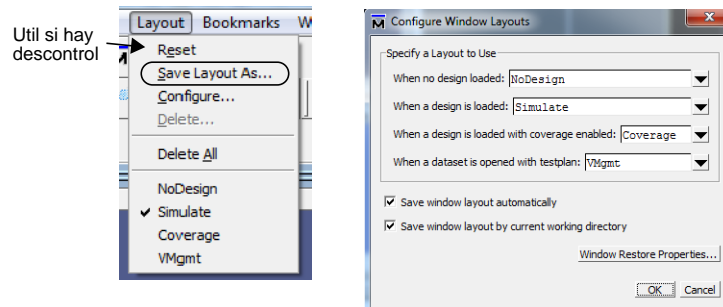


**Figura 1.28** Las órdenes en la paleta son función de la ventana seleccionada.

La ubicación espacial de los rótulos, correspondientes a la interface gráfica, se modifica en función del tamaño del marco del simulador Modelsim.

Una ventana se puede desligar del marco de Modelsim pulsando en el rótulo  (“Dock/Undock”). Posteriormente, pulsando el mismo rótulo se vuelve a posicionar en el lugar original.

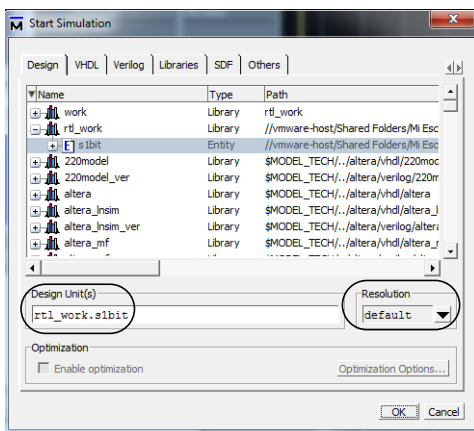
La disposición espacial de las ventanas en Modelsim se puede almacenar para futuras sesiones (Figura 1.29).



**Figura 1.29** Gestión del almacenamiento de la disposición espacial de las ventanas en el marco de Modelsim.

Pulsando con el ratón el simbolo “+”, ubicado a la izquierda de “rtl\_work” dentro de la ventana “Library”, se observan los ficheros incluidos en el diseño.

Para iniciar la simulación hay que dar la orden “Simulate -> Start Simulation”. Emerge la pantalla que se muestra en la Figura 1.30. Hay que seleccionar el fichero. No modifique la “Resolution”. El defecto son ps. La orden que ejecuta Modelsim es la que se muestra en la Figura 1.30.



### Ordenes

```
vsim +altera -do s1bit_run_msim_rtl_vhdl.do -l msim_transcript -gui -t ns rtl_work.s1bit
```

**Figura 1.30** Inicio de la simulación en Modelsim.

Una vez que finaliza el proceso de iniciar la simulación hay una nueva ventana, denominada “sim”, que se identifica mediante una pestaña en la parte central izquierda del marco de Modelsim (Figura 1.31). También las ventanas “Objects” y “Processes (Active)” tienen información. En la primera de ellas se muestran los puertos de entrada y salida del diseño<sup>11</sup>. En la ventana “Processes (Active)” Modelsim identifica las sentencias de asignación de señal (concurrentes) como procesos.

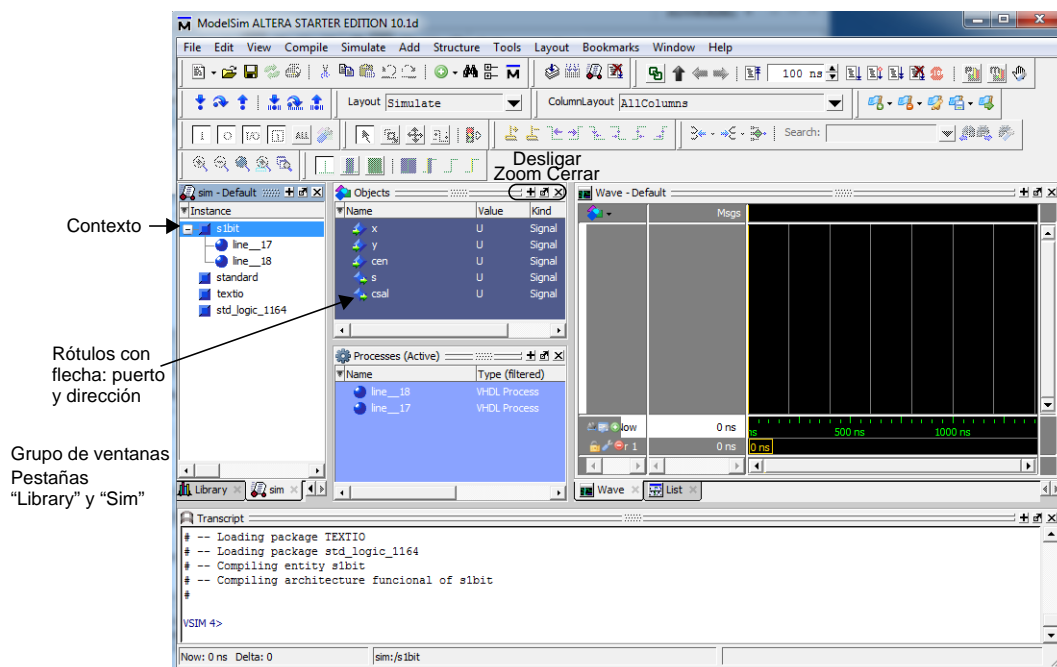


Figura 1.31 Ventana después de haber iniciado la simulación.

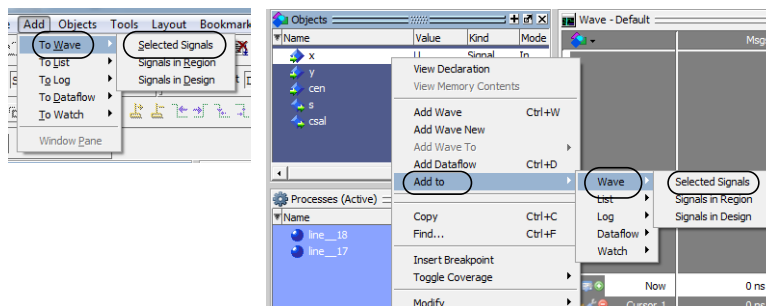
## Preparación de la simulación

Todas las órdenes que se explican utilizando las funcionalidades de la interface gráfica, pueden darse utilizando el teclado, después de haberse posicionado en la ventana de órdenes y de mensajes ("Transcript"), usualmente denominada ventana de mensajes, de Modelsim. La orden que debe darse es la que se observa en la ventana de mensajes cuando se utiliza la interface gráfica.

Para efectuar la simulación hay que establecer valores en los puertos de entrada. La salida de la simulación se puede observar en la ventana "Wave" y en la ventana "List". En la primera de ellas se muestra en formato de ondas de señal y en la segunda de forma textual.

- Otra forma de iniciar la simulación es pulsar en el símbolo "+" en la parte izquierda de "rtl\_work" en la ventana "Library". Posteriormente hay que pulsar dos veces con el botón derecho en la etiqueta "s1bit". La orden que se ejecuta es "vsim rtl\_work.s1bit".

Para observar las señales en las ventanas temporal (“Wave”) y textual (“List”) hay que añadirlas. Una posibilidad es efectuarlo de forma individual. Se selecciona un objeto señal (ventana de objetos) y se da la orden “Add -> ToWave->SelectedSignals”. También se pueden añadir pulsando el botón del ratón una vez seleccionada la señal. Emerge una ventana que permite efectuar varias selecciones, entre ellas añadir la señal a la ventana temporal (“Wave”, Figura 1.32).



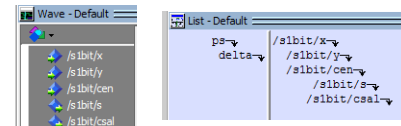
#### Ordenes

add wave \  
sim:/s1bit/x

**Figura 1.32** Añadir señales en la ventana temporal.

Si se quieren añadir todas las señales del diseño se selecciona la ventana “Objects”. Posteriormente se da la orden “Add -> ToWave->SignalsinDesign”. La orden que ejecuta Modelsim es la que se muestra en la Figura 1.33<sup>12</sup>. Para añadir las señales en la ventana lista se sigue el mismo proceso seleccionando “ToList” en lugar de ToWave”.

Ordenes	Comentario
add wave -r /*	añade objetos a la ventana temporal
add list -r /*	añade objetos a la ventana textual

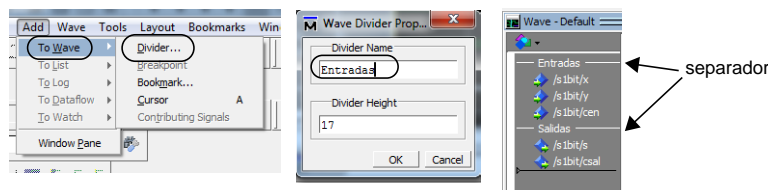


**Figura 1.33** Orden para añadir todas los objetos en las ventanas temporal y textual.

12. Otra posibilidad es seleccionar s1bit en la pestaña “sim” y arrastrarlo hasta la primera columna de la ventana temporal (“Wave”). La orden que se ejecuta para el objeto x es “add wave -position end sim:/s1bit/x”. Para los otros objetos es la misma orden cambiando el nombre del objeto.



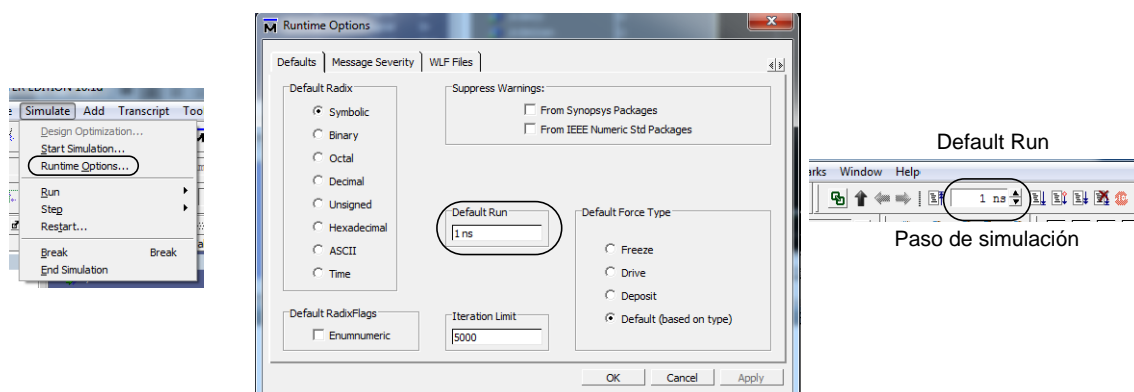
Para facilitar la identificación de señales en la ventana temporal se pueden añadir separadores que identifiquen grupos de señales. Para ello, hay que dar la orden “Add -> To Wave -> Divider” (Figura 1.34). En la pantalla emergente se establece un nombre para identificar la división. En el caso de que un separador no quede posicionado en el lugar que se quiere, puede ser seleccionado y moverse al lugar deseado.



**Figura 1.34** Añadir separadores en la ventana temporal.

**Trabajo:** Efectúe los pasos descritos para ubicar las señales en las ventanas temporal (“Wave”) y textual (“List”). Además, añada los separadores en la ventana temporal.

**Paso de simulación.** Seguidamente se establece el tiempo de un paso de simulación. Para ello hay que dar la orden “Simulate -> Runtime Options”. En la pantalla emergente se establece 1ns (“Default Run”) como paso de simulación. El valor establecido se observa en la ventana de Modelsim (Figura 1.35).



**Figura 1.35** Paso de simulación.

## Simulación

### Estimulación de las entradas

Para establecer un valor en una señal hay que dar la orden "Wave -> Force". Otra posibilidad es seleccionar la señal en la ventana temporal y pulsar el ratón. Emerge un ventana donde hay que seleccionar "Force". En esta ventana hay que establecer un valor en "Value" y en "Delay For". En "Value" se establece el valor de la señal. En "Delay For" se indica cuándo se establece el valor (Figura 1.36). En la misma figura también se muestra la orden que se observa en la ventana de mensajes de Modelsim. A la señal x se le asigna el valor 1 al inicio de la simulación. El valor de esta señal se mantiene ("freeze") hasta que se modifica con otra orden "Force" o con una orden NoForce". Posteriormente, a la misma señal, se le asigna el valor 0 a partir de 5 ns (si no se especifican las unidades se supone el valor por defecto, el cual es ps, que es el tiempo establecido en "Resolution", Figura 1.30).

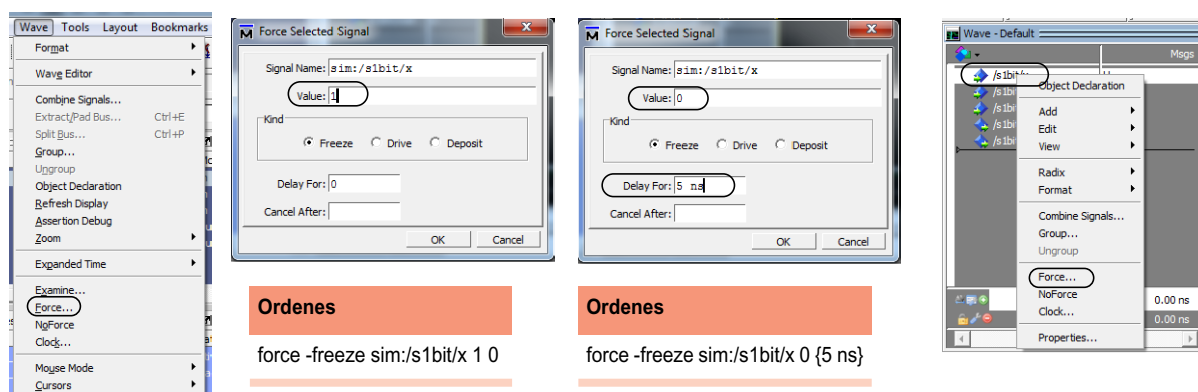
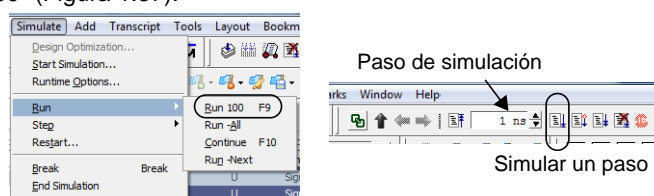


Figura 1.36 Orden para establecer un valor en una señal.

### Activación de la simulación

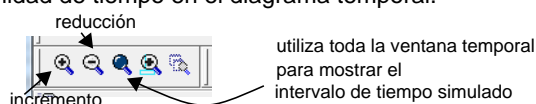
*Trabajo: Asigne los valores 1, 1, 0 a las señales x, y, cen respectivamente. Establezca el valor cero en la señal x después de 5 pasos de simulación (5 ns).*

Una vez establecidos valores en las señales de entrada se puede iniciar la simulación. Para ello hay que pulsar en el rótulo que hay a la izquierda de la ventana donde se muestra el tiempo de un paso de simulación. Una alternativa es dar la orden “Simulate -> Run -> Run 100” (Figura 1.37).



**Figura 1.37** Efectuar un paso de simulación.

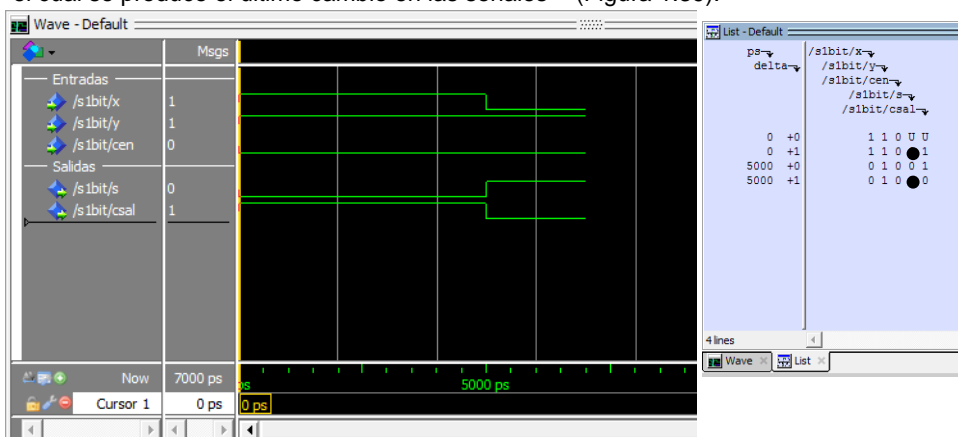
Cada vez que se da la orden previa, el simulador simula 1ns (valor de un paso de simulación, “Default Run”). Para visualizar de forma cómoda las señales, en la ventana temporal, pueden utilizarse los rótulos que permiten incrementar o disminuir el espacio que ocupa una unidad de tiempo en el diagrama temporal.



**Figura 1.38** Herramientas para visualizar las señales en la ventana de temporal.

**Trabajo:** Pulse varias veces en el rótulo que da la orden de realizar un paso de simulación y compruebe el resultado.

La ventana temporal y la ventana textual tienen el aspecto que se muestra en la Figura 1.39. Observe que en la ventana textual se muestran valores hasta 5 ns; es el instante en el cual se produce el último cambio en las señales<sup>13</sup> (Figura 1.39).



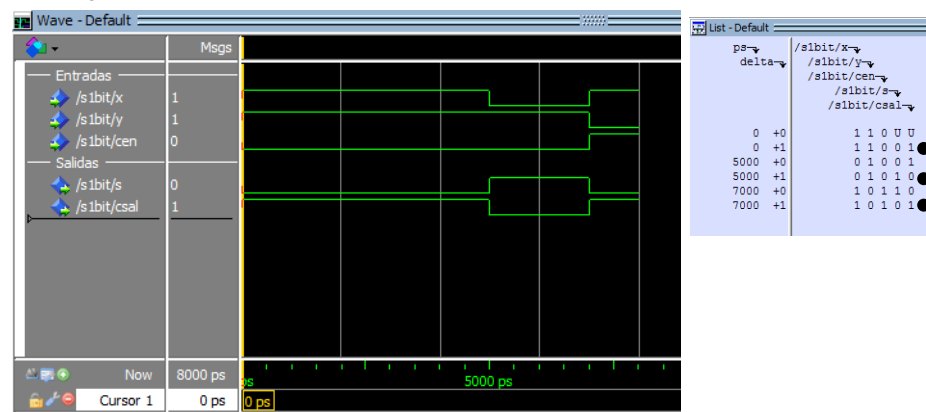
**Figura 1.39** Valores de las señales en las ventanas temporal y textual.

Para observar el valor de una señal, en un punto de la ventana temporal, es suficiente posicionar el ratón encima de la traza de la señal deseada, en el lugar deseado. Al cabo de cierto tiempo emerge una ventana con información.

Para comprobar otros valores de entrada es suficiente establecer nuevos valores en los puertos de entrada.

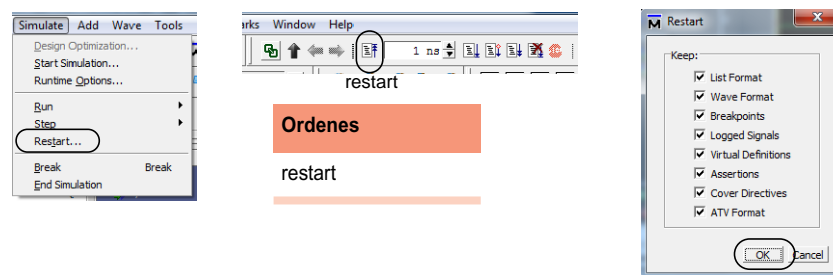
*Trabajo: Asigne los valores 1, 0, 1 a las señales x, y, cen respectivamente. Pulse varias veces en el rótulo que da la orden de realizar un paso de simulación y compruebe el resultado.*

En la Figura 1.40 se observa el resultado de la simulación.



**Figura 1.40** Resultado de la simulación.

Para empezar una nueva simulación (tiempo cero) se utiliza la orden "Simulate -> Restart". Después de dar la orden, emerge una ventana donde debe pulsarse OK (Figura 1.41).

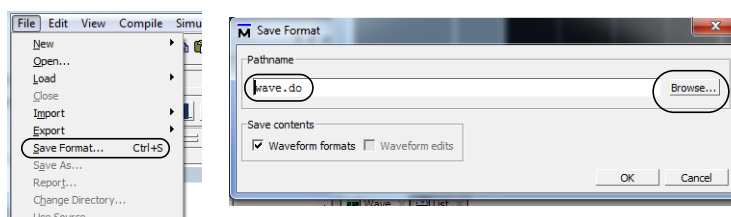


**Figura 1.41** Poner a cero la simulación.

13. En la ventana textual ("List") no tenga en cuenta la columna etiquetada como "delta". Es un parámetro interno del simulador. Dado un grupo de filas con el mismo tiempo, la fila de interés es la que tiene un "delta" mayor (en la Figura 1.39 han sido marcadas con un círculo).

## Almacenamiento de la configuración de la simulación

**Almacenamiento del formato de las ventanas temporal y textual.** El formato de la ventana temporal y de la ventana textual puede almacenarse para futuras simulaciones. Para ello hay que seleccionar la ventana cuyo formato (señales que se observan) se quiere guardar y posteriormente hay que dar la orden que se muestra en la Figura 1.42. En la ventana que emerge se propone un nombre para fichero y el directorio donde se almacena. Para conocer el directorio pulse en "Browse".



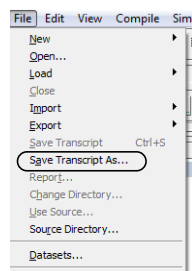
### Ordenes

```
write format wave -window .main_pane.wave.interior.cs.body.pw.wf {/vmware-
host/Shared Folders/Mi Escritorio/. . /LAB1/QUARTUS/simulation/modelsim/wave.do}
```

**Figura 1.42** Almacenamiento del formato de la ventana temporal.

**Utilización del formato almacenado de las ventanas temporal y textual.** En una activación posterior del simulador Modelsim, después de activar la simulación, hay que dar las órdenes, en la ventana de mensajes ("transcript"), "camino-del-sistema-de-ficheros/do wave.do", "camino-del-sistema-de-ficheros/do list.do". Mediante la orden "pwd" se conoce el directorio donde está posicionado el intérprete de la ventana "transcript". Si los ficheros están en el mismo directorio, en el cual está posicionado el intérprete, no es necesario especificar el "camino-del-sistema-de-ficheros".

**Almacenamiento de los mensajes en la ventana de órdenes y mensajes.** Todos los mensajes que han sido observados en la ventana de mensajes (“transcript”) puede almacenarse en un fichero. Para ello, después de seleccionar la ventana de mensajes, hay que dar la orden que se muestra en la Figura 1.43. Emerge una ventana que permite nombrar el fichero e indicar el directorio donde se quiere almacenar.



**Figura 1.43** Almacenamiento de la ventana de órdenes y mensajes.

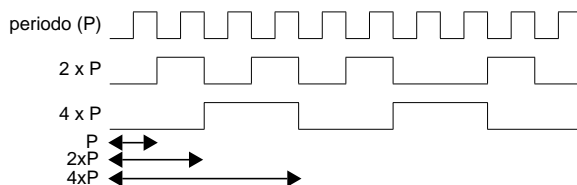
## Comprobación del funcionamiento lógico

Para comprobar el funcionamiento del dispositivo utilizaremos la tabla de verdad del sumador de 1bit (Figura 1.44). Ahora bien, utilizaremos una forma metódica; en lugar de utilizar la orden “Force” utilizaremos la orden “Clock”.

entradas			salidas	
cen	y	x	csal	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

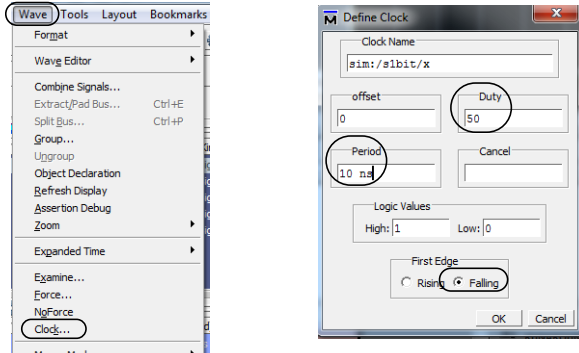
**Figura 1.44** Tabla de verdad de un sumador de 1 bit.

Para cada puerto de entrada se describe una señal periódica. El periodo de la señal del segundo puerto es el doble del periodo de la señal de primer puerto. El periodo de la señal del tercer puerto es el doble del periodo de la señal del segundo puerto.



**Figura 1.45** Señales periódicas.

En la Figura 1.46 se muestra la forma de establecer una señal periódica en un puerto de entrada. La señal es cuadrada (Duty: 50%), el periodo es de 10 ns y empieza en un flanco descendente. Deben de especificarse las unidades de tiempo. El defecto son ps.

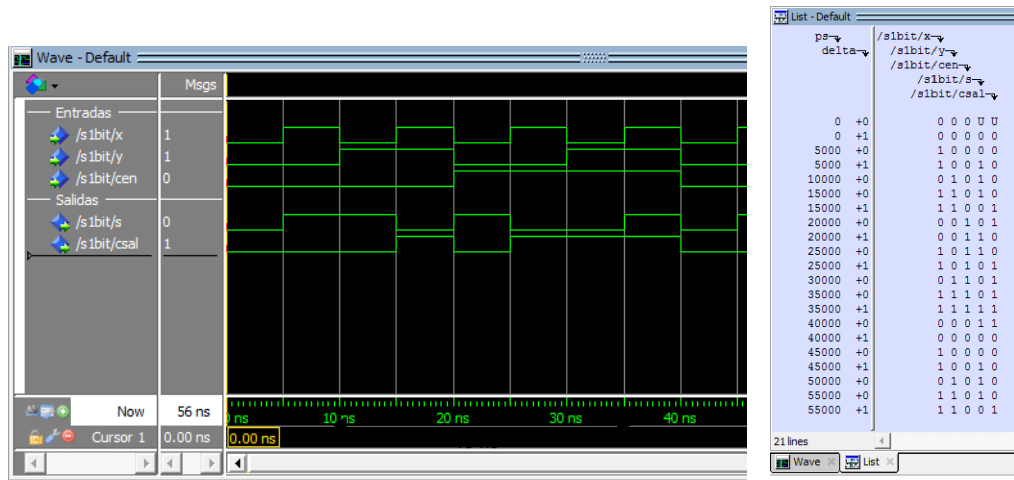


#### Ordenes

```
force -freeze sim:/s1bit/x 0 0, 1 {5000 ps} -r {10 ns}
```

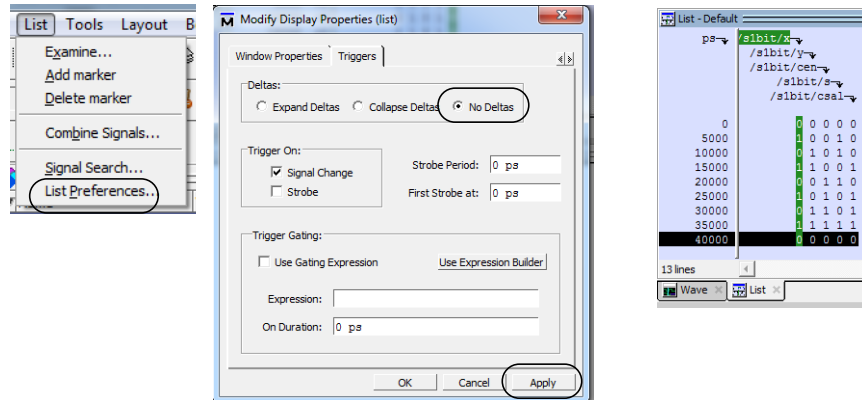
**Figura 1.46** Señal periódica. Utilización de la orden "Clock" para establecer valores en una señal.

En la Figura 1.47 se muestra la salida de la simulación, en la ventana temporal y en la ventana textual, después de establecer señales periódicas.



**Figura 1.47** Ventanas temporal y textual mostrando el resultado de la simulación.

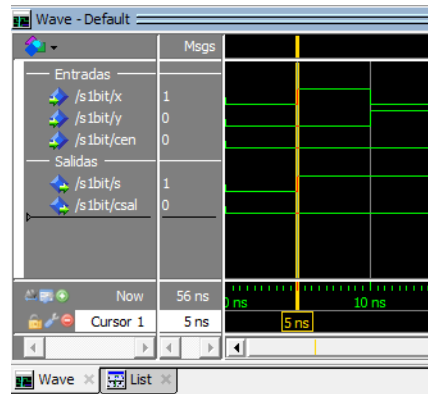
En la ventana textual puede eliminarse la columna “delta” (Figura 1.48).



**Figura 1.48** Supresion de la columna delta en la ventana textual.

## Análisis de las señales en la ventana temporal

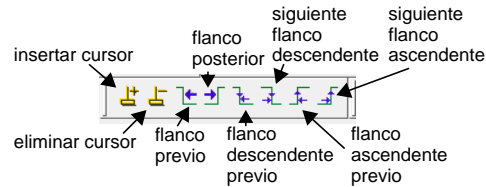
Para identificar el valor de las señales en un instante determinado se puede utilizar un cursor en la ventana temporal. En suficiente posicionar el ratón en el lugar elegido de la ventana temporal y pulsar el ratón. Emerge una línea vertical (cursor). En la segunda columna del diagrama temporal se observa el valor textual de las señales y en la parte inferior de la ventana el instante de tiempo (Figura 1.49). Es el cursor por defecto que al empezar la simulación está en tiempo cero. Una vez seleccionado el cursor puede moverse por el diagrama temporal moviendo el ratón.



**Figura 1.49** Cursor en la ventana temporal.

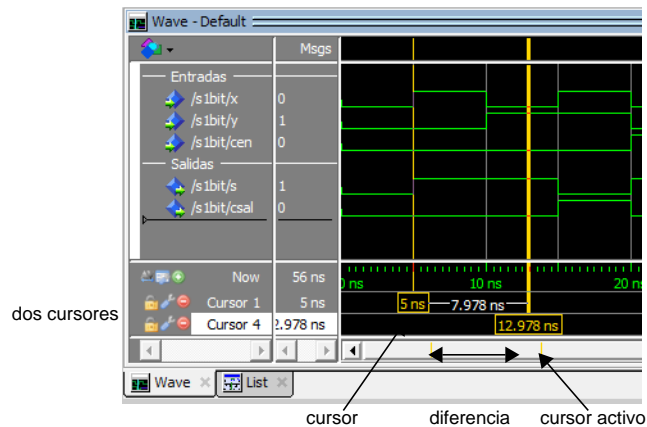


Una vez seleccionada una señal en la ventana temporal, podemos utilizar funcionalidades de Modelsim, accesibles en los rótulos, para mover un cursor de flanco en flanco (Figura 1.50).



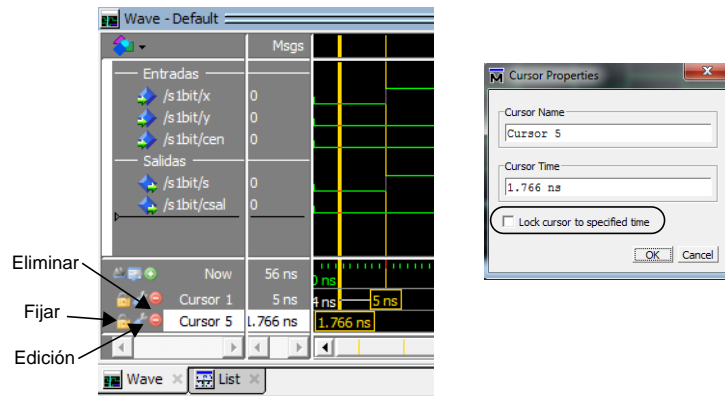
**Figura 1.50** Ordenes para analizar las señales en la ventana temporal.

Otra funcionalidad disponible es la inserción de cursores en la ventana temporal. Cada cursor tiene asociada una fila en la parte inferior de la ventana temporal (Figura 1.50). Mediante esta funcionalidad se observan en la ventana de tiempos diferencias temporales (Figura 1.51, tiempo en cada cursor y diferencia). Seleccionando una señal y un cursor éste puede moverse entre flancos utilizando las funcionalidades descritas previamente. El cursor activo o seleccionado se muestra con una línea más gruesa.



**Figura 1.51** Varios cursores en la ventana temporal.

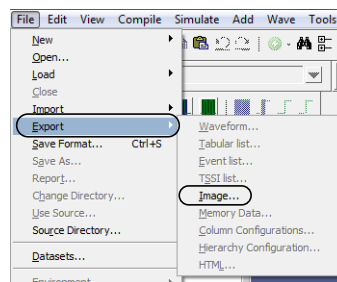
Un cursor se puede fijar en una posición determinada o establecer textualmente una posición determinada en el diagrama temporal. Una posibilidad es editarlo. Hay que pulsar en el rótulo “llave inglesa” asociado al cursor y emerge una ventana donde se puede establecer el tiempo donde se posiciona el cursor y si se queda fijo (“lock”). Las otras posibilidades se muestran en la Figura 1.52.



**Figura 1.52** Gestión de los cursores.

## Almacenamiento de resultados

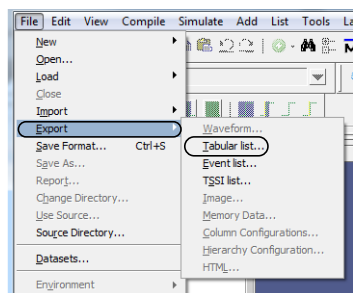
**Almacenamiento de la ventana temporal.** La ventana temporal puede almacenarse en un fichero en formato \*.bmp. Para ello hay que seleccionar la ventana temporal y efectuar los pasos que se muestran en la Figura 1.53. Emerge una ventana que permite posicionarse en el directorio donde se quiere almacenar el fichero (RESULTADOS).



**Figura 1.53** Almacenamiento de la ventana temporal.

La ventana temporal también puede almacenarse en formato postscript. Para ello de la orden “File -> Print Postscript”.

**Almacenamiento de la ventana textual.** La ventana textual puede almacenarse en un fichero. Para ello hay que seleccionar la ventana textual y seguir los pasos que se muestran en la Figura 1.54. Emerge una ventana que permite posicionarse en el directorio donde se quiere almacenar el fichero (RESULTADOS). Hay que seleccionar el formato “Tabular List”. Es el formato que se observa en la ventana.



#### Ordenes

```
write list -window .main_pane.list.interior.cs.body {/vmware-
host/Shared Folders/Mi Escritorio/. .LAB1/RESULTADOS/list.lst}
```

**Figura 1.54** Almacenamiento de la ventana textual.

*Trabajo: Comprobación del funcionamiento lógico. Establezca señales periódicas en los puertos x, y, cen. Efectúe una simulación para comprobar el valor de las variables de salida para todas las combinaciones posibles de valores de entrada.*

## Modelo VHDL con retardo

El modelo VHDL que hemos utilizado previamente se puede denominar modelo de comportamiento (“Behavioral Model”). No se especifican elementos básicos tecnológicos que den lugar directamente a una implementación. En otra sesión de laboratorio se mostrará otra forma de especificar este tipo de modelos, la cual se asemeja en parte a un lenguaje de programación usual.

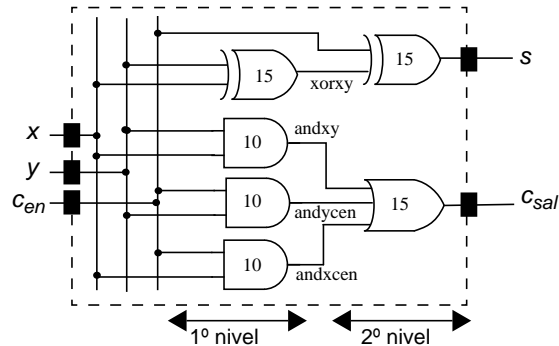
En esta sección utilizaremos un modelo denominado de flujo de datos (“Dataflow Model”) y además especificaremos retardos. En un modelo de flujo de datos se muestra todo el camino de datos y las señales de control utilizando elementos básicos.

**Modelo de flujo de datos (“Dataflow Model”).** Se especifica la sucesión temporal con la que evolucionan las señales de entrada para obtener la salida.

Estamos interesados en una simulación detallada del circuito donde se modelen las transiciones de todas las señales en una implementación física. Entonces, además de las señales de la interface queremos modelar las señales intermedias del circuito.

En la Figura 1.55 se muestra un esquema con puertas lógicas de un sumador de 1bit y el retardo asociado a cada puerta. El número de niveles de puertas del sumador depende de los tipos de puertas disponibles y del número de entradas de las puertas. Las puertas

utilizadas tiene 2 entradas (AND y XOR) y 3 entradas (OR). En esta sección se describe cómo especificar el cuerpo de la arquitectura, para describir con detalle el esquema de la Figura 1.55. Notemos que la descripción de la interface (“entity”) no se modifica.



**Figura 1.55** Esquema de puertas lógicas de un sumador de 1 bit. Los números en el interior de las puertas indican el retardo en ns.

En el esquema de la Figura 1.55 se observan señales internas, las cuales deben describirse en el modelo. Así mismo debe especificarse el retardo de cada puerta lógica.

Las sentencias de asignación de señales concurrentes, en la especificación de la arquitectura, se comunican mediante señales (“signals”), las cuales representan las entradas y salidas y cables del circuito digital. En la declaración de la interface se declaran los puertos de entrada y salida del módulo. Cuando son necesarios cables adicionales, se declaran señales (“signals”) en la parte declarativa del cuerpo de la arquitectura. Esto es, en la parte donde se describe el funcionamiento.

Teniendo en cuenta la declaración de señales y la especificación de constantes para especificar el retardo, el aspecto del cuerpo de la arquitectura es el siguiente.

Ubicación	<b>architecture</b>	architecture_name of NAME_OF_ENTITY is	
	<b>constant</b> . . .		parte declarativa
	<b>signal</b> . . .		
	<b>begin</b>		
		-- Concurrent Statements	
	<b>end</b> architecture_name;		

## Objetos de datos en VHDL: Señales y constantes

Un objeto de datos (“Data Object”) es un ítem en VHDL, que se crea mediante una declaración, tiene un nombre (identificador asociado), un valor y tipo asociado. Un objeto puede ser una constante (“Constant”) o una señal (“Signal”). Posteriormente explicaremos otros objetos. Hasta ahora hemos visto que los puertos de entrada y salida son señales. Las señales pueden considerarse cables en un esquema de circuito, las cuales

pueden tener un valor actual y valores futuros (historia de valores), y estos valores son función de las sentencias de asignación de señales utilizadas. Una constante asocia un valor a un símbolo de un tipo de datos. Seguidamente observaremos que la declaración de los distintos objetos de datos es muy similar.

**Constante (“Constant”).** Una constante se inicializa con un valor al declararla y este valor no se puede modificar durante la simulación. La declaración se efectúa de la siguiente forma.

Ejemplo	Sintaxis
<b>constant</b> retardoand: <b>time</b> := 2ns;	<b>constant</b> constant_name: <b>type</b> := initial value;

En la declaración de una constante se especifica el nombre del objeto, el tipo y su valor. Las constantes se declaran al principio del cuerpo de la arquitectura y entonces pueden ser utilizadas en cualquier sentencia dentro de la arquitectura. Posteriormente se indica otro lugar donde se pueden declarar constantes y su visibilidad.

**Tipo “time”.** Es un tipo predefinido en el estandar de VHDL. Se especifica un valor numérico, un espacio y las unidades (fs, ps, ns,  $\mu$ s, ms, sec, min, hr).

La disponibilidad del tipo “time” es una consecuencia natural del modelado que se quiere obtener al efectuar la simulación.

**Señales (“Signals”).** Las señales se declaran al principio del cuerpo de la arquitectura utilizando la siguiente sentencia.

Ejemplo	Sintaxis
<b>signal</b> andxy, andxcen: std_logic;	<b>signal</b> signal_names: type [ := initial value] ;

En la declaración de una señal se especifica el nombre del objeto, el tipo y un posible valor inicial. Los objetos “signal” por defecto están indefinidos.

El objeto “signal” representa una señal lógica transportada por un cable en un circuito. Una señal no tiene memoria. Por tanto, si la fuente de la señal se elimina, la señal no tiene ningún valor. Esto es, está indefinida.

## Especificación de retardos en VHDL

Un retardo en una sentencia de asignación de señal se especifica de la siguiente forma.

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

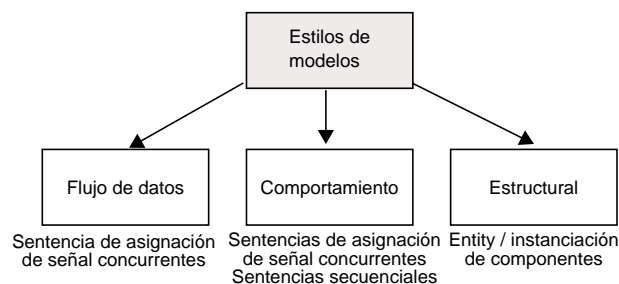
La especificación anterior refleja la realidad del hardware, donde la lógica tiene retardos de propagación. El retardo incorporado en una sentencia de asignación de señal tiene en cuenta el retardo de la lógica (evaluar “value expresion”) y cualquier otro retardo de propagación necesario para que el nuevo valor se observe al final del cable.

En resumen, una señal actualiza su valor cuando se ejecuta su sentencia de asignación de señal y ha transcurrido el retardo especificado. Si no se especifica un retardo, la señal se actualiza después de un retardo denominado delta utilizado por el simulador para ordenar los eventos.

## Descripción en VHDL del sumador de 1bit con retardos

Al modelar un circuito el diseñador especifica eventos, retardos y concurrencia. Los eventos se especifican con sentencias de asignación de señales. Los retardos se especifican en las sentencias de asignación de señales. La concurrencia se especifica utilizando una sentencia de asignación de señal para cada señal que se quiere modelar. El orden de ejecución de las sentencias depende del flujo de datos y es independiente del orden en el que se han especificado las sentencias de asignación de señales.

En la Figura 1.56 se muestran tres clases de modelos de arquitectura al describir un diseño. La descripción estructural de una arquitectura se efectúa en el próximo capítulo. La descripción de una arquitectura mediante sentencias secuenciales se efectúa dentro de dos capítulos.



**Figura 1.56** Modelos utilizados para describir una arquitectura.

Cuando se utiliza un modelo de flujo de datos y se especifican los retardos, la Figura 1.57 muestra el cuerpo de la arquitectura para el sumador de 1 bit de la Figura 1.55.

Se especifica un retardo para cada tipo de puerta mediante objetos "Constant". También se especifican en la parte declarativa las cuatro señales internas del circuito (Figura 1.55). La parte correspondiente a la declaración "entity" no se modifica (Figura 1.3).

<b>Ejemplo</b>	<pre> <b>architecture</b> flujodatos <b>of</b> s1bit <b>is</b>   <b>constant</b> retardoxor: <b>time</b> := 15 ns;   <b>constant</b> retardoand: <b>time</b> := 10 ns;   <b>constant</b> retardoor: <b>time</b> := 15 ns;   <b>signal</b> xorxy, andxy, andxcen, andycen : std_logic; <b>begin</b>     xorxy &lt;= x <b>xor</b> y <b>after</b> retardoxor;     s &lt;= xorxy <b>xor</b> cen <b>after</b> retardoxor;     andxy &lt;= x <b>and</b> y <b>after</b> retardoand;     andxcen &lt;= x <b>and</b> cen <b>after</b> retardoand;     andycen &lt;= y <b>and</b> cen <b>after</b> retardoand;     csal &lt;= andxy <b>or</b> andxcen <b>or</b> andycen <b>after</b> retardoor; <b>end</b> flujodatos; </pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;">parte declarativa</div> <div style="border: 1px solid black; padding: 5px;">cuerpo</div>
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Figura 1.57** Cuerpo de la arquitectura para un sumador de 1 bit, especificando retardos y utilizando un modelo de flujo de datos.

## Simulación lógica o funcional

*Trabajo:* En el directorio LAB1 cree el directorio FLUJODATOS y, dentro de éste, cree un nuevo conjunto de directorios para efectuar un diseño. Seguidamente cree un proyecto utilizando Quartus. Utilice la especificación de la arquitectura de la Figura 1.57 como cuerpo de la arquitectura. Compruebe el funcionamiento lógico o funcional del del circuito. Para ello especifique señales periódica en los puertos de entrada cuando utilice el simulador Modelsim.

## Simulación temporal del circuito para medir el retardo

En una simulación con Modelsim se puede observar el retardo de propagación de las señales en las ventanas temporal y textual.

En el modelo de retardos que estamos utilizando, el tiempo de retardo de los cables es cero. El tiempo de retardo de las puertas es el establecido en el modelo VHDL.

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

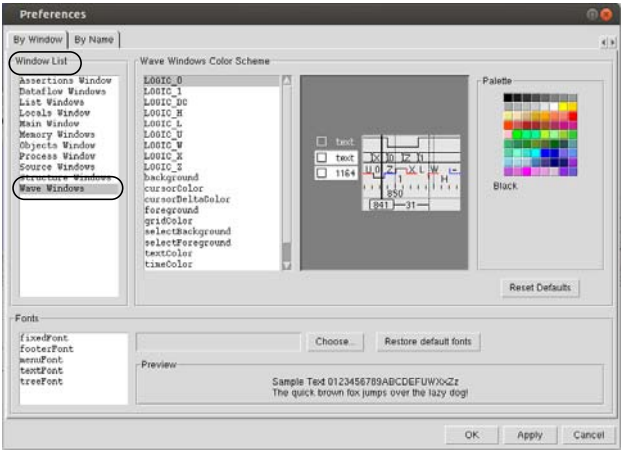
**Preparación del circuito.** Una forma de garantizar que, al modificar el valor de una señal de entrada, se observe el retardo de propagación de los elementos del circuito, es establecer previamente el valor indefinido en los cables del circuito.

*Nota: Una puerta AND-2 que tiene una entrada con el valor lógico cero no muestra un retardo de propagación si la otra entrada modifica su valor lógico. La situación es idéntica con una puerta OR-2 que tiene una de sus entradas con el valor lógico 1.*

Al iniciarse una simulación en Modelsim las entradas y salidas están indefinidas (símbolo U). Entonces, si se estimulan las entradas con valores se puede determinar el retardo del circuito para esas entradas.

Para observar las señales en la ventana temporal con un valor numérico o símbolo es necesario seleccionar la señal deseada (pueden ser varias) y pulsar el botón derecho del ratón. En la ventana emergente hay que seleccionar “Format -> Literal”<sup>14</sup>.

Por otro lado, teniendo en cuenta que las ventanas temporales, que se han mostrado previamente, no se observan bien cuando se imprimen, es necesario modificar los parámetros de configuración de la ventana temporal. Para ello hay que dar la orden “Tools -> Edit Preferences”. En la ventana emergente seleccione “Wave Windows” en el marco “Window List” Figura 1.58. Seleccione cada una de las entradas en el marco “Wave Windows Color Scheme” y establezca los valores que se muestran en la tabla de la Figura 1.58.



acónimo	color
LOGIC_0	Black
LOGIC_1	Black
LOGIC_DC	blue
LOGIC_H	grey90
LOGIC_L	grey90
LOGIC_U	red
LOGIC_W	red
LOGIC_X	red
LOGIC_Z	blue

acónimo	color
background	grey50
cursorColor	Black
cursorDeltaColor	Black
foreground	White
gridColor	grey50
selectBackground	White
selectForeground	Black
textColor	#000000
timeColor	#000000

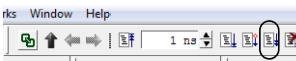
**Figura 1.58** Valores para configurar la ventana temporal.

*Trabajo: Establezca el formato literal para las señales, establezca la configuración para la ventana temporal y siga los pasos que se indican seguidamente.*

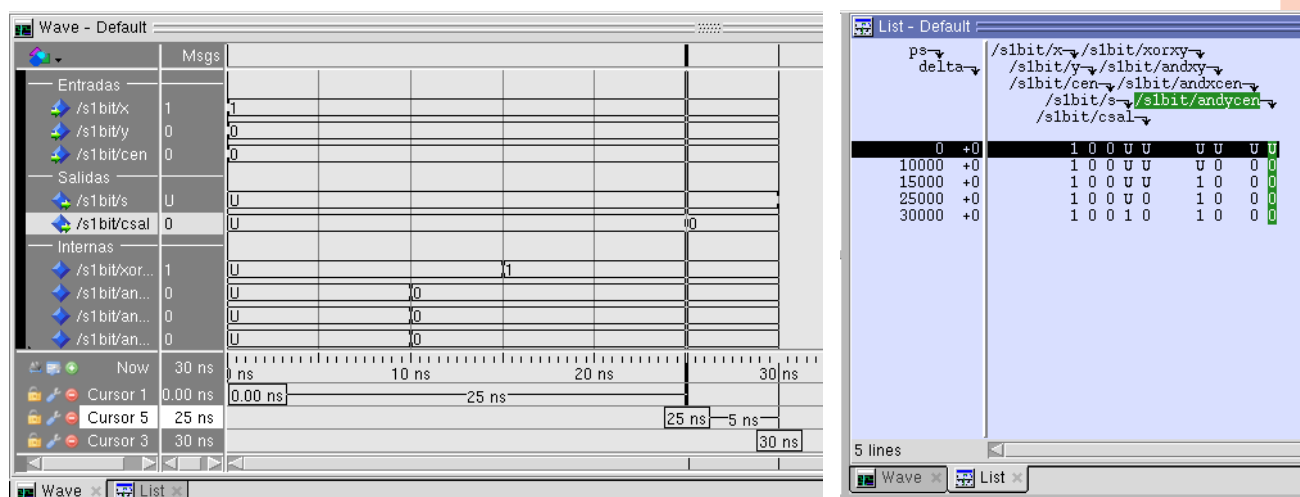
14. Otra posibilidad es pulsar dos veces en una variable en la ventana temporal y emerge una ventana donde hay que seleccionar la pestaña “Format”. En esta ventana se selecciona “Literal”.



En la Figura 1.59 se muestra el resultado de la simulación después de haber dado las siguientes órdenes al simulador Modelsim, mediante la interface gráfica.

Ordenes	Interface gráfica
force -freeze sim:s1bit/x 1 0	Wave -> Force
force -freeze sim:s1bit/y 0 0	Wave -> Force
force -freeze sim:s1bit/cen 0 0	Wave -> Force
run -all	

La simulación se suspende cuando las señales de entrada y salida permanecen estables. Ahora todos las señales del circuito tienen un valor lógico que depende de los valores lógicos de las entradas.



**Figura 1.59** Resultado de la simulación en un diseño con retardos.

Las entradas tienen los valores  $x = 1$ ,  $y = 0$ ,  $cen = 0$ . Las señales  $s$  y  $csal$  mantienen el valor indefinido hasta el instante 30 ns (Cursor 3 en la Figura 1.59) y 25 ns (Cursor 5 en la Figura 1.59) respectivamente, a partir del cual toman los valores  $s = 1$  y  $csal = 0$ . Entonces, el retardo son 30 ns.

En el tiempo  $t = 0$  ns se ha establecido el valor en las señales de entrada (1ª fila en la ventana textual). Se observa que las señales de salida están indefinidas. Después de 25 ns la señal de salida csal toma el valor 0 (4ª fila). Este retardo se corresponde con el retardo de un nivel de puertas AND (3 puertas) y una puerta OR. Después de 5 ns la señal de salida s toma el valor 1 (5ª fila). Este retardo se corresponde con el retardo de dos niveles de puertas XOR. A partir de este instante no se modifican las señales de salida.

---

*Nota: Una vez los valores de las señales de salida son estables, Modelsim suspende la simulación si no se modifica algún valor de una de las señales de entrada. Entonces, si se simula dos veces consecutivas con la misma entrada, la segunda vez se obtiene un retardo nulo. Si se modifica algún valor de las señales de entrada el retardo observado puede depender de los valores previos de las señales de entrada.*

---

## Utilización de la orden force

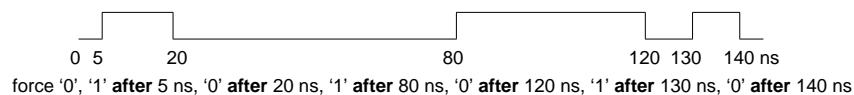
Una forma menos tediosa de comprobar los retardos es poder especificar, en una única orden, todos los posibles valores en un puerto de entrada. En esta sección se muestra cómo utilizar la orden “force” directamente en la ventana de órdenes y mensajes para establecer varios valores de excitación o estimulación en un puerto de entrada, en los instantes deseados (frente de onda).

### Ordenes

```
force [-freeze | -drive | -deposit] [-cancel <time>] [-repeat <time>] <item_name> <value> [<time>] [, <value> <time> ...]
```

Si no se especifica “[ -freeze | -drive | -deposit] ” el defecto es “-freeze”.

Por ejemplo, el frente de onda de la parte superior de la Figura 1.60 se obtiene mediante la orden “force” mostrada en la parte inferior de la misma figura. Notemos que los tiempos son todos relativos al primer elemento del frente de onda.



**Figura 1.60** Frente de onda especificado mediante una orden “force”.

Entre dos conjuntos de valores de entrada distintos hay que conseguir que las salidas del circuito queden indeterminadas o indefinidas. Entonces, entre dos conjuntos de valores de entrada se inserta un conjunto de valores que consiga indeterminar las salidas. La forma de especificar un valor indeterminado en un puerto de entrada de 1 bit es “U” (símbolo U).

Por otro lado, hay que determinar un retardo entre dos estimulaciones del circuito. Este retardo debe ser suficiente para que se propague la estimulación de las señales de entrada a las señales de salida. En caso contrario, si se cambian las entradas antes de que las salidas sean estables, las salidas del circuito no serán correctas. Analizando el circuito de la Figura 1.55 se determina que el retardo del circuito es 30 ns. En consecuencia un posible valor para el retardo entre estimulaciones son 50 ns.

En la tabla de la Figura 1.61 se muestran los frentes de onda para cada uno de los puertos de entrada. En una orden “force”, después de la señal, se especifican los pares <valor> <tiempo> separados por comas. Finalmente se da la orden “run-all”.

#### Ordenes

```
force s1bit/x U 0 ns, 0 50 ns, U 100 ns, 1 150 ns, U 200 ns, 0 250 ns, U 300 ns, 1 350 ns, U 400 ns, 0 450 ns, U 500 ns, 1 550 ns, U 600 ns, 0 650 ns, U 700 ns, 1 750 ns
```

```
force s1bit/y U 0 ns, 0 50 ns, U 100 ns, 0 150 ns, U 200 ns, 1 250 ns, U 300 ns, 1 350 ns, U 400 ns, 0 450 ns, U 500 ns, 0 550 ns, U 600 ns, 1 650 ns, U 700 ns, 1 750 ns
```

```
force s1bit/cen U 0 ns, 0 50 ns, U 100 ns, 0 150 ns, U 200 ns, 0 250 ns, U 300 ns, 0 350 ns, U 400 ns, 1 450 ns, U 500 ns, 1 550 ns, U 600 ns, 1 650 ns, U 700 ns, 1 750 ns
```

```
run -all
```

**Figura 1.61** Ordenes de estimulación para medir el retardo del circuito.

El conjunto de órdenes se puede almacenar en un fichero, denominado `estimulos.do` y posteriormente dar la orden `do estimulos.do` en la ventana de mensajes de Modelsim.

*Trabajo: Edite el fichero `estimulos.do` con las órdenes de la Figura 1.61 y almacénelo en el directorio PRUEBAS. Posteriormente, reinicie el simulador con la orden “restart” y estimule las entradas del circuito con la orden “do camino\_sistema\_ficheros/estimulos.do”.*

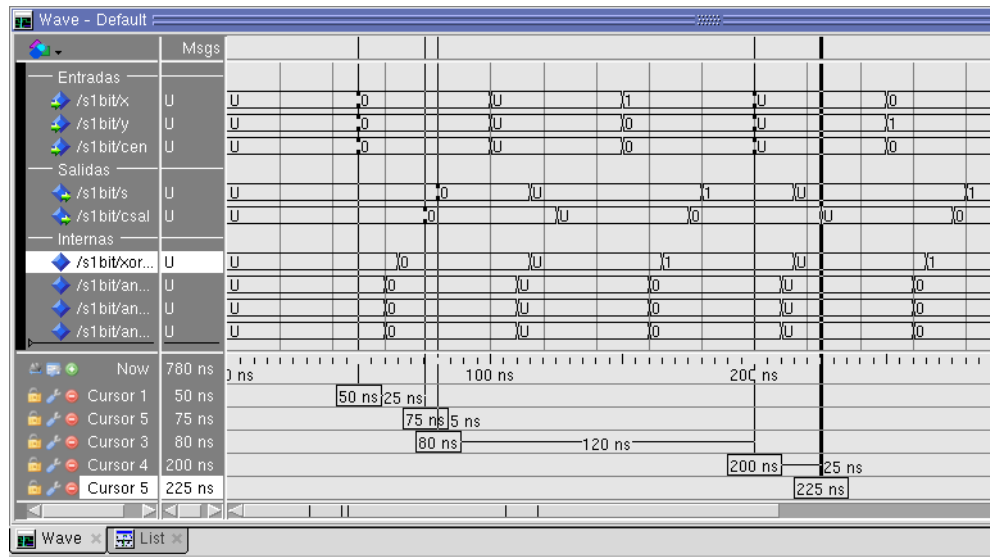
En la Figura 1.62 se muestra parte de la ventana textual. En la Figura 1.63 se muestra la traza de las señales para algunas combinaciones de entrada (ventana temporal).

		Entradas				Salidas		Internas			
		ps	/s1bit/x	s1bit/y	/s1bit/cen	/s1bit/s	/s1bit/csai	/s1bit/xorxy	/s1bit/andxy	/s1bit/andxcen	s1bit/andycen
estimulación	↑	0	U	U	U	U	U	U	U	U	U
		50000	0	0	0	U	U	U	U	U	U
		60000	0	0	0	U	U	U	0	0	0
		65000	0	0	0	U	U	0	0	0	0
		75000	0	0	0	U	0	0	0	0	0
finalización de la propagación	↓	80000	0	0	0	0	0	0	0	0	0
		100000	U	U	U	0	0	0	0	0	0
estímulos para indeterminar las salidas	↑	110000	U	U	U	0	0	0	U	U	U
		115000	U	U	U	U	0	U	U	U	U
		125000	U	U	U	U	U	U	U	U	U
30 ns	↑	150000	1	0	0	U	U	U	U	U	U
		160000	1	0	0	U	U	U	0	0	0
		165000	1	0	0	U	U	1	0	0	0
		175000	1	0	0	U	0	1	0	0	0
		180000	1	0	0	1	0	1	0	0	0
		200000	U	U	U	1	0	1	0	0	0

**Figura 1.62** Parte de la ventana textual.

El primer vector de comprobación estimula las entradas en  $t = 50$  ns (cursor 1 en la Figura 1.63). Las salidas se estabilizan en  $t = 80$  ns (cursor 3 en la Figura 1.63). Entonces el retardo son 30 ns. El vector de comprobación (1, 0, 0) estimula las entradas en  $t = 150$  ns y el retardo para estabilizar las salidas son también 30 ns. Observemos como entre

vectores de comprobación de interés se inyectan estímulos para indefinir las señales en el circuito. Por ejemplo, en  $t = 200$  ns el vector de estímulos es (U, U, U) y las salidas quedan indefinidas después de 25 ns (Figura 1.63).



**Figura 1.63** Parte de la ventana temporal.

