GRUPO DE DISEÑO
HARDWARE SOFTWARE

DATCCCIA – ETSII– URJC

Introducción a ModelSim
Sumador binario de n bits

Resumen:

El presente informe presenta una introducción sencilla paso a paso del simulador VHDL ModelSim mediante la construcción de un sumador binario parametrizable de n bits materializado como una red iterativa de sumadores binarios de 1 bit en cascada – *ripple carry*.

José Ignacio Martínez Torre
Pablo Huerta Pellitero

Universidad Rey Juan Carlos

DOCENCIA - VHDL

Introducción a ModelSim

© **Grupo de diseño Hardware Software – DATCCIA – ESCET – URJC**
C/ Tulipán s/n, 28933, Móstoles, Madrid, ESPAÑA

Tabla de contenidos

Capítulo

1

Objetivo

El objetivo de este manual es presentar a grandes rasgos el entorno de simulación de circuitos lógicos digitales descritos mediante un lenguaje de descripción de hardware como VHDL.

En la Figura 1 se muestra el diagrama general de flujo que se deberá llevar a cabo para comprobar el correcto funcionamiento de cualquier diseño.

- En primer lugar, se realizará un estudio analítico a mano del circuito propuesto para conocer su funcionamiento y tener una idea certera de qué debemos esperar como resultado de la aplicación de las herramientas informáticas. Este estudio se deberá realizar antes de utilizar dichas herramientas y es parte fundamental para la correcta asimilación de los conocimientos que se imparten.

- A continuación, se hará **un estudio asistido por ordenador utilizando el paquete informático ModelSim®** de Mentor Graphics inc. Este paquete informático está compuesto por un conjunto de aplicaciones que permiten diseñar circuitos digitales en VHDL y simular y analizar su comportamiento de una forma sencilla dentro de un entorno integrado para PC. La secuencia habitual de pasos se lleva a cabo del siguiente modo:

    - Se introduce el diseño que se pretende simular mediante la edición del código VHDL, bien a través del editor de la herramienta (**Edit Window**) o bien a través de otro editor convencional. Además, se debe crear un entorno de trabajo para gestionar las bibliotecas utilizadas y creadas (**Library Browser**).

    - Se compila el diseño de forma ordenada jerárquicamente desde el nivel más bajo al nivel más alto. ModelSim detecta de forma automática el orden de compilación de los distintos archivos.

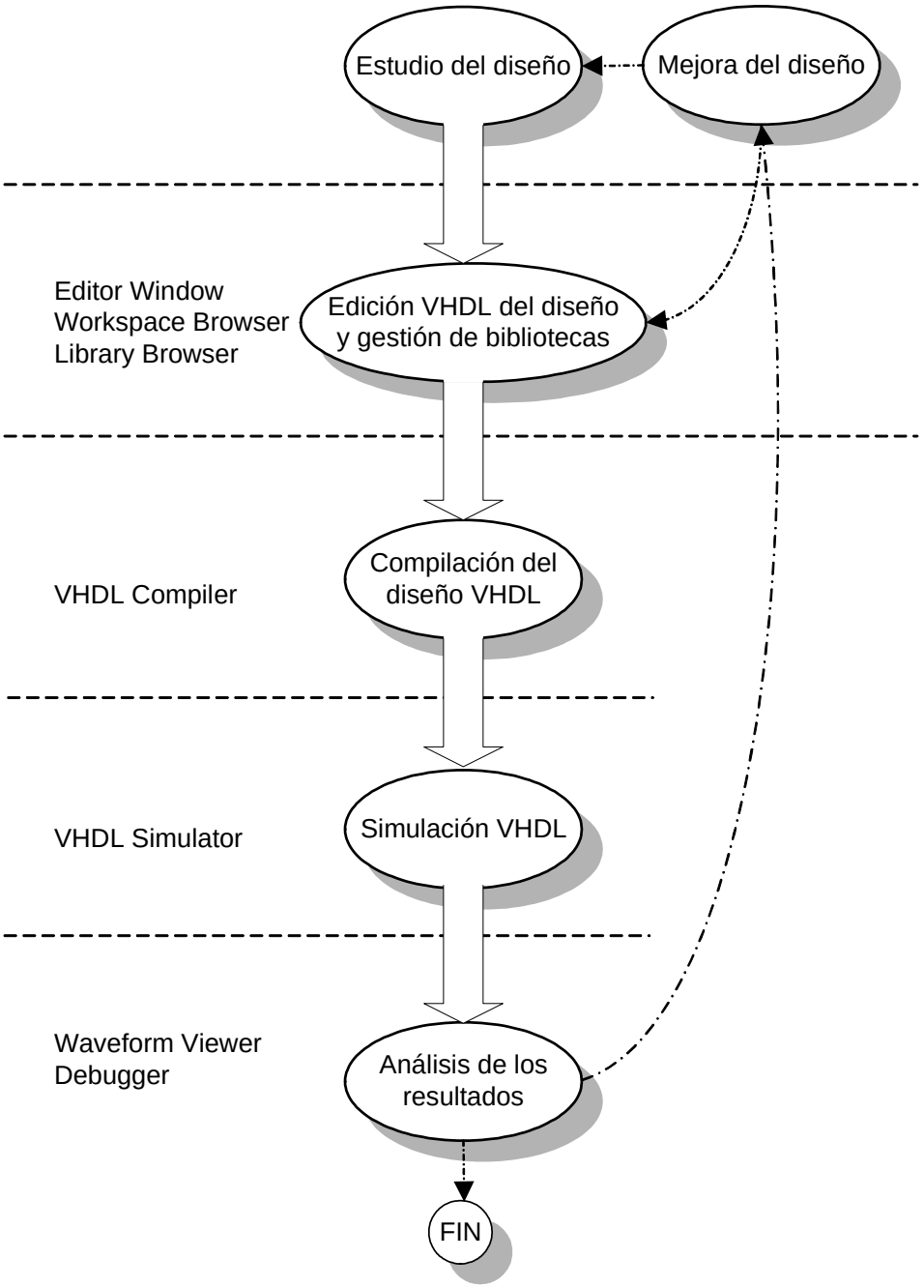


Figura 1: Método de trabajo.

- Se simula el diseño mediante el simulador.

- Se comprueba el correcto o incorrecto funcionamiento del diseño, analizando las formas de onda de las señales necesarias mediante el visualizador (**Waveform Viever)** y/o se depura el código utilizando el depurador. En caso de que los resultados no sean los esperados, se deberá mejorar el diseño, normalmente comenzando con el estudio del propio diseño o re-escribiendo parte o todo el código VHDL.

La explicación de los pasos básicos se realiza materializando una práctica introductoria sencilla en el Capítulo 2, de modo que se siga el flujo de diseño de la Figura 1. En [1,2,3,4,5] se hace referencia a bibliografía que aporta información complementaria sobre este paquete informático, el lenguaje VHDL y diseño lógico.

Capítulo

2

Práctica guiada

La práctica se orienta hacia el aprendizaje del entorno de simulación de ModelSim aplicándolo a la descripción en el dominio estructural y funcional de un sumador binario parametrizable de n bits materializado como una red iterativa de sumadores binarios de 1 bit en cascada – *ripple carry*.

El sumador binario parametrizable de n bits se va a describir utilizando una metodología de arriba-abajo (top-down) para su concepción y una metodología de abajo-arriba (*bottom-up*) para su descripción VHDL.

Concepción del sumador de arriba-abajo

El sumador binario parametrizable de n bits (Figura 2) es un sistema con dos entradas vectoriales de datos de n bits (a y b), una entrada de acarreo de 1 bit (c_in), una salida vectorial de resultados de n bits (s) y una salida de acarreo de 1 bit (c_out).

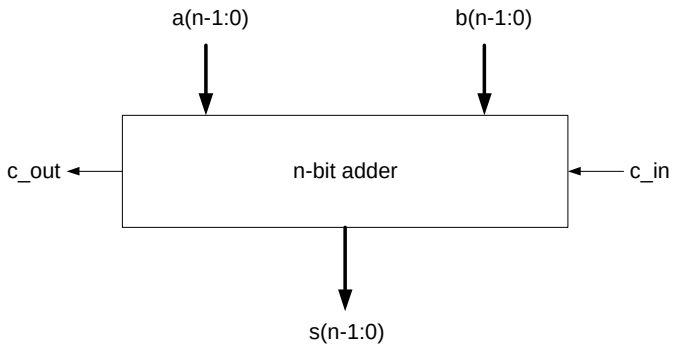


Figura 2: sumador binario parametrizable de n bits.

Este sumador binario parametrizable de n bits será descrito en los dominios conductual y estructural. En ambos dominios el sumador se puede describir y realizar de distintas maneras. Para el dominio estructural, el sumador va a ser descrito como una red iterativa en cascada o ripple-carry.

En Figura 3 se muestra la estructura de bloques e interconexiones del sumador.

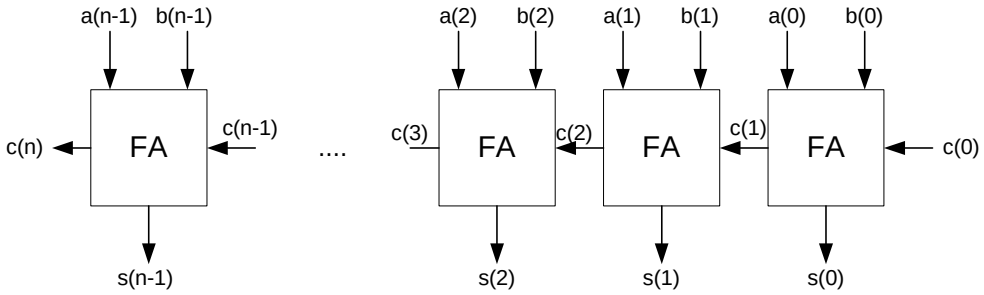
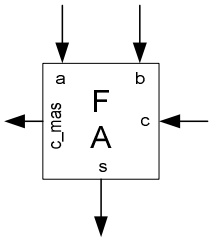


Figura 3: sumador binario en cascada (*ripple-carry*).

Los componentes de la red iterativa son sumadores binarios completos de 1 bit (FA, *Full Adder*) y son idénticos entre sí. De nuevo estos sumadores de 1 bit pueden ser descritos en distintos dominios.

En Figura 4 se muestra una descripción estructural de un sumador binario completo de 1 bit.



$$s = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} + a \cdot b \cdot c + a \cdot \bar{b} \cdot \bar{c}$$

$$c_{mas} = ab + ac + bc$$

Figura 4: descripción estructural del FA.

.

Descripción VHDL del sumador de abajo-arriba

En el apéndice A se incluye el código VHDL completo de los distintos diseños desde el sumador básico de un bit, hasta la verificación (*test*) del sumador completo parametrizable.

El objetivo ahora es aprender a utilizar el entorno de ModelSim realizando el diseño del sumador binario parametrizable de n bits.

Edición VHDL y gestión de espacios de diseño

En primer lugar, se arranca la aplicación ModelSim XE desde el menú de Inicio de Windows, como se indica en la Figura 5.

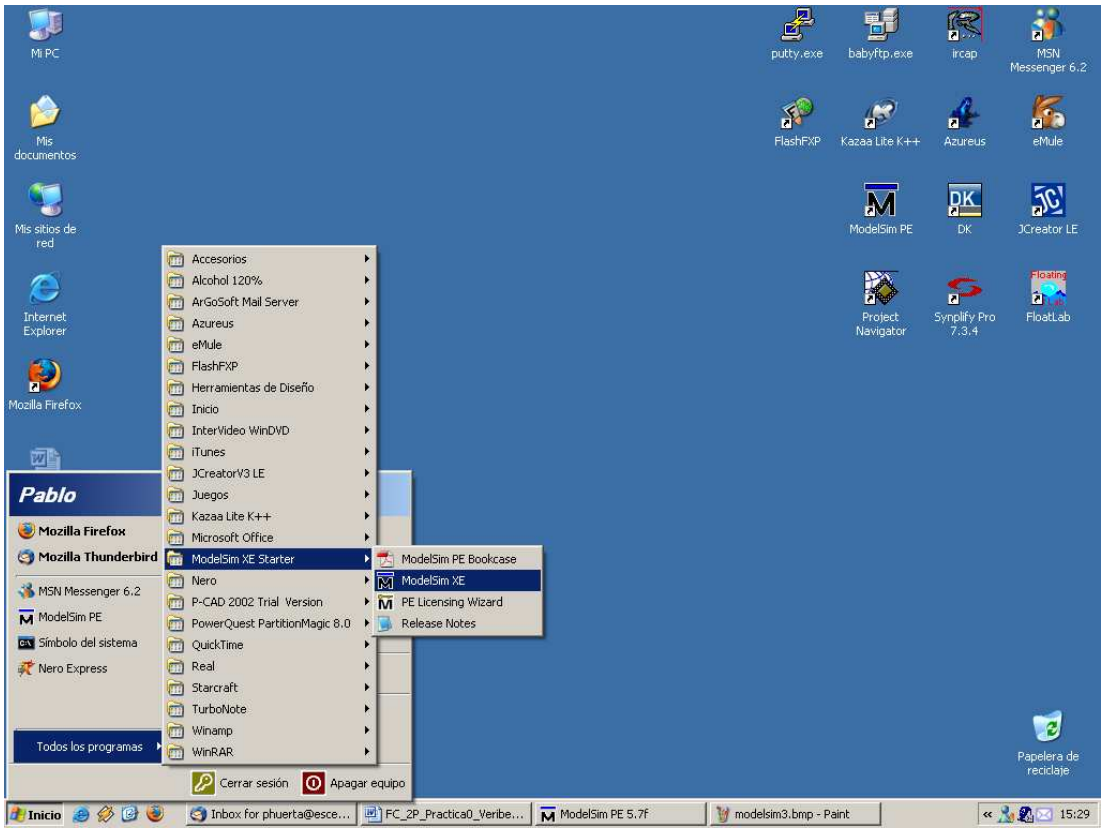


Figura 5: Ventana de inicio de ModelSim

Se comienza creando un Proyecto de Trabajo (Menu *File->Project->New*). Se introduce el nombre del proyecto y el lugar donde se quiere guardar, y se pulsa OK. (Figura 6).

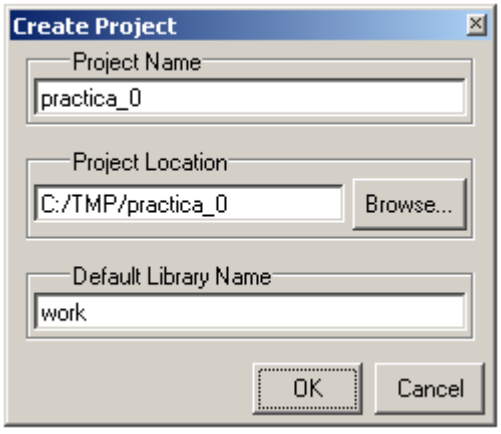


Figura 6: Creación del Proyecto.

Una vez hecho esto, aparece una ventana que nos permite crear un nuevo archivo, añadir al proyecto archivos existentes, etc. (Figura 7).

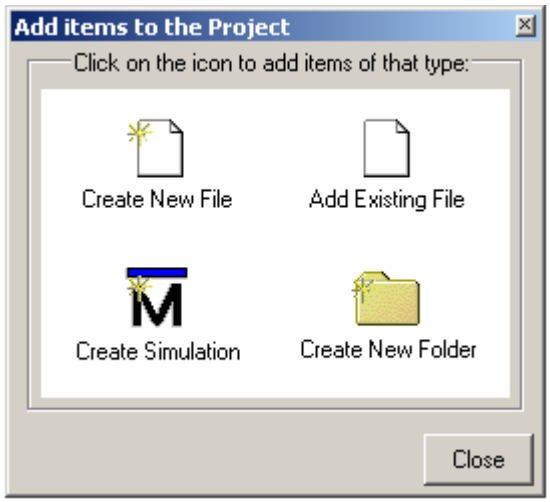


Figura 7: Añadir Archivos.

El siguiente paso consiste en crear un archivo de VHDL para introducir un diseño. Para ello, se pulsa dos veces en *Create New File* en la ventana anterior, o si la hemos cerrado podemos hacerlo desde el menú *File->Add to Project->New File*. Nos aparecerá una ventana donde debemos introducir el nombre del archivo, y el lenguaje que se va a usar, en nuestro caso VHDL. (Figura 8).

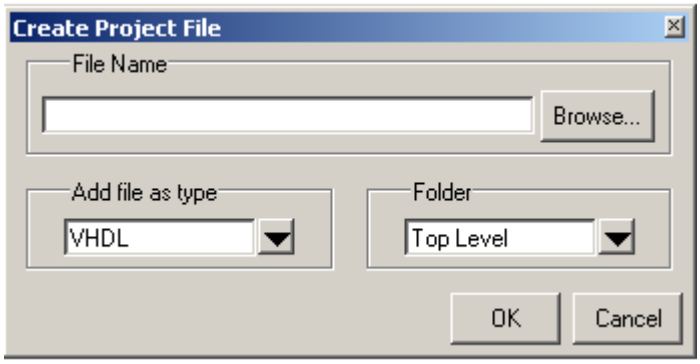


Figura 8: Creación de un archivo VHDL.

Aparecerá el archivo creado en la parte derecha de la pantalla (o en la izquierda, depende de la configuración del ModelSim), y haciendo doble click sobre el, se nos abrirá el editor de texto en el que podremos teclear directamente el código VHDL deseado (Figura 9).

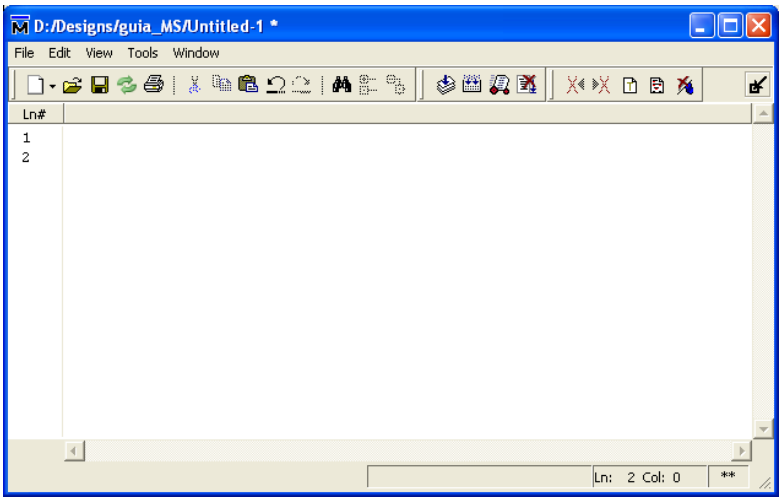


Figura 9: Hoja en blanco del editor de VHDL.

En un proceso de diseño completo normal, deberíamos escribir aquí el código VHDL; sin embargo para este ejemplo ya disponemos del código VHDL que se debe copiar del apéndice A de este documento. Se puede colocar cada unidad compilable en un archivo distinto, mezcladas en varios archivos o todas juntas en un único archivo. En este caso se ha decidido tener varios archivos donde cada uno contenga todas las unidades compilables relacionadas con una entidad. Si se opta por poner todo el código en un solo archivo, hay que tener en cuenta que para cada módulo que se describa (una entidad y una o varias arquitecturas) es necesario incluir las librerías que se van a usar para ese módulo usando las sentencias *library* y *use*. Todo el proceso de escritura se puede hacer directamente dentro de ModelSim o primero en otro editor y después añadir al proyecto los archivos .vhd. Los archivos que se propone crear a partir del código que se encuentra en el Apéndice A son:

- add_1.vhd
- add_1_test.vhd
- add_8.vhd
- add_8_test.vhd
- add_n.vhd
- add_n_test.vhd

Nota: A veces al crear un nuevo archivo desde el editor integrado en ModelSim, no lo reconoce como archivo fuente VHDL (se tiene que ver en la columna *Type* que el tipo de archivo es VHDL). Si sucede esto, hay que eliminar el archivo del proyecto (click botón derecho sobre el archivo y seleccionar *Remove from Project*) y posteriormente añadirlo de nuevo desde *File-> Add to Project-> Existing File*.

Compilación del diseño vhdl

Una vez hemos introducido en el espacio de trabajo todos los archivos VHDL, éstos se deben compilar siguiendo un orden lógico. Es decir que se han de compilar primero aquellas unidades de diseño que son elementos constituyentes de otras. Para cambiar el orden en que se compila cada archivo, basta con ir al menú *Compile->Compile Order*. Desde ahí se puede cambiar el orden en el que se compilan los archivos, o dejar que la herramienta los ordene automáticamente pulsando en el botón *Auto Generate*.

Una vez preparados para hacer la compilación, la llevamos a cabo pulsando *Compile All* en el menú *Compile* o su icono correspondiente  , lo que dará lugar a la aparición de mensajes en la ventana de salida del entorno (Figura 10). Estos mensajes nos indicarán si la compilación ha transcurrido con normalidad o, si por el contrario han ocurrido errores.

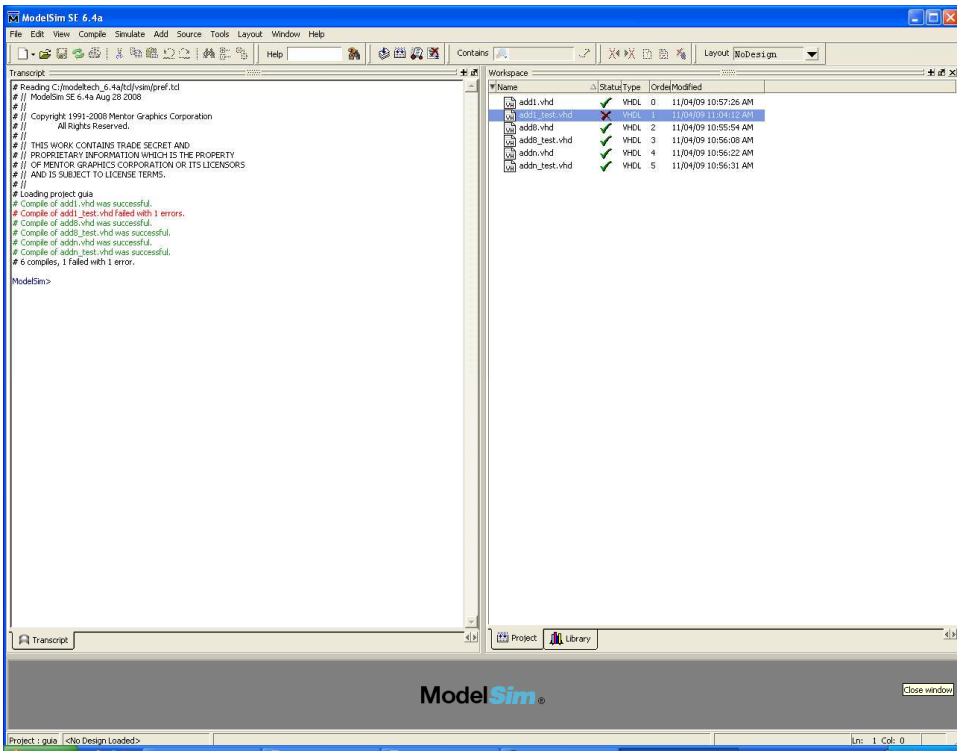


Figura 10: Resultados de la compilación.

Si todo ha salido correctamente, se puede pasar ya al siguiente paso, que sería la simulación. Si ha habido errores, estarán resaltados en rojo. Haciendo doble clic sobre el mensaje de error, se abrirá una ventana con más información acerca del error.

Antes de continuar, echemos un vistazo al esquema de bibliotecas utilizado el Library Browser. Si seleccionamos la pestaña Library en la parte inferior derecha de la ventana, veremos lo siguiente (Figura 11).

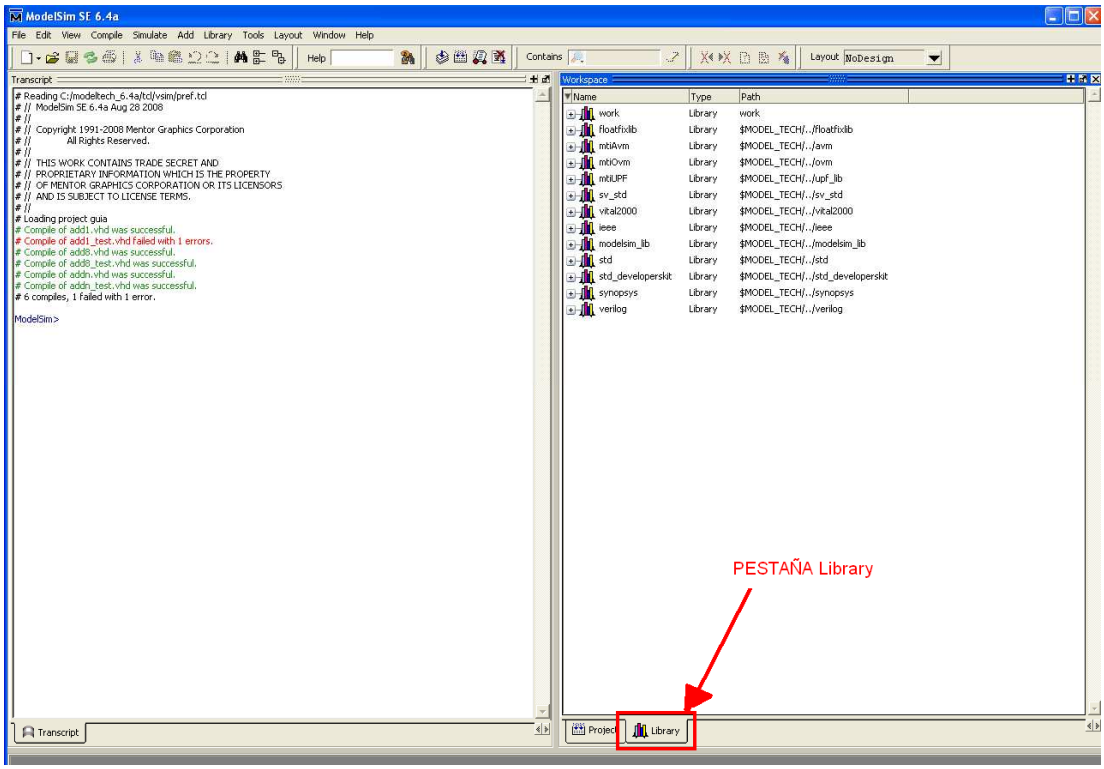


Figura 11: Library Browser.

Las bibliotecas VHDL consisten en un nombre lógico y un camino físico a la propia biblioteca, que se pueden ver si hacemos click con el botón derecho y seleccionamos Propiedades. Las unidades de diseño se compilan por defecto en la biblioteca WORK, que puede ser abierta para mostrar todas las unidades existentes.

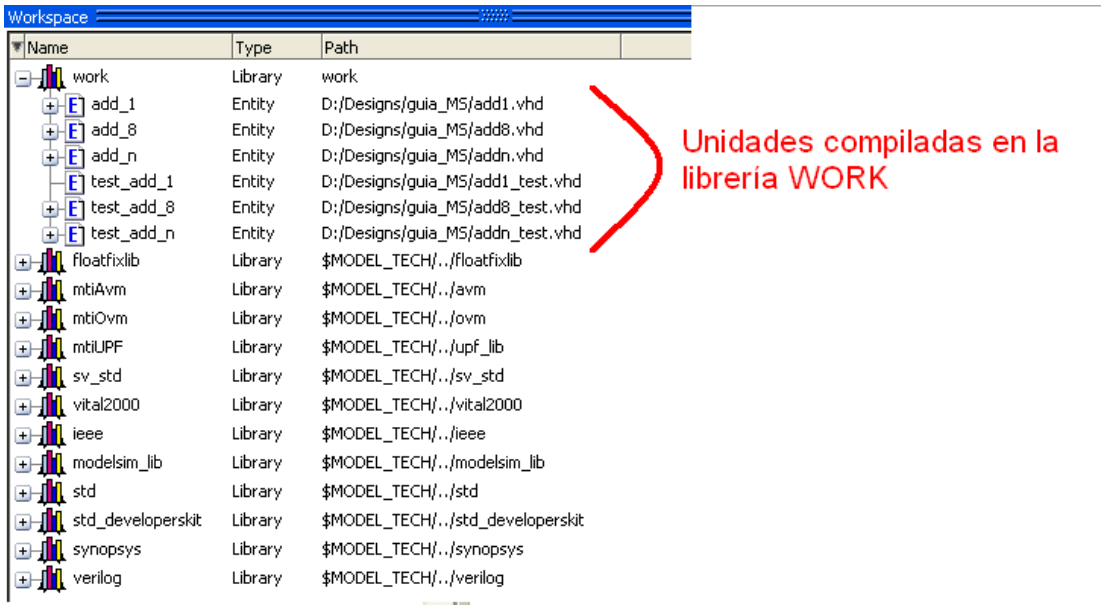


Figura 12: unidades compiladas en la librería work

Simulación vhdl y análisis de resultados

El siguiente paso lógico consiste en especificar la unidad de diseño de más alto nivel que va a ser simulada, normalmente denominada test. Para ello ya hemos creado previamente un archivo de test, cuyo código hemos copiado del Apéndice de este documento. En este ejemplo vamos a simular sólo el sumador de un bit.

Para comenzar la simulación nos iremos a la pestaña Library. Expandimos la librería *work*. Dentro de la librería *work* seleccionamos el fichero que queremos simular, en este caso test_add_1, y pulsando con el botón derecho sobre el elegimos *Simulate*. En la mitad derecha de la pantalla se nos abrirá una nueva pestaña, con los módulos simulables (Figura 13).

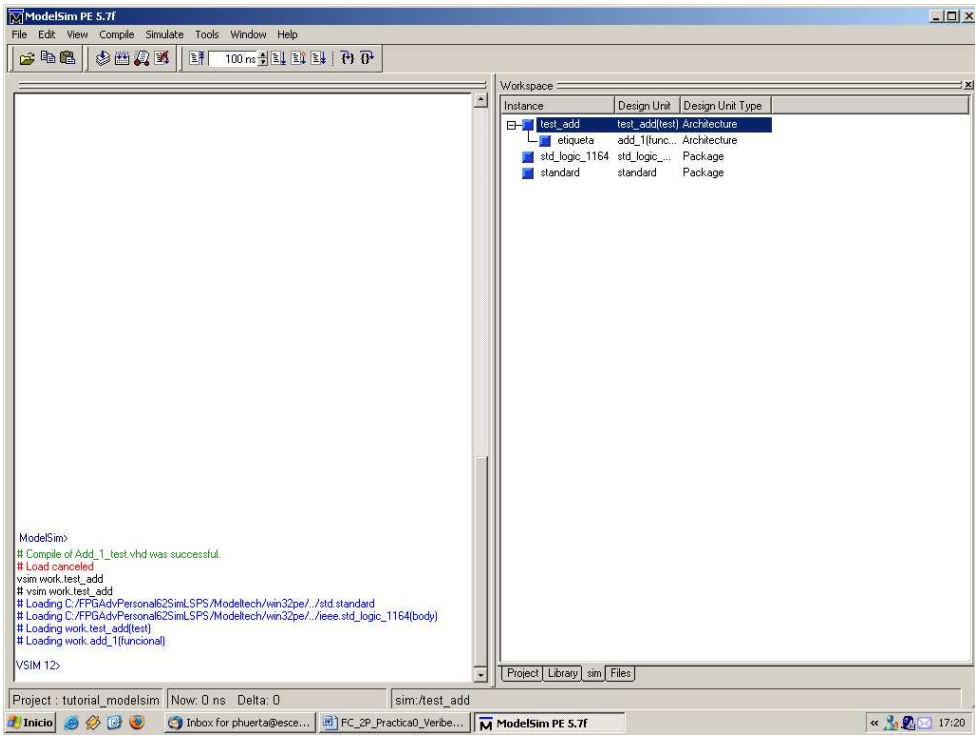


Figura 13: Pestaña de Simulación.

Ahora solo tenemos que hacer click con el botón derecho sobre test_add_1 y escoger *Add->Add to Wave* (Figura 14).

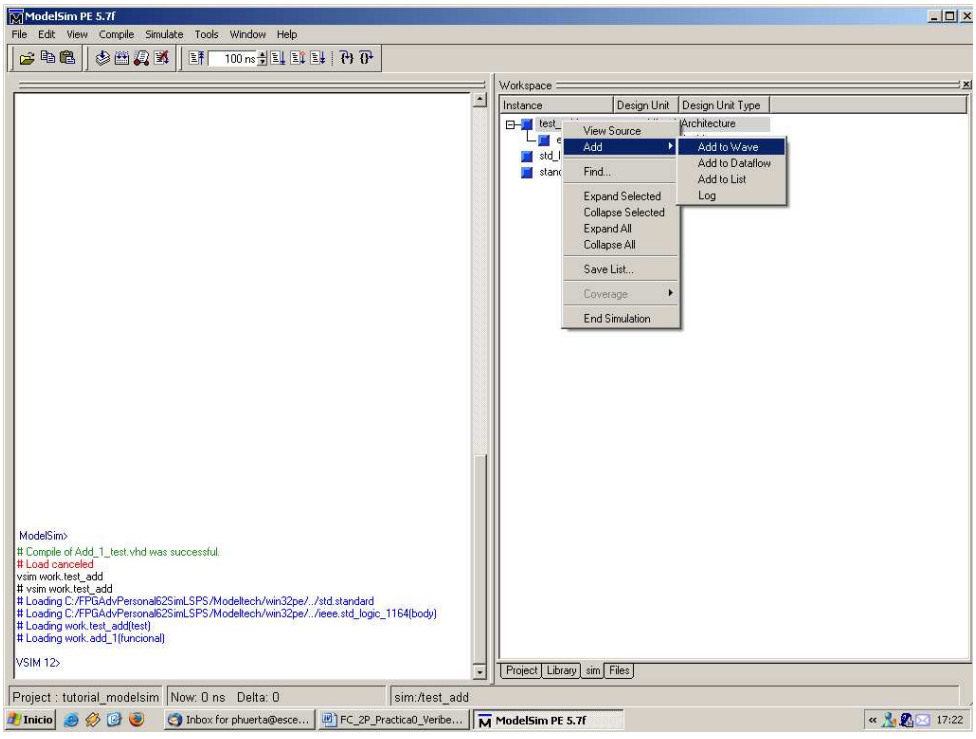


Figura 134: Adición de señales a la ventana de visualización.

Una vez hecho esto, se nos abrirá la ventana del visor de ondas (Wave) con todas las señales visibles desde el testbench. La forma de añadir señales individualmente y de niveles más bajos de la jerarquía de módulos se verá más adelante.

Desde la ventana del visor de ondas podremos ver los resultados de la simulación. Pulsando sobre el icono  iniciaremos la simulación y veremos como van apareciendo los resultados. Cada pulsación sobre este icono hace que avance el tiempo de la simulación 100 ns, así que podemos pulsarlo varias veces si es necesario. En la siguiente figura se ve el resultado de la simulación.

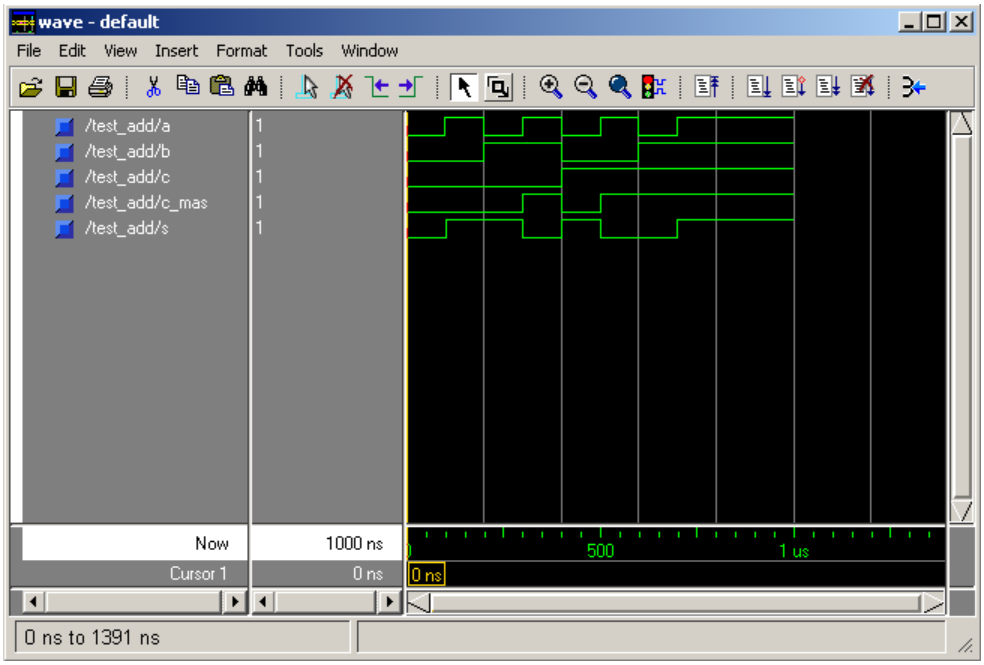


Figura15: Resultado de la simulación.

A partir de aquí existen la serie habitual de posibilidades para realizar ampliaciones o reducciones que recojan toda la información en la pantalla (Zoom to Fit), Zooms parciales (Scale factor), visualización de valores, selección del tipo de información a mostrar (valores binarios, hexadecimales, buses, etc.), medida de tiempo entre transiciones de señales, adición de cursores, etc.

Utilizando estas posibilidades es posible analizar el resultado de la simulación sobre el diseño y comprobar si su funcionamiento es correcto o no. Dependiendo de esto último podremos finalizar el trabajo (Save y Exit) o revisar el diseño, bien desde el principio o desde el propio sistema.

Para esto último es muy útil otra de las herramientas adicionales que existen y que es el Depurador. Su formato es idéntico al de cualquier depurador de cualquier lenguaje de programación, luego no se hace hincapié en ella.

Cuando se simula un circuito utilizando ModelSim es posible visualizar cualquier señal de cualquier módulo interno del circuito. Para visualizar una señal interna del circuito, se deben seguir los siguientes pasos:

1. Seleccionar en la pestaña "S*im*" el módulo del cual queremos ver una señal interna. Para ello se puede navegar en la jerarquía de módulos desplegándolos con un click en el símbolo ⊞
2. Una vez seleccionado el módulo, se debe abrir la ventana de selección de señales: *View->Debug Windows -> Signals*
3. Desde esta ventana seleccionamos las señales que queremos añadir al visor de ondas y con click en el botón derecho se selecciona "*Add to Wave*"

Para poner en práctica este concepto, simularemos el sumador de 8 bits, y seleccionaremos algunas señales de módulos internos para ser visualizadas.

Para comenzar esta nueva simulación, se debe finalizar antes la simulación anterior, desde el menú *Simulate -> End Simulation*.

Ahora ya se puede iniciar la simulación del sumador de 8 bits. Para ello, al igual que con el ejemplo anterior del sumador de 1 bit, se debe buscar en la pestaña *Library* dentro de la librería *Work* el test del sumador de 8 bits, como se muestra en la figura 16.

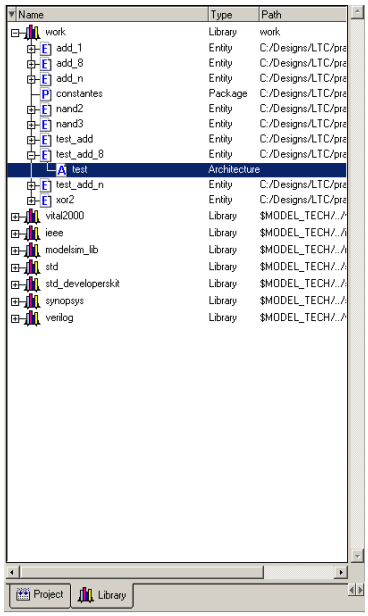


Figura 16: test sumador 8 bits

Haciendo doble clic sobre el test, se arranca la simulación. Desde la pestaña *Sim* podemos añadir como antes las señales del test al visor de ondas, haciendo click con el botón derecho sobre *test_add_8* y seleccionando *Add->Add to Wave*, como se muestra en la siguiente figura:

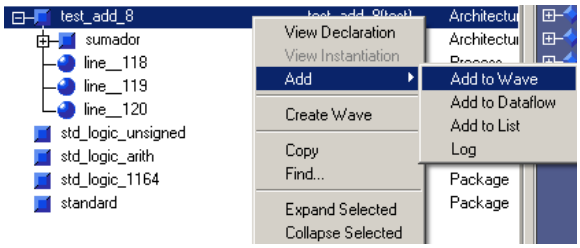


Figura 17: añadir señales al visor de ondas

Para poder añadir señales de módulos más internos, debemos expandir el módulo *sumador*, haciendo click en ⊞. Dentro de sumador hay más módulos, que también se pueden expandir. En la siguiente figura, se muestran varios módulos expandidos, hasta llegar al sumador 0 (el que suma los bits menos significativos):

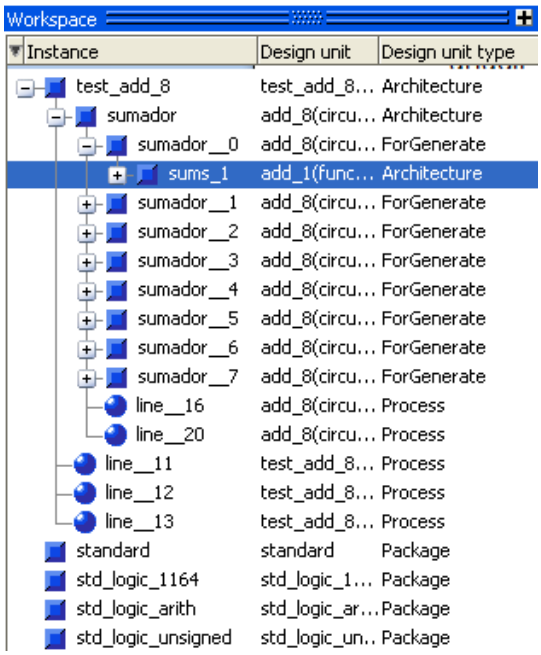


Figura 18: exploración de los diferentes submódulos

Una vez seleccionado un submódulo, se pueden añadir sus señales internas a la ventana del visor de ondas. Para ello, se utiliza el menú *View->Debug Window->Signals*. Desde la nueva ventana que se abre, se puede añadir cualquier señal del módulo al visor de ondas

mediante un click con el botón derecho y seleccionando *Add to Wave->Selected Signals*. En la siguiente figura se muestra como se añadiría la señal *a* del sumador 0:

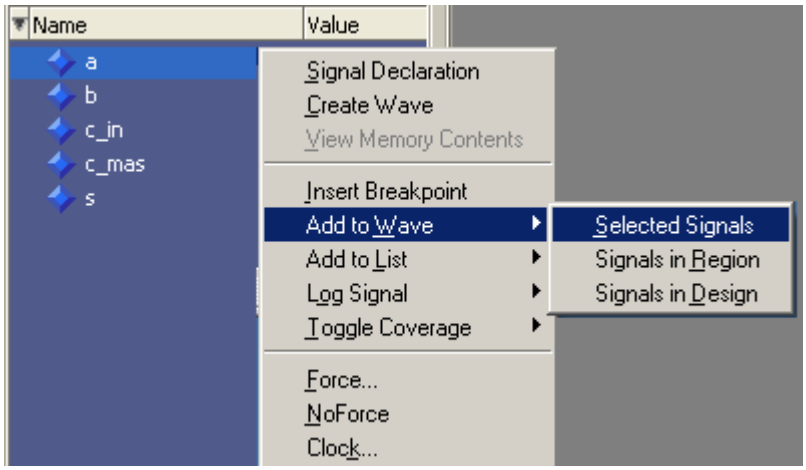


Figura 19: Añadiendo una señal al visor de ondas

Una vez añadidas las señales que se quieren visualizar, ya se puede iniciar la simulación desde la ventana de visualización de ondas, haciendo que avance el tiempo de la simulación:

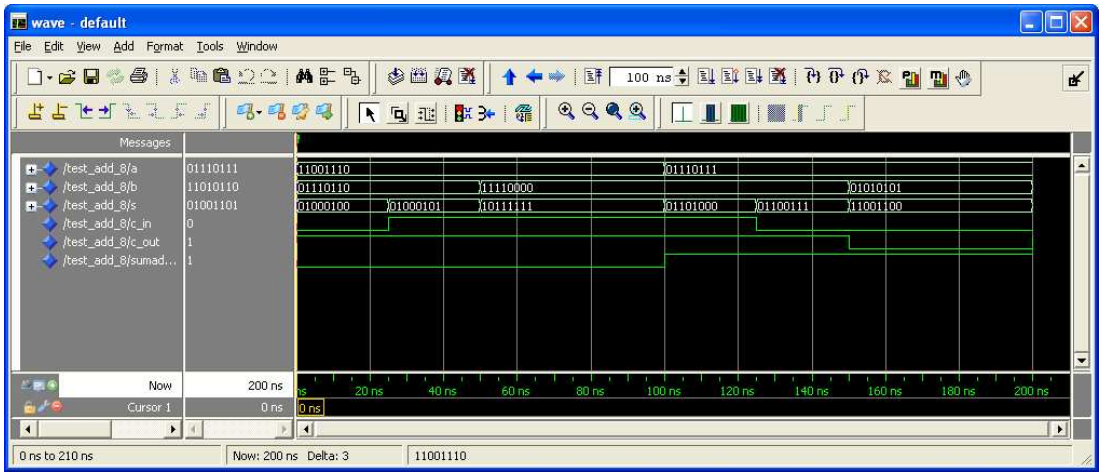


Figura 20: simulación del sumador de 8 bits

Como se ha visto hasta ahora, en la ventana del visor de ondas, los valores de las señales aparecen por defecto en binario. Para algunos tests puede ser más fácil analizar los resultados visualizando la información en otro formato. Para este ejemplo del sumador, sería interesante poder ver los valores de los números a sumar, así como el resultado de la suma en un formato más cómodo. Para cambiar el formato de representación de una señal, basta con seleccionarla y con el botón derecho seleccionar *Radix->* y el formato que se desee. En

la siguiente figura, se muestra como se visualizarían los resultados habiendo puesto las señales *a, b y s* en formato *unsigned*:

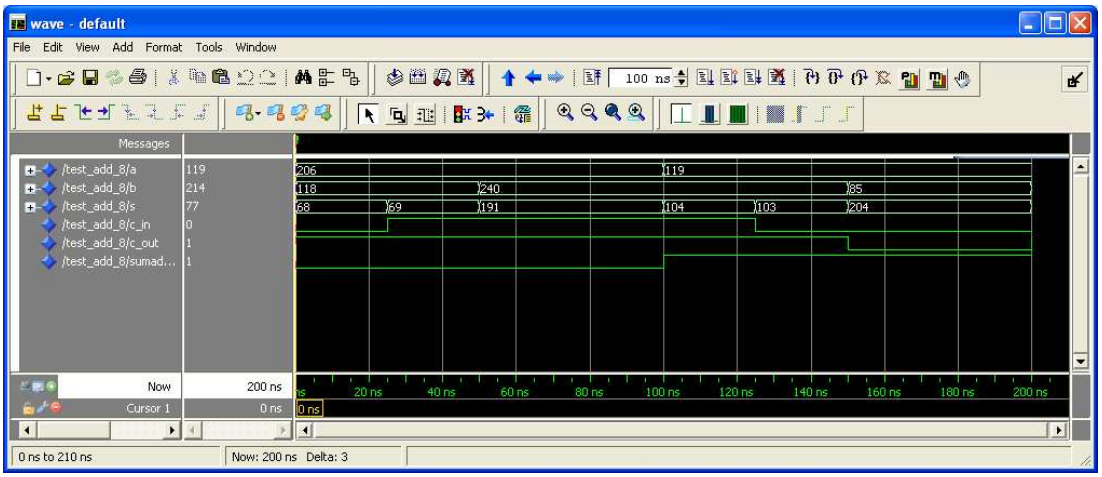


Figura 21: representación de los resultados en formato unsigned

Con esto queda concluida, esta introducción sobre el sistema de simulación VHDL de ModelSim.

Apéndice

A

Código VHDL del sumador binario de n bits

A continuación se da el código VHDL de los componentes y sistemas que se deben utilizar.

Add_1:

```
library ieee;
use ieee.std_logic_1164.all;

entity add_1 is
	port (a, b, c_in: in std_logic; c_mas, s: out std_logic);
end add_1;

architecture funcional of add_1 is
begin
	s <= ((not a) and (not b) and c_in) or ((not a) and b and (not c_in)) or (a
and b and c_in) or (a and (not b) and (not c_in));
	c_mas <= (a and b) or (a and c_in) or (b and c_in) ;
end funcional;
```

Add_1_test:

```
library ieee;
use ieee.std_logic_1164.all;

entity test_add_1 is end test_add_1;

architecture test of test_add_1 is
	signal a: std_logic := '0';
	signal b: std_logic := '0';
	signal c_in: std_logic := '0';
	signal c_mas, s: std_logic;
begin
	inst_1: entity work.add_1 port map(a, b, c_in, c_mas, s);
	a <= not a after 10 ns;
	b <= not b after 20 ns;
	c_in <= not c_in after 40 ns;
end test;
```

Add_8:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add_8 is
	port (a, b: in std_logic_vector(7 downto 0);
		c_in: in std_logic;
		s: out std_logic_vector(7 downto 0);
		c_out: out std_logic);
end add_8;

architecture circuito of add_8 is
	signal c: std_logic_vector(8 downto 0);
begin
	c(0) <= c_in;
	sumador: for i in 0 to 7 generate
	sums_1: entity work.add_1 port map (a(i), b(i), c(i), c(i+1), s(i));
	end generate;
	c_out <= c(8);
end circuito;

architecture funcional of add_8 is
	signal long_a, long_b, long_carry, long_s: std_logic_vector(8 downto 0);
begin
	long_a <= '0'&a; long_b <= '0'&b;
	long_carry <= "00000000"&c_in;
	long_s <= long_a + long_b + long_carry;
	s <= long_s(7 downto 0);
	c_out <= long_s(8);
end funcional;
```

Add_8_test:

```
library ieee;
use ieee.std_logic_1164.all;

entity test_add_8 is end test_add_8;

architecture test of test_add_8 is
	signal a, b,s: std_logic_vector(7 downto 0);
	signal c_in, c_out: std_logic;
begin
	sumador: entity work.add_8(circuito) port map (a, b, c_in, s, c_out);
	c_in <= '0', '1' after 25 ns, '0' after 125 ns, '1' after 225 ns;
	a <= "11001110", "01110111" after 100 ns;
	b <= "01110110", "11110000" after 50 ns, "01010101" after 150 ns, "11010110"
after 200 ns;
end test;
```

Add_n:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add_n is
	generic(n: integer);
	port (a, b: in std_logic_vector(n-1 downto 0);
		c_in: in std_logic;
		s: out std_logic_vector(n-1 downto 0);
		c_out: out std_logic);
end add_n;
```

```
architecture circuito of add_n is
	signal c: std_logic_vector(n downto 0);
begin
	c(0) <= c_in;
	sumador: for i in 0 to n-1 generate
	puertas: entity work.add_1 port map (a(i), b(i), c(i), c(i+1), s(i));
	end generate;
	c_out <= c(n);
end circuito;

architecture funcional of add_n is
	signal long_a, long_b, long_carry, long_s: std_logic_vector(n downto 0);
begin
	long_a <= '0'&a; long_b <= '0'&b;
	long_carry <= conv_std_logic_vector(0,n)&c_in;
	long_s <= long_a + long_b + long_carry;
	s <= long_s(n-1 downto 0);
	c_out <= long_s(n);
end funcional;
```

Add_n_test:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity test_add_n is end test_add_n;

architecture test of test_add_n is
	constant ancho_add: integer := 8;
	signal a, b, s: std_logic_vector(ancho_add-1 downto 0);
	signal c_in, c_out: std_logic;
begin
	sumador: entity work.add_n(circuito) generic map(ancho_add) port map (a, b,
c_in, s, c_out);
	c_in <= '0', '1' after 25 ns, '0' after 125 ns, '1' after 225 ns;
	a <= conv_std_logic_vector(15, ancho_add), conv_std_logic_vector(78,
ancho_add) after 100 ns, conv_std_logic_vector(178, ancho_add) after 200 ns;
	b <= conv_std_logic_vector(21, ancho_add), conv_std_logic_vector(78,
ancho_add) after 50 ns, conv_std_logic_vector(17, ancho_add) after 100 ns,
conv_std_logic_vector(98, ancho_add) after 150 ns;

end test;
```

Bibliografía

[1] Ayuda *on-line* de ModelSim.

[2] J-P. Deschamps. *Síntesis de Circuitos Digitales*. 1ª edición. Thomson. 2002.

[3] Z. NAVABI, VHDL, “Analysis and Modeling of Digital Systems”, McGraw-Hill, 1993

[4] Ll. TERÉS, Y. TORROJA, S. OLCOZ, E. VILLAR, “VHDL - Lenguaje Estándar de Diseño Electrónico”, McGraw-Hill, 1998.

[5] J.-P. DESCHAMPS, J.Mª. ANGULO, “Diseño de Sistemas Digitales”, Paraninfo, 1992 (2ª edición).