GRUPO DE DISEÑO
HARDWARE SOFTWARE

DATCCCIA – ETSII– URJC

Introducción a Veribest
Sumador binario de n bits

Resumen:

El presente informe presenta una introducción sencilla paso a paso del simulador VHDL de Veribest 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 Veribest

C/ Tulipán s/n, E-28933, Móstoles, Madrid, ESPAÑA

Tabla de contenidos

Capítulo

1

Objetivo

El objetivo de este manual introductorio 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 VHDL Simulator®** de Veribest 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 (**Editor 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 indica el orden de compilación de los archivos que contienen el diseño y las características de compilación deseadas (**Workspace Browser**) y se

compila el diseño de forma ordenada jerárquicamente desde el nivel más bajo al nivel más alto.

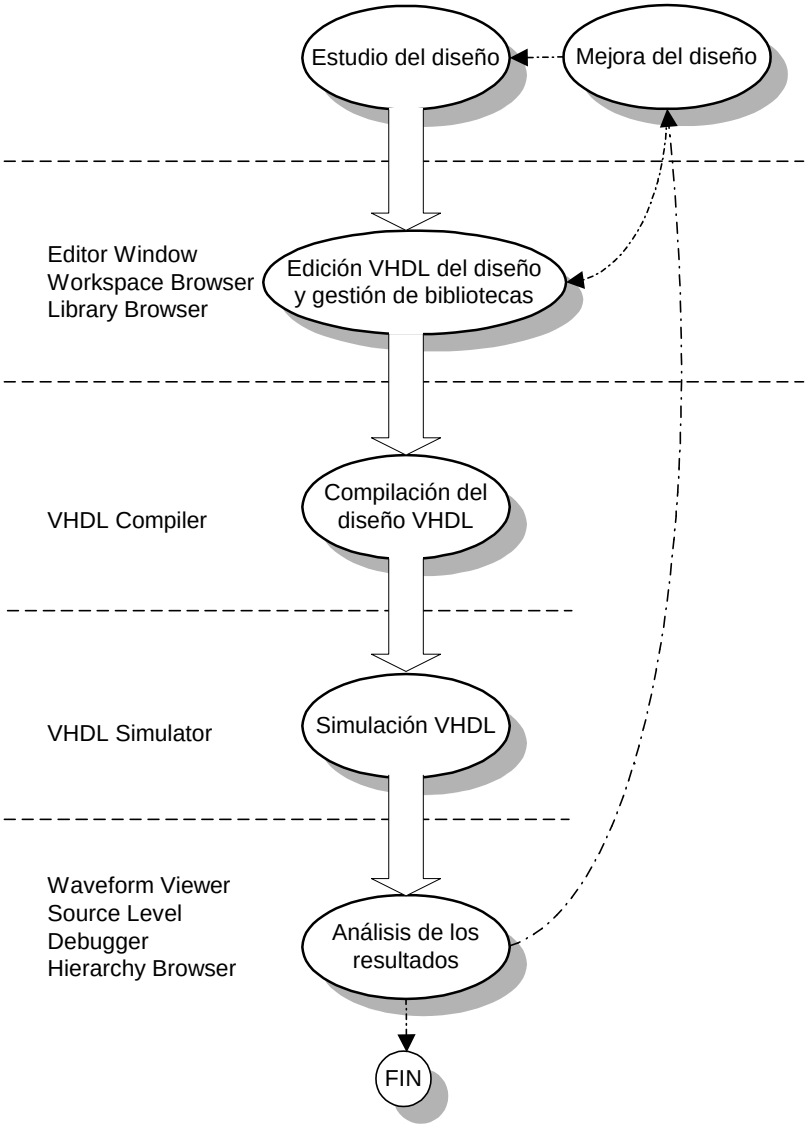


Figura 1: Método de trabajo.

- Se simula el diseño mediante el simulador VHDL especificando el tiempo de simulación (**VHDL Simulator**).

- 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 (**Source Level Debugger**). 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.

El paquete VeriBest VHDL Simulator (VBVHDL) **versión VB99.0 de evaluación** permite la utilización del estándar VHDL 1993 IEEE, y puede ejecutarse bien mediante el entorno gráfico o bien mediante línea de órdenes. La limitación de esta versión de evaluación consiste en que sólo se pueden compilar y simular 2000 líneas de 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 Veribest 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).

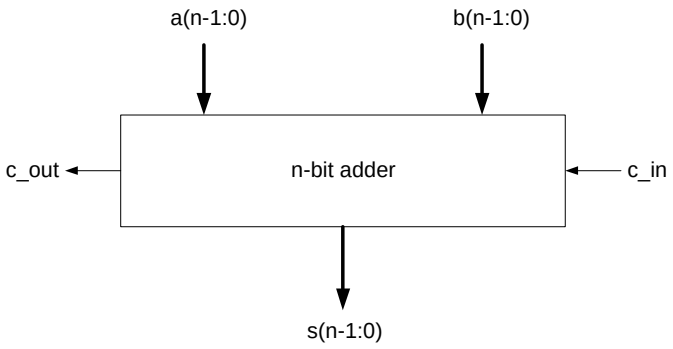


Figura 2: sumador binario parametrizable de n bits.

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

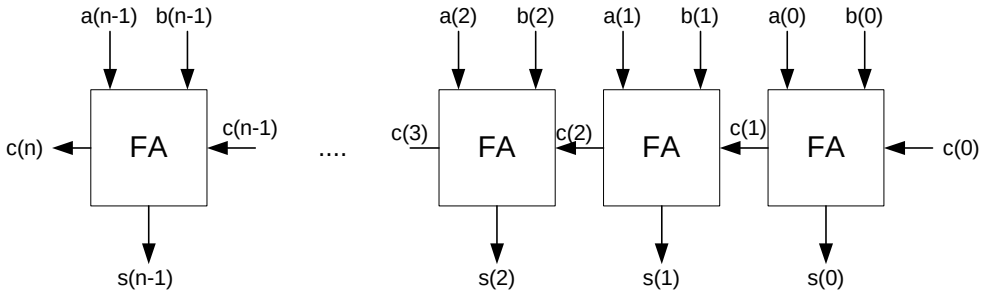
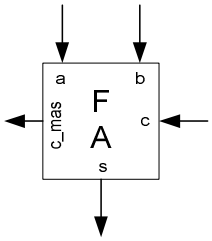


Figura 3: sumador binario en cascada (*ripple-carry*).

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 en forma de expresiones de conmutación 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 mediante expresiones de conmutación 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 Veribest 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 Veribest VHDL Simulator desde el menú de Inicio de Windows, como se indica en la Figura 5.

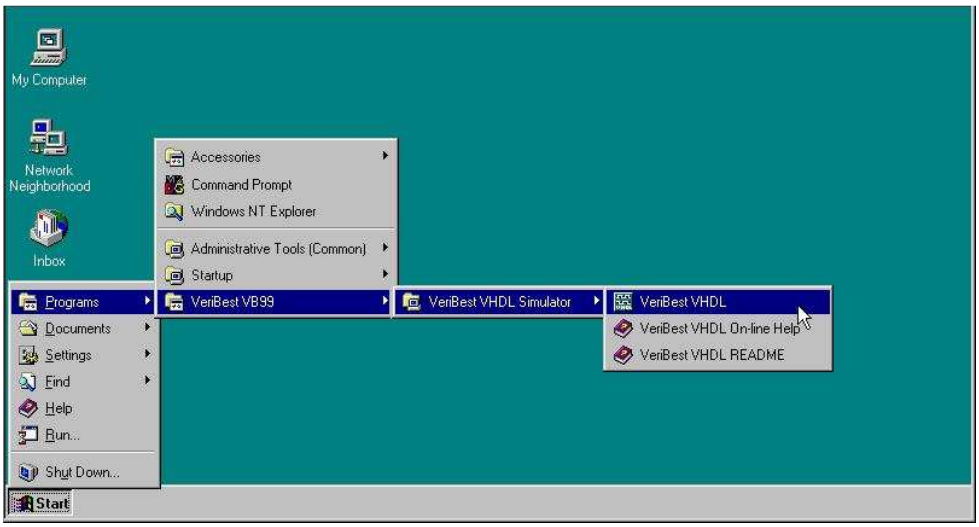


Figura 5: Ventana de inicio de Veribest.

Se comienza creando un Espacio de Trabajo (*Workspace)* seleccionando **New** desde el menú **File**. Se selecciona ***VHDL Workspace*** y se pulsa **OK** (Figura 6).

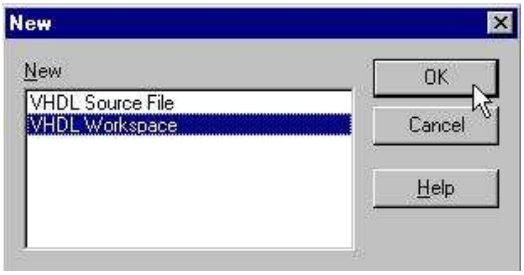


Figura 6: Creación del Espacio de Trabajo.

El cuadro de diálogo "***Create Workspace***" pide el nombre y la situación del mismo (Figura 7). Se crea el espacio de trabajo en alguna localización permitida. ES MUY IMPORTANTE NO UTILIZAR NOMBRES DE DIRECTORIOS QUE CONTENGAN ESPACIOS EN BLANCO O CARACTERES ESPECIALES. Una vez elegida la localización se teclea el nombre del espacio de trabajo, que será el más adecuado a cada diseño, por ejemplo *Add_N*. **Se debe marcar también la opción "Use Synopsys IEEE Library".**

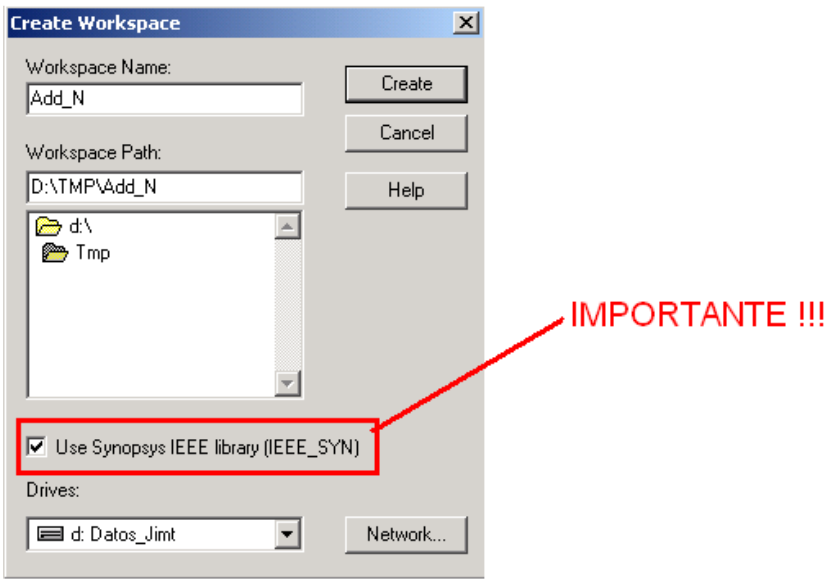


Figura 7: creación del espacio de trabajo.

Se pulsa Create y aparece una ventana de espacio de trabajo con el nombre especificado en su barra de título (Figura 8).

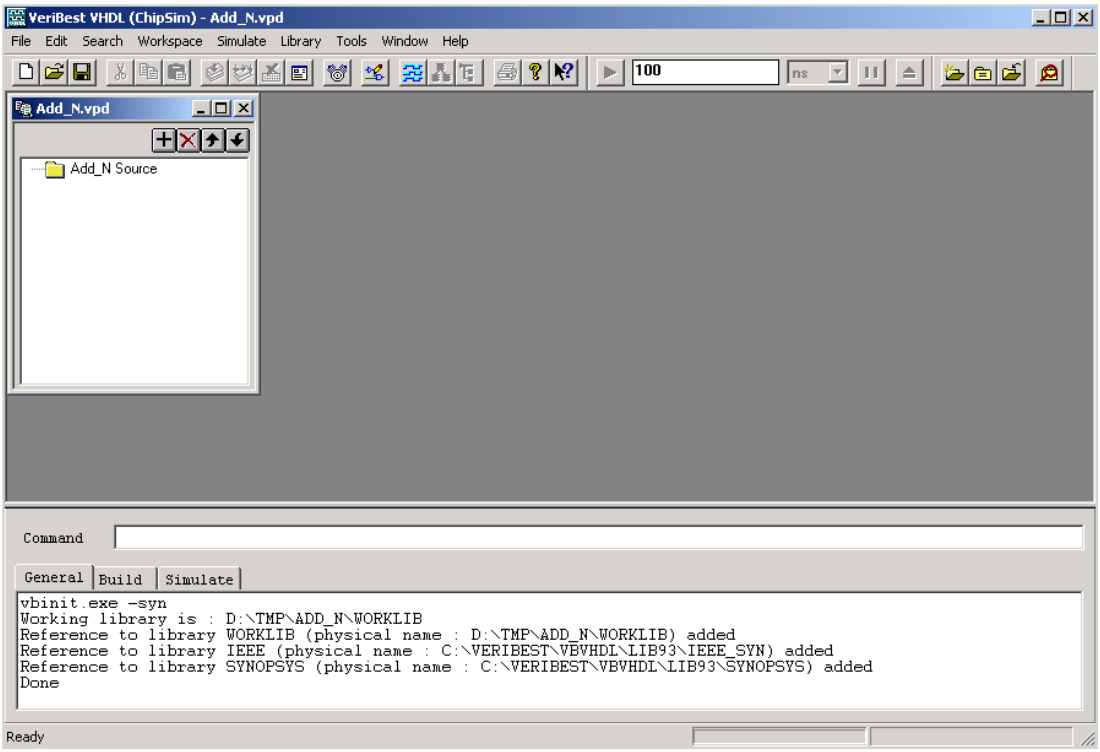


Figura 8: Espacio de trabajo creado.

El siguiente paso consiste en crear un archivo VHDL para introducir un diseño. Para ello, se pulsa **New** en el menú **File** pero ahora se pulsa sobre VHDL Source File (Figura 9), y a continuación OK.

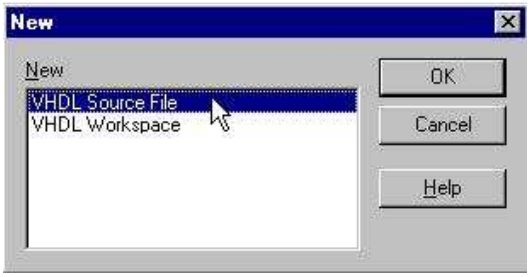


Figura 9: Creación de un archivo VHDL.

Aparece una ventana con una hoja vacía del editor de texto en la que se puede teclear directamente el código VHDL deseado (Figura 10), que puede salvarse en cualquier localización, aunque se recomienda que se guarde dentro del directorio creado para el espacio de trabajo.

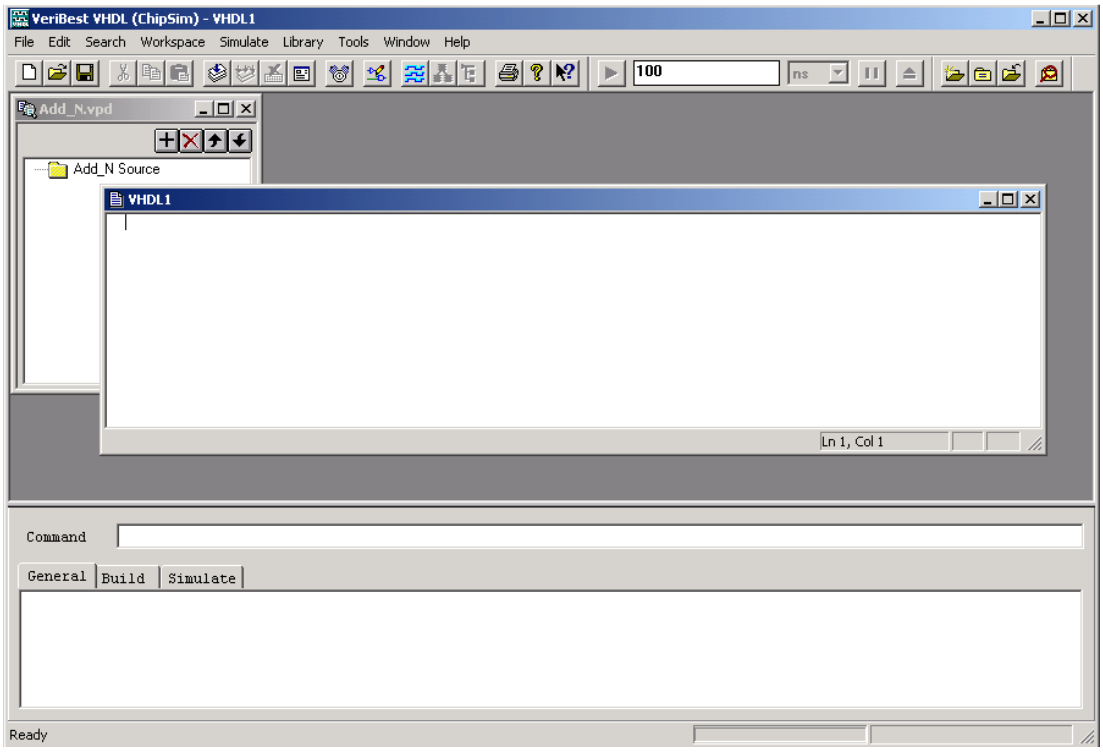


Figura 10: Hoja en blanco del editor de VHDL.

En un proceso de diseño completo normal, deberíamos escribir aquí el código VHDL; sin embargo, para este ejemplo ya disponemos del código VHDL, basta con copiarlo del apéndice A de este documento. Se pueden colocar cada unidad compilable en un archivo distinto, mezcladas en varios archivos o todas juntas en un único archivo. En este caso se

propone utilizar varios archivos donde cada uno contenga todas las unidades compilables relacionadas con una entidad. Todo el proceso de escritura se puede hacer directamente dentro de Veribest o primero en otro editor y después añadir 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

Compilación del diseño vhdl

El siguiente paso lógico consiste en incorporar el archivo o archivos VHDL al espacio de trabajo actual, lo que permitirá que el entorno sepa qué archivos debe compilar. Para ello, activamos la ventana del entorno de trabajo pulsando en su barra de títulos y ejecutamos **Add Files into Workspace** del menú **Workspace** o pulsamos el botón + en su esquina superior derecha.

Nos desplazamos por el árbol de directorios y localizamos los archivos que queremos añadir. (Figura 11).

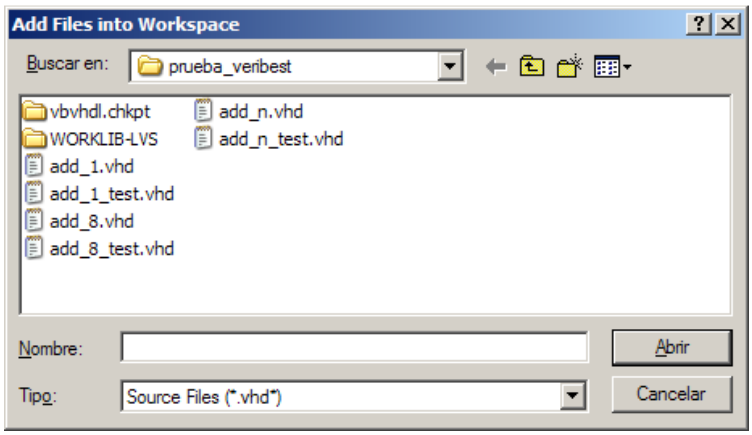


Figura 11: Adición de archivos VHDL al espacio de trabajo.

Una vez se encuentran en el espacio de trabajo los archivos VHDL, circunstancia habitual en diseños con múltiples 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.

El orden de compilación se especifica en la lista que aparece en la ventana del Workspace Browser, compilándose primero aquellos archivos que se encuentran más arriba en la lista. Se cambia el orden, seleccionando el archivo que se desea mover y pulsando las flechas que indican en qué dirección va a moverse (hacia arriba o hacia abajo).

En el caso del ejemplo, el orden debería ser el que aparece en la Figura 12. Esto es dependiente de las unidades compilables del diseño y de sus relaciones. Obviamente, aquellas que sean necesarias para otras deberán ser compiladas antes.

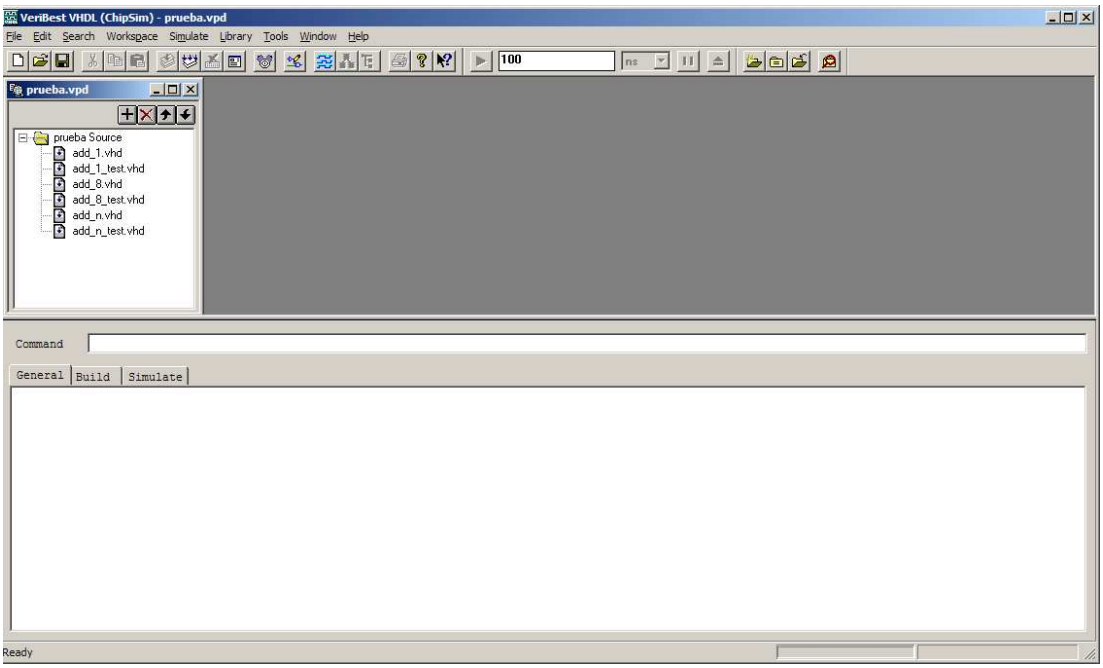


Figura 12: Orden de compilación adecuado para el ejemplo.

A continuación, hay que indicarle al compilador qué opciones debe establecer para realizar su tarea correctamente. Para ello, se selecciona **Settings** en el menú **Workspace** y se activa el separador **Compile** (Figura 13) en la ventana **Workspace Settings** que aparece. Ahí se pueden seleccionar opciones sobre si se desea depurar el código, qué comprobaciones se hacen sobre índices, cuál es la biblioteca de compilación, etc. En principio, se deja como está y se pulsa OK.

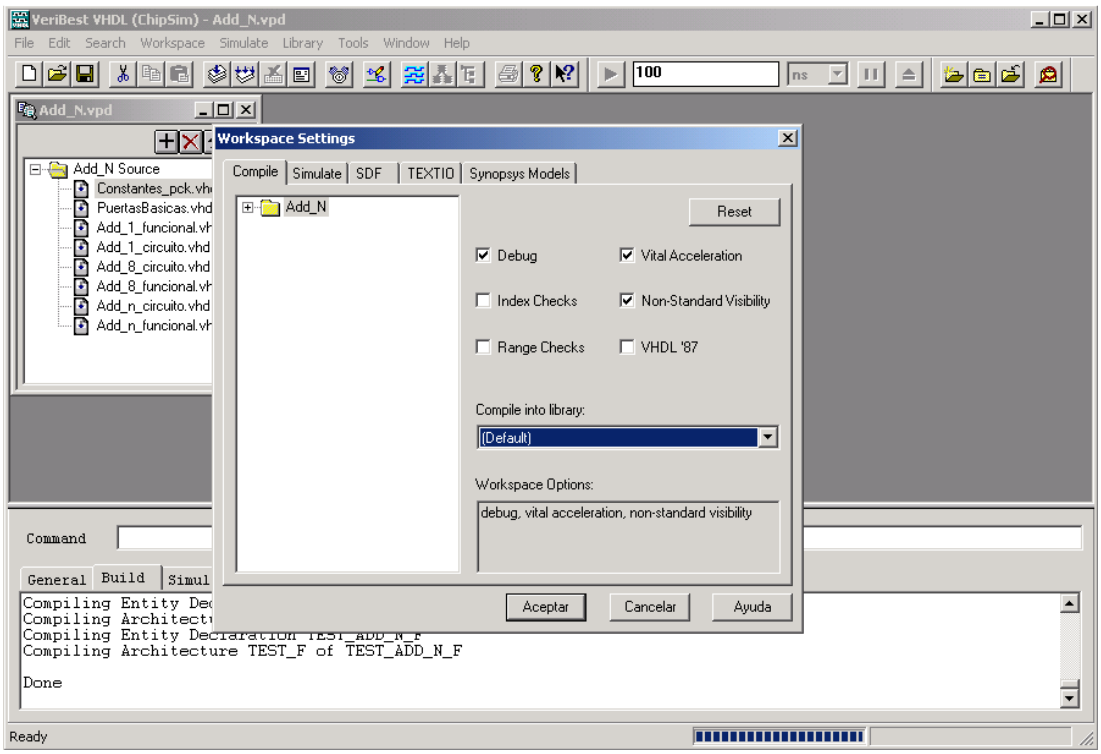


Figura 13: Opciones de compilación.

Una vez preparados para hacer la compilación, la llevamos a cabo pulsando **Compile All** en el menú **Workspace** o su icono correspondiente  , lo que dará lugar a la aparición de mensajes en la ventana de salida del entorno (Figura 14). Estos mensajes nos indicarán si la compilación ha transcurrido con normalidad o, si por el contrario han ocurrido errores.

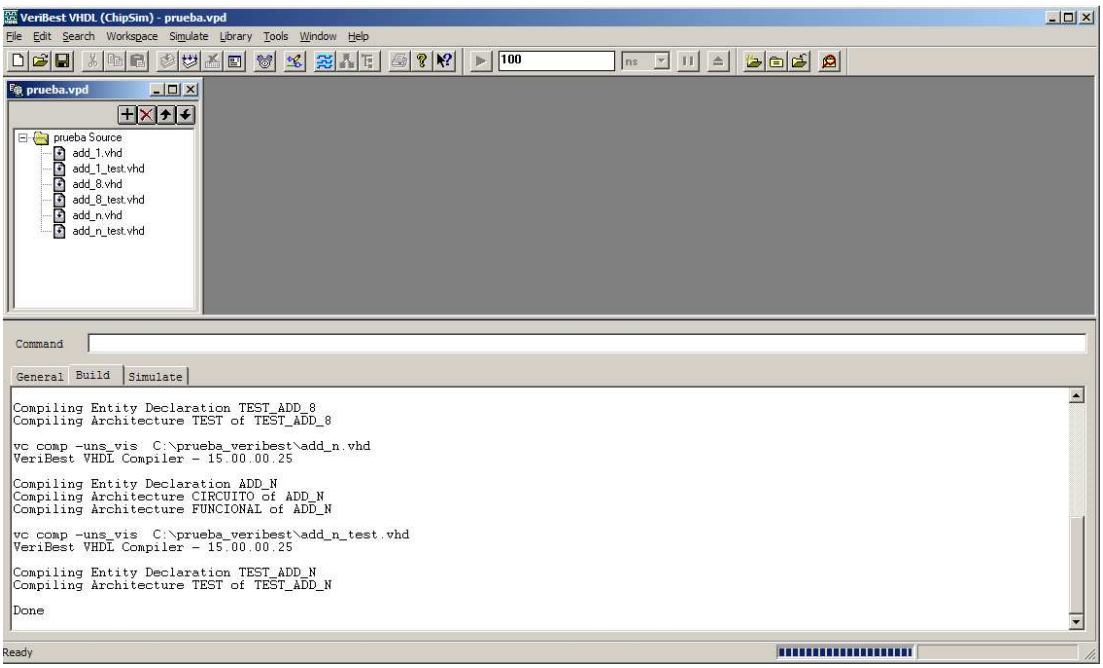


Figura 14: Resultados de la compilación.

Si todo ha salido correctamente, se puede proceder a salvar el diseño (Save).

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, se vuelve a abrir la ventana **Workspace Settings** y se selecciona el separador **Simulate** (Figura 15).

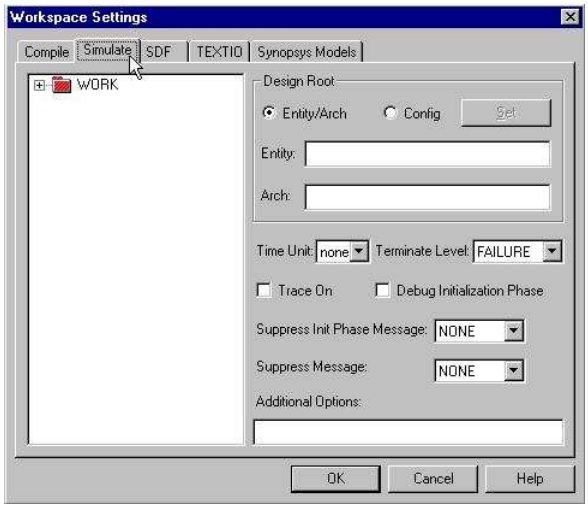


Figura 15: Opciones de simulación.

Se expande la biblioteca **WORK** para poder seleccionar la unidad de diseño que se desea simular. En este caso, se selecciona **TEST_ADD** (Test del sumador de 1 bit) de la lista (Figura 16).

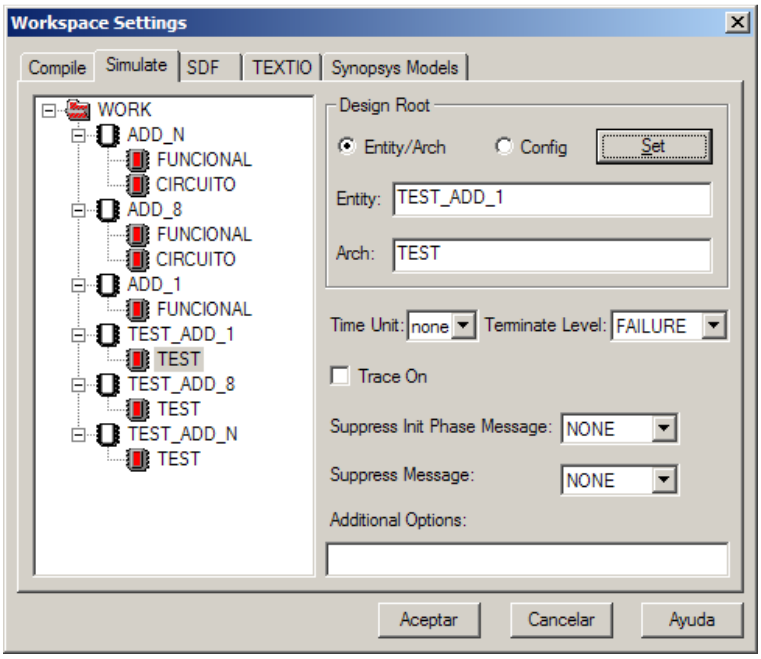


Figura 16: Selección de la arquitectura y la entidad para simulación.

Se pulsa **Aceptar** para cerrar la ventana de opciones y se está en disposición de comenzar la simulación. Se arranca el simulador seleccionando **Execute Simulator** en el menú **Workspace** o pulsando sobre el icono correspondiente  .

Se acepta el mensaje que informa de que no existe licencia para correr código VHDL de más de un determinado número de líneas (Figura 17).

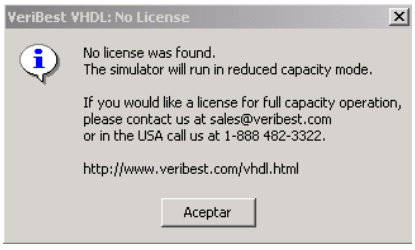


Figura 17: mensaje de falta de licencia.

Siempre que el mensaje que aparezca en la ventana de salida indique que todos los pasos previos han sido llevados a cabo correctamente, se activan las barras de herramientas de simulación y depuración (Figura 18). Conviene mirar bien los mensajes de la línea de comandos porque los errores hay que corregirlos todos.

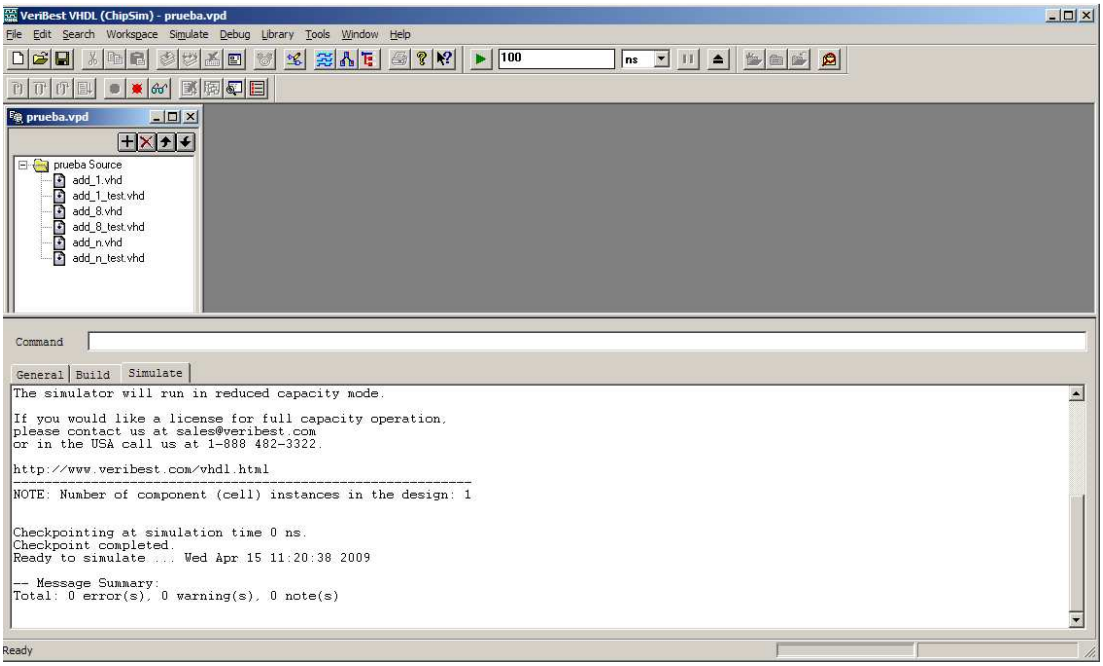


Figura 18: Simulación lista para comenzar.

Para poder ver los resultados de simulación es necesario abrir una ventana de representación de formas de onda. Esto se consigue seleccionando **New Waveform Window** del menú Tools (Figura 19), o pulsando en el botón  .

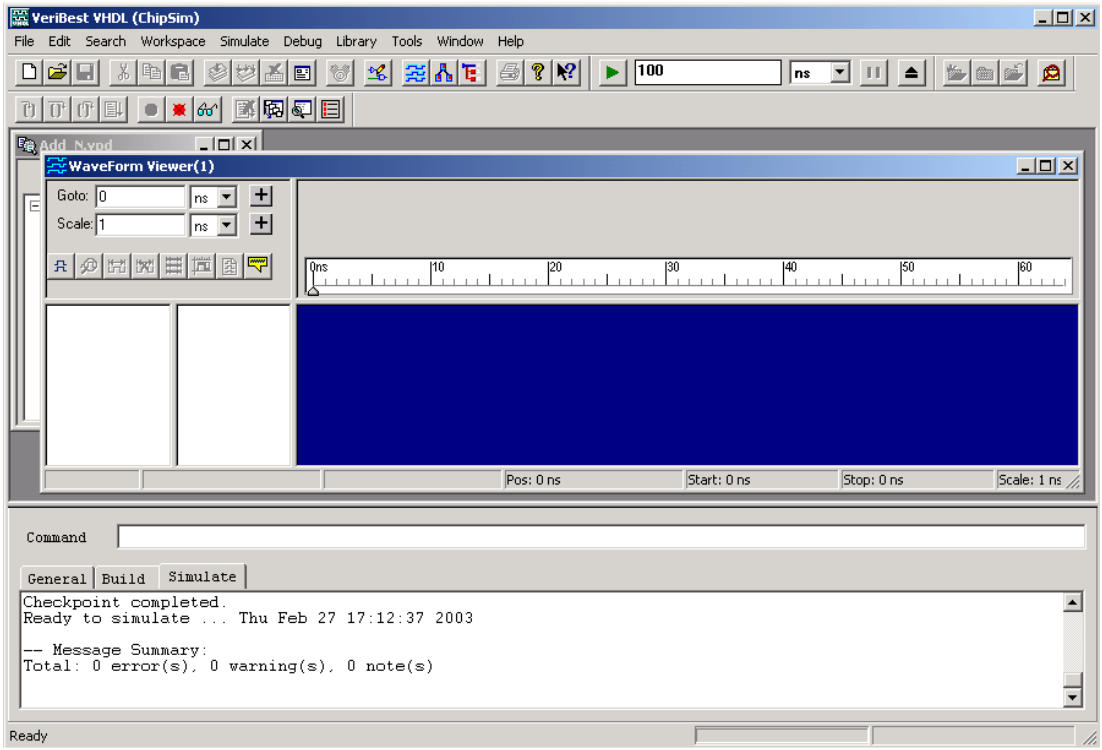


Figura 19: Ventana de representación de formas de onda.

Una vez abierta la ventana se añaden aquellas señales que se desean representar mediante el botón  de la ventana (Figura 20).

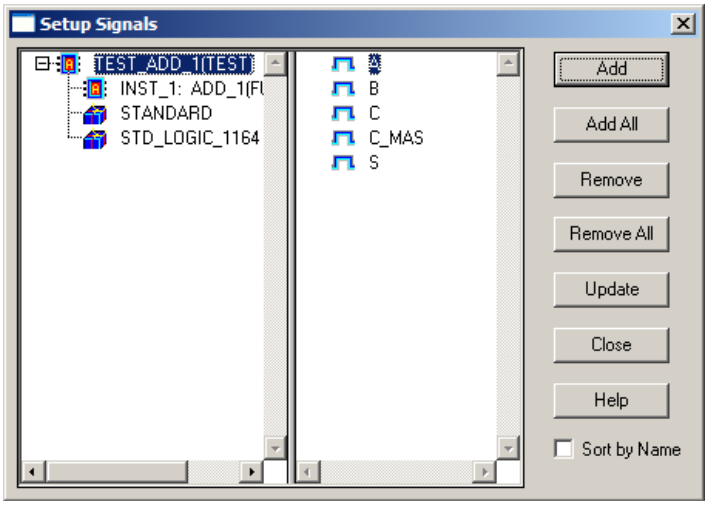


Figura 20: Adición de señales a la ventana de visualización.

En el ejemplo que nos ocupa seleccionaremos todas las señales del diseño pulsando en el botón **Add All**. En la mayoría de casos, se seleccionarán exclusivamente aquellas señales que se necesiten y no todas, ya que no todas tienen porque ser relevantes.

Seleccionadas las señales, se indica al simulador cuánto tiempo queremos que simule el diseño. Este valor dependerá de las formas de onda de las señales de entrada del test. Ya estamos en disposición de comenzar la simulación y para ello pulsamos en el botón 

el tiempo a la derecha del botón indica cuanto tiempo avanza la simulación por cada pulsación del botón.

Cuando haya terminado la simulación, se mostrarán las formas de onda de todas las señales seleccionadas previamente (Figura 21). Se puede comprobar que el sumador de 1 bit cumple con su tabla de verdad.

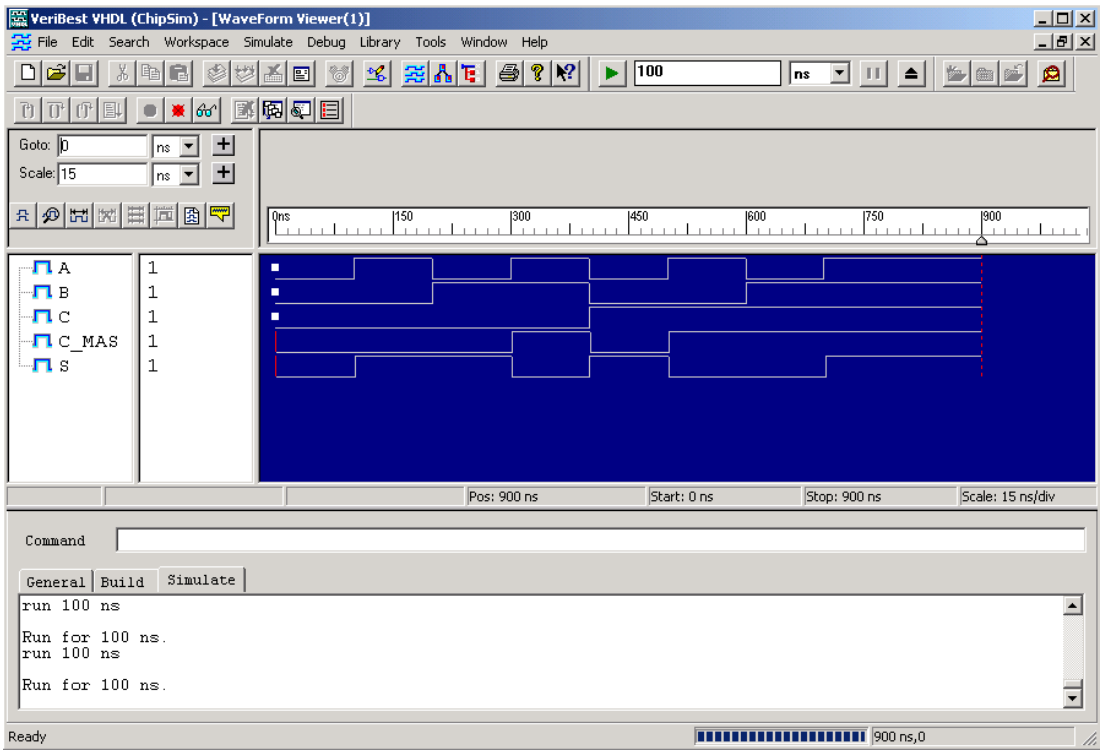


Figura 21: Formas de onda de las señales del ejemplo.

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. Se puede obtener más información desde el menú de Ayuda de Veribest.

Cuando se simula un circuito utilizando VeriBest es posible visualizar cualquier señal de cualquier módulo interno del circuito. 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 -> Quit.*

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 ir al menú *WorkSpace->Settings* y en la pestaña de *Simulate* escoger el Test que está dentro de TEST_ADD_8 y pulsar en el botón *Set.*.

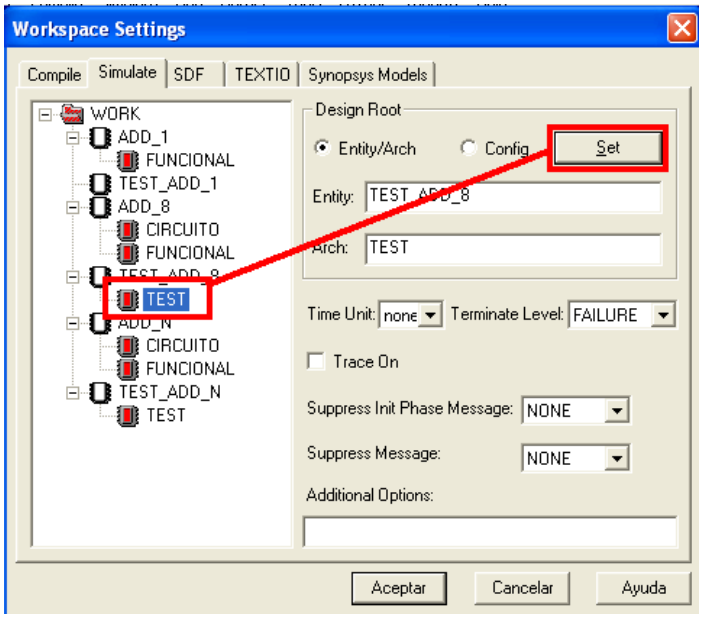


Figura 16: test sumador 8 bits

Una vez seleccionado el test-bench que se quiere simular, se arranca la simulación desde el menú *Simulate->Run* y posteriormente ya se pueden añadir ondas al visor de ondas pulsando en el botón correspondiente:

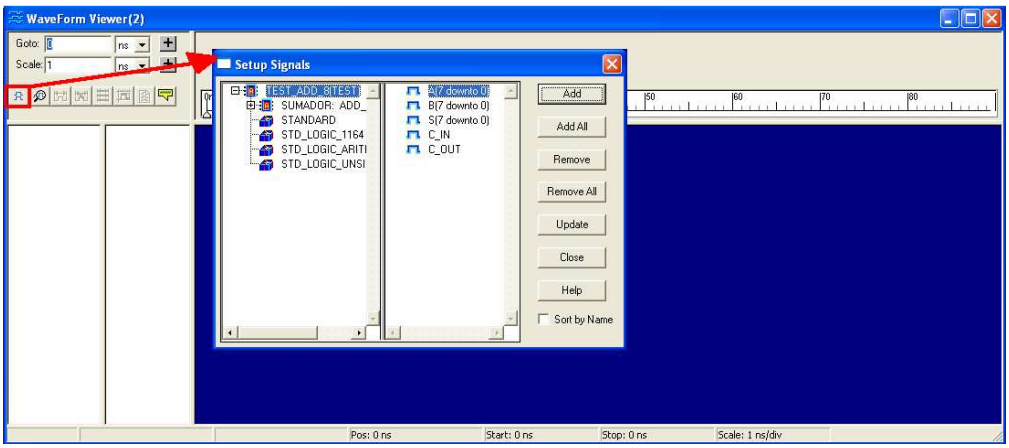


Figura 17: añadir señales al visor de ondas

Al igual que en el ejemplo del sumador de 1 bit, podemos añadir todas las señales del test-bench pulsando en el botón *Add* All. 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):

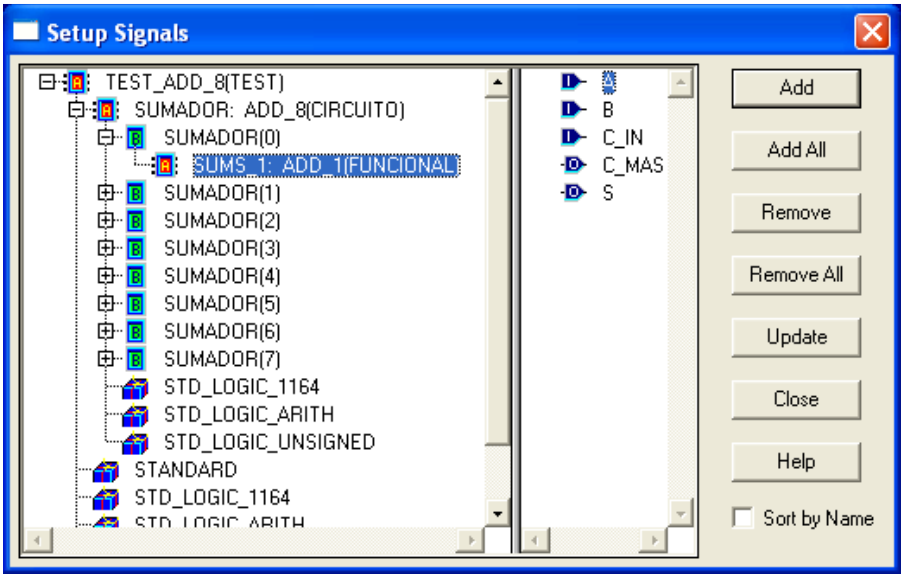


Figura 18: exploración de los diferentes submódulos

Una vez seleccionado un submódulo aparecerán sus señales en la parte derecha de la ventana. Se puede añadir cualquier señal al visor de ondas seleccionándola y pulsando en el botón *Add*, o también se pueden añadir todas las señales del módulo pulsando en el botón *Add All*. Para este ejemplo añadiremos sólo la señal *A* del módulo SUMADOR(0).

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:

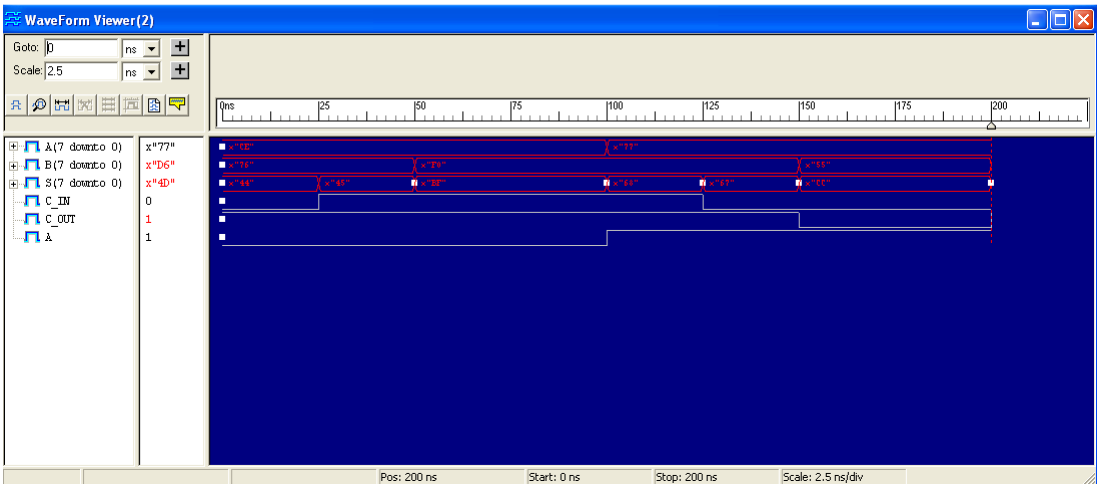


Figura 20: simulación del sumador de 8 bits

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 en el visor de ondas y con el botón derecho seleccionar *Bus Display>* 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 *decimal*:

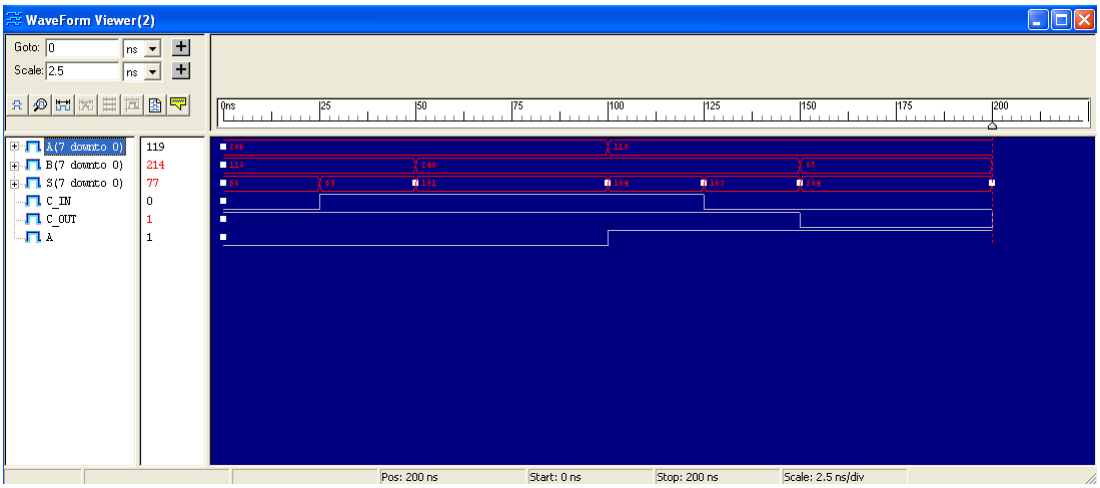


Figura 21: representación de los resultados en formato unsigned

Apéndice

A

Código VHDL del sumador binario de n bits

A continuación se da el código VHDL de los componentes y sistemas que se pueden diseñar.

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 Veribest.

[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).