P2

Funciones booleanas elementales descritas con VHDL

1.- Introducción y objetivos

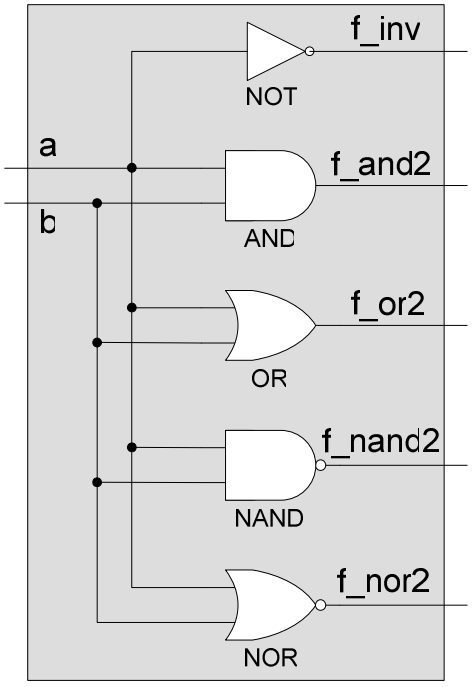
En esta práctica se van a implementar y estudiar las funciones booleanas elementales. Al mismo tiempo se pretende familiarizar al alumno en el entorno de desarrollo Xilinx ISE Design Suite, e introducir los conceptos y la sintaxis de la descripción de lógica digital empleando el lenguaje de descripción de hardware VHDL, ampliamente usado en el diseño de sistemas digitales complejos.

2.- Descripción del entorno de desarrollo

Lo primero que haremos será ejecutar el ISE Design Suite 12.4. El profesor describirá brevemente cada una de las ventanas que aparecen.

3.- Creación de nuestro proyecto

En esta práctica vamos a implementar las funciones lógicas que se muestran en la figura.



Toda función lógica puede describirse unívocamente a través de su tabla de la verdad. Abajo aparecen las tablas de verdad de las funciones que han de ser implementadas y, en una fase posterior, comprobadas mediante simulación.

		Tabla de la verdad				
a	**b**	**f_inv**	**f_and2**	**f_or2**	**f_nand2**	**f_nor2**
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	1	0
1	1		1	1	0	0

Preguntas:

- ¿Cuántas entradas tiene el diseño a implementar?
- ¿Cuántas salidas?

Para empezar el diseño tenemos que crear un nuevo proyecto, empleando un asistente. El profesor indicará cómo realizar este proceso. En el proyecto se incluyen todos los ficheros asociados determinado diseño.

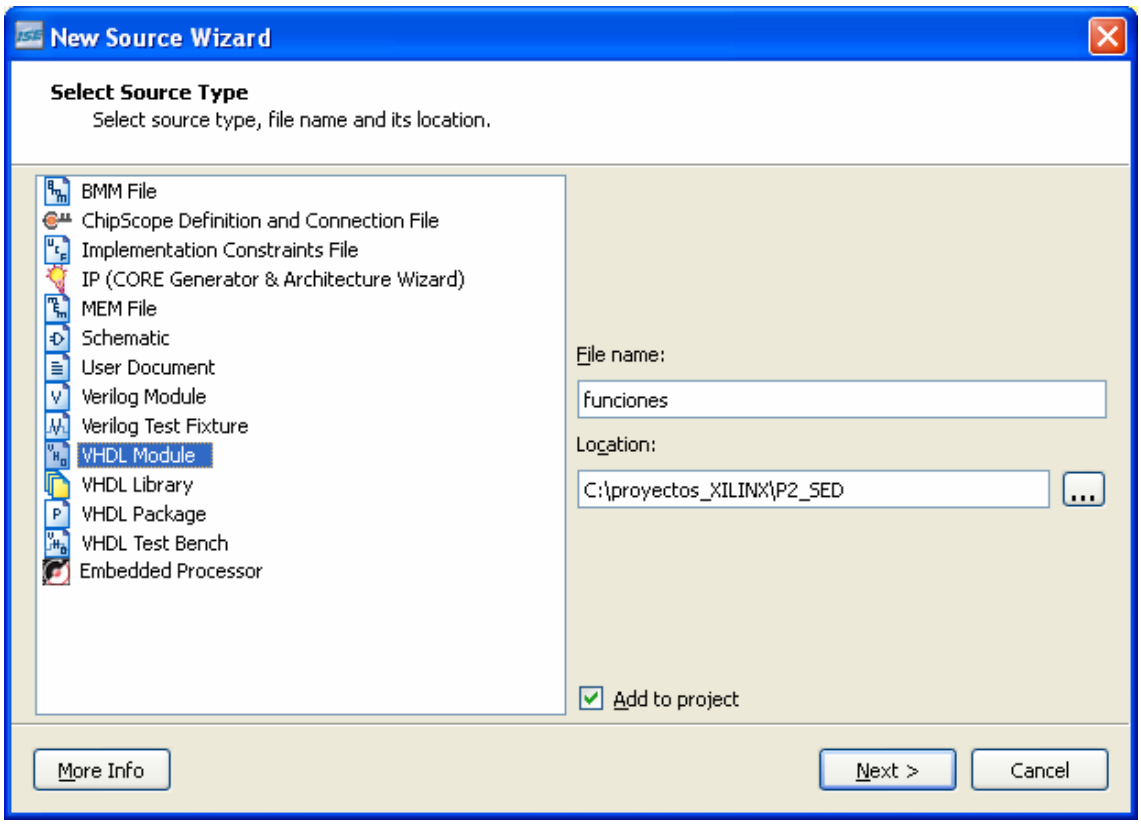
IMPORTANTE: En este asistente deberemos introducir una ruta para guardar nuestro proyecto, así como un nombre. Para el correcto funcionamiento de la aplicación tanto la ruta como el nombre NO deben contener espacios en blanco, ni caracteres 'raros', como eñes, comas, guiones y cosas por el estilo.

Una vez creado el proyecto, la apariencia de las ventanas que teníamos en el monitor ha cambiado. El profesor realizará los comentarios pertinentes.

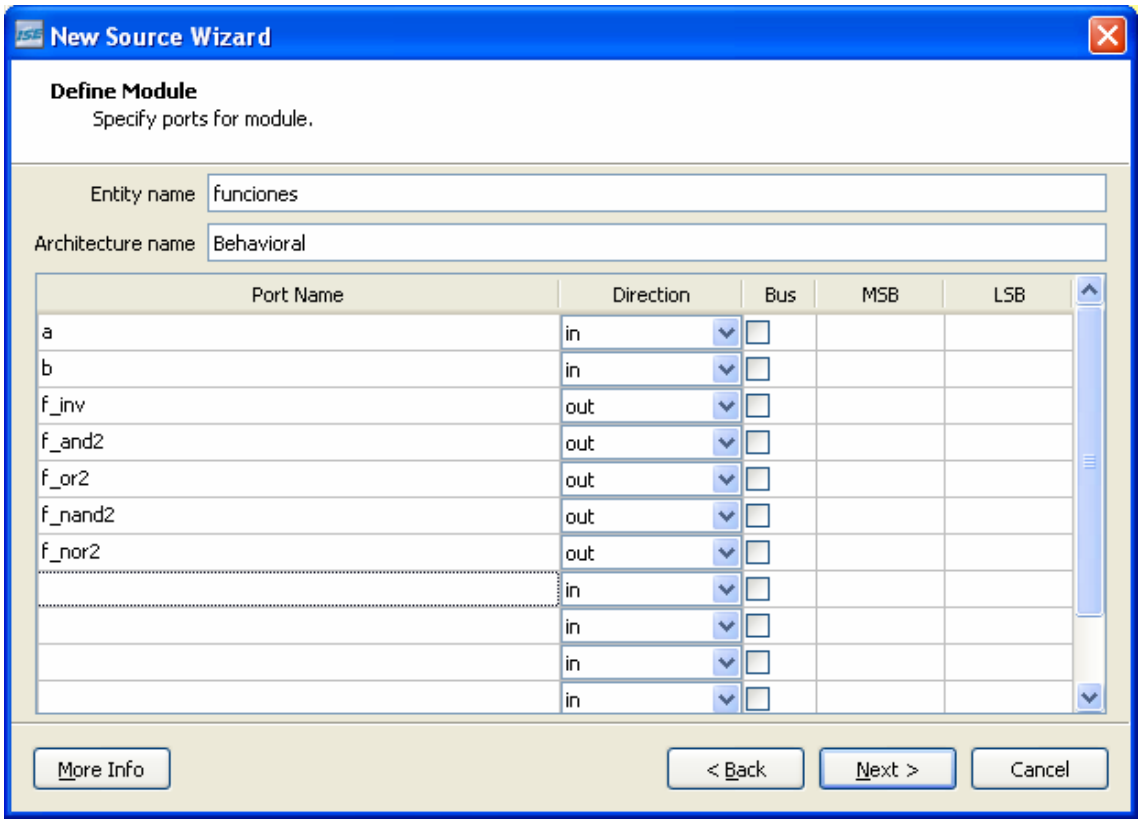
4.- Creación del esqueleto de un código VHDL

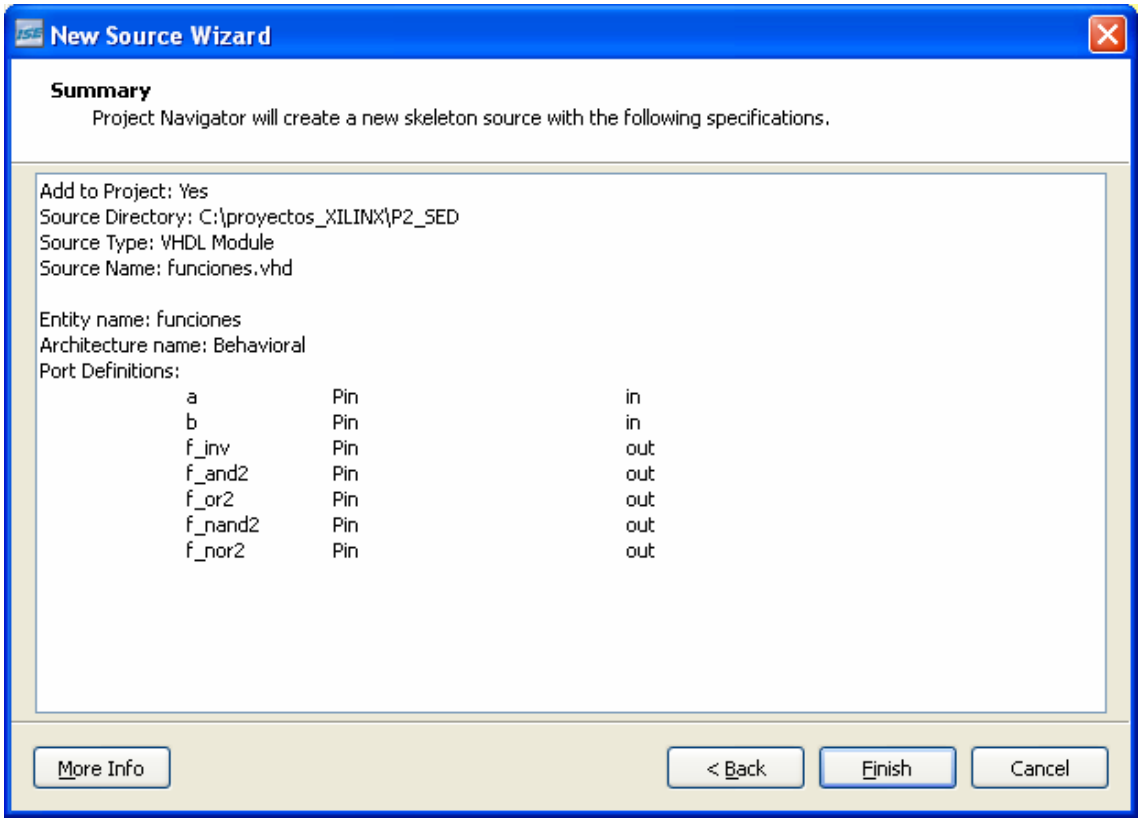
Para empezar a describir nuestro diseño debemos crear un código VHDL. A través de un asistente generaremos un esqueleto de código VHDL que luego tendremos que rellenar con la descripción de nuestro diseño.

El asistente se abrirá ejecutando el comando *New Source*. En la primera pantalla del mismo seleccionamos *VHDL Module* de todos los tipos de fuente que se pueden generar. En el margen derecho escribimos el nombre del fichero y la ubicación.



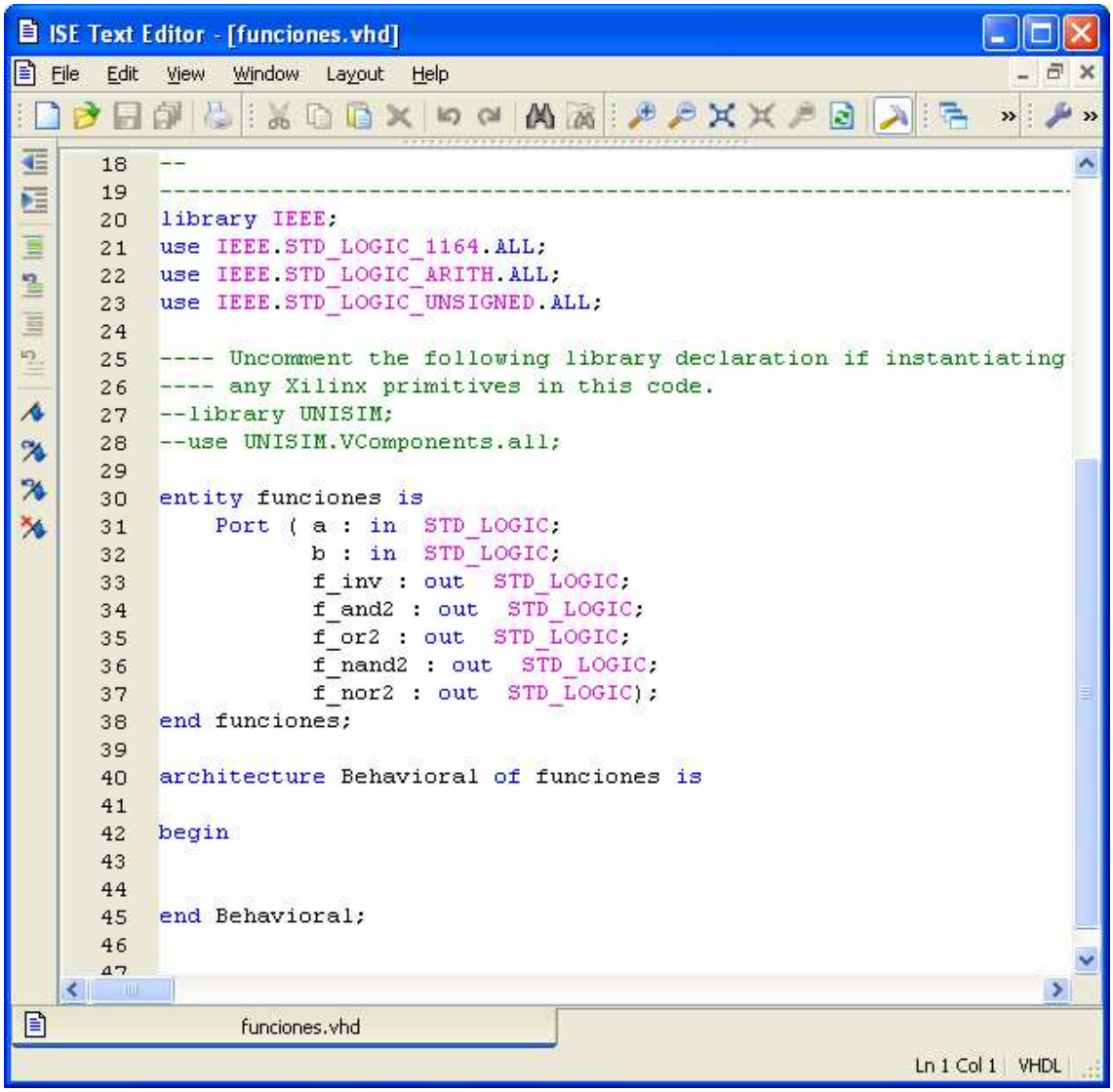
En la siguiente ventana del asistente, debemos introducir los puertos de entrada y salida del diseño:





Los puertos pueden ser compuestos (tipo bus), pero en nuestro todos los puertos son señales simples. Nótese que debemos indicar si los puertos son de entrada o salida.

La última ventana resume todos los parámetros que hemos seleccionado en el asistente. Con esto se genera el código **funciones.vhd**. El profesor explicará la estructura del código.



5.- Completando el código, comprobando la sintaxis y sintetizando

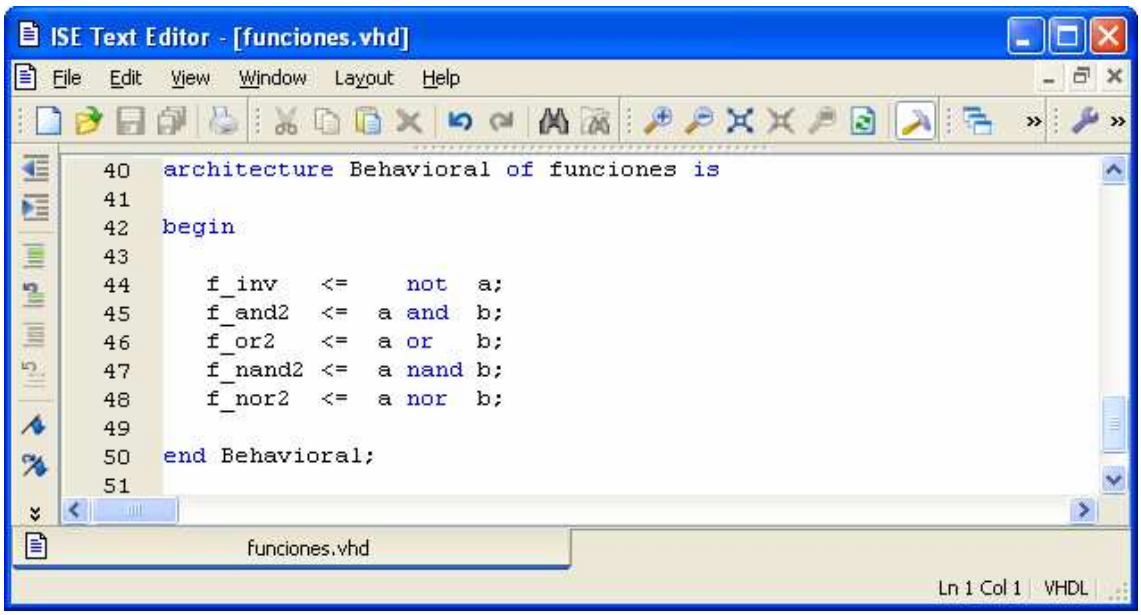
En la parte descriptiva de la arquitectura debemos escribir las asignaciones concurrentes que generarán la lógica que deseamos implementar.

Entra dentro de lo probable que escribamos mal las instrucciones (nos falta alguna coma o cosas así) o describamos lógica que no puede sintetizarse. Estos tipos de errores saltarán si ejecutamos *Check Syntax* y *Synthesize-XST* respectivamente.

A continuación se realizarán ejercicios comprobando los errores más comunes de un diseñador principiante y la principal diferencia con otros lenguajes, VHDL es un lenguaje concurrente:

- Errores de sintaxis
- Asignaciones múltiples
- Lectura de puertos de salida
- Escritura en puertos de entrada
- Asignación concurrente (reordenar las asignaciones)
- Señales internas

Después de todas las tareas de arriba, la descripción final del diseño queda como se ve en la figura:



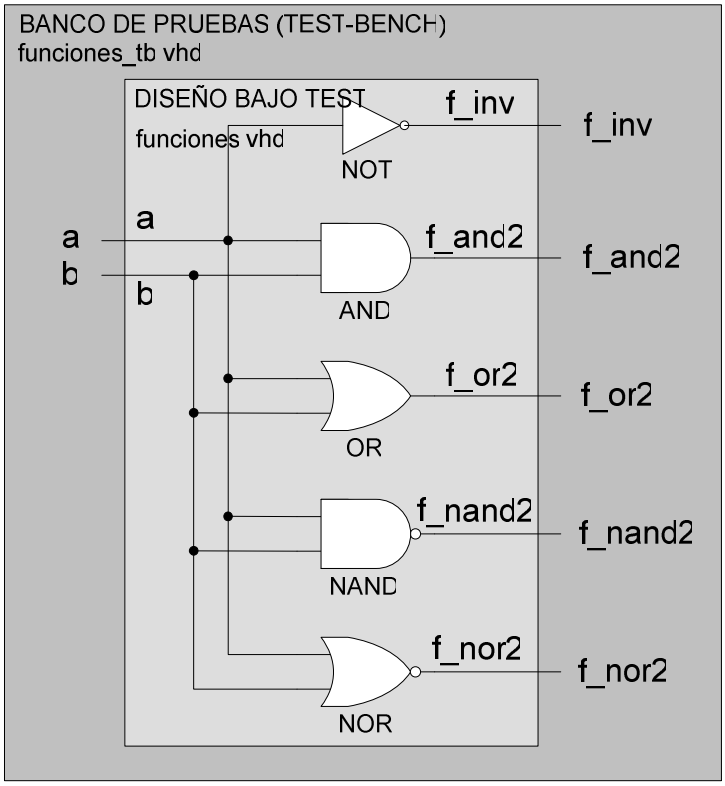
6.- Creando un banco de pruebas (test-bench)

Ya hemos terminado con la descripción del diseño. Pero eso sólo es la primera fase del diseño digital.

El siguiente paso necesario es comprobar que la lógica que se sintetiza realmente implementa lo que queremos. Este paso SIEMPRE es necesario, aun cuando la comprobación de la sintaxis y la síntesis no den problemas (supongamos que en la línea 48 de la figura de arriba escribimos **and** en lugar de **nor**, todo irá de perlas, pero la función implementada NO es la deseada, y por tanto el diseño está mal, y esto no lo sabremos a no ser que simulemos).

Para comprobar el correcto funcionamiento de las funciones implementadas, hay que comprobar los valores de tabla de verdad que están al principio de este documento.

Esta comprobación se realiza mediante un banco de pruebas, también llamado en inglés test-bench. El test-bench encapsula nuestro diseño en otro en un nivel jerárquicamente superior, tal como se ve en la figura.



El test-bench es un código VHDL NO sintetizable (no se puede implementar físicamente), en el que generamos estímulos a la entrada de la unidad bajo test (unit under test – uut) y observamos las salidas que provocan esos estímulos.

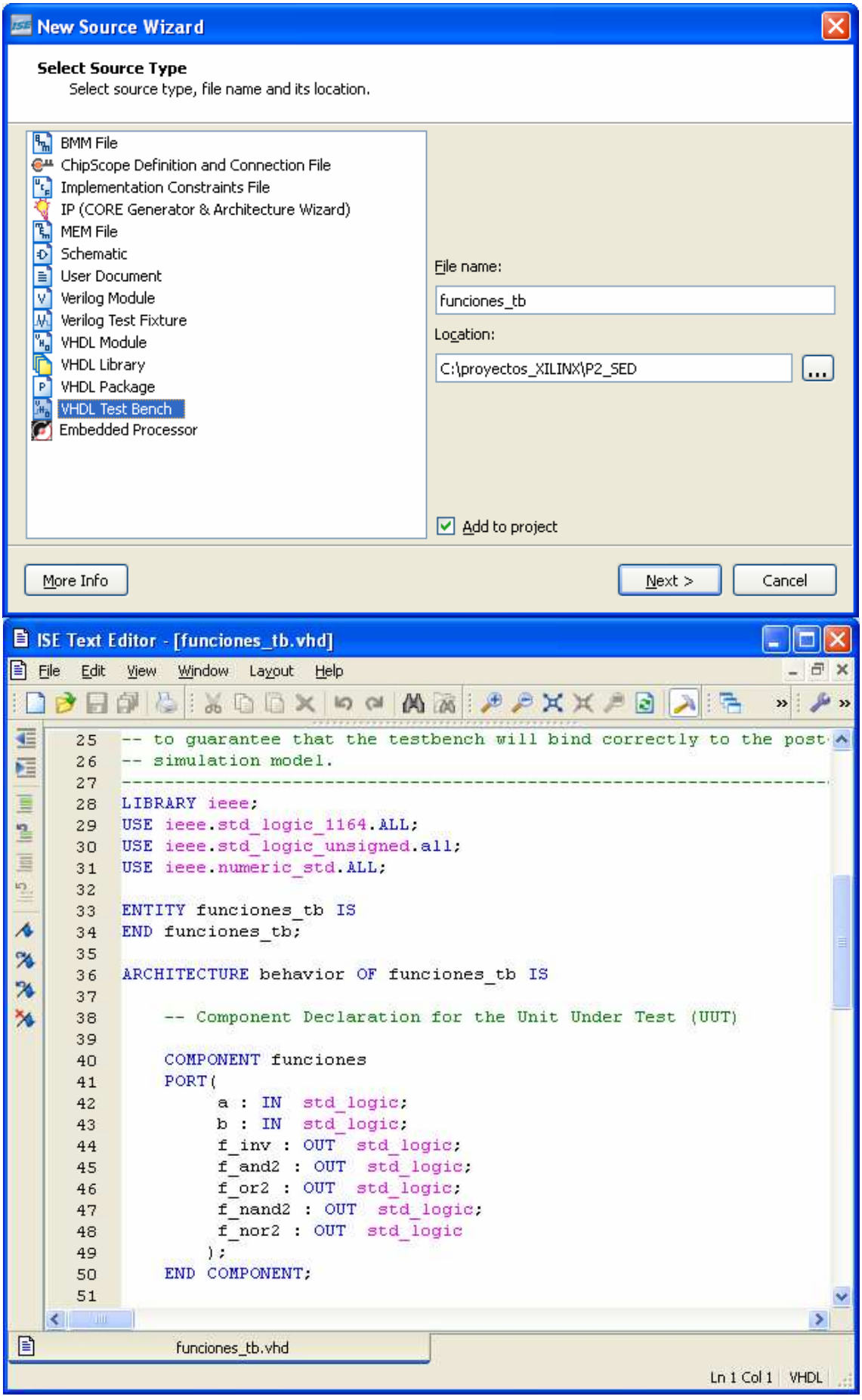
Preguntas:

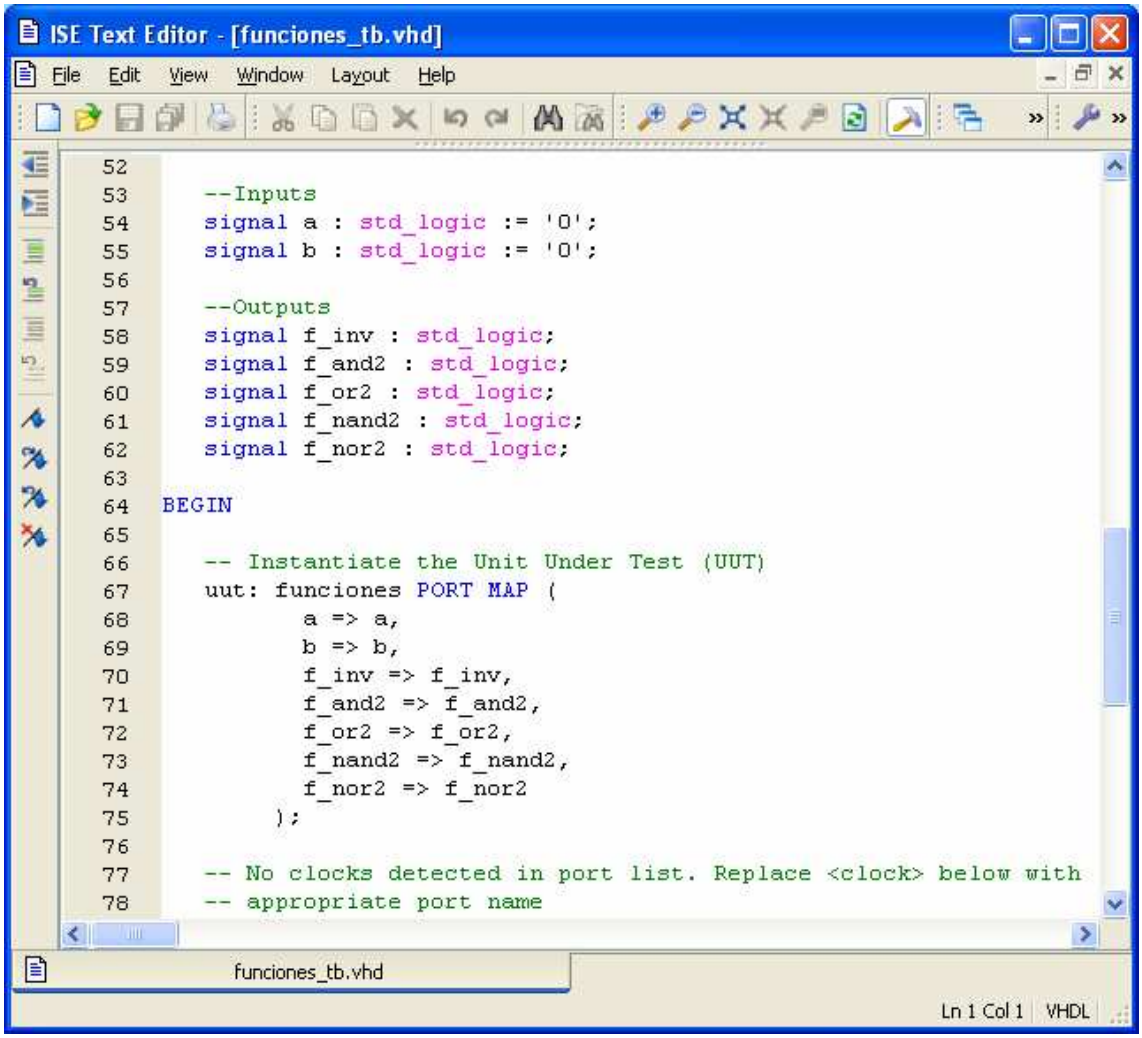
- ¿Cuántas entradas tiene el banco de pruebas?
- ¿Cuántas salidas?

Para generar el banco de pruebas, ejecutamos el comando *New Source*. En el asistente esta vez seleccionamos *VHDL test bench*, en lugar de *VHDL Module*, como se aprecia en la figura.

NOTA:
Es una buena costumbre poner un nombre al test-bench que lo asocie al nombre del diseño que va a ser simulado. En esta práctica, si queremos simular el diseño de **funciones.vhd**, el tech-bench debería llamarse **funciones_tb.vhd**, por ejemplo.

Al terminar todos los pasos del asistente, se genera el código que se muestra en la figura de la página siguiente:


New Source Wizard
Select Source Type
Select source type, file name and its location.
BMM File
ChipScope Definition and Connection File
Implementation Constraints File
IP (CORE Generator & Architecture Wizard)
MEM File
Schematic
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor
File name:
funciones_tb
Location:
C:\proyectos_XILINX\P2_SED
...
Add to project
More Info
Next >
Cancel
ISE Text Editor - [funciones_tb.vhd]
File Edit View Window Layout Help
25 -- to guarantee that the testbench will bind correctly to the post
26 -- simulation model.
27 --
28 LIBRARY ieee;
29 USE ieee.std_logic_1164.ALL;
30 USE ieee.std_logic_unsigned.all;
31 USE ieee.numeric_std.ALL;
32
33 ENTITY funciones_tb IS
34 END funciones_tb;
35
36 ARCHITECTURE behavior OF funciones_tb IS
37
38 -- Component Declaration for the Unit Under Test (UUT)
39
40 COMPONENT funciones
41 PORT(
42 a : IN std_logic;
43 b : IN std_logic;
44 f_inv : OUT std_logic;
45 f_and2 : OUT std_logic;
46 f_or2 : OUT std_logic;
47 f_nand2 : OUT std_logic;
48 f_nor2 : OUT std_logic
49);
50 END COMPONENT;
51
funciones_tb.vhd
Ln 1 Col 1 VHDL



El profesor explicará todas las instrucciones que aparecen en el código, haciendo especial hincapié en las instrucciones COMPONENT y PORT MAP.

NOTA:
De la línea 77 en adelante, las instrucciones conforman la depuración de un sistema secuencial. Como no es el caso, estas líneas serán eliminadas SALVO la línea 104 (la instrucción END de final de código).

Para depurar completamente el diseño, debemos introducir por las entradas (a, b) todas las combinaciones de entradas posibles, que en este caso son cuatro: 00, 01, 10 y 11, confrontar con la tabla de la verdad del principio del documento.

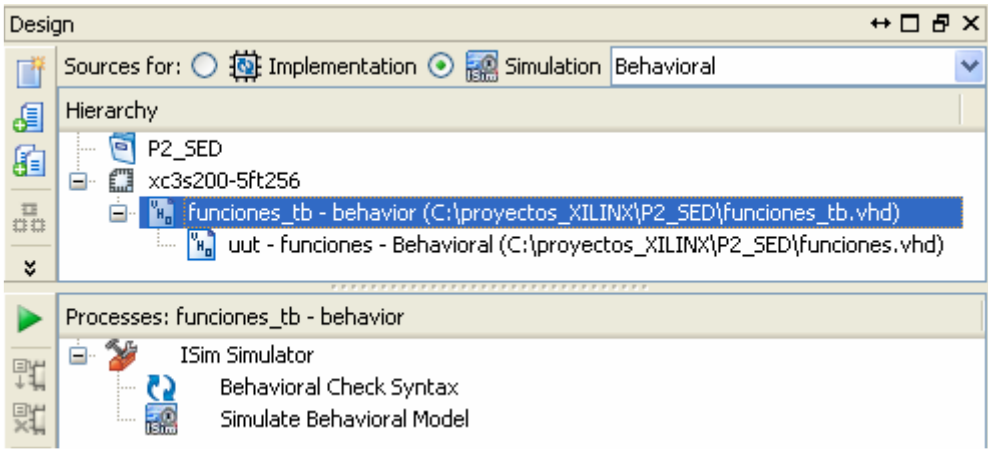
Ello se consigue con las siguientes asignaciones concurrentes:

```
77    a <= not a after 10 ns;
78    b <= not b after 20 ns;
```

El tiempo que aparece en cada instrucción es arbitrario. Lo importante es que el valor de los estímulos cambie de manera adecuada para obtener las 4 combinaciones de entrada, y esto se consigue haciendo que el periodo de cambio de los estímulos se duplique. Si tuviéramos 4 señales de entrada, por ejemplo, sería 10, 20, 40 y 80 ns respectivamente.

7.- Simulando

Una vez completado el banco de pruebas. Tenemos que seleccionar las fuentes para simulación (Source for simulation).

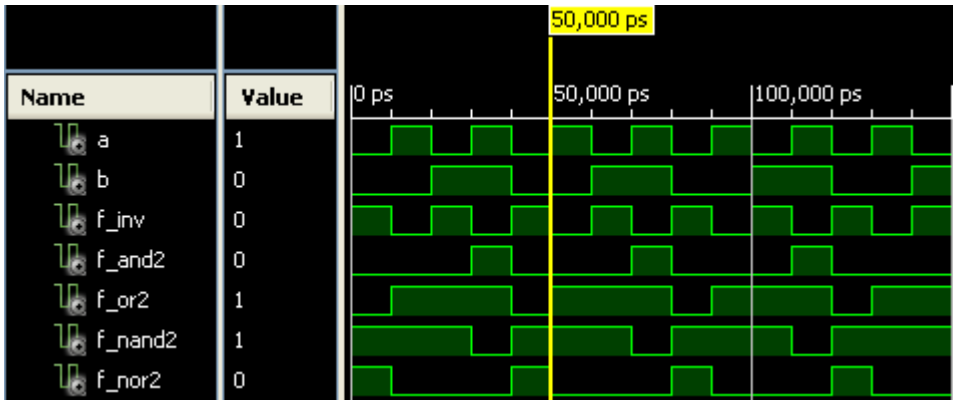


Aquí se puede observar la jerarquía. En la imagen se comprueba que la unidad bajo test (**utt**) cuelga de **funciones_tb**.

Para simular, basta con ejecutar el comando *Simulate Behavioral Model*.

IMPORTANTE:
Debe estar seleccionado el test-bench, como aparece en la figura. Si seleccionamos la **uut**, ésta no es un banco de pruebas, y no tiene estímulos de entrada, por lo que no observaremos nada en la simulación.

Obtenemos el siguiente cronograma:



Sólo debemos comprobar los primeros 4 valores, el resto son valores cíclicos. Observando la tabla de la verdad comprobamos que el diseño está bien.

		Tabla de la verdad				
a	**b**	**f_inv**	**f_and2**	**f_or2**	**f_nand2**	**f_nor2**
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	1	0
1	1		1	1	0	0

IMPORTANTE: Es importante CERRAR la ventana de la simulación si queremos realizar una nueva. El entorno de desarrollo está programado de tal manera que sólo se permite la ejecución de una simulación cada vez.

8.- Ejercicios

Crear un nuevo proyecto e implementar las siguientes funciones lógicas:

- $f1 = abc$
- $f2 = a + b + c$
- $f3 = ab + ac$
- $f4 = a\bar{b} + c$
- $f5 = (a + c)(\bar{b} + c)$
- $f6 = a \oplus b \oplus c$ Usar el operador XOR
- $f7 = abcd$
- $f8 = \overline{(a + b)(c + d)}$

1. Escribir la tabla de la verdad de todas las funciones
2. Observar las tablas de f4 y f5. ¿Qué pasa con estas dos funciones?
3. En términos de puertas básicas, ¿qué implementación es más eficiente, f4 ó f5?
4. Implementar las funciones en VHDL, chequear la sintaxis y sintetizar
5. Simular y comprobar la síntesis correcta contrastando con las tablas de verdad