

DISEÑO DIGITAL 1. BLOQUE TEMÁTICO 1**TÍTULO DE LA ACTIVIDAD:**
Tutorial ModelSim: simulación**CÓDIGO:**
BT1.P2**FECHA:****NOMBRE:****APELLIDOS:****MODALIDAD:**

Tutorial. Individual

TIPO:

Presencial

DURACIÓN:60
minutos**CALENDARIO:**

Sesión Sincrónica P2

REQUISITOS:Conocimientos adquiridos en
Electrónica 2**CRITERIO DE
ÉXITO:**

COMENTARIOS E INCIDENCIAS:

TIEMPO DEDICADO:

Minutos

AUTOEVALUACIÓN:
[entre 0 y 10 puntos]

No procede

PARTE I. Introducción:Test-Bench y simulación lógica

En esta actividad se continúa con el aprendizaje del proceso de depuración de modelos VHDL, empleando el simulador ModelSim, que se inició en la actividad P1. En el tutorial de aquella actividad se practicaron los procedimientos de creación de proyectos y ficheros, así como los de edición y depuración sintáctica de código HDL. En esta actividad se van a revisar las operaciones relacionadas con la validación funcional de los modelos lógicos VHDL, operaciones que aparecen numeradas en la figura 1, y son:

- Creación de un *Test-Bench*: Consiste en la realización del *modelo VHDL de una prueba de simulación* –para ello se aplicarán los procedimientos aprendidos en la actividad P1. Como resultado de esta operación se obtiene un *Test-Bench* VHDL.
- Ejecución y configuración de una simulación lógica: En esta fase se ordena la ejecución del *Test-Bench*. Para ello resulta necesario configurar algunos parámetros de simulación y, también, las herramientas de verificación de resultados que se utilizarán posteriormente. Tras la ejecución de la simulación se podrá observar la respuesta del modelo a las pruebas modeladas en el *Test-Bench*.
- Análisis de resultados de simulación: Esta tarea consiste en comparar el funcionamiento del modelo VHDL en la simulación (observable en un visor de formas de onda) con el del sistema modelado; como resultado puede obtenerse la validación del modelo VHDL, si la comparación es satisfactoria, o, en caso contrario, un diagnóstico de error que permita corregir el modelo.

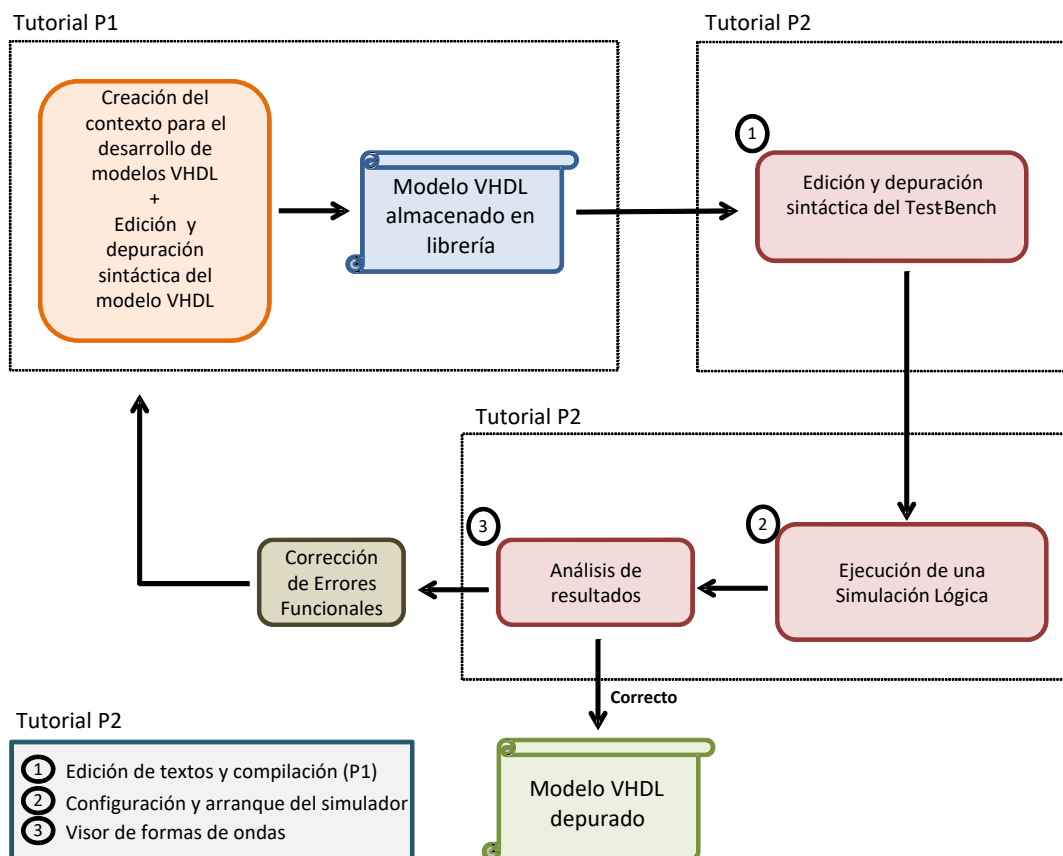


Figura 1.- Desarrollo de modelos VHDL

Instrucciones para la realización del tutorial

Aunque las siguientes indicaciones ya fueron hechas en el tutorial que realizó en P1, se las recordamos nuevamente:

1. Debe ejecutar todas las operaciones que se le indican mediante el predicado “**Realice las siguientes operaciones**”.
2. El texto en cursiva contiene explicaciones relativas a elementos del lenguaje VHDL o reglas que debe aplicar en la realización de código. Léalas con atención y, si tiene cualquier duda o curiosidad sobre ellas, no dude en preguntar a su profesor.
3. Comunique a su profesor cualquier problema o incidencia con que se encuentre durante la realización del tutorial.
4. Cuando termine de realizar el tutorial comuníquese a su profesor.

PARTE II. Tutorial

Realice las siguientes operaciones:

1. Arranque el simulador –si no recuerda cómo hacerlo, consulte el procedimiento T.1 de la actividad P1¹.

Observe que el simulador arranca tomando como proyecto activo *Proy_P1*, el proyecto que creó en la actividad P1, que era el que estaba activo la última vez que cerró ModelSim.

Este comportamiento del entorno facilita la continuidad del trabajo en un proyecto a lo largo de diversas sesiones -como en este tutorial que está comenzando a realizar, que va a consistir en la verificación del modelo del decodificador BCD a 7 segmentos desarrollado en la actividad P1 y va a desarrollarse sobre el proyecto *Proy_P1*.

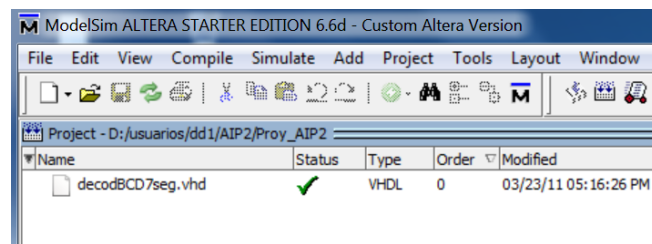


Figura 2.- Proyecto en el arranque: *Proy_P1*

Bancos de Prueba de Sistemas Electrónicos

Cuando se desea verificar experimentalmente el funcionamiento de un sistema electrónico se realizan pruebas funcionales, pruebas en las que se excitan las entradas del sistema con estímulos y tras las que se observan las respuestas de las salidas; dichas respuestas se comparan con las del sistema modelado para realizar así un diagnóstico de la corrección del modelo.

La realización de pruebas de funcionamiento se realiza en lo que se conoce como Banco de Pruebas o Banco de Test (Test-Bench). En un Banco de Pruebas (figura 3) puede distinguirse:

- ***El sistema hardware que se desea probar:*** Es el objeto de la prueba –suele denominarse, abreviadamente, DUT (Device Under Test).
- ***El conjunto de equipos necesarios para generar los estímulos y medir la respuesta del DUT:*** Forman parte de él la instrumentación de test (para la generación y medida de señales) y los elementos de conexión entre la instrumentación y el DUT.

Por otra parte, es necesario destacar que la realización de las pruebas conlleva la ejecución de diversas tareas de ingeniería, como el diseño de las pruebas, el diseño del propio Banco de Test o el análisis de los resultados. En la figura 3 se representan los recursos materiales y las tareas de ingeniería involucradas en la realización de pruebas en un Banco de Test.

¹ En adelante, cuando deba repetir un procedimiento que ya ha realizado, y para evitar la recurrencia a esta farragosa fórmula que le indica, por si necesita repasarlo, la referencia del mismo, se abreviará adjuntando, entre paréntesis, el código del procedimiento precedido por la actividad donde se explicó; por ejemplo, la instrucción que contiene *la nota al pie* que le ha traído hasta aquí, quedaría de la siguiente forma: “Arranque el simulador (P1.T.1)”.

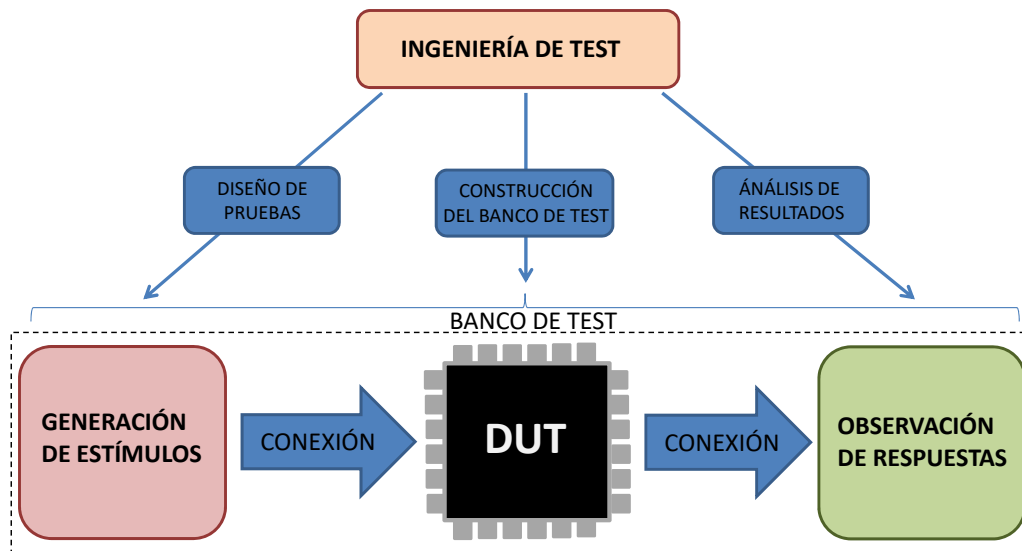


Figura 3.- Banco de Test de un Sistema Digital

Bancos de Prueba VHDL

Un simulador VHDL ejecuta lo que se conoce como **Test-Bench** VHDL, que no es otra cosa que un modelo del banco de test de la figura 3, porque, como ya sabe:

- En un Test-Bench VHDL se inserta el modelo VHDL –que hace las veces del DUT– cuyo funcionamiento se desea verificar.
- En un Test-Bench VHDL se conecta el modelo (DUT) a señales (recursos de conexión) que determinan el discurrir de las pruebas (estímulos) o permiten observar la respuesta del modelo.
- En un Test-Bench VHDL hay procesos con sentencias WAIT –que cumplen la función de los equipos que generan estímulos– que materializan pruebas asignando valores a señales.

Las tareas de ingeniería que hay que realizar para verificar un modelo VHDL son también análogas a las indicadas en la figura 1:

1. En primer lugar hay que diseñar las pruebas que se van a realizar.
2. A continuación hay que construir el Test-Bench –para lo que resulta necesario codificar los estímulos que materializan las pruebas diseñadas, emplazar el modelo VHDL que se desea validar y conectarlo a los estímulos.
3. Seguidamente hay que ejecutar una simulación.
4. Por último hay que analizar los resultados de la simulación para realizar un diagnóstico del modelo.

El curso que va a seguir esta actividad, a partir de aquí, es el marcado por estas tareas, que se completarán para llevar a cabo la verificación con ModelSim del modelo del decodificador BCD a 7 segmentos realizado en el tutorial de la actividad P1.

Diseño de pruebas

El diseño de pruebas de simulación está fuera del alcance de esta actividad. Sólo se trata aquí –de manera tangencial– porque es la tarea que especifica las pruebas a partir de las que se desarrolla el código de los procesos que manejan los estímulos de la simulación.

El punto de partida para el diseño de las pruebas de validación de un modelo VHDL es el propio modelo lógico del sistema a partir del que se creó éste (figura 4).

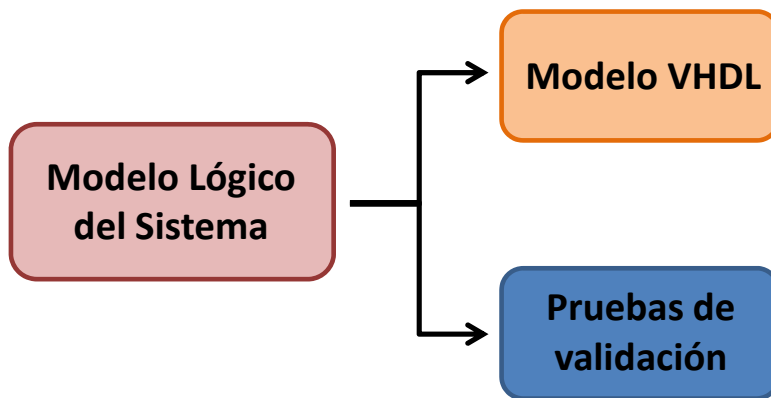


Figura 4.- Diseño de las pruebas de simulación

Observando el modelo lógico del decodificador BCD a 7 segmentos (figura 5) resulta evidente que, dada su simplicidad, es posible plantear la realización de una prueba exhaustiva. En la simulación se realizará, por tanto, una prueba en la que se irán presentando en la entrada del decodificador, y en orden binario creciente, las 16 combinaciones de '0's y '1' que pueden formarse con 4 bits.

DECODIFICADOR BCD A 7 SEGMENTOS									
A	s_a	s_b	s_c	s_d	s_e	s_f	s_g	D	C
B	s_a	s_b	s_c	s_d	s_e	s_f	s_g	D	C
C	s_a	s_b	s_c	s_d	s_e	s_f	s_g	D	C
D	s_a	s_b	s_c	s_d	s_e	s_f	s_g	D	C
Resto									

Figura 5.- Modelo lógico del decodificador

Construcción del Test-Bench

El desarrollo del código de un Test-Bench VHDL² puede dividirse en dos partes:

1. La primera parte tiene una naturaleza sistemática, es decir, no requiere ningún tipo de aportación creativa por parte del diseñador³. Comprende la creación de una Declaración de Entidad sin puertos y de un Cuerpo de Arquitectura donde se declaran señales y se emplaza el modelo VHDL a verificar.
2. La segunda parte consiste en la codificación de las pruebas de simulación diseñadas. La complejidad de esta tarea depende de las peculiaridades que presenten las pruebas que se hayan diseñado -en el ejemplo de este tutorial, y en los que realizará en esta primera parte del curso, la codificación de pruebas resultará ser siempre muy sencilla; a medida que el curso de la asignatura avance, las cosas se irán complicando (un poco).

Se va a abordar primero el desarrollo de la parte del código que puede generarse de manera sistemática. Para editar el TB hay que crear un fichero de texto. Esto es lo que se va a hacer a continuación, pero siguiendo un procedimiento distinto al que aprendió en la actividad P1.

Procedimiento T.1: Creación de un fichero añadido al proyecto

Realice las siguientes operaciones:

1. Pulse el botón derecho del ratón con el cursor situado en el área de trabajo de la pestaña **Project** y seleccione la opción **Add to Project -> New File...**

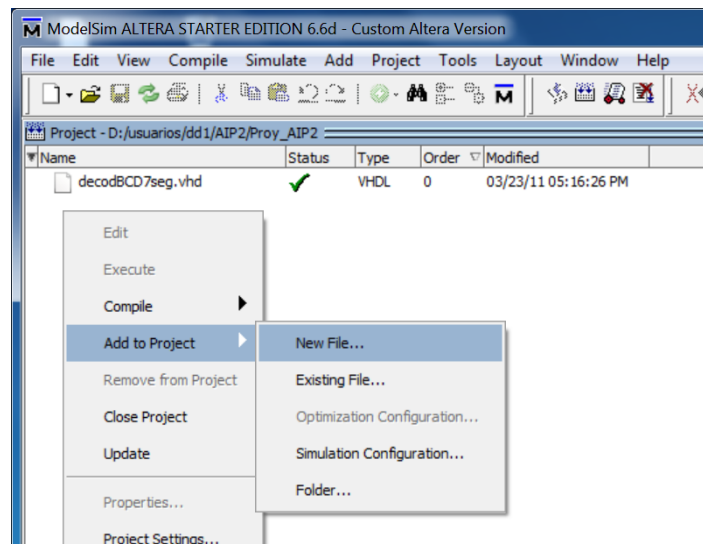


Figura 6.- Nuevo fichero VHDL

² En adelante TB VHDL

³ Esto es cierto en un grado tal que resulta posible generar automáticamente este código; ModelSim, por ejemplo, facilita herramientas para hacerlo –no las vamos a utilizar porque se genera un código con construcciones VHDL que *no nos parece conveniente* presentar en esta introducción al uso del lenguaje.

2. En la ventana que aparece, escriba en el campo **File Name** el nombre del nuevo fichero, **tb_decodBCD7seg.vhd** (figura 7), y pulse el botón **OK**.

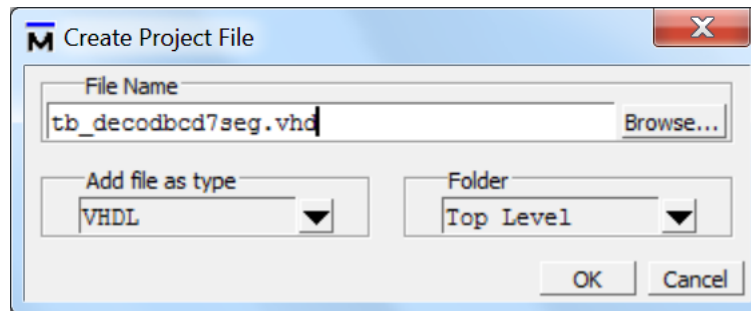


Figura 7.- Nombre del fichero

Observe que el nuevo fichero se añade al proyecto, apareciendo su nombre al lado del fichero que contiene el modelo del decodificador (figura 8).

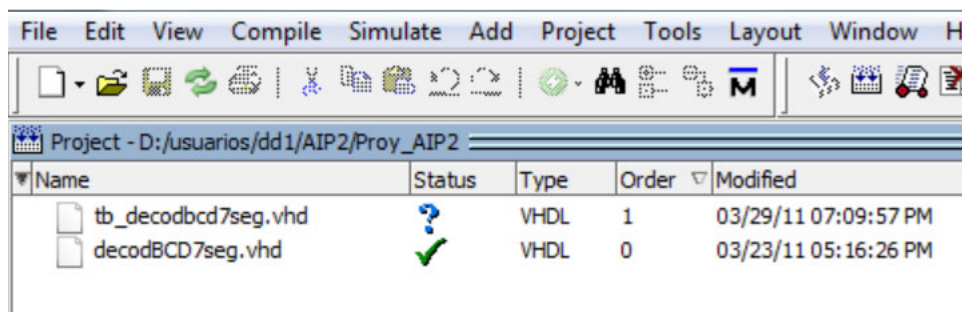


Figura 8.- Fichero añadido al proyecto

3. Para abrir el nuevo fichero y comenzar la edición, realice una doble pulsación con el botón izquierdo del ratón sobre el nombre del fichero.

Si lo desea, despegue el editor de texto de la ventana principal de ModelSim, o amplíe el área del editor en dicha ventana.

Declaración de Entidad del Test-Bench

En la figura 9 se muestra el código correspondiente a la declaración de entidad del TB del decodificador BCD a 7 segmentos.

```
2      -- Test-Bench de un decodificador
3      -- BCD a 7 segmentos realizado para
4      -- el tutorial de la actividad AIP3
5      -- (29-3-2011)
6
7      -- Cláusulas de visibilidad
8      library ieee;
9      use ieee.std_logic_1164.all;
10
11     -- Declaración de Entidad sin puertos
12     entity tb_decodBCD7seg is
13     end entity;
14
```

Figura 9.- Declaración de Entidad de un TB

En este código se puede constatar que:

- Las cláusulas de visibilidad que preceden a la Declaración de Entidad del TB son idénticas a las que se emplean en la realización de modelos lógicos VHDL. El motivo de su inclusión aquí es que los tipos de datos cuyo uso facilitan, **std_logic** y **std_logic_vector**, se van a utilizar para declarar señales en el Cuerpo de Arquitectura del TB, que adquiere visibilidad sobre estos tipos porque la hereda de la Declaración de Entidad.
- La única aportación que la Declaración de Entidad hace al TB es darle nombre; normalmente el nombre que se elige es el del modelo que se prueba (**decodbcd7seg**, en el ejemplo), precedido o seguido por un acrónimo o palabra que indica que se trata de un TB (**tb_decodbcd7seg**, **test_decodbcd7seg**, **decodbcd7seg_tb**, etcétera).

Realice la siguiente operación:

1. Introduzca, en el editor de ModelSim, el código comprendido entre las líneas 2 y 13 de la figura 9.

Declaración de señales y emplazamiento del DUT en el Cuerpo de Arquitectura

El código de la figura 10 muestra la cabecera del cuerpo de arquitectura, la sentencia de inserción del decodificador y la declaración de las señales que se conectan al mismo.

```
14
15  architecture test of tb_decodBCD7seg is
16      signal digBCD: std_logic_vector(3 downto 0); --Estímulo
17      signal segDisp: std_logic_vector(6 downto 0); --Respuesta
18  begin
19
20      --Modelo a verificar: decodificador BCD a 7 segmentos
21      dut: entity Work.decodBCD7seg(rtl)
22          port map(digBCD => digBCD,
23                  segDisp => segDisp);
24
25
```

Figura 10.- Emplazamiento del decodificador

Observe que como nombre de arquitectura se ha elegido **test**, para indicar que contiene un TB y no el modelo de un sistema –podría haberse elegido también **t_b**, **test_bench**, etcétera.

Las señales se declaran en la zona de declaración del Cuerpo de Arquitectura, que es el espacio comprendido entre la cabecera y **begin** (en esta zona pueden declararse diversos tipos de objetos: a lo largo de este curso, a medida que resulte necesario, irá conociendo algunos de ellos). Por estar declaradas ahí, las señales son visibles en todo el Cuerpo de Arquitectura, lo que quiere decir que les pueden asignar valor los procesos que se incluyan en la arquitectura y, también, que se pueden conectar al modelo que se emplaza en el TB.

En la sentencia de emplazamiento, la etiqueta elegida, **dut**, advierte sobre la condición del modelo (es un modelo bajo prueba). La identificación del modelo se realiza indicando su nombre jerárquico, que se construye enlazando la librería donde está almacenado con el nombre de su Declaración de Entidad y añadiendo, entre paréntesis, el nombre de la Arquitectura que modela su funcionamiento. (la librería es **Work**, la Entidad, **decodbcd7seg**, y la arquitectura, **rtl**). **Work** es una palabra

reservada en VHDL; hace referencia a la librería donde está almacenada la unidad de código en la que se emplea dicha palabra: como el decodificador y su TB van a quedar almacenados en la misma librería (la del proyecto actualmente activo), en el TB puede hacerse referencia a la misma como librería **Work**.

La lista de conexión define la conexión entre puertos del modelo y señales (estímulos y respuestas) declaradas en el TB. Observe que el tipo de datos de cada señal debe coincidir con el del puerto al que se conecta.

La creación del código de la figura 10 se reduce, en cualquier TB, al cumplimiento del siguiente conjunto de instrucciones:

- Creación de la cabecera del Cuerpo de Arquitectura incluyendo su nombre (**test** o similar) y el nombre de la Declaración de Entidad del TB.
- Declaración de tantas señales como puertos tenga el modelo que se desea verificar, asignando a cada una el mismo nombre y tipo de datos que el puerto correspondiente.
- Emplazamiento del modelo, refiriendo su nombre jerárquico y conectando cada puerto con la señal homónima declarada en el Cuerpo de Arquitectura del TB.

Realice la siguiente operación:

1. Introduzca, en el editor de ModelSim, el código comprendido entre las líneas 15 y 23 de la figura 10.

Codificación de las pruebas de simulación

Las pruebas de simulación se materializan asignando valor a señales (estímulos) en procesos que contienen sentencias **WAIT**. Esta tarea debe abordarse cuando se dispone de una especificación de las pruebas que se desea realizar. En el caso del decodificador BCD a 7 segmentos se debe generar la secuencia de estímulos que se representa en la figura 11.

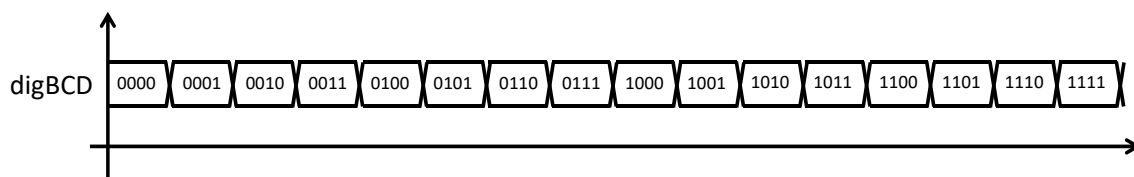


Figura 11.- Secuencia de estímulos

Teniendo en cuenta que cuando se ejecuta una sentencia **WAIT** en un proceso, la ejecución del proceso se suspende, la generación de la secuencia de estímulos se reduce a repetir, 16 veces, las siguientes operaciones:

- Se asigna valor al estímulo.
- Se suspende el proceso por el tiempo que se desee mantener el valor asignado.

En VHDL, la anterior pareja de operaciones se puede codificar mediante una sentencia de asignación de valor y una sentencia *WAIT FOR*, tal y como se indica en la figura 12.

```
Estimulo <= Valor;  
WAIT FOR tiempo_de_mantenimiento;
```

Figura 12.- Definición de un estímulo

Utilizando estas instrucciones, el código del proceso que materializa la prueba de funcionamiento del decodificador, generando la secuencia de estímulos de la figura 11, se muestra en la figura 13.

```
27  -- Generación de estímulos  
28  process  
29  begin  
30      digBCD <= "0000"; --En T = 0 ns  
31      wait for 100 ns;  
32      digBCD <= "0001"; --En T = 100 ns  
33      wait for 100 ns;  
34      digBCD <= "0010"; --En T = 200 ns  
35      wait for 100 ns;  
36      digBCD <= "0011"; --En T = 300 ns  
37      wait for 100 ns;  
38      digBCD <= "0100"; --En T = 400 ns  
39      wait for 100 ns;  
40      digBCD <= "0101"; --En T = 500 ns  
41      wait for 100 ns;  
42      digBCD <= "0110"; --En T = 600 ns  
43      wait for 100 ns;  
44      digBCD <= "0111"; --En T = 700 ns  
45      wait for 100 ns;  
46      digBCD <= "1000"; --En T = 800 ns  
47      wait for 100 ns;  
48      digBCD <= "1001"; --En T = 900 ns  
49      wait for 100 ns;  
50      digBCD <= "1010"; --En T = 1000 ns  
51      wait for 100 ns;  
52      digBCD <= "1011"; --En T = 1100 ns  
53      wait for 100 ns;  
54      digBCD <= "1100"; --En T = 1200 ns  
55      wait for 100 ns;  
56      digBCD <= "1101"; --En T = 1300 ns  
57      wait for 100 ns;  
58      digBCD <= "1110"; --En T = 1400 ns  
59      wait for 100 ns;  
60      digBCD <= "1111"; --En T = 1500 ns  
61      wait;  
62  end process;
```

Figura 13.- Estímulos para el TB del decodificador

En este código se ha elegido una duración de 100 ns para cada combinación de entrada (podría haberse elegido cualquier otra). Observe que la última sentencia *WAIT*, en la línea 61 de la figura 13, suspende definitivamente el proceso; si se hubiera puesto en su lugar una sentencia *WAIT FOR*, la secuencia de estímulos se repetiría periódicamente, de manera indefinida, puesto que, como ya sabe, la ejecución de sentencias en un proceso con sentencias *WAIT* es cíclica y continua, de modo que tras la ejecución de la última sentencia del proceso se continúa por la primera.

Realice la siguiente operación:

1. Añada al TB, con el editor de ModelSim, el código comprendido entre las líneas 27 y 62 de la figura 13 –para acelerar la edición utilice comandos de **Copy+Paste** y, si lo desea, no edite los comentarios VHDL.

Realice las siguientes operaciones:

1. Para completar el código del TB, añada al fichero el código de la figura 14.

```
64      end test;
```

Figura 14

2. Salve el fichero (P1.T7) y ordene su compilación (P1.T9). Si no ha cometido errores sintácticos en la edición del código, el Test Bench se almacenará en la librería de trabajo actual –si en la compilación se detecta algún error, intente corregirlo (P1.T13), pero solicite la ayuda de su profesor si no es capaz de solucionar el problema rápidamente.

En la figura 15 se muestra el código completo del TB del decodificador BCD a 7 segmentos.

```

1  -- Test-Bench de un decodificador
2  -- BCD a 7 segmentos realizado para
3  -- el tutorial de la actividad AIP3
4  -- (29-3-2011)
5
6  -- Cláusulas de visibilidad
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10
11 -- Declaración de Entidad sin puertos
12 entity tb_decodBCD7seg is
13 end entity;
14
15 architecture test of tb_decodBCD7seg is
16     signal digBCD: std_logic_vector(3 downto 0); --Estímulo
17     signal segDisp: std_logic_vector(6 downto 0); --Respuesta
18 begin
19
20 --Modelo a verificar: decodificador BCD a 7 segmentos
21     dut: entity Work.decodBCD7seg(rtl)
22         port map(digBCD => digBCD,
23                 segDisp => segDisp);
24
25
26
27 -- Generación de estímulos
28 process
29 begin
30     digBCD <= "0000"; --En T = 0 ns
31     wait for 100 ns;
32
33     digBCD <= "0001"; --En T = 100 ns
34     wait for 100 ns;
35     digBCD <= "0010"; --En T = 200 ns
36     wait for 100 ns;
37     digBCD <= "0011"; --En T = 300 ns
38     wait for 100 ns;
39     digBCD <= "0100"; --En T = 400 ns
40     wait for 100 ns;
41     digBCD <= "0101"; --En T = 500 ns
42     wait for 100 ns;
43     digBCD <= "0110"; --En T = 600 ns
44     wait for 100 ns;
45     digBCD <= "0111"; --En T = 700 ns
46     wait for 100 ns;
47     digBCD <= "1000"; --En T = 800 ns
48     wait for 100 ns;
49     digBCD <= "1001"; --En T = 900 ns
50     wait for 100 ns;
51     digBCD <= "1010"; --En T = 1000 ns
52     wait for 100 ns;
53     digBCD <= "1011"; --En T = 1100 ns
54     wait for 100 ns;
55     digBCD <= "1100"; --En T = 1200 ns
56     wait for 100 ns;
57     digBCD <= "1101"; --En T = 1300 ns
58     wait for 100 ns;
59     digBCD <= "1110"; --En T = 1400 ns
60     wait for 100 ns;
61     digBCD <= "1111"; --En T = 1500 ns
62     wait;
63 end process;
64 end test;
```

Figura 15.- Test-Bench del decodificador

Ejecución de una simulación

Una vez construido el modelo de la prueba de funcionamiento, el Test-Bench VHDL, hay que ejecutarlo, es decir, realizar una **simulación**. Para realizar una simulación es necesario procesar el Test-Bench, y el modelo insertado en él, para generar una red de procesos y, a partir de esta, un código ejecutable por el procesador sobre el que funciona el entorno de simulación –en última instancia es el microprocesador del PC en el que está instalado ModelSim el que tiene que ejecutar el *programa* generado a partir del Test-Bench. No se va a entrar aquí en los entresijos del proceso de **elaboración** del código ejecutable de un Test-Bench, pero sí en cómo se ordena la realización de esta tarea en Modelsim y en los resultados que se pueden obtener al hacerlo.

Procedimiento T.2: Arranque de la simulación

Realice las siguientes operaciones:

1. En la ventana principal de ModelSim, seleccione la pestaña **Library**.
2. Expanda la librería **Work**, sitúe el cursor del ratón sobre el icono del TB del decodificador (figura 16) y realice una doble pulsación con el botón izquierdo del ratón.

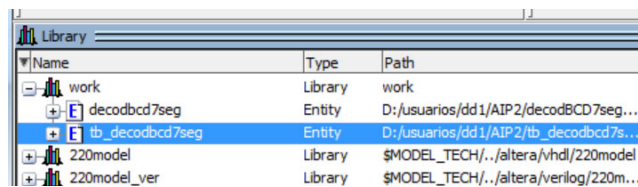


Figura 16.- Arranque de simulación

Al realizar esta acción se ordena la creación del código *ejecutable* de simulación y, durante este proceso, pueden detectarse *errores* que impidan obtener dicho ejecutable (por ejemplo, que no exista el modelo VHDL que se inserta en el TB); si se encuentran errores, se diagnostican y reportan en la ventana **Transcript** –en este caso, deberá corregirlos y volver a arrancar la simulación. Si el ejecutable de simulación se genera sin problemas, el aspecto de la ventana principal de ModelSim debe cambiar y asemejarse al que muestra la figura 17.

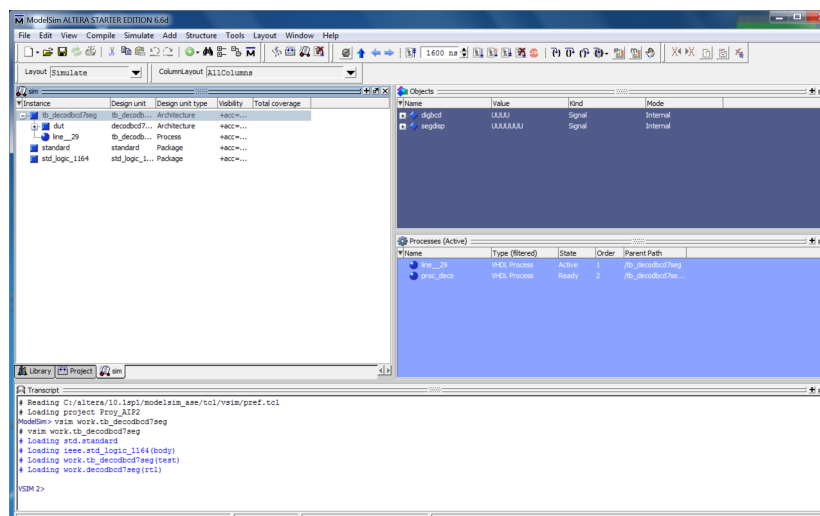


Figura 17.- Entorno de Simulación

Puede observar que cambian los íconos que aparecen en la barra de herramientas de la ventana de ModelSim y que se añade una nueva pestaña, **Sim**, a las dos previamente existentes (**Library** y **Project**). En esta ventana se representa la jerarquía de diseño con que se ha elaborado el ejecutable de simulación (figura 18).

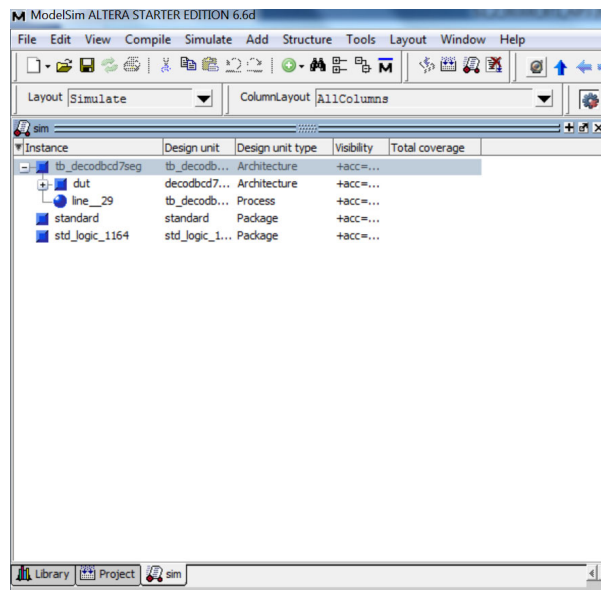


Figura 18.- Pestaña **Sim**

Los íconos de la pestaña **Sim** identifican unidades de código VHDL y procesos: permiten abrir –en modo *lectura*- los ficheros donde están almacenados.

Procedimiento T.3: Acceso al código durante la simulación

Realice la siguiente operación:

1. Sitúe el cursor del ratón sobre el icono circular etiquetado como **line_29** y realice una doble pulsación con el botón izquierdo del ratón.

Observe que se abre la ventana de edición de código (figura 19). Aunque intente editarlo no podrá hacerlo. En este modo el código se muestra para que pueda insertar *breakpoints* y seguir el flujo de ejecución en modo *paso a paso*.

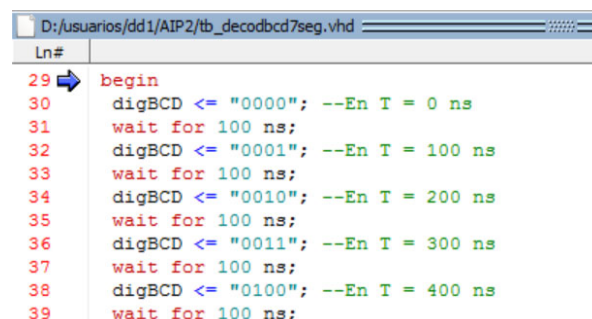


Figura 19.- Ventana de código

La flecha azul que apunta a la línea 29 en la ventana de código de la figura 19 marca la primera instrucción a ejecutar cuando comienza la simulación (en T = 0).

Además de la pestaña **Sim**, y por efecto de haber arrancado la simulación, en la ventana de ModelSim pueden observarse otras dos ventanas: las ventanas **Object** y **Processes** (figura 20).

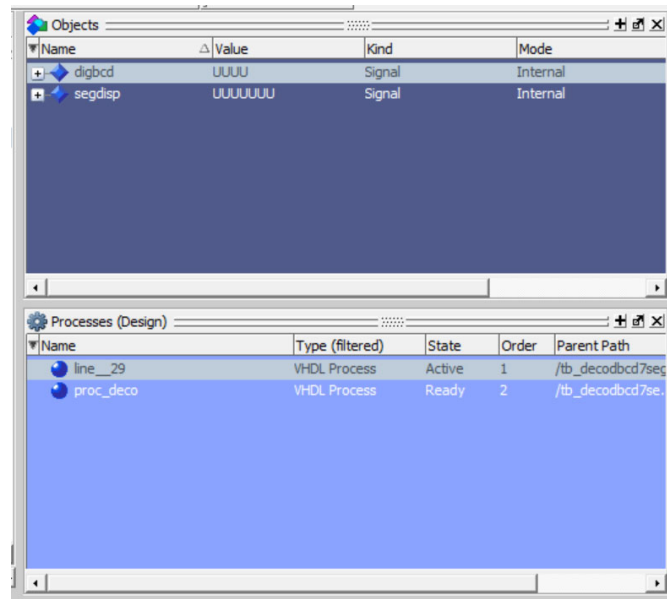


Figura 20.- Ventanas **Objects** y **Processes**

En la ventana **Processes** se listan los procesos que forman parte de la jerarquía del Test-Bench. En el ejemplo de este tutorial hay dos: **proc_deco**, que forma parte del modelo del decodificador BCD a 7 segmentos y el proceso que genera los estímulos en el TB, que se denomina **line__29** –en el código no tiene nombre, pero para poder identificarlo el simulador le asigna uno automáticamente. Esta ventana permite saber en qué proceso se encuentra el flujo de ejecución de la simulación.

En la ventana **Objects** se listan los objetos declarados en la unidad de código seleccionada en la pestaña **Sim**. En la figura 20 se listan las dos señales declaradas en el TB del decodificador, **digbcd** y **segdisp**. En la ventana se indica, además del nombre del objeto, su clase (señal) y su valor actual: en la figura 20 se puede observar que el valor inicial de las dos señales es ‘U’.

Valor inicial de los objetos VHDL

*La primera operación que se realiza en la ejecución de una simulación, en $T=0$, consiste en inicializar todos los **objetos** que forman parte de la **jerarquía** del Test-Bench. Debe entenderse aquí que:*

- Un objeto es cualquier elemento del modelo que puede tomar valor, como por ejemplo una señal (según avance el curso irá conociendo nuevos tipos de objetos). Una característica importante de los objetos VHDL es que tienen que tener asociado un tipo de datos, porque VHDL es un lenguaje que dispone de un control riguroso de los tipos de datos: sólo permite que a un objeto se le asigne un valor que pertenezca a su tipo de datos –en los lenguajes que no tienen esta característica, cuando a un objeto se le asigna un valor que no es de su tipo de datos se convierte de manera automática a otro que sí lo es.*
- La jerarquía del Test-Bench está compuesta por el propio Test-Bench y por las unidades de código donde se codifica el modelo que se inserta en él.*

Pues bien, el valor inicial que se asigna a un objeto es:

1. El que se indica en la declaración del objeto. La inicialización de un objeto en su declaración es optativa en el caso de las señales –en general, las señales no suelen inicializarse en la declaración.
2. Si el objeto no se inicializa en la declaración, se aplica un mecanismo de inicialización por defecto; es pronto para explicar este mecanismo, pero resulta suficiente con saber, de momento, dos cosas sobre él: que el valor inicial que asigna a un objeto depende de su tipo de datos y que cuando el tipo de datos es **std_logic** el valor inicial es 'U' –si el tipo es **std_logic_vector**, el valor inicial es un string de 'U's: "UUUU...UU".

La semántica del valor 'U' en el tipo **std_logic** es precisamente un valor metalógico que indica que el objeto mantiene su valor de inicialización en $T=0$, es decir, que aún no se le ha asignado valor en el transcurso de la simulación.

La ventana **Objects** se puede utilizar para preparar el visor de formas de onda que se empleará para revisar los resultados de la simulación.

Procedimiento T.4: Preparación del visor de formas de onda

Realice la siguiente operación:

1. Seleccione con el ratón, en la pestaña **Sim**, el fichero del TB.

Esta acción provoca que en la ventana **Objects** se listen los objetos declarados en el TB: las señales que hacen las veces de estímulos y respuestas de la simulación.

2. Seleccione las dos señales que aparecen en la ventana **Objects**: **digbcd** y **segdisp**.
3. Con las señales seleccionadas, active el botón derecho del ratón y, en el menú que se despliega (figura 21), seleccione la opción **Add -> To Wave -> Selected Signals**.

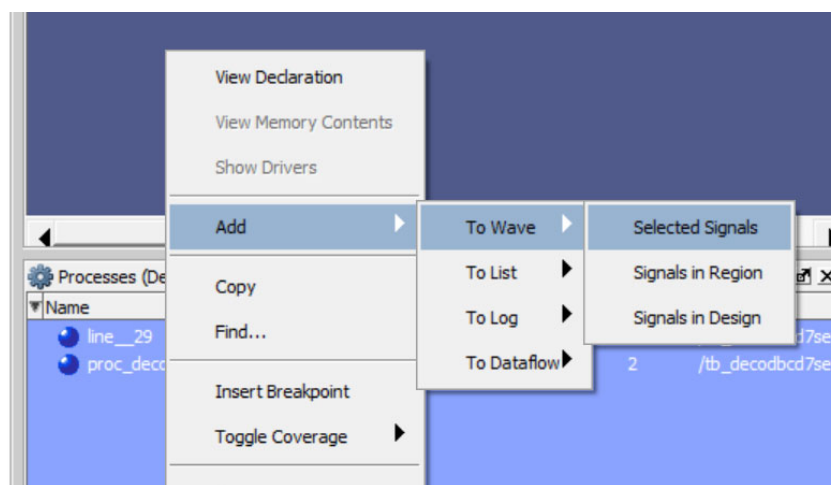


Figura 21.- Preparación del visor de formas de onda

Observe que aparece una nueva ventana, la del visor de formas de onda, en el área de trabajo de ModelSim.

4. Desprenda la ventana y maximícela (figura 22).

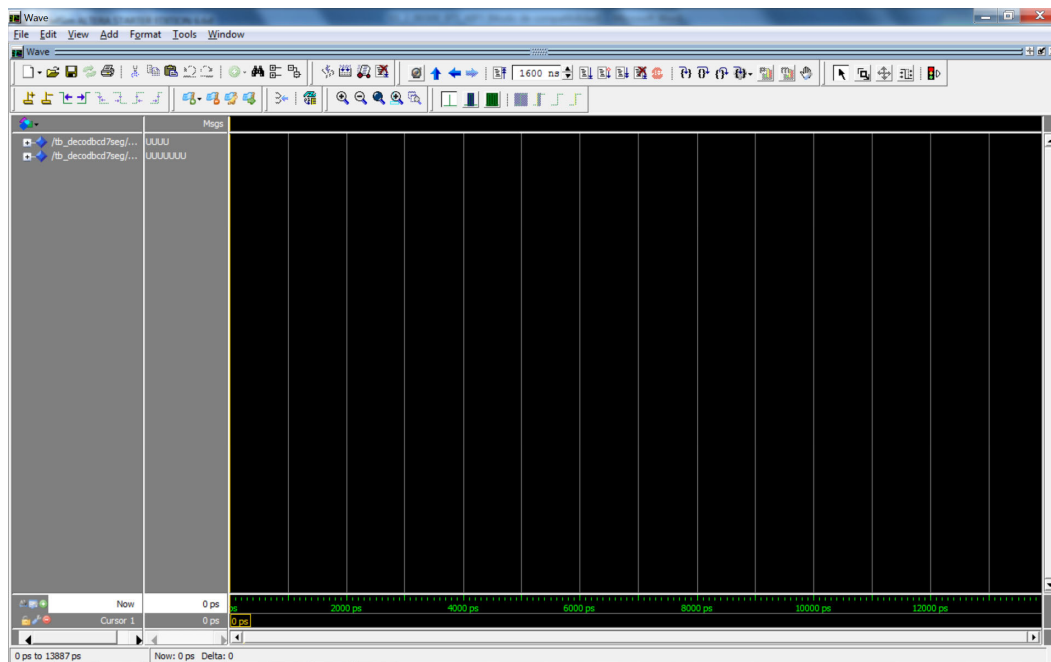


Figura 22.- Visor de formas de onda

En la ventana pueden distinguirse las siguientes zonas:

- La zona de menús y herramientas de acceso directo, en la parte superior de la ventana.
- La zona de visualización, donde se muestran las señales y dónde se representa el resultado de la simulación, ocupa la mayor parte de la ventana y tiene tres campos donde se muestra el nombre de las señales, su valor y la forma de la señal tras la ejecución de la simulación.
- El eje de tiempos, en la parte inferior de la pantalla.
- La zona de mensajes, situada entre la zona de visualización y la zona de herramientas

A la izquierda del eje de tiempos hay seis iconos que dan acceso a opciones de configuración de uso frecuente. Vamos a utilizar dos de ellos para acondicionar la información que muestra la ventana.

Procedimiento T.5: Visor de formas de onda: nombres simples y completos de señales

Al arrancar el visor de formas de onda las señales que se muestran se denotan por su nombre completo, que incluye el de la unidades de código en la jerarquía del TB donde están declaradas (figura 23).

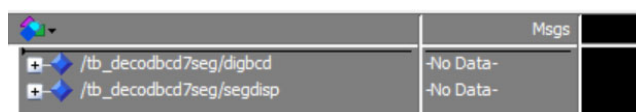


Figura 23.- Nombres jerárquicos

1. Pulse el botón izquierdo del ratón cuando el cursor esté situado sobre el icono remarcado en la figura 24 –está situado en la esquina inferior izquierda de la ventana.

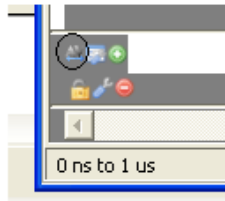


Figura 24

Observe que los nombres jerárquicos de las señales son sustituidos por los simples –esto facilita la identificación de las señales; para recuperar el nombre jerárquico basta con volver a pulsar sobre el mismo icono –cada vez que se pulsa conmuta el modo de representación.

Procedimiento T.6: Visor de formas de onda: cambio de las unidades de tiempo de simulación

Si observa la escala de tiempos del visor de formas de onda, podrá comprobar que, por defecto, el tiempo de simulación se expresa en picosegundos.

1. Sitúe el cursor del ratón sobre el icono que se encuentra remarcado en la figura 25 y pulse el botón izquierdo del ratón.

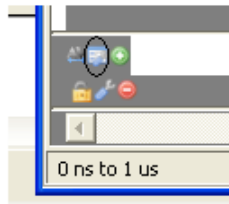


Figura 25

Aparecerá una ventana como la de la figura 26.

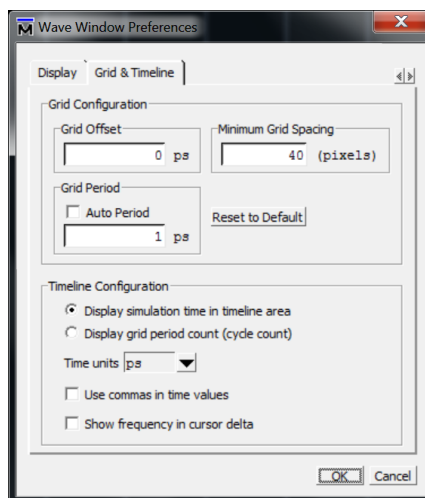


Figura 26.- Configuración del visor

2. En la lista desplegable **Time units**, seleccione nanosegundos (**ns**) y pulse **OK**.

Ejecución de una simulación

La ejecución de una simulación puede ordenarse desde la ventana principal de ModelSim o desde la ventana del visor de formas de onda –en ambos casos se sigue el mismo procedimiento y se utilizan los mismos iconos, situados casi exactamente en la misma zona de las ventanas. Aquí se va a describir este proceso referido a la ventana del visor de formas de onda, que es desde donde se suele *lanzar*. Antes de ordenar la ejecución de la simulación se suele fijar el tiempo total de simulación.

Procedimiento T.7: Visor de formas de onda: establecimiento del tiempo de simulación

Si observa la barra de herramientas del visor de formas de onda, observará que, aproximadamente en el centro de la barra, existe una *casilla* editable que contiene un valor numérico y un factor de escala de tiempos. Este valor es el lapso de tiempo que avanzará la ejecución de la simulación cuando se dé la orden correspondiente. La prueba codificada en el TB dura 1600 ns; se va a elegir, por ejemplo, una duración total de la simulación de 2000 ns.

1. Escriba en ese campo: **2000 ns** (figura 27)

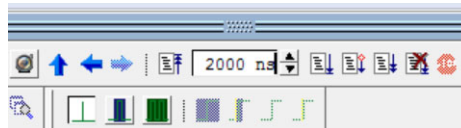


Figura 27.- Tiempo total de simulación

Procedimiento T.8: Visor de formas de onda: ejecución de la simulación

1. Pulse el botón izquierdo del ratón sobre el icono situado a la derecha de la casilla donde ha indicado la duración de la simulación (figura 27) –el símbolo del icono representa una hoja de papel con una flecha azul, boca abajo, a la derecha.

Observe que, en breves instantes se completa la simulación y aparece una representación gráfica de las señales (figura 28). En la zona del eje de tiempos se muestra el tiempo de simulación actual: 2000 ns.

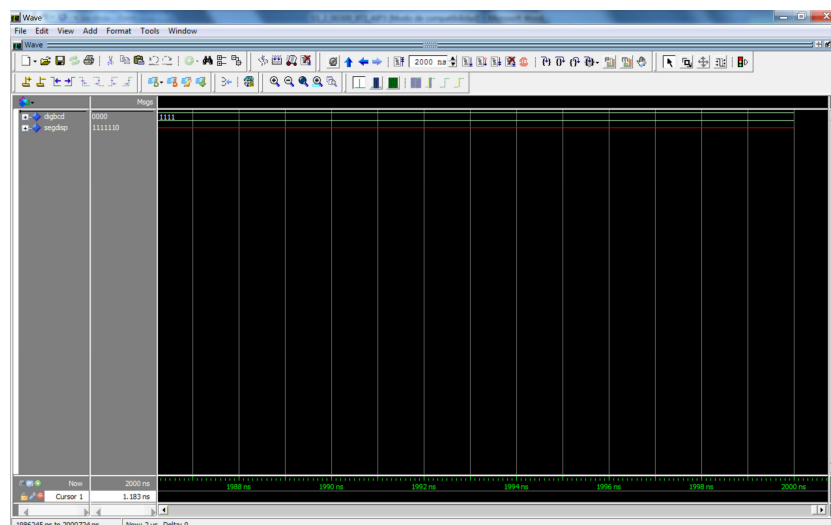


Figura 28.- Simulación completa

Procedimiento T.9: Visor de formas de onda: zoom completo y cursor

Para revisar el resultado de la prueba de simulación hay que realizar un *zoom* que muestre los resultados entre 0 y 2000 ns.

Realice la siguiente operación:

1. Con la ventana del visor de formas de onda activa, pulse la tecla **F** –efe mayúscula.

El visor deberá adoptar una apariencia como la que se muestra en la figura 29.

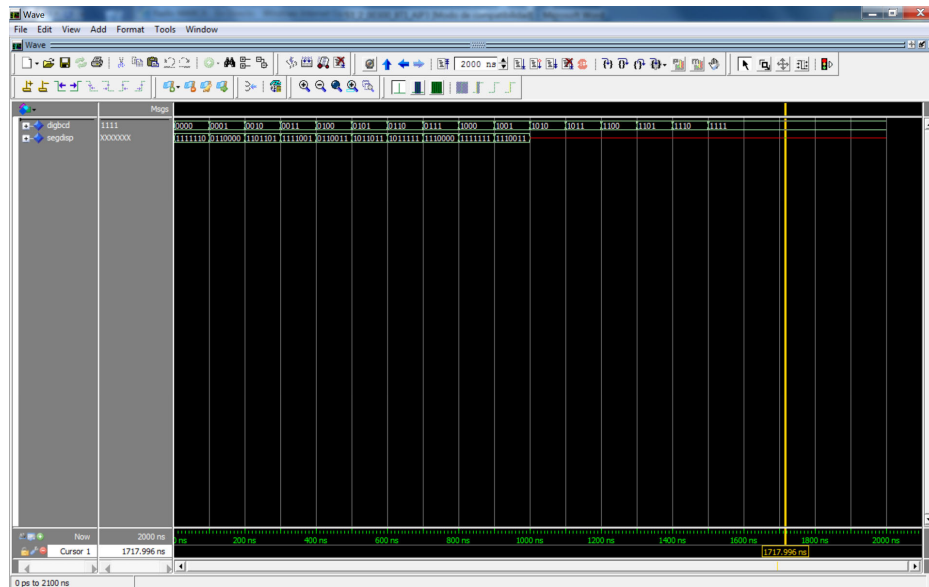


Figura 29.- Visión de la simulación completa

2. Sitúe el cursor del ratón en un punto del área de visualización de formas de onda y pulse el botón izquierdo del ratón.

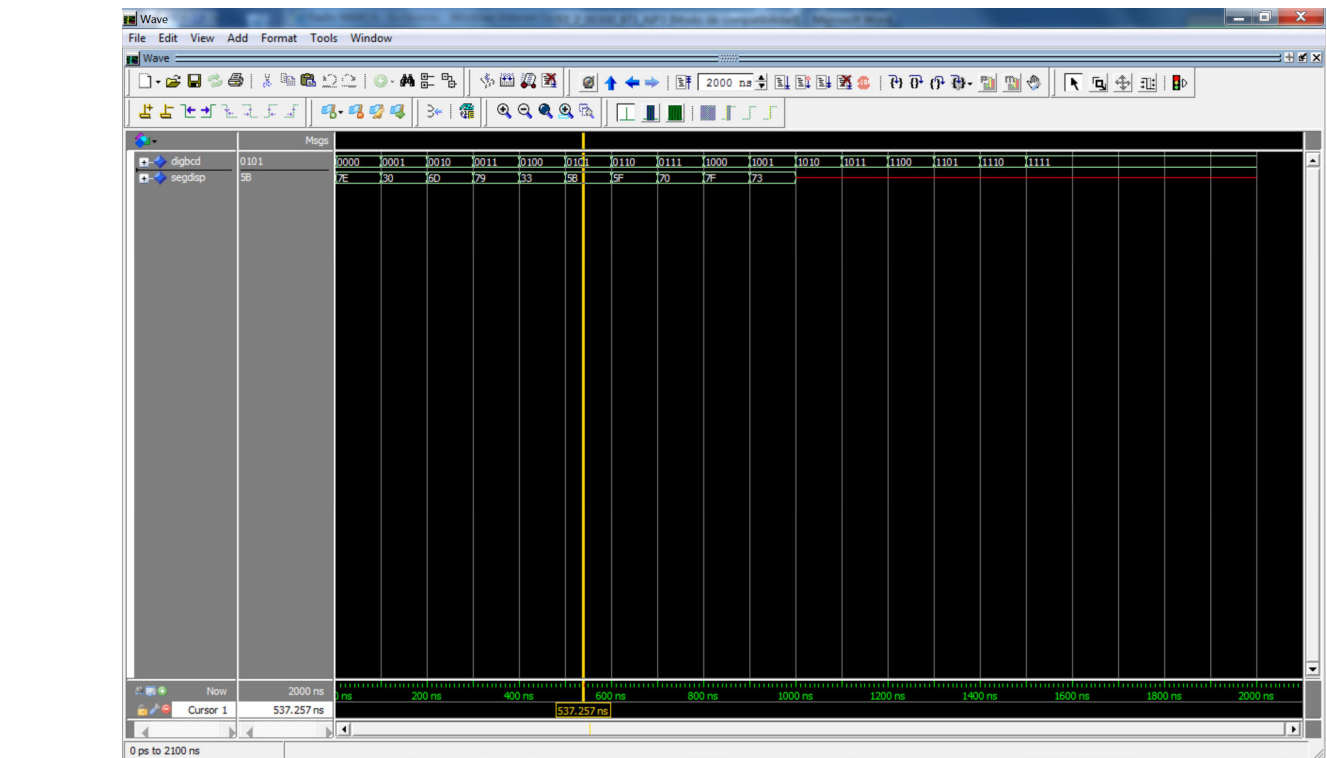
Observe que aparece un cursor amarillo, con una marca que indica el tiempo de simulación donde está situado, y que el valor de las señales del visor, que se indica a la derecha del nombre, se corresponde con el que tienen las señales en la posición del cursor –puede comprobarlo variando la ubicación del cursor.

Procedimiento T.10: Visor de formas de onda: Cambio del formato de representación de valores

Para facilitar la verificación de los resultados de la simulación es conveniente evitar que los valores de los buses con un gran número de bits se representen en binario natural. El visor de formas de onda permite que elijamos entre diversos formatos de presentación de los valores de los buses.

Realice las siguientes operaciones:

1. Seleccione la señal **segdisp**, situando el cursor del ratón sobre el nombre de la señal y pulsando el botón izquierdo.
2. Pulse ahora el botón derecho del ratón y seleccione la opción **Radix -> Hexadecimal**.



DÍGITO	0	1	2	3	4	5	6	7	8	9
HEXA	0x7E	0x30	0x6D	0x79	0x33	0x5B	0x5F	0x70	0x7F	0x73

Ejecución paso a paso de la simulación

Realice la siguiente operación:

1. En la ventana principal del entorno ModelSim, seleccione, en el menú **Simulate**, la opción **Run ->Restart...**

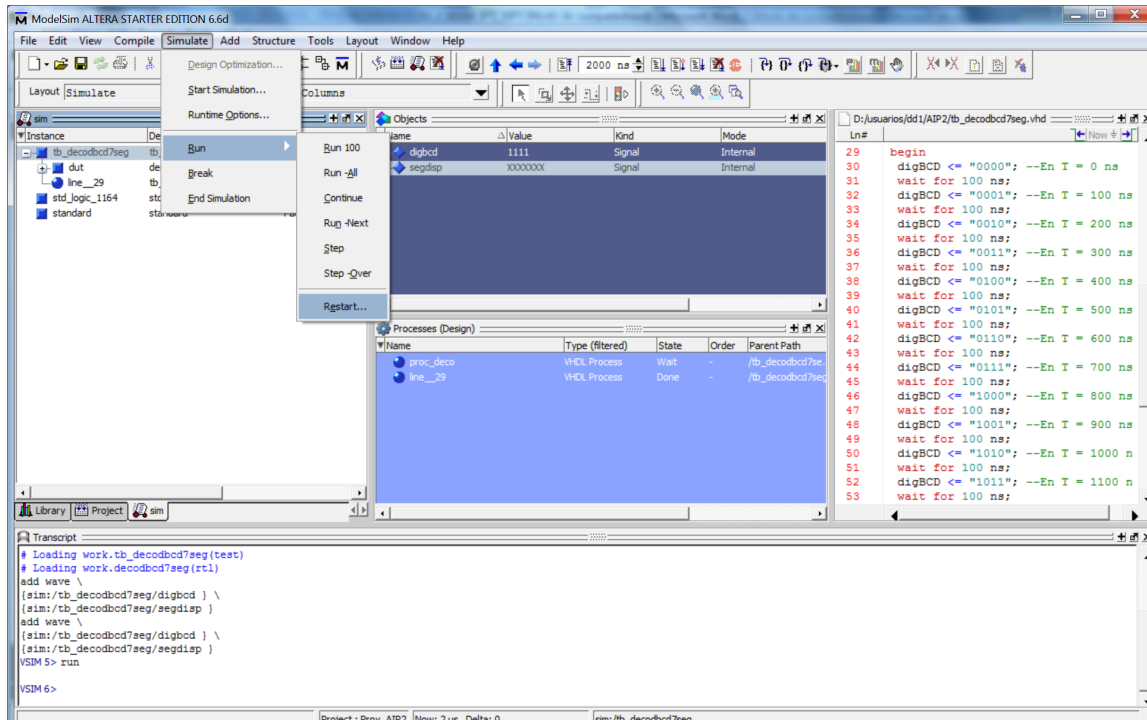


Figura 31.- Reinicio de la simulación

2. En la ventana que aparece (figura 32), pulse el botón **OK**.

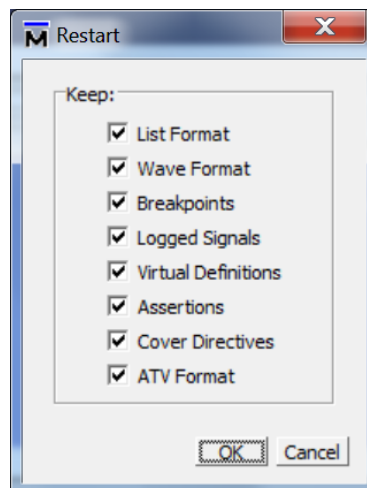


Figura 32

Esta acción que acaba de realizar reinicia la simulación y sitúa el tiempo de simulación en $T=0$. Las siguientes operaciones van a servir para conseguir una disposición de la ventana del entorno que le facilite el seguimiento de la ejecución paso a paso de la simulación.

Realice las siguientes operaciones:

1. En la ventana **Processes** realice una pulsación simple, con el botón izquierdo del ratón, tras situar el cursor sobre el proceso **line_29**.

La ventana del entorno deberá adoptar un aspecto similar al que se observa en la figura 33.

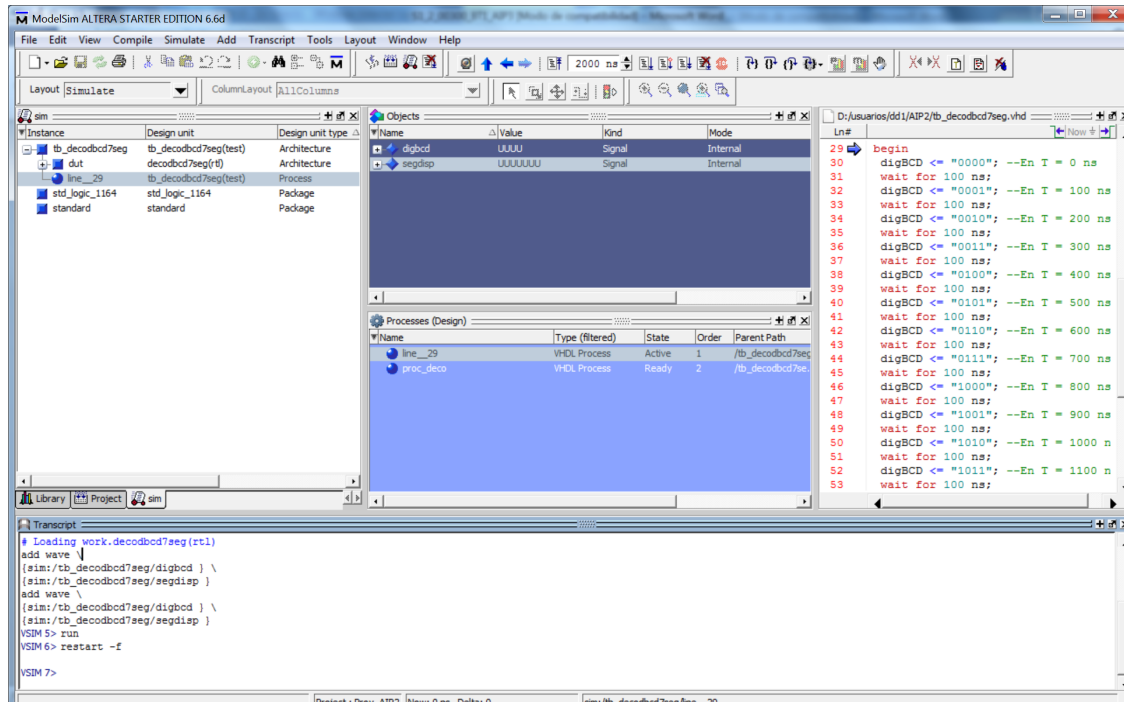


Figura 33

Inicio de la simulación

Al iniciarse la simulación, en $T = 0$, se realizan en orden las siguientes operaciones:

1. Primero se da valor inicial a todos los objetos de la jerarquía.
2. Después se ejecutan todos los procesos de la jerarquía.

En el ejemplo de este tutorial, se inicializan las señales **digBCD** y **segdisp** (con todos sus valores a 'U') y a continuación se ejecutan los dos procesos que forman parte de la jerarquía del TB: el que modela el funcionamiento del decodificador y el que asigna valor a los estímulos.

En VHDL el modelo de ejecución de los procesos es concurrente (los procesos se ejecutan en paralelo). En la práctica, esta condición implica que si en un instante del tiempo de simulación (como ahora en $T=0$) deben ejecutarse varios procesos, dicha ejecución debe equivaler a la ejecución simultánea e independiente de los procesos.

En realidad los procesos se ejecutan secuencialmente y el orden en que se ejecutan no está predeterminado, pero el resultado que se obtiene tras la ejecución de los procesos es independiente del orden de ejecución y equivale al que se obtendría con su ejecución en paralelo: esta característica se deriva del mecanismo de actualización de señales de los simuladores VHDL.

Actualización de señales y ciclo de simulación

Las señales cambian de valor por la ejecución de las sentencias de asignación que forman parte de los procesos VHDL. Pero la ejecución de una sentencia de asignación no tiene un efecto instantáneo sobre la señal a la que asigna valor:

- Una sentencia de asignación proyecta un cambio de valor para una señal.
- El cambio de valor se verifica cuando el proceso que contiene la sentencia de asignación, y cualquier otro concurrente con él, se suspenden –entiéndase: cuando han completado su ejecución y se encuentran suspendidos⁴.

Este mecanismo se ilustra en la figura 34. La ejecución paralela o secuencial (en cualquier orden) de estos tres procesos arroja siempre los mismos resultados: los tres procesos proyectan tres transacciones sobre las señales A, B y C. Cuando la ejecución de los tres procesos se ha completado y los procesos pasan a estar suspendidos, se efectúa la actualización de las señales.

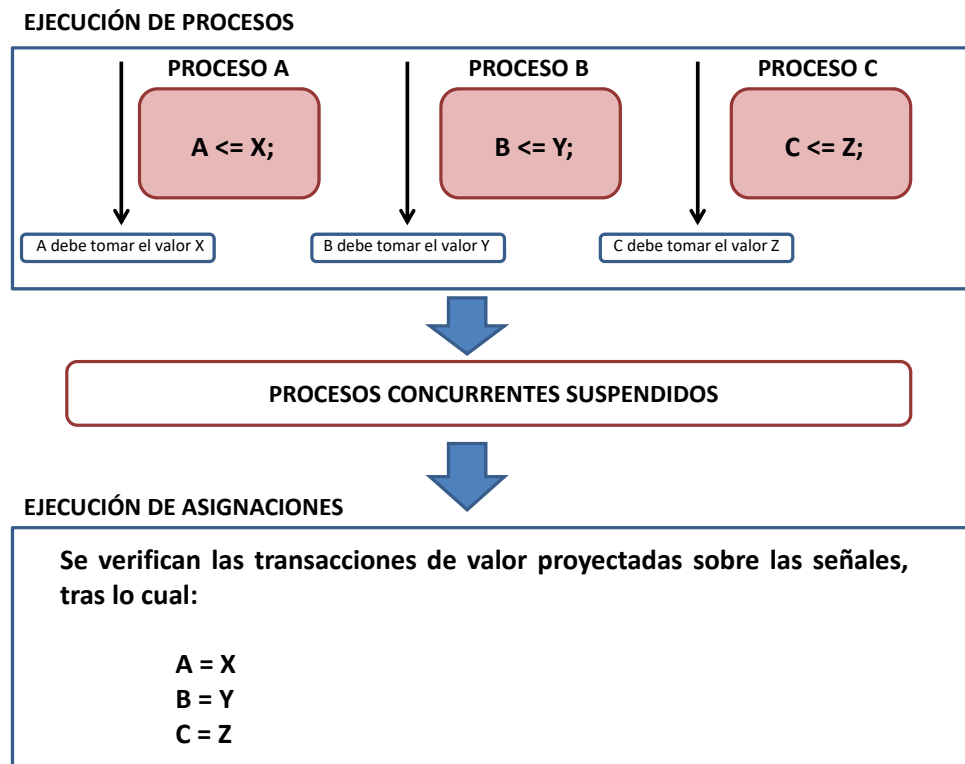


Figura 34.- Ejecución concurrente de procesos

El mecanismo ilustrado en la figura 34 se conoce como ciclo de simulación. Los ciclos de simulación ocurren en determinados instantes del tiempo de simulación (para $T=0$, por ejemplo) y es la tarea que de manera repetitiva ejecuta un simulador VHDL durante la ejecución de una simulación. El ciclo de simulación consta de dos fases: en la primera se ejecutan procesos, en la segunda se actualizan las señales para las que se hayan realizado proyecciones de asignación de valor en la fase anterior.

⁴ No se ha tratado aún el modelado de retardos; cuando llegue el momento y se trate este asunto, se ampliará la descripción del modo de actualización de las señales VHDL.

Observe que como consecuencia del modo en que se actualizan las señales, tras la ejecución de una sección de código como la siguiente:

```
A <= '0';  
B <= A;
```

El valor que se asigna a B no es '0', porque la sentencia `A <= '0';` no modifica el valor de A, simplemente proyecta una asignación sobre A. El valor que se proyecta para B es el de A al comienzo de la ejecución del proceso.

Va a comprobar ahora que el mecanismo descrito es el que guía el curso de las simulaciones. El estado inicial de la simulación es el siguiente:

- Tiempo de simulación: $T = 0$;
- Valor de las señales: el inicial, todas valen 'U'
- Primera sentencia a ejecutar: la primera del proceso del TB

Para acceder a los iconos que permiten controlar la ejecución *paso a paso* de una simulación primero debe *desanclar* la ventana *waves* utilizando el botón *dock/undock* que está en la esquina superior izquierda de la ventana (figura 35, izquierda).

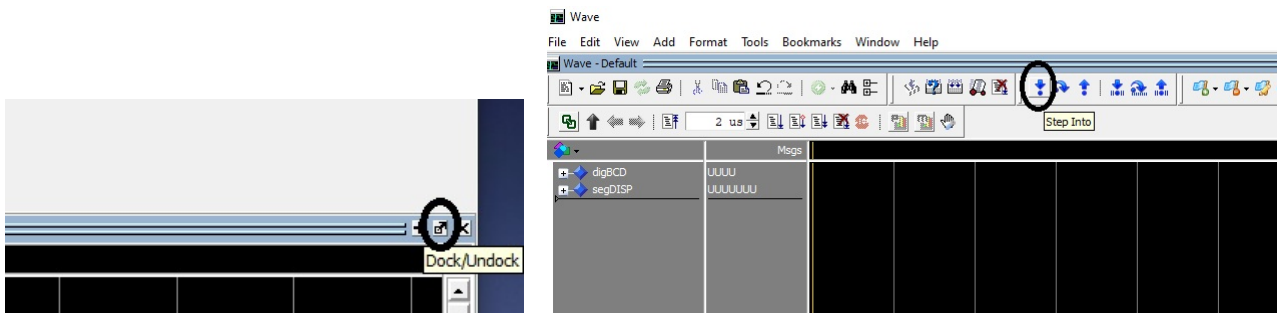


Figura 35

A partir de ahora, cada vez que se le indique que ordene una ejecución paso a paso (se resumirá como p-a-p), pulse el botón izquierdo del ratón con el cursor situado sobre el icono *step into* (figura 35, derecha).

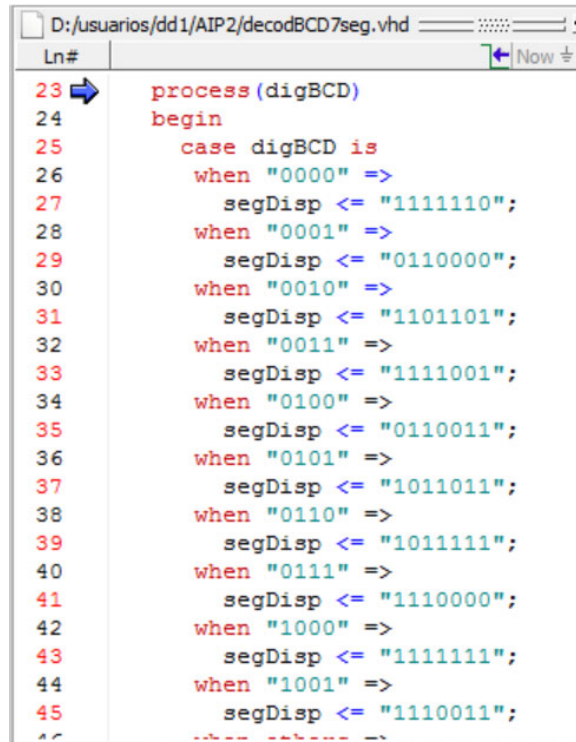
Realice las siguientes operaciones:

1. Ordene ejecuciones p-a-p hasta que la flecha azul, que marca la próxima sentencia a ejecutar se sitúe en la línea 31 (figura 36).

```
27  -- Generación de estímulos  
28  process  
29  begin  
30      digBCD <= "0000"; --En T = 0 ns  
31  →  wait for 100 ns;  
32      digBCD <= "0001"; --En T = 100 n
```

Figura 36

2. Observe, en la ventana **Objects**, que aunque se ha ejecutado una sentencia de asignación de valor a la señal **digBCD**, ésta no se ha actualizado con el valor asignado ("0000") –pero el simulador ha *apuntado* que tiene que actualizar la señal en la segunda fase del ciclo de simulación actual.
3. Ordene otra ejecución p-a-p. Esta orden provoca que se ejecute una sentencia WAIT (línea 31 de la figura 36): el proceso se suspende por 100 ns; se reiniciará su ejecución, por tanto, en $T=100\text{ns}$.
Observe que el flujo de ejecución pasa al proceso que modela el funcionamiento del decodificador (figura 37).

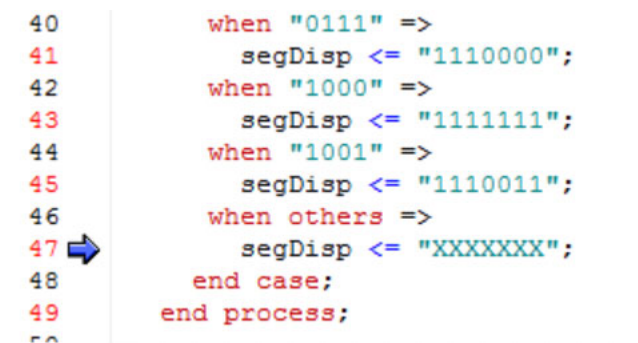


```
Ln# |
23 | process(digBCD)
24 | begin
25 |   case digBCD is
26 |     when "0000" =>
27 |       segDisp <= "1111110";
28 |     when "0001" =>
29 |       segDisp <= "0110000";
30 |     when "0010" =>
31 |       segDisp <= "1101101";
32 |     when "0011" =>
33 |       segDisp <= "1111001";
34 |     when "0100" =>
35 |       segDisp <= "0110011";
36 |     when "0101" =>
37 |       segDisp <= "1011011";
38 |     when "0110" =>
39 |       segDisp <= "1011111";
40 |     when "0111" =>
41 |       segDisp <= "1110000";
42 |     when "1000" =>
43 |       segDisp <= "1111111";
44 |     when "1001" =>
45 |       segDisp <= "1110011";
```

Figura 37

4. Ordene dos ejecuciones p-a-p.

La sentencia CASE evalúa digBCD ("UUUU") y deriva la ejecución a la última rama (figura 38), donde se asigna –se proyecta la asignación- a **segDisp** del valor "XXXXXXXX".



```
40 |     when "0111" =>
41 |       segDisp <= "1110000";
42 |     when "1000" =>
43 |       segDisp <= "1111111";
44 |     when "1001" =>
45 |       segDisp <= "1110011";
46 |     when others =>
47 |       segDisp <= "XXXXXXXX";
48 |   end case;
49 | end process;
```

Figura 38

5. Ordene otra ejecución p-a-p.

El cursor se sitúa en la última línea del proceso. Cuando se ordene el siguiente paso de ejecución, se habrá completado la primera fase del ciclo de simulación y podrá observar el efecto de la segunda fase: la actualización de las asignaciones proyectadas.

6. Ordene otra ejecución p-a-p.
7. Compruebe, en la ventana **Objects**, el valor al que se han actualizado las señales.

Ciclos delta

La secuencia de ejecución que acaba de verificar ilustra el desarrollo de un ciclo de simulación. Como ha comprobado, en la segunda parte del ciclo se actualizan señales, por lo que en dichas señales se pueden producir cambios de valor –habrá eventos en esas señales, salvo que se dé la casualidad de que los valores proyectados para las señales sean los mismos que ya tenían-, y como ya sabe los procesos que modelan el funcionamiento de los sistemas digitales se ejecutan cuando se producen eventos en las señales que forman parte de su lista de sensibilidad. Por ello las actualizaciones de señales que se producen en la segunda parte de un ciclo de simulación pueden dar lugar a un nuevo ciclo que se sucede sin que avance el tiempo de simulación. Estos ciclos se denominan **ciclos delta**. En la figura 39 se ilustra este mecanismo.

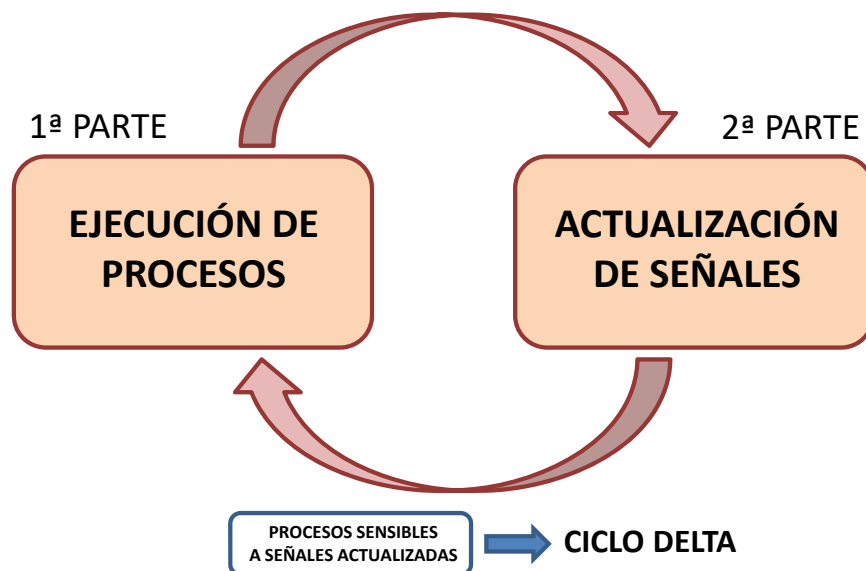
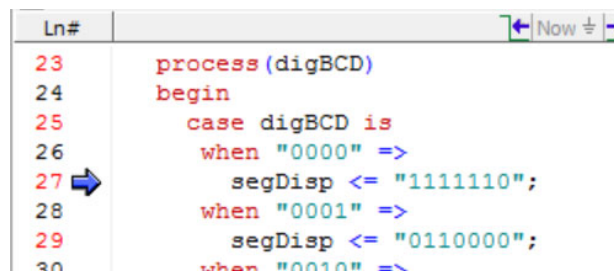


Figura 39.- Ciclos Delta

En el ejemplo del tutorial, la señal **digBCD** se ha actualizado y ha cambiado de valor (de “UUUU” a “0000”); como el proceso que modela el funcionamiento del decodificador es sensible a **digBCD** se desencadena un nuevo ciclo de simulación, un **ciclo delta**, pero ahora, en la primera parte del ciclo de simulación sólo se ejecutará el proceso **proc_deco** –porque es el único *sensible* a **digBCD**.

Realice las siguientes operaciones:

1. Observe que el cursor de ejecución está en el proceso **proc_deco**.
2. Ordene ejecuciones p-a-p hasta que el cursor se sitúe en la línea 27 (figura 40).



```
Ln# | Now
23  | process(digBCD)
24  | begin
25  |   case digBCD is
26  |     when "0000" =>
27  |       segDisp <= "1111110";
28  |     when "0001" =>
29  |       segDisp <= "0110000";
30  |     when "0010" =>
```

Figura 40

3. Ordene otras dos ejecuciones p-a-p para que se complete la ejecución del proceso y finalice la primera parte del ciclo de simulación y, además, se actualice la señal **segDisp** con el valor asignado en la línea 27, “1111110”.

Puede observar los valores actualizados de las señales en la ventana **Objects**.

Avance de la simulación

*Como no hay ningún proceso sensible a **segDisp**, no puede haber nuevos ciclos de simulación en el tiempo de simulación $T=0$, de modo que, si existe algún proceso suspendido por la existencia de una sentencia **WAIT FOR**, el simulador avanzará el tiempo de simulación hasta aquel instante en que deba continuar con la ejecución de ese proceso, instante en el que se iniciará la ejecución de un nuevo ciclo de simulación –si no hubiera ningún proceso suspendido por una sentencia **WAIT FOR**, la simulación habrá terminado.*

En el ejemplo del tutorial, el proceso que asigna valor a los estímulos está suspendido por 100 ns, de modo que el simulador avanzará 100 ns el tiempo de simulación y continuará ejecutando dicho proceso.

Realice las siguientes operaciones:

1. Ordene una ejecución p-a-p.

Observe que el cursor de ejecución vuelve a la sentencia **WAIT** donde se había suspendido el proceso. El tiempo de simulación es ahora 100ns. Puede comprobarlo en el visor de formas de onda.

2. Ordene ejecuciones p-a-p hasta que el proceso se suspenda por 100 ns en la línea 33.

Tras suspenderse el proceso, y haberse completado, por tanto, la primera parte del ciclo de simulación, se actualiza **digBCD** con el valor proyectado en la sentencia de asignación de la línea 32 (“0001”), y se desencadena un ciclo delta de simulación en el que se ejecutará el proceso que modela al decodificador.

3. Ordene la ejecución p-a-p del proceso **proc_deco**.

Observará que se completa la ejecución, asignándose ahora a la salida el valor asociado a **digBCD** = “0001”. Nuevamente se habrá agotado la ejecución de ciclos de simulación para el instante de tiempo actual y el simulador avanzará el tiempo otros 100 ns, para continuar con la ejecución del proceso que asigna valor a los estímulos.

Puede inferir ya cuál va a ser la rutina de ejecución de este TB:

- Se ejecuta un ciclo de simulación que da lugar a la actualización de **digBCD** con un nuevo valor y a la suspensión del proceso que asigna valor a los estímulos por 100 ns.
 - Como consecuencia de la actualización de **digBCD** se desencadena un ciclo delta de simulación en el que el proceso que modela al decodificador actualiza el valor de salida; tras ejecutarse el proceso y actualizarse **segdisp**, se avanza el tiempo de simulación para que se repita nuevamente la misma rutina de ejecución.
4. Ordene ejecuciones p-a-p para observar el desarrollo de la simulación hasta que se alcance el tiempo de simulación T= 500 ns. Revise de vez en cuando, en el visor de formas de onda, el avance del tiempo y el desarrollo de la prueba.

Para terminar con la revisión de los aspectos clave del desarrollo de la simulación, vamos a ver cómo la simulación finaliza cuando el tiempo de simulación no puede continuar avanzando.

Realice las siguientes operaciones:

1. Seleccione, en la ventana **Processes**, el proceso **line_29**.
2. En la ventana de código, sitúe el cursor del ratón sobre la línea 60 y active el botón izquierdo (figura 41) para establecer un punto de ruptura (**breackpoint**).

```
52      digBCD <= "1011"; --En T = 1100 n
53      wait for 100 ns;
54      digBCD <= "1100"; --En T = 1200 n
55      wait for 100 ns;
56      digBCD <= "1101"; --En T = 1300 n
57      wait for 100 ns;
58      digBCD <= "1110"; --En T = 1400 n
59      wait for 100 ns;
60      digBCD <= "1111"; --En T = 1500 n
61      wait;
```

Figura 41.- Breackpoint

Observe que un marcador rojo marca la posición del punto de ruptura.

3. Pulse sobre el icono de la figura 42 para que la simulación continúe hasta el punto de ruptura.

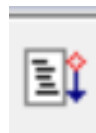


Figura 42

Ejecutando p-a-p el código desde este punto podremos observar cómo finaliza la simulación.

4. Ejecute p-a-p las sentencias del proceso **line_29** hasta que se complete su ejecución.

Tras ejecutar la sentencia **WAIT;**, el proceso queda definitivamente suspendido –no vuelve a reanudarse y, por tanto, no puede hacer avanzar el tiempo de simulación. Con la ejecución de este proceso termina la primera parte del ciclo de simulación, se actualiza **digBCD** (a “1111”) y comienza un ciclo delta con la ejecución del proceso que modela el funcionamiento del decodificador.

5. Continúe con la ejecución p-a-p hasta completar la ejecución de **proc_dec**. Cuando esta ejecución se complete desaparecerá la flecha del cursor de ejecución y, si continúa dando órdenes de ejecución p-a-p, se le indicará en la ventana **Transcript** que no se puede continuar: “**Nothing left to do**”. La simulación ha terminado porque no hay ningún proceso que pueda continuar ejecutándose:
 - El proceso de asignación de estímulos quedó anulado por la sentencia **WAIT;**
 - El proceso que modela el decodificador sólo se ejecuta cuando hay eventos en **digBCD**, eventos que ya no pueden ocurrir.
6. Puede comprobar, en el visor de formas de onda, que el resultado de este modo de ejecución se refleja en el cronograma que representa el resultado de la simulación (figura 43).

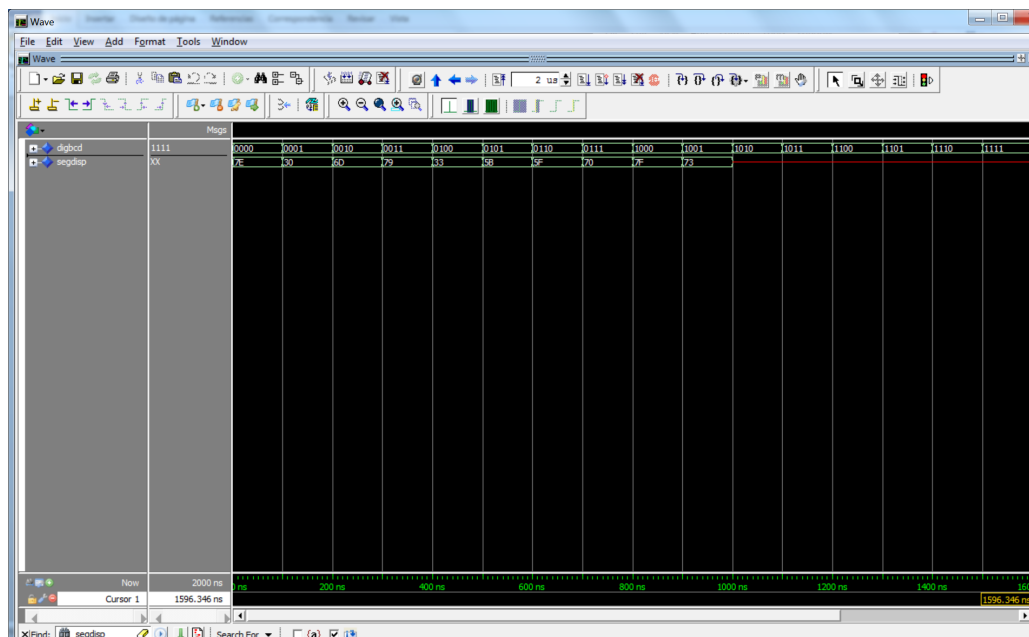


Figura 43