

DISEÑO DIGITAL 1. BLOQUE TEMÁTICO 1**TÍTULO DE LA ACTIVIDAD:****Tutorial VHDL ModelSim: creación de proyectos y ficheros VHDL. Compilación de modelos.****CÓDIGO:
BT1.P1****FECHA:****NOMBRE:****APELLIDOS:****MODALIDAD:**

Tutorial. Individual

TIPO:

Presencial

DURACIÓN:60
minutos**CALENDARIO:**

Sesión Sincrónica P1

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: metodología de desarrollo y depuración de modelos VHDL

En la figura 1 se representa el conjunto de tareas que hay que realizar para completar el desarrollo del modelo lógico de un sistema digital en VHDL. Para realizar estas operaciones resulta imprescindible utilizar un tipo de entornos de CAD, los Simuladores HDL, que integran herramientas capaces de dar soporte a todo este proceso.

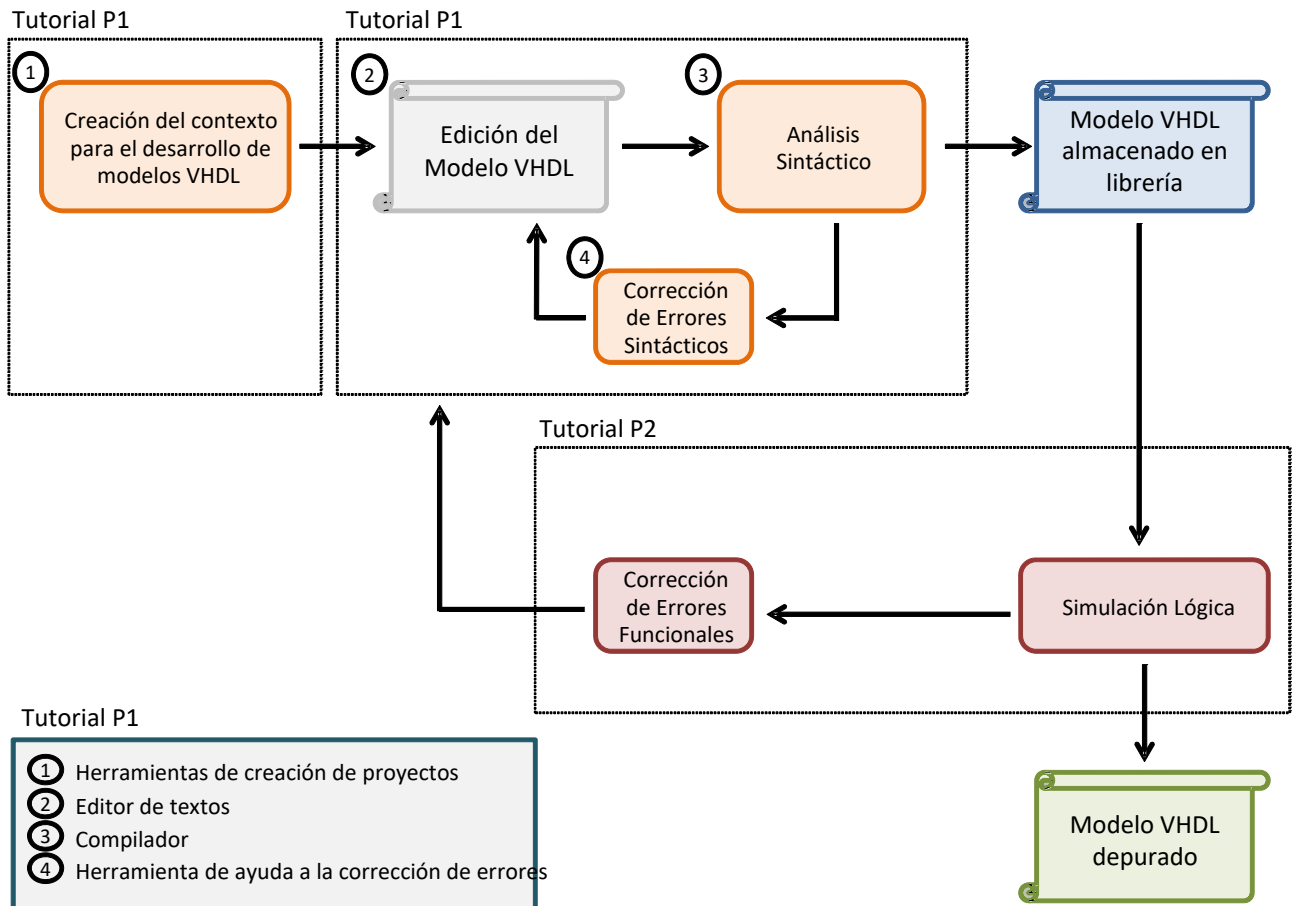


Figura 1.- Desarrollo de modelos VHDL

Como puede observarse en la figura 1, en el ciclo de desarrollo de un modelo VHDL se pueden distinguir tres conjuntos de operaciones:

1. Creación del contexto de desarrollo del modelo: Su objetivo es definir una librería de trabajo VHDL para almacenar los modelos que se vayan a desarrollar. Cuando se completa esta fase se dispone de un entorno adecuado para editar, compilar y simular modelos VHDL.
2. Almacenamiento del modelo en la librería de trabajo: Consiste en la edición y compilación del modelo para depurarlo sintácticamente y conseguir su almacenamiento en la librería de trabajo VHDL. Al finalizar esta fase se dispone de un modelo VHDL sintácticamente correcto almacenado en una librería.
3. Simulación del modelo: En esta fase del proceso se verifica la corrección funcional del modelo. Los errores detectados durante la simulación conllevarán la edición del modelo y la

repetición de su depuración sintáctica. Una vez que se consigue realizar una simulación satisfactoria, se dispone de un modelo VHDL depurado que puede trasladarse a una herramienta de síntesis automática para proceder a su realización tecnológica.

En esta actividad se va a realizar un tutorial cuyo objetivo es que aprenda a completar las dos primeras fases del ciclo de desarrollo de modelos VHDL (las operaciones 1, 2, 3 y 4 de la figura 1), empleando el entorno de simulación HDL *ModelSim* de *Mentor Graphics*¹.

Instrucciones para la realización del tutorial

A la hora de realizar esta actividad presencial tenga en cuenta que:

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 modelos. Léalos con atención y, si tiene cualquier duda o curiosidad sobre ellos, 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.

¹ En la actividad P2 se describirá el proceso de simulación del modelo.

Proyectos en ModelSim

Para poder desarrollar el modelo lógico VHDL de un sistema digital es necesario disponer de una librería VHDL activa –conocida como librería de trabajo o librería de diseño- en la que almacenar los modelos a medida que se van completando (editando y compilando).

Una librería VHDL es una colección de unidades de código VHDL (declaraciones de entidad y cuerpos de arquitectura, por ejemplo) que han sido compiladas satisfactoriamente.

El entorno ModelSim crea una librería de diseño VHDL cuando se crea un proyecto. La creación de un proyecto materializa la primera tarea de la figura 1: la creación del entorno de desarrollo de modelos VHDL.

Procedimiento T.2: Creación de un proyecto

Realice las siguientes operaciones:

1. En el menú **File**, de la ventana principal de ModelSim, active la opción **New** y seleccione, entre las alternativas que aparecen, **Project...** (figura 4)

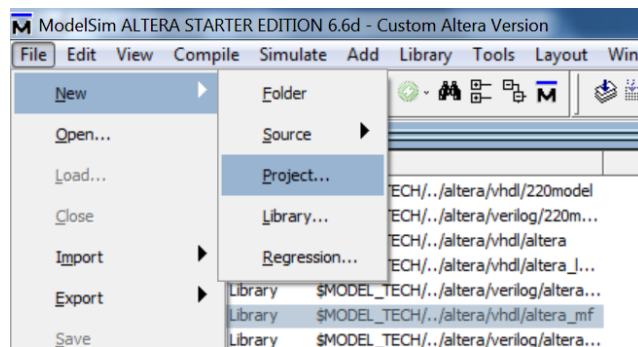


Figura 4.- Menú para la creación de proyectos

2. En la ventana que aparece, indique:
 - a. En el campo **Project Name**, el nombre del proyecto: *Proy_P1*.
 - b. En el campo **Project Location**, su ubicación: en el directorio de su cuenta, en un subdirectorio de nombre *P1/modelsim*.
 - c. En el resto de campos deje los valores por defecto (figura 5).

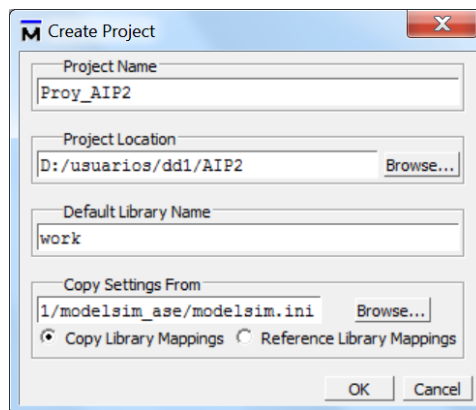


Figura 5.- Definición del proyecto

3. Pulse el botón **OK**.
4. En la ventana que aparece, confirme la creación del directorio del proyecto –esta confirmación se solicita cuando, como en este caso, el directorio del proyecto no existe.
5. Aparece una ventana que permite añadir o crear ficheros para el proyecto. Ciérrela pulsando el botón **Close**.

Como resultado de la creación del proyecto, se añade una nueva pestaña, **Project**, al espacio de trabajo de la ventana del simulador. Ahora hay dos pestañas, **Project** y **Library**. En la primera, que ahora se encuentra vacía, irán apareciendo los ficheros con código VHDL que se vayan creando en el proyecto P1; en la segunda aparece una lista de librerías VHDL (Figura 6).

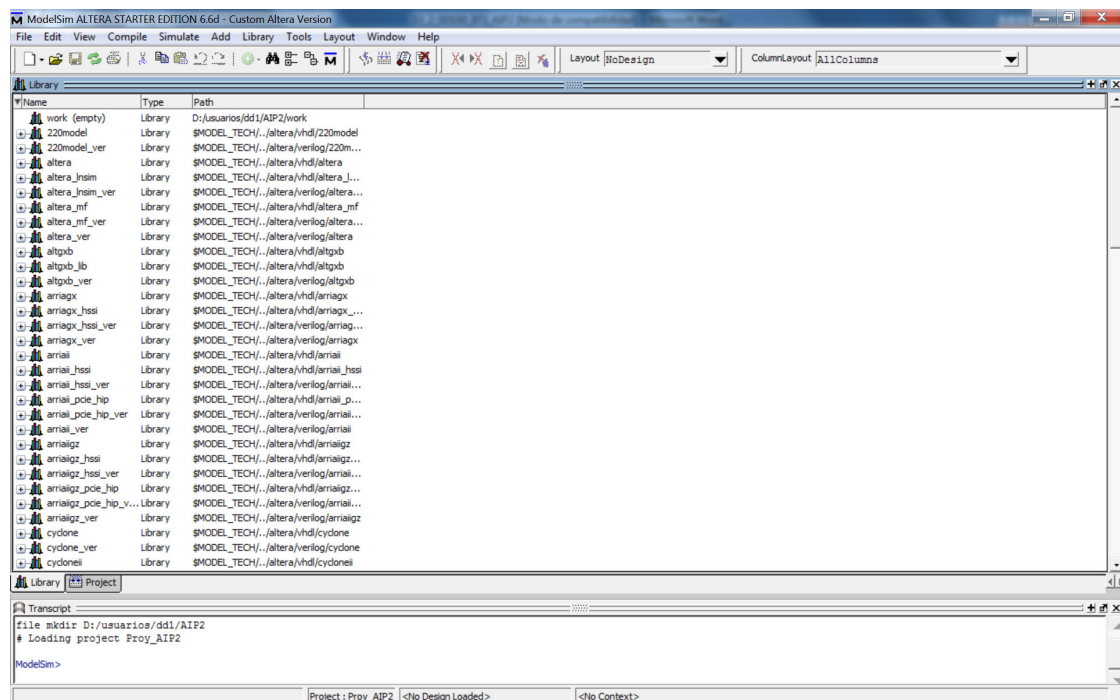


Figura 6.- Lista de librerías

La primera librería de la lista, la librería **Work**, es la librería de diseño VHDL correspondiente al proyecto de trabajo actual. El resto son **librerías de recursos**: contienen elementos que pueden utilizarse para la construcción de modelos VHDL.

Una vez que se dispone de una librería de diseño activa, puede comenzar el proceso de desarrollo del modelo VHDL. En primer lugar hay que crear un fichero VHDL.

Procedimiento T.3: Creación de un fichero VHDL

Realice las siguientes operaciones:

1. En el menú **File**, de la ventana principal de ModelSim, seleccione la opción **New**, luego **Source** y, por último, **VHDL** (figura 7).

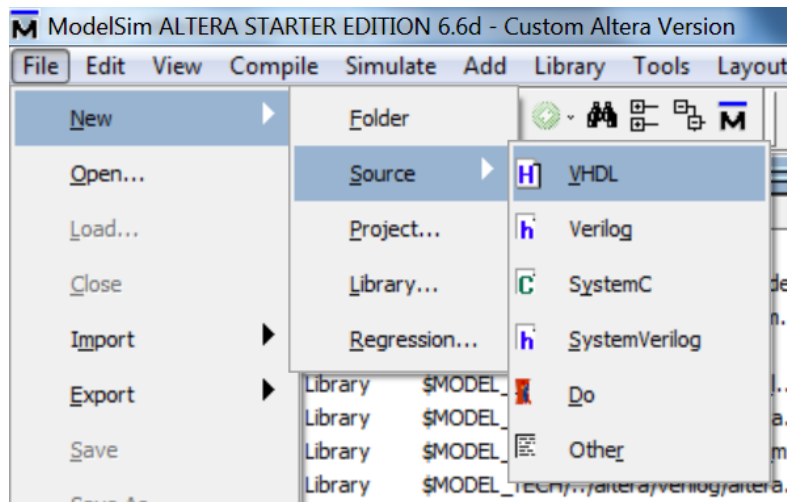


Figura 7.- Creación de un fichero VHDL

2. Como resultado de esta acción, se activa un editor de texto en la parte derecha del área de trabajo. Puede *desanclar* la hoja de edición pulsando el botón izquierdo del ratón sobre el icono situado en la esquina superior izquierda de la hoja y seleccionando la opción **Dock/Undock** (figura 8).

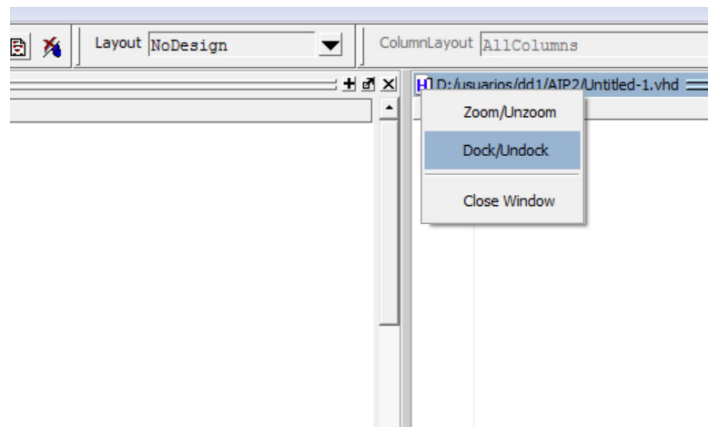


Figura 8.- Desanclado de la hoja de edición

Como resultado de esta acción dispone de una ventana independiente de edición.

Modelado de un decodificador BCD a 7 segmentos

El requisito imprescindible para la realización del modelo VHDL de un sistema digital es el conocimiento de su modelo lógico (su interfaz y funcionamiento), ya que el modelo VHDL no es otra cosa que dicho modelo lógico expresado en un lenguaje formal. En este tutorial se va a desarrollar el modelo VHDL del decodificador BCD a 7 segmentos diseñado en el ejercicio 1 de la actividad AINP3. Su modelo lógico se representa en la figura 9.

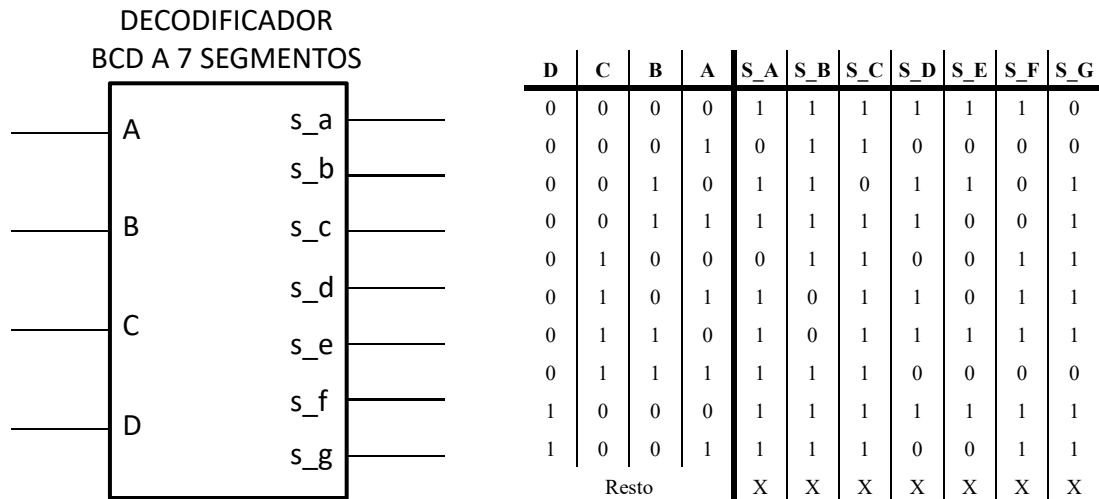


Figura 9.- Modelo lógico del decodificador BCD a 7 segmentos

El código de un modelo VHDL de este decodificador se muestra en la figura 10. En los siguientes apartados se editará este código y se comentarán algunos aspectos interesantes o novedosos del mismo.

```
-- Modelo de un decodificador BCD a 7 segmentos
-- realizado para el tutorial de la actividad AINP2
-- del bloque I de Diseño digital I
-- (17-3-2011) versión 1.0

-- Cabecera: cláusulas de visibilidad
library ieee;
use ieee.std_logic_1164.all;

-- Entidad: unidad primaria VHDL que modela la interfaz
entity decodBCD7seg is
port(
    digBCD: in std_logic_vector(3 downto 0);
    segDisp: buffer std_logic_vector(6 downto 0)
);
end entity;

-- Arquitectura: unidad secundaria (rtl) asociada a una
-- declaración de entidad (decodBCD7seg)
architecture rtl of decodBCD7seg is
begin
    proc_deco:
    process(digBCD)
    begin
        case digBCD is
            when "0000" =>
                segDisp <= "1111110";
            when "0001" =>
                segDisp <= "0110000";
            when "0010" =>
                segDisp <= "1101101";
            when "0011" =>
                segDisp <= "1111001";
            when "0100" =>
                segDisp <= "0110011";
            when "0101" =>
                segDisp <= "1011011";
            when "0110" =>
                segDisp <= "1011111";
            when "0111" =>
                segDisp <= "1110000";
            when "1000" =>
                segDisp <= "1111111";
            when "1001" =>
                segDisp <= "1110011";
            when others =>
                segDisp <= "XXXXXXX";
            end case;
        end process;
    end rtl;
```

Figura 10.- Modelo VHDL de un decodificador BCD a 7 segmentos

Procedimiento T.4: Edición de comentarios VHDL

Es frecuente que los modelos VHDL vayan precedidos por un comentario de cabecera relativo al modelo contenido en el fichero y con información complementaria de interés (fecha de realización, versión, revisión, etcétera).

En la figura 11 se muestra la sintaxis de los comentarios VHDL: en una unidad de código VHDL un comentario es cualquier texto precedido por dos guiones.

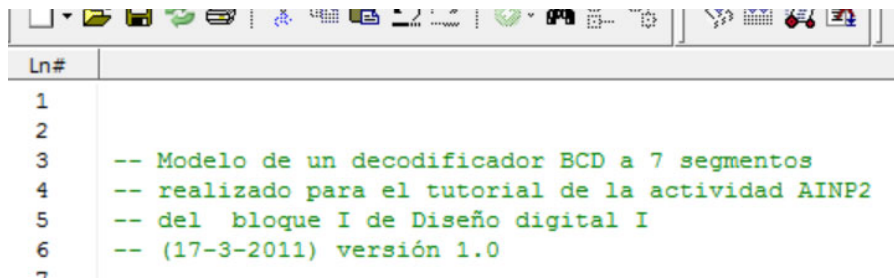


Figura 11.- Comentarios VHDL

Realice la siguiente operación:

1. Introduzca, en el editor de ModelSim, el código comprendido entre las líneas 3 y 6 de la figura 11.

Procedimiento T.5: Edición de cláusulas de visibilidad

En VHDL la interfaz del sistema se modela en una **Declaración de Entidad**. Cuando se realizan modelos sintetizables, el código de la Declaración de Entidad debe estar precedido por cláusulas de visibilidad que permitan utilizar los **tipos de datos idóneos** para los modelos sintetizables –tipos de datos creados expresamente para el desarrollo de modelos sintetizables y encapsulados en unidades de código dentro de una librería de recursos VHDL.

En la figura 12 se muestran las cláusulas de visibilidad que anteceden siempre el código de cualquier modelo VHDL sintetizable. Más adelante, en este curso, se verá que en dichos modelos estas cláusulas vienen frecuentemente acompañadas de alguna otra.

```
8 | -- Cabecera: cláusulas de visibilidad
9 | library ieee;
10 | use ieee.std_logic_1164.all;
11 |
```

Figura 12.- Cláusula de visibilidad

El código de la línea 9 hace visible la librería **IEEE** (una librería de recursos) a la unidad de código VHDL que va a continuación de estas líneas: en el ejemplo, la declaración de entidad del decodificador; en general, la declaración de entidad de un modelo sintetizable.

El código de la línea 10 da visibilidad sobre el contenido de una unidad de código VHDL, denominada **std_logic_1164** y almacenada en la librería anterior (IEEE). Esta unidad de código es de un tipo diferente a las que conoce hasta ahora (declaraciones de entidad y cuerpos de arquitectura), es un **Paquete VHDL**. En los paquetes VHDL se pueden definir, entre otras cosas, tipos de datos y en el paquete **std_logic_1164** están definidos, en concreto, los dos tipos de datos que se utilizan en los modelos sintetizables: los tipos **std_logic** y **std_logic_vector**.

En la cláusula de visibilidad del paquete **std_logic_1164** se hace referencia al mismo mediante su nombre jerárquico, que incluye la librería donde está almacenado (IEEE) –la cláusula que da visibilidad a esta librería debe preceder a la del paquete para hacer descifrable su nombre jerárquico, **ieee.std_logic_1164**. El sufijo **all** que se añade al paquete indica que la cláusula proporciona visibilidad sobre **todo** el contenido del paquete.

El tipo **std_logic** se usa para representar el valor que puede tomar un **nodo lógico**. Tiene definidos 9 valores: cuatro de ellos permiten distinguir los niveles lógicos alto y bajo, los cinco restantes sirven para representar niveles metalógicos (indeterminación, alta impedancia, etcétera). Cada valor de este tipo de datos es un carácter y se representa, por tanto, entre apóstrofes; los nueve valores son:

- '0' y '1': niveles lógicos bajo y alto
- 'L' y 'H': niveles lógicos bajo y alto débiles³
- 'Z': alta impedancia
- 'X' y 'W': indeterminación (fuerte y débil)
- '-': indiferencia
- 'U': No inicializado

El tipo **std_logic_vector** es un array de valores **std_logic** y se utiliza para modelar el valor de grupos de bits (de buses). Un valor de este tipo es un array de caracteres (un string) y se representa entre comillas -el valor de un objeto de tipo **std_logic_vector** de 4 bits puede ser, por ejemplo, "011X".

Realice la siguiente operación:

1. Añada, con el editor de ModelSim, el código comprendido entre las líneas 8 y 10 de la figura 12.

Procedimiento T.6: Edición de la Declaración de Entidad

La Declaración de Entidad sirve para modelar la interfaz de los sistemas digitales. En ella se especifica:

- El nombre del sistema.
- El nombre, dirección y número de bits de cada puerto del sistema digital.

En la figura 13 se muestra la sección de código que modela la interfaz del decodificador BCD a 7 segmentos.

```
11
12  -- Entidad: unidad primaria VHDL que modela la interfaz
13  entity decodBCD7seg is
14  port (
15      digBCD: in std_logic_vector(3 downto 0);
16      segDisp: buffer std_logic_vector(6 downto 0)
17  );
18  end entity;
19
```

Figura 13.- Declaración de Entidad del decodificador

En la Declaración de Entidad deben elegirse nombres adecuados tanto para el sistema (**decodBCD7seg**) como para sus puertos (**digBCD** y **segDisp**). En el modelo de la figura 13 se ha modificado –reinterpretado– la interfaz lógica de la figura 9; para simplificar el código del modelo

³ Un nivel lógico débil es el forzado por una salida con una impedancia alta como, por ejemplo, una resistencia de *pull-up*.

se han sustituido las cuatro entradas y siete salidas del decodificador por dos buses de cuatro y siete bits.

La dirección **in** de puerto **digBCD** lo distingue como puerto de entrada; su tipo de datos, **std_logic_vector(3 downto 0)** define su anchura, 4 bits. Cuando un puerto, como en este caso, está compuesto por varios bits, puede hacerse referencia a él mediante su nombre simple o añadiendo a éste el rango del puerto:

- **digBCD**
- **digBCD(3 downto 0)**

Para referirse a un bit del puerto o a un subconjunto de bits del puerto:

- **digBCD(3)** es el bit de la izquierda de **digBCD**
- **digBCD(3 downto 2)** son los dos bits de la izquierda de **digBCD**

En VHDL los puertos de salida pueden definirse con dos tipos de direccionalidad: **buffer** y **out**. Aunque en muchos casos son equivalentes, el uso de **out** puede en ocasiones causar problemas⁴, por lo que usaremos siempre el otro tipo, **buffer**.

Realice la siguiente operación:

1. Añada, con el editor de ModelSim, el código comprendido entre las líneas 12 y 18 de la figura 13.

Con esta última operación se ha completado el código de la declaración de entidad (figura 14), que es una unidad de código y, por tanto, es susceptible de ser compilada separadamente.

```
--
2
3  -- Modelo de un decodificador BCD a 7 segmentos
4  -- realizado para el tutorial de la actividad AINP2
5  -- del bloque I de Diseño digital I
6  -- (17-3-2011) versión 1.0
7
8  -- Cabecera: cláusulas de visibilidad
9  library ieee;
10 use ieee.std_logic_1164.all;
11
12 -- Entidad: unidad primaria VHDL que modela la interfaz
13 entity decodBCD7seg is
14 port(
15     digBCD: in std_logic_vector(3 downto 0);
16     segDisp: buffer std_logic_vector(6 downto 0)
17 );
18 end entity;
```

Figura 14.- Código de la Declaración de Entidad del decodificador

⁴ En la última revisión del lenguaje (VHDL-2008) se han solventado los problemas asociados a la direccionalidad **out**, pero todavía hay pocas herramientas que soportan esta versión del lenguaje, por lo que continúa siendo aconsejable el uso de la direccionalidad **buffer**.

Procedimiento T.7: Guardando un fichero VHDL

Realice las siguientes operaciones:

1. Para salvar el código editado hasta el momento seleccione la opción **Save** en el menú **File** de la ventana de edición (figura 15).

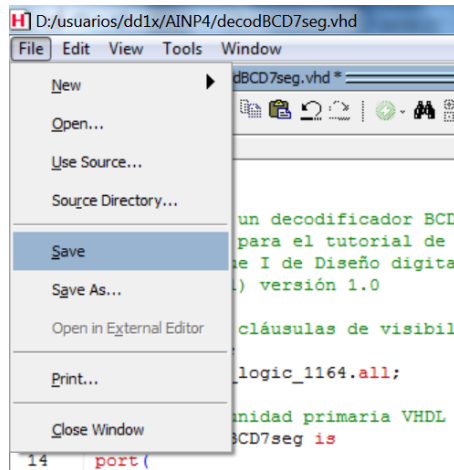


Figura 15.- Salvando el fichero

*A los ficheros que contienen modelos VHDL se les suele dar el mismo nombre que a la declaración de entidad que contienen. La extensión de los ficheros VHDL suele ser **vhd** – en algunos sistemas (en ModelSim, por ejemplo) se acepta también la extensión **vhdl**.*

2. En la ventana que aparece (figura 16), salve el fichero –no en el directorio del proyecto actual, que es el que se ofrece por defecto, sino en **P1/hdl**, - como **decodBCD7seg.vhd**.

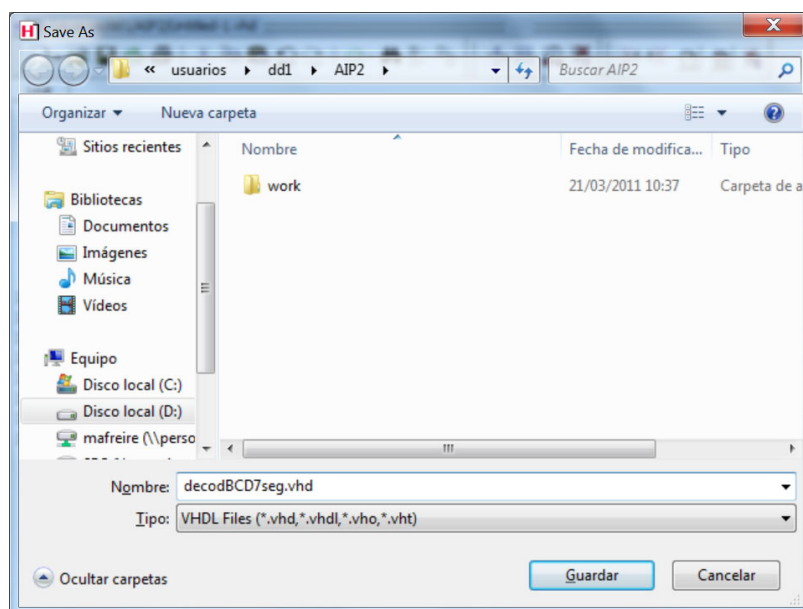


Figura 16.- Nombre del fichero

3. Cierre la ventana de edición y, en la ventana principal de ModelSim, seleccione la pestaña **Project** (figura 17).

Procedimiento T8: *Añadiendo ficheros al proyecto activo*

Para poder compilar la Declaración de entidad del decodificador es necesario que el fichero que la contiene pertenezca al proyecto activo.

Realice las siguientes operaciones:

1. Sitúe el puntero del ratón sobre el área de trabajo de la pestaña **Project**, pulse el botón derecho y seleccione la opción **Add to Project -> Existing File** (figura 17).

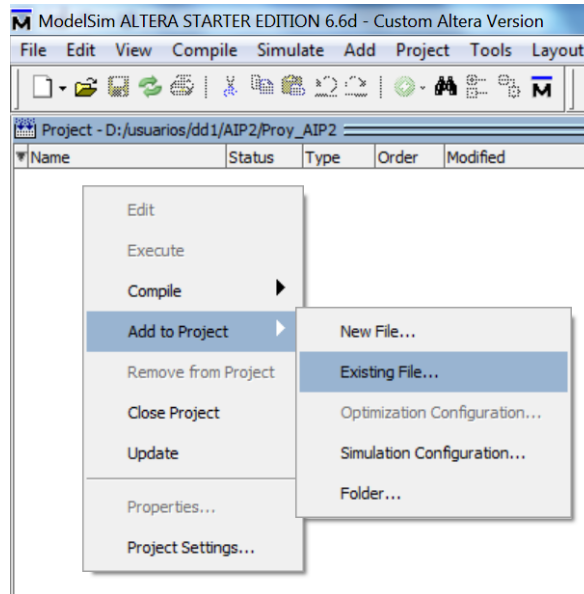


Figura 17.- Añadiendo ficheros

2. En la ventana de selección de ficheros que aparece, pulse el botón **Browse** para seleccionar el fichero **decodBCD7seg.vhd** del directorio **hdl**.
3. Pulse el botón **OK** para añadir el fichero al proyecto (figura 18).

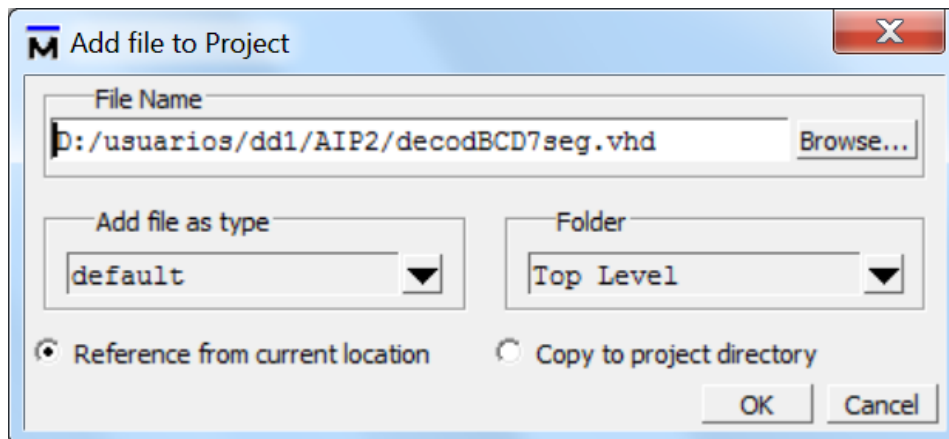


Figura 18.- Ventana de selección de ficheros

Observe que el fichero aparece en el área informativa de la pestaña **Project** (figura 19).

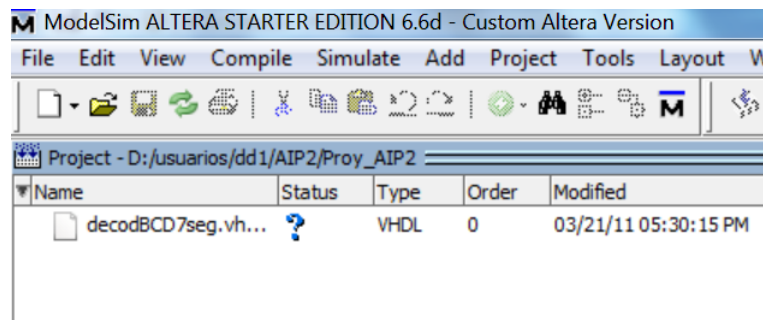


Figura 19.- Ficheros del proyecto

Procedimiento T9: *Compilación de ficheros VHDL*

Una vez añadido el fichero al proyecto se puede ordenar su compilación.

Realice las siguientes operaciones:

1. Sitúe el puntero del ratón sobre el icono del fichero, pulse el botón derecho y seleccione la opción **Compile -> Compile Selected** (figura 20).

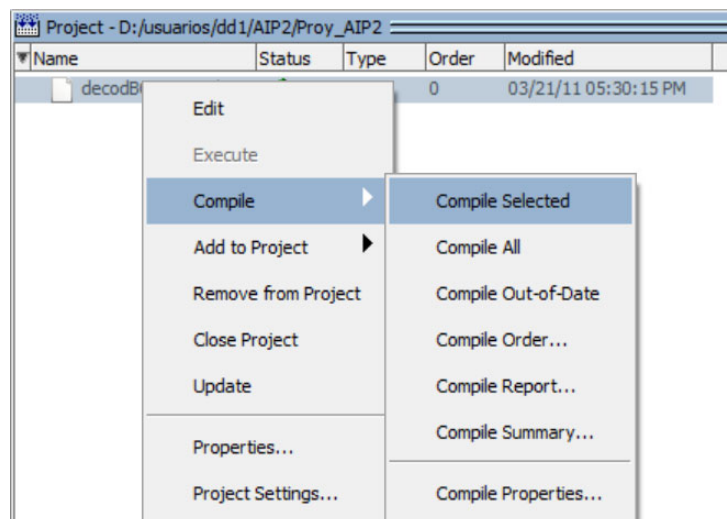


Figura 20.- Compilación de un fichero VHDL

Si no ha cometido errores editando el texto, debe aparecer, en la ventana **Transcript** situada en la parte inferior de la ventana de ModelSim, el mensaje que se muestra en la figura 21.

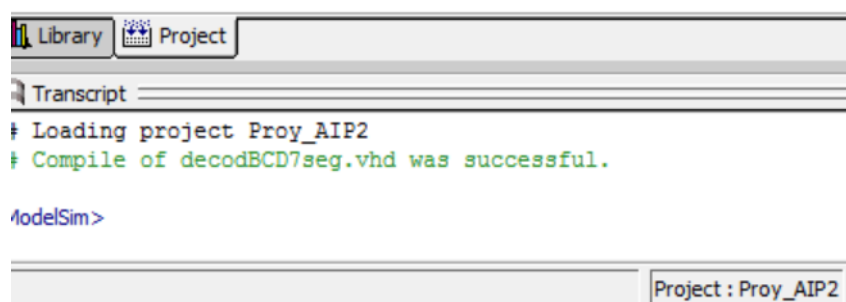


Figura 21.- Mensaje de resultado de la compilación

Si en lugar de este mensaje aparece otro, en rojo, indicando la existencia de errores, comuníquese inmediatamente a su profesor para que le ayude a subsanar el problema, con el fin de que pueda continuar con el desarrollo previsto del tutorial.

Procedimiento T10: Visualización del contenido de la librería de trabajo

Como consecuencia de la compilación exitosa de la declaración de entidad, la unidad de código se añade a la librería de trabajo del proyecto actual. Puede comprobarlo siguiendo el procedimiento que se describe seguidamente.

Realice las siguientes operaciones:

1. Seleccione la pestaña **Library** y, situando el puntero de ratón sobre el icono de la librería de trabajo -la primera de la lista: **Work**- pulse el botón izquierdo para expandirla y ver su contenido.

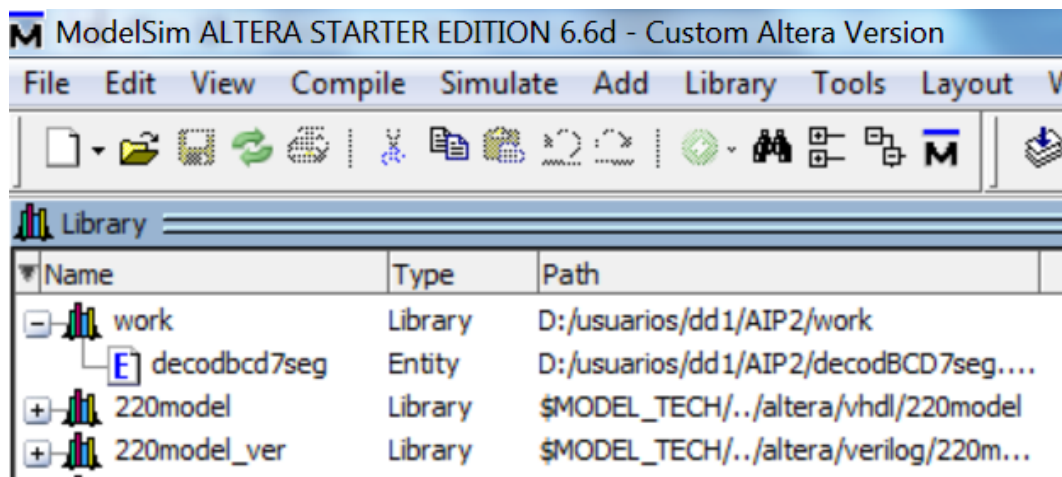


Figura 22.- Unidades almacenadas en la librería Work

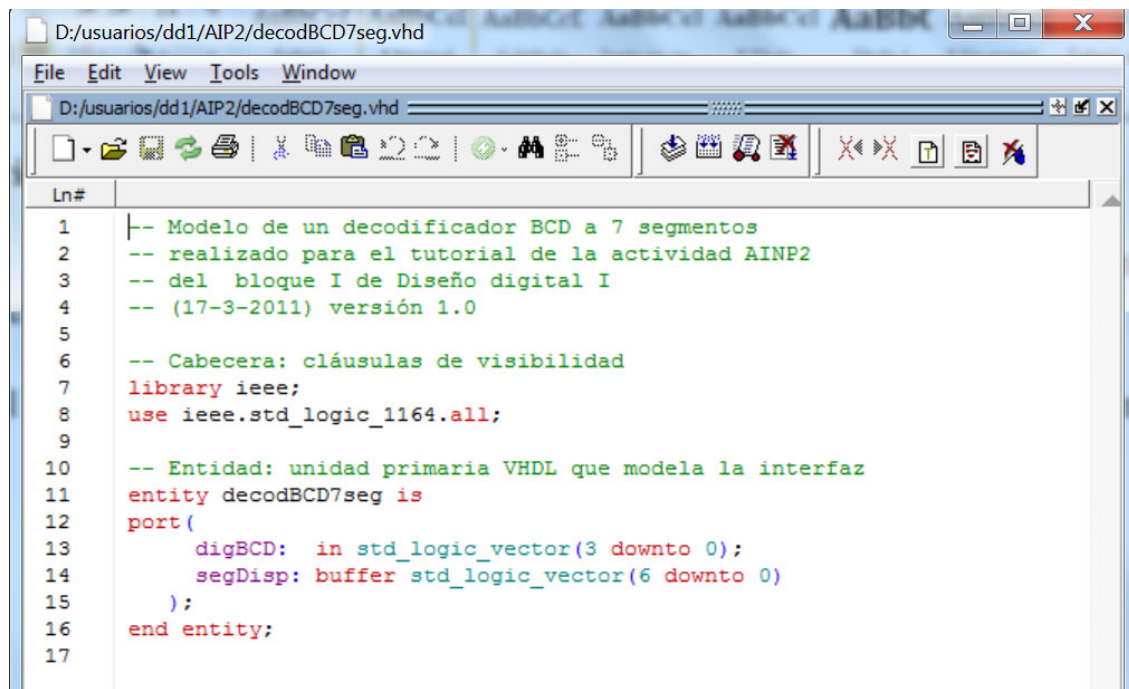
Procedimiento T11: Apertura de un fichero del proyecto

Para completar el modelo del decodificador hay que describir su funcionamiento en un Cuerpo de Arquitectura. Podría optarse por crear un nuevo fichero de texto para editar esta unidad de código, pero, por cuestiones prácticas, el Cuerpo de Arquitectura de un modelo se añade siempre al fichero donde está la Declaración de Entidad. Por tanto, para definir el funcionamiento del decodificador hay que abrir el fichero **decodBCD7seg.vhd**.

Realice las siguientes operaciones:

1. Seleccione la pestaña **Project** y realice una doble pulsación del botón izquierdo del ratón sobre el icono del fichero **decodBCD7seg.vhd**.

Como resultado se abre el editor de texto de ModelSim con el código editado hasta el momento (figura 23).



```
D:/usuarios/dd1/AIP2/decodBCD7seg.vhd
File Edit View Tools Window
D:/usuarios/dd1/AIP2/decodBCD7seg.vhd
Ln#
1  |-- Modelo de un decodificador BCD a 7 segmentos
2  |-- realizado para el tutorial de la actividad AINP2
3  |-- del bloque I de Diseño digital I
4  |-- (17-3-2011) versión 1.0
5
6  -- Cabecera: cláusulas de visibilidad
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 -- Entidad: unidad primaria VHDL que modela la interfaz
11 entity decodBCD7seg is
12 port (
13     digBCD:  in std_logic_vector(3 downto 0);
14     segDisp:  buffer std_logic_vector(6 downto 0)
15 );
16 end entity;
17
```

Figura 23.- Editor de texto

Modelo de Funcionamiento

El modelo lógico de funcionamiento de un sistema digital se describe en un Cuerpo de Arquitectura. Un Cuerpo de Arquitectura está asociado siempre a una Declaración de Entidad, pues describe el funcionamiento del sistema identificado por ella. Esta relación entre la Declaración de Entidad y el Cuerpo de Arquitectura se refleja en la estructura y reglas del lenguaje de la siguiente manera:

- *La Declaración de Entidad se considera una unidad de código primaria, el Cuerpo de Arquitectura como una unidad secundaria dependiente de la Declaración de Entidad y, por ello, deben compilarse de la siguiente manera: primero la Declaración de Entidad y después, cuando esta compilación se haya efectuado satisfactoriamente, el Cuerpo de Arquitectura.*
- *Todos los objetos (puertos) declarados en una Declaración de Entidad, así como los objetos sobre los que ésta haya adquirido visibilidad (los tipos **std_logic** y **std_logic_vector**) son visibles –pueden ser utilizados– en el Cuerpo de Arquitectura para describir el modelo de funcionamiento.*

Procedimiento T12: Edición de la cabecera de un Cuerpo de Arquitectura

Como ya se ha comentado, el código de los Cuerpos de Arquitectura se edita en el mismo fichero, y a continuación, del código de la Declaración de Entidad de la que depende –de este modo, al ordenar la compilación del fichero, se procesan las unidades de acuerdo con las reglas descritas anteriormente.

En la cabecera que identifica que una unidad de código es un Cuerpo de Arquitectura hay que indicar:

- *El nombre de la declaración de entidad de la que depende -para establecer la relación entre ambas unidades de código.*
- *El nombre de la arquitectura –porque el lenguaje admite que una declaración de entidad pueda tener varias arquitecturas asociadas (que se diferencian por este nombre); no se va a entrar ahora en el porqué de esta posibilidad, y utilizaremos siempre, mientras no se indique lo contrario, la etiqueta **RTL** como nombre de cuerpos de arquitectura de modelos sintetizables.*

En la figura 24 se muestra el código de la cabecera del Cuerpo de Arquitectura del decodificador.

```
17  
18      -- Arquitectura: unidad secundaria (rtl) asociada a una  
19      -- declaración de entidad (decodBCD7seg)  
20      architecture rtl of decodBCD7seg is  
21      begin  
22
```

Figura 24.- Cabecera de Cuerpo de Arquitectura

Realice la siguiente operación:

1. Añada, con el editor de ModelSim, el código comprendido entre las líneas 18 y 21 de la figura 24.

Modelado del funcionamiento de sistemas combinacionales simples

El funcionamiento de un sistema combinacional simple –entendiendo aquí por simple, un sistema cuyo funcionamiento puede ser definido sin necesidad de recurrir a la conexión entre dos o más (sub)sistemas- puede ser descrito mediante un proceso con lista de sensibilidad, siguiendo las siguientes reglas:

- *La lista de sensibilidad del proceso debe estar formada por todas las entradas del sistema combinacional.*
- *El código del proceso debe asignar valor a todos los bits de salida para todas y cada una de las combinaciones de entrada.*

En la figura 25 se muestra un proceso que materializa una descripción correcta y sintetizable del funcionamiento del decodificador BCD a 7 segmentos,

*Un primer detalle interesante que puede observarse en este código es que se puede poner nombre a un proceso; el nombre es opcional y, en caso de que se defina, se añade como una etiqueta (en el ejemplo **proc_deco**) seguida de dos puntos “:” antes de la palabra **process**. Nombrar los procesos puede resultar de ayuda durante la fase de depuración funcional de modelos complejos.*

El proceso que modela el decodificador:

- Es sensible a todas sus entradas (a los cuatro bits de entrada del decodificador), es decir, a **digBCD**.
- Para definir la relación funcional que liga las combinaciones de entrada y salida del decodificador se ha recurrido al uso de una sentencia **CASE** que asigna, a las diez combinaciones BCD de **digBCD**, las correspondientes combinaciones de salida en **segDisp**. La sentencia CASE se convierte, por tanto, en una traslación directa de la tabla de verdad que representa el funcionamiento del decodificador.

*La última opción de la sentencia CASE utiliza la fórmula **others** para indicar que agrupa todos los valores que no han aparecido en las opciones anteriores de la sentencia. La asignación que se hace a la salida en este caso, un string de 'X' ("XXXXXXXX"), indica que el valor de salida para estas combinaciones resulta indiferente –una alternativa para señalar indiferencias (de muy escaso uso) es el uso de guiones ("-----").*

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

Figura 25.- Modelado del funcionamiento del decodificador

Realice la siguiente operación:

1. Añada, con el editor de ModelSim, el código comprendido entre las líneas 23 y 50 de la figura 25.

Una vez editado el proceso que modela el funcionamiento del decodificador, hay que completar el modelo añadiendo la sentencia de cierre del Cuerpo de Arquitectura (figura 26).

```
50  
51     end rtl;  
52
```

Figura 26.- end RTL

Realice la siguiente operación:

1. Añada, con el editor de ModelSim, el código de la línea 51 de la figura 26, pero omita el punto y coma (;) final –para cometer un error sintáctico que permita describir la herramienta de detección de errores de ModelSim.

Realice las siguientes operaciones:

1. Salve el código editado (procedimiento T.7).
2. Compile el modelo (procedimiento T.9).

Observe que en la ventana **Transcript** aparece un mensaje, en rojo, advirtiendo de la detección de errores en el código (figura 27).

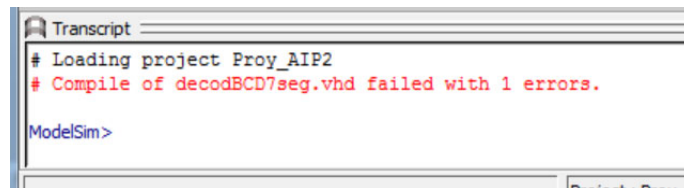


Figura 27.- Error de compilación

Procedimiento T13: Corrección de errores sintácticos

Realice las siguientes operaciones:

1. Sitúe el cursor del ratón sobre el mensaje de error de la ventana **Transcript** y realice una doble pulsación con el botón izquierdo del ratón.

Observe que aparece una ventana con una explicación del error detectado (figura 28).

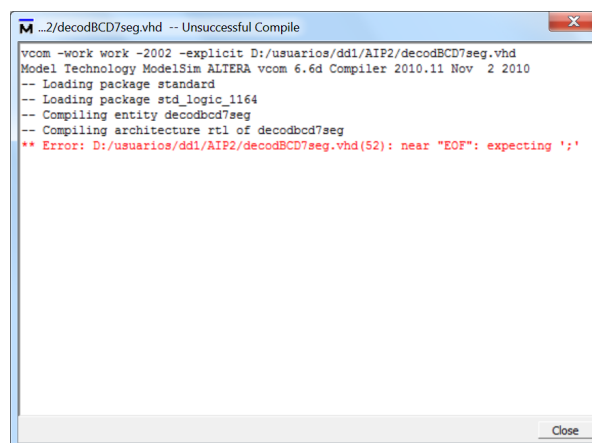


Figura 28.- Error sintáctico

El mensaje que aparece en la ventana describe el origen del error: *near EOF: expecting ';' ('cerca del fin de fichero se espera un ';')*.

2. Sitúe el cursor del ratón sobre el mensaje de error y realice una doble pulsación con el botón izquierdo del ratón.

Observe que se abre la ventana del editor de texto con el cursor situado en la línea donde se ha detectado el error (figura 29).

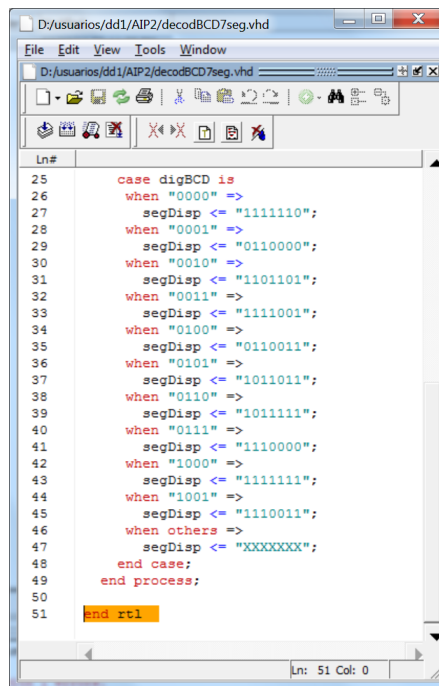


Figura 29.- Línea sombreada

3. Corrija el error, añadiendo el punto y coma, salve el fichero y recompílo.

La compilación se completa ahora con éxito. Como consecuencia se almacena en la librería de diseño (**Work**) el Cuerpo de Arquitectura del decodificador (figura 30).

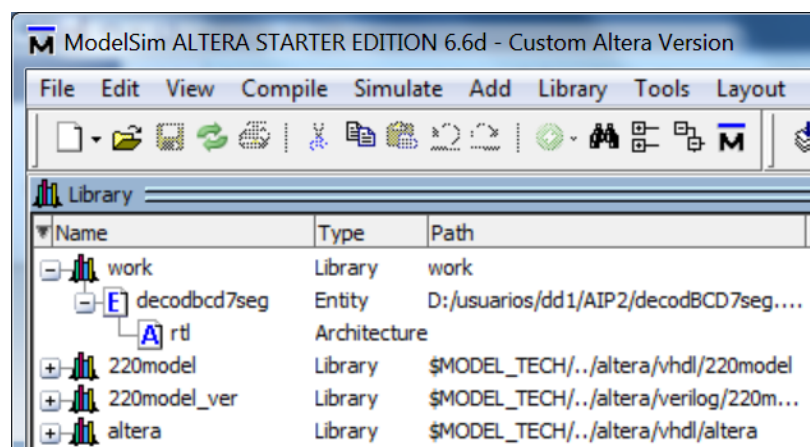


Figura 30.- Modelo en librería

Procedimiento T14: *Cerrando el entorno ModelSim*

Realice las siguientes operaciones:

1. En el menú **File** de la ventana principal de ModelSim, pulse la opción **Quit** y confirme, en la ventana que aparece, que desea cerrar el simulador.