Universidad de Alcalá

Tutorial del entorno ISE/ISim

Laboratorio de Diseño

Grado en Ingeniería Electrónica de Comunicaciones

Curso 2014/15

Contenido

1.- Introducción y objetivos.

El presente documento presenta un tutorial que tiene como objetivo guiar al alumno en el proceso de diseño en VHDL para FPGAs. Para ello, se estudia mediante un ejemplo de modelado, el flujo de diseño utilizando la herramienta ISE (*Integrated Software Environment*) de la firma comercial *Xilinx*.

Con el objetivo de familiarizarse con la herramienta anteriormente mencionada, se considera imprescindible el estudio del tutorial de una forma detallada. Con ello se podrá afrontar con mayor facilidad el desarrollo de la práctica libre. El tutorial permitirá al alumno conocer y dominar los siguientes aspectos:

a. Lista de los pasos necesarios en el proceso de diseño para FPGAs particularizado para los dispositivos de la firma comercial *Xilinx*.
b. Sintetizar e implementar diseños basados en FPGAs.
c. Detectar y corregir errores sintácticos en código especificado en VHDL.
d. Simular modelos en VHDL para extraer tanto su comportamiento funcional como temporal.
e. Comprobar el correcto funcionamiento del modelo en la placa de pruebas.

2.- Introducción al flujo de diseño para las FPGAs de *Xilinx*

En líneas generales el flujo de diseño (Figura 1) comienza con la creación de código VHDL que modela el comportamiento del diseño. Una vez especificado el diseño en VHDL se procede a la simulación funcional del mismo. Cuando el diseño funciona según las especificaciones, se puede continuar con las siguientes fases que consisten la síntesis y la implementación del modelo para poder realizar la simulación temporal. La simulación temporal permitirá comprobar que la temporización del diseño cumple con las restricciones impuestas al mismo, fundamentalmente relacionadas con la señal del reloj. Finalmente, se genera el fichero de descarga en placa llamado *bitstream* y se configura la FPGA para verificar que el modelo funciona en la placa del laboratorio.

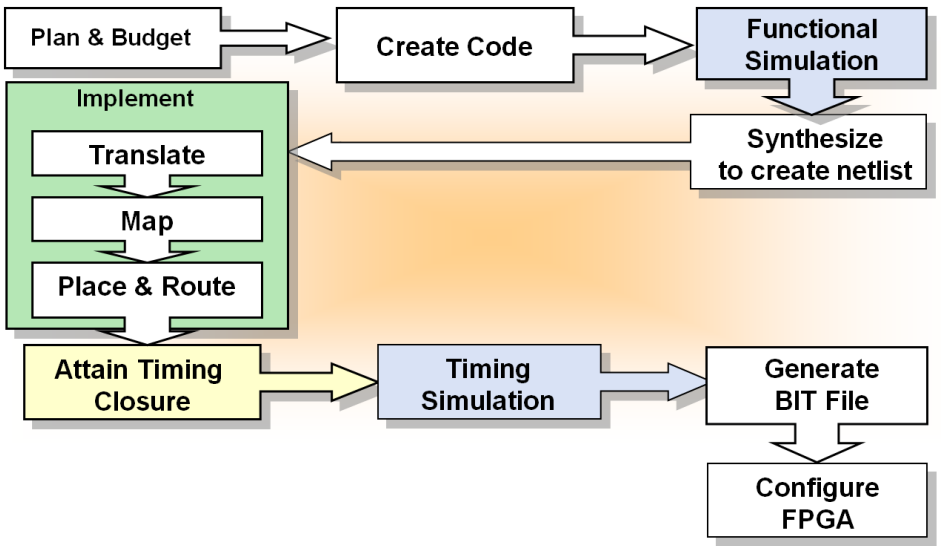


Figura 1.- Flujo de diseño para FPGAs.

En el flujo de diseño entran en juego diferentes ficheros que, en la mayoría de los casos tienen la misión de servir como ficheros intermedios entre los distintos procesos que se ejecutan para realizar las distintas fases de diseño (Figura 2). En los siguientes apartados se analizaran cada una de las herramientas y los ficheros más relevantes que cada una de ellas genera.

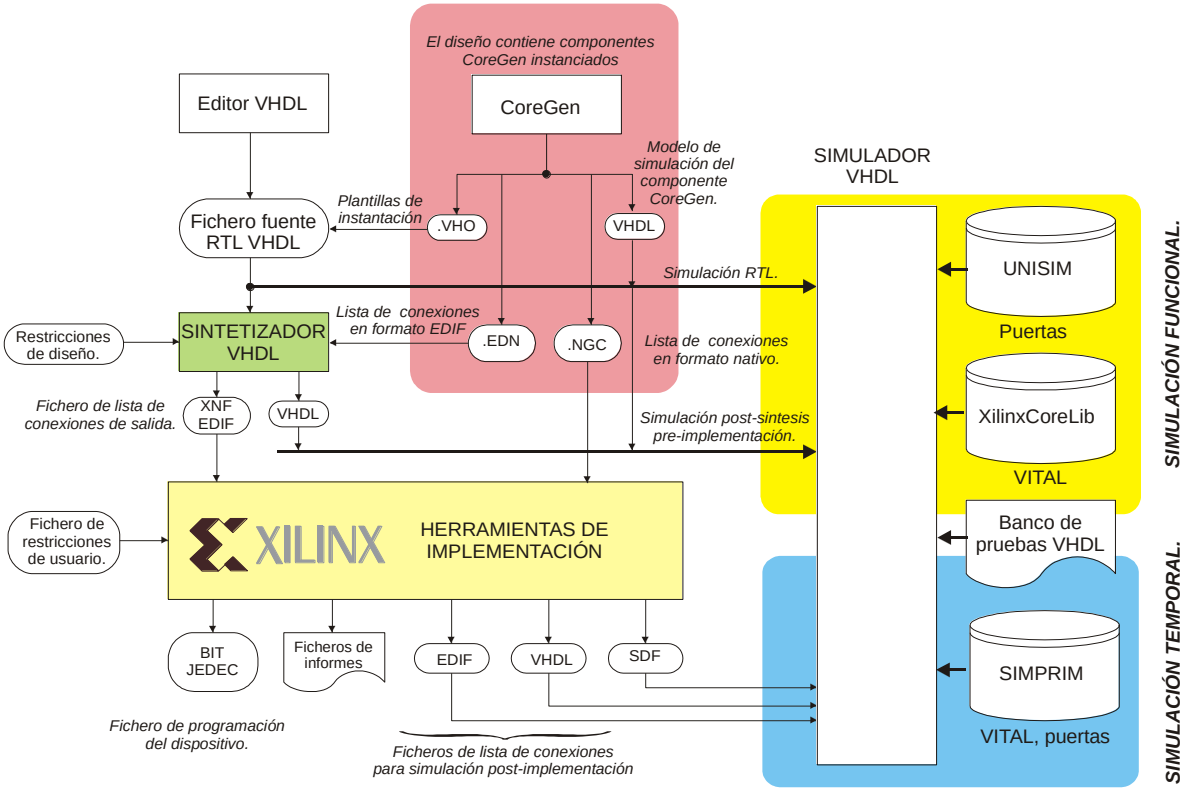


Figura 2.- Herramientas en el diseño basado en FPGAs de Xilinx.

A través de la herramienta ISE de *Xilinx* se pueden utilizar entre otros, dos métodos para introducir el diseño: esquemas o lenguajes de descripción hardware (HDLs). El flujo basado en HDLs permite describir la funcionalidad usando lenguajes como VHDL o Verilog para después obtener un descripción estructural del mismo después de un proceso de síntesis. El proceso termina con la obtención del diseño a nivel físico, lo que en el punto anterior llamamos la generación del *bitstream*. Para ello, se utiliza el emplazamiento y ruteado. A diferencia del flujo basado en HDLs los esquemas obligan a descomponer el diseño en componentes que se interconectan a través de un editor de esquemas.

También existe la posibilidad de introducir los diseños a través de diagramas de estados. Este tutorial pondrá especial énfasis en la especificación del diseño a través del lenguaje VHDL.

Después de la codificación del diseño en un lenguaje de descripción hardware y su posterior simulación funcional, se procede a su síntesis. Se entiende como síntesis al proceso de convertir descripciones en alto nivel de abstracción de un diseño en representaciones optimizadas a nivel de puertas (netlist). Por lo tanto la herramienta ISE durante el proceso de síntesis compila el diseño

para transformar los modelos especificados en VHDL en una *netlist* específica para una determinada arquitectura. El entorno ISE soporta el uso de la tecnología XST (*Xilinx Synthesis Technology*) que no es más que la herramienta de síntesis de *Xilinx* la cual está integrada en el entorno ISE. También se pueden utilizar herramientas de síntesis de terceras partes, es decir, de otros fabricantes incluyendo *Synplify*, *Synplify* and *Precision software.* En el laboratorio se utilizará la herramienta que viene integrada en el ISE y que se llama XST (Xilinx Synthesis Technology).

La herramienta de síntesis XST toma como entrada los modelos especificados en VHDL o Verilog. Adicionalmente, la herramienta puede aceptar como entrada ficheros de restricciones de XST en los cuales se puede especificar restricciones de síntesis, temporización e implementación. Con los ficheros de entrada mencionados anteriormente, la herramienta de síntesis realiza los siguientes pasos:

- Análisis sintáctico del código fuente.
- Compilación que transforma el código HDL en un conjunto de componentes genéricos que la herramienta puede reconocer y que se pueden ver como un esquema a nivel RTL.
- Mapeado tecnológico que traslada los componentes anteriores en componentes de la tecnología destino.

Al aplicar los procesos anteriores la herramienta XST genera a su vez los siguientes ficheros de salida:

- Informe de síntesis que contiene los resultados del proceso de síntesis incluyendo estimaciones del área y temporización.
- Esquemático a nivel RTL que representa el diseño preoptimizado a nivel de transferencia entre registros (RTL). Esta representación es en términos de símbolos genéricos, tales como sumadores, multiplicadores, contadores, puertas AND y OR.
- Esquemático a nivel tecnológico almacenado en un fichero con extensión NGC representa el circuito en términos de elementos lógicos optimizados para la arquitectura o tecnología destino. Se trata de modelo de circuito una vez realizado el mapeado tecnológico. Los elementos que constituyen el circuito son componentes especificados de la tecnología como por ejemplo LUTs, lógica de acarreo, buffers de entrada/salida y otros componentes. Se trata de una representación a nivel tecnológico del código HDL optimizado para la tecnología destino.

Después de la síntesis se debe llevar a cabo el proceso de implementación que consiste en tres fases:

- *Translate:* se trata de un proceso durante el cual en enlazan todos los ficheros de diseño que se quieren implementar en uno con formato NGD. El fichero NGD (*Native Generic Database*) representa una lista de conexiones (*netlist*) que describe el diseño a nivel lógico en forma de primitivas de *Xilinx*. A veces recibe el nombre de fichero de diseño y sirve como entrada al mapeador. (Figura 3)

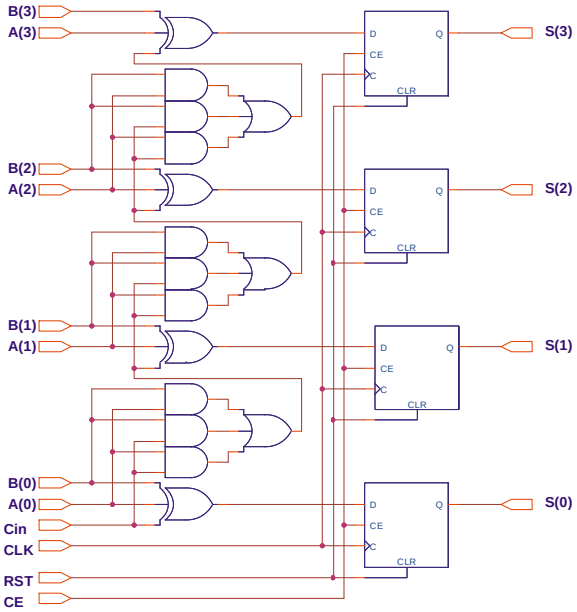


Figura 3.- Circuito ejemplo resultado de ejecutar el proceso *Translate.*

- *Map:* agrupa las primitivas (símbolos lógicos o puertas) especificadas en el fichero anterior (NGD) en componentes físicos (slices e IOBs). En otras palabras, se trata de un proceso que mapea la lógica definida en el fichero NGD en elementos que constituyen la FPGA, tales como CLBS e IOBs. El proceso genera como salida un fichero formato NCD (*Native Circuit Description*) que físicamente representa el diseño mapeado en componentes de la FPGA destino. La Figura 4 muestra el circuito de la Figura 3 mapeado en CLBs.

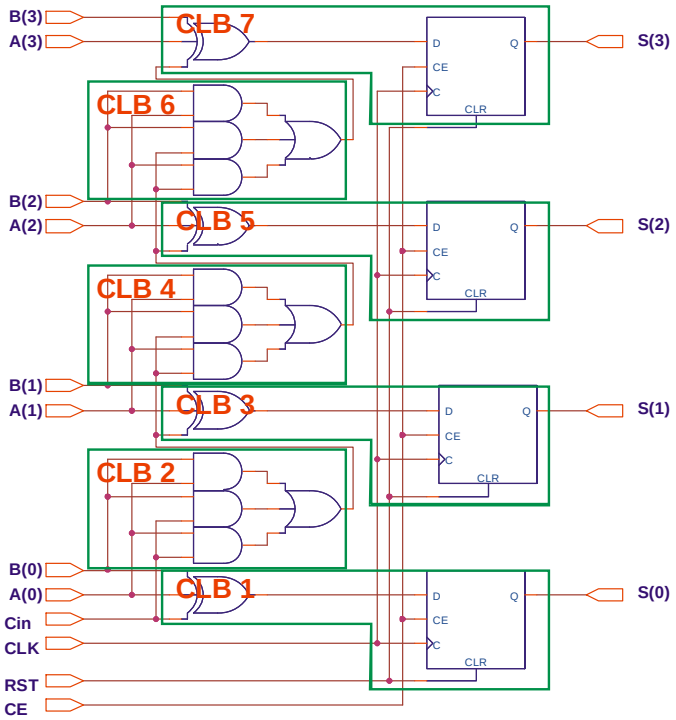


Figura 4.- Mapeado.

- *Place & Route:* este proceso toma como punto de partida el fichero mapeado NCD emplazándolo y conectándolo físicamente dentro de la FPGA. El resultado de este proceso genera un fichero NCD que se utiliza para la generación del fichero de configuración (*bitstream*). Finalmente la Figura 5 muestra un ejemplo de emplazamiento y ruteado dentro de una FPGA.

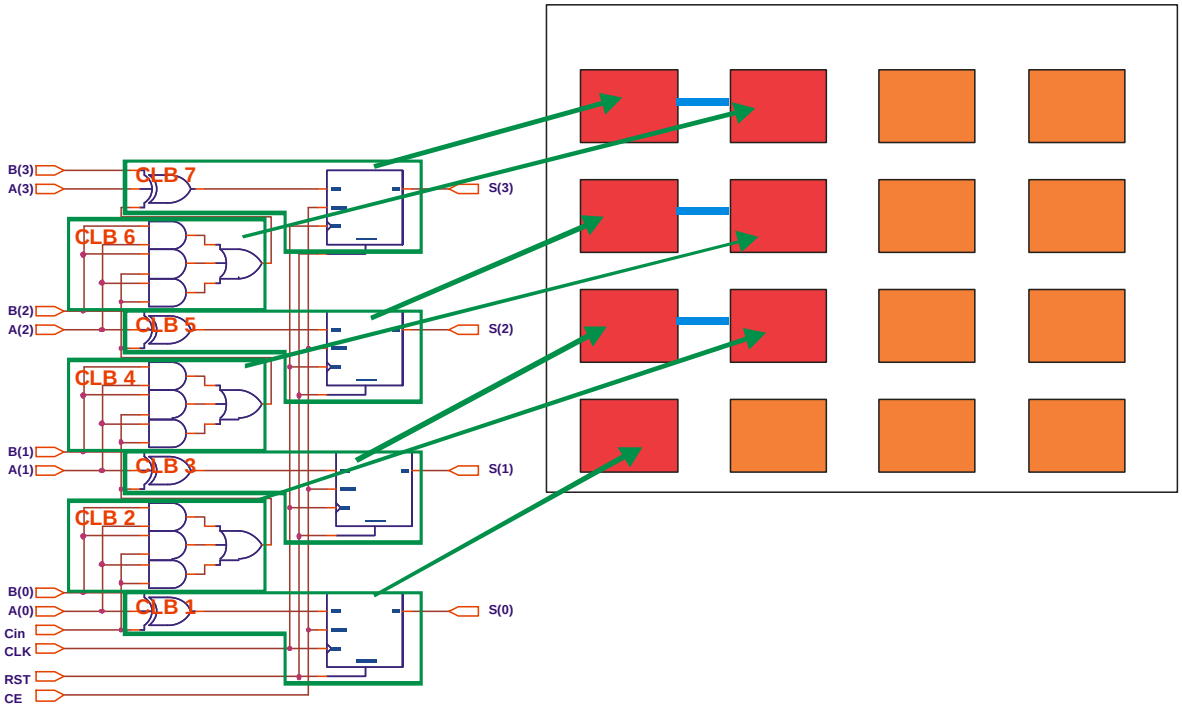


Figura 5.- Emplazamiento y ruteado

El proceso de simulación (Figura 1) se lleva a cabo en dos fases distintas dentro de la etapa de diseño. Por un lado, antes del proceso de síntesis se puede llevar a cabo una simulación funcional o de comportamiento. Este tipo de simulación emplea modelos de diseño descritos en HDL con un nivel de abstracción alto. Así, por ejemplo, el diseño podría incluir un operador suma sin especificar como el circuito se implementa en la FPGA. La herramienta de síntesis extraer en el siguiente paso la estructura de puertas y conexiones del sumador generando la correspondiente lista de conexiones.

La simulación funcional por lo tanto es la más rápida ya que no incluye en los modelos ningún tipo de información temporal y se basa, en la mayoría de los casos en la ejecución del código HDL. Por ello es la simulación que menos información proporciona. Sin embargo, permite verificar la funcionalidad sin información temporal. Durante el proceso de diseño, la mayor parte de las simulaciones son funcionales. Los errores que se detectan en esta fase de simulación son menos costosos de corregir, en términos del tiempo utilizado para ello. Después de que la funcionalidad requerida se consigue, se pueden llevar a cabo simulaciones temporales para obtener mayor detalle del comportamiento del circuito.

La simulación temporal, por el contrario, incluye información temporal. Esta información es importante a la hora de verificar el comportamiento del circuito después de que los retardos del

camino crítico, es decir en el peor escenario, hayan sido calculados para el diseño. Indicar que solamente se tendrá información temporal después de que se haya llevado a cabo el proceso de place and route (PAR). Durante este proceso, se genera una lista de conexiones de componentes con los retardos anotados en formato SDF (*Standard Delay Format*). Las simulaciones temporales pueden identificar condiciones de carrera, violaciones de tiempos de mantenimiento (*hold*) y de configuración (*setup*) de los elementos de almacenamiento basadas en las condiciones de operación para la funcionalidad especificada. Este tipo de simulación es más lenta pero proporciona mayor información del comportamiento del circuito, no solamente funcional sino también temporal.

Las simulaciones se llevan al cabo mediante herramientas específicas como por ejemplo *ModelSim*. En el laboratorio de diseño se empleará el simulador *ISIM* que viene integrado en el entorno *ISE*. Como se puede observar en la Figura 6 los ficheros de entrada al simulador dependen del tipo de simulación.

Para la simulación funcional se requieren los siguientes ficheros:

- Descripción funcional del modelo a simular en forma de esquemático o con un lenguaje de descripción hardware.
- Fichero de estímulos normalmente en forma de banco de pruebas (test bench) descrito en HDL. También se puede definir los estímulos en forma de comandos del simulador que se esté utilizando.
- Biblioteca con los modelos funcionales.

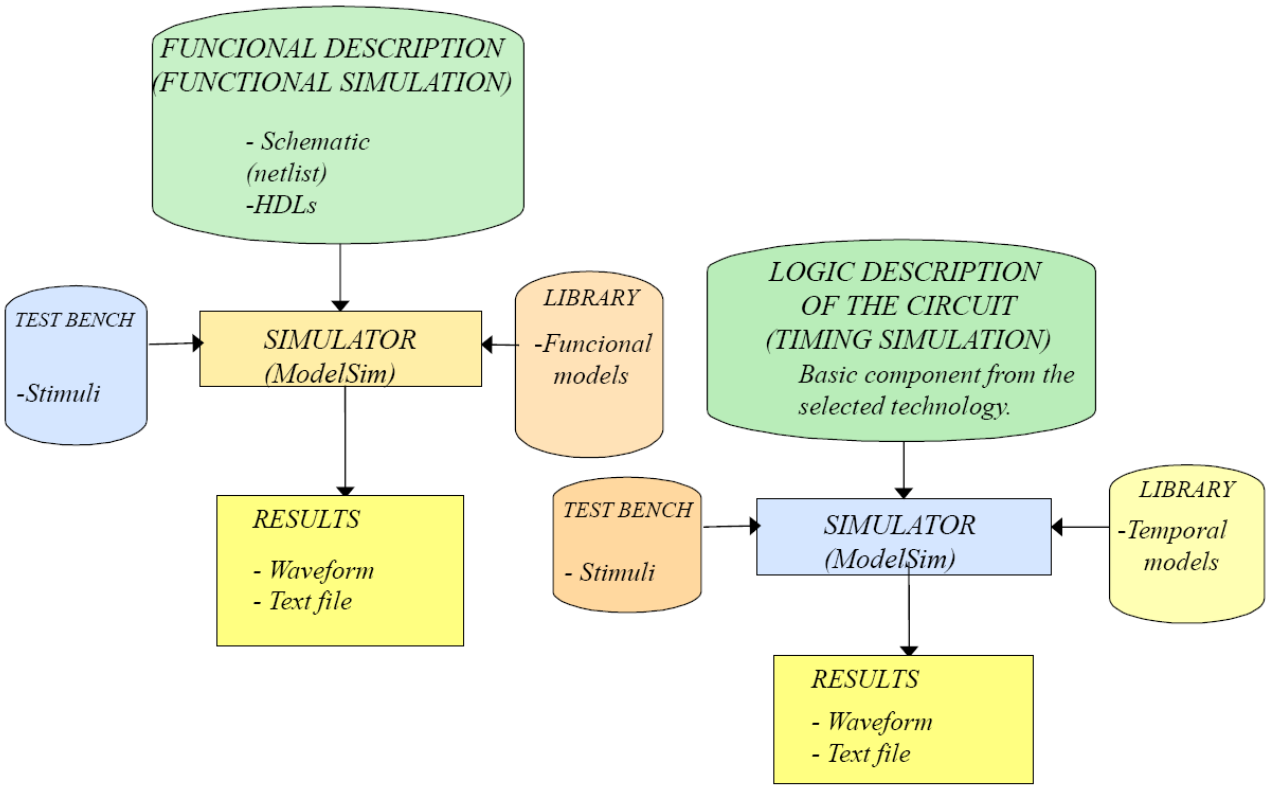


Figura 6.- Entradas y salidas en el proceso de simulación.

La herramienta de simulación muestra los resultados como fichero de texto o como formas de ondas.

La simulación temporal en comparación con la funcional toma como entrada la descripción lógica del circuito que incluye los retardos anotados después del proceso de Place and Route. Asímismo, los componentes de las bibliotecas incluyen información temporal. Como en el caso anterior los resultados se muestran mediante ficheros de texto o como formas de ondas.

Una vez que el diseño ha sido implementado, se genera por parte de la herramienta el fichero *.bit (*bitstream*) para poder configurar la FPGA. El fichero *.bit contiene toda la información de configuración para definir la lógica interna y las conexiones dentro de la FPGA. Este fichero se puede descargar de dos formas:

- Directamente a la FGPA usando un cable USB.
- A través de una memoria externa que previamente debe programarse con el fichero de configuración *bitstream*.

Para poder generar el *bitstream* se requiere un fichero NCD totalmente ruteado.

3.- Ejemplo de diseño.

Para guiar al alumno dentro del flujo de diseño de *Xilinx* y con el objetivo de facilitar la compresión del mismo, se plantea un ejemplo sencillo basado en un registro de desplazamiento que se descargará en la FPGA Spartan-6 núcleo central de la placa de laboratorio Atlys de Digilent.

La Figura 7 muestra la interface del diseño que consta de las siguientes entradas y salidas.

- *Clk*: entrada de reloj de 100 MHz procedente de un oscilador incluido en la placa de pruebas.
- *Rst:* señal de inicialización del proceso de representación. Se implementa a través del pulsador 0 de la placa.
- *Polarity:* señal que cambia el nivel de polaridad de los leds en la representación de la salida del registro de desplazamiento. Así, por ejemplo, cuando la entrada *polarity* toma el valor 0 se desplaza un bit a nivel bajo, es decir, solamente un led estará en *off* y el resto en *on*. Cuando la entrada *polarity* toma un 1 entonces se desplaza un nivel alto estando todos los leds en *off* excepto uno.
- *Direction:* esta entrada se usa para definir la dirección de desplazamiento: con un cero en esta entrada el desplazamiento es a derechas, es decir desde el LED de menor peso al LED de mayor peso. Cuando la entrada toma el valor 1 el desplazamiento se lleva a cabo en el sentido contrario.

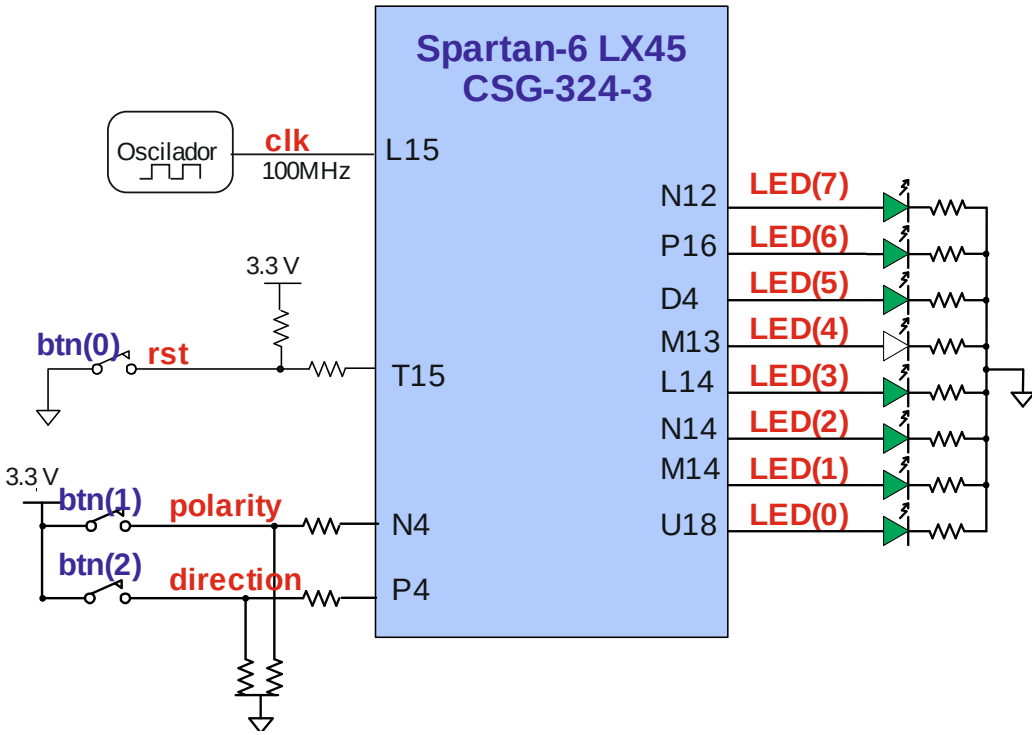


Figura 7.- Interfaz para el ejemplo de diseño.

Para la implementación del diseño del ejemplo se proponen cuatro modelos diferentes. El objetivo es poner de manifiesto no solamente aspectos importantes de diseño sino también presentar herramientas que facilitan considerablemente el desarrollo de mismo.

Todos los modelos comparten la estructura interna del desplazador basada en un contador binario de 3 bits y un decodificador 3 a 8. En general todos los modelos constan de los siguientes módulos VHDL que establecen la jerarquía de diseño (Figura 9):

- *Decoder_3_8*: se trata de un decodificador 3 a 8 con polaridad programable que se implementa con la estructura de la Figura 8. El modelo del decodificador incluye dos multiplexores: uno para definir el valor de salida activo controlado por la entrada *dec_in* y otro para definir la polaridad, en este caso controlada por la señal *polarity*.
- *Countermod8:* se trata de un contador *up/down* de 3 bits o módulo 8 (rango de cuenta de 0 a 7).
- *Shifter8:* este módulo VHDL interconecta los dos anteriores para implementar el desplazador.
- *Prescaler:* la frecuencia de desplazamiento será de un Hz. Esto obliga a introducir un prescaler para pasar de una frecuencia de 100 MHz a una frecuencia de 1 Hz.
- *Top_system:* se trata del módulo VHDL sintetizable más alto en la jerarquía de diseño. Este módulo interconecta el *prescaler* con el módulo del desplazador (*shifer8*)
- *Top_system_tb*: este módulo implementa el banco de pruebas (test bench) para poder llevar a cabo la simulación del modelo completo.

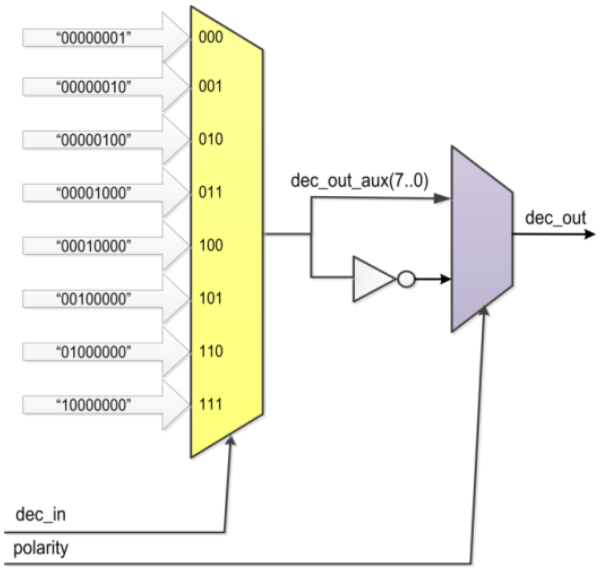


Figura 8.- Estructura del decodificador basada en dos multiplexores.

El código VHDL del modelo del decodificador se muestra el siguiente listado 1.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity decoder_3_to_8 is
  port (
    polarity : in  std_logic;
    dec_in   : in  std_logic_vector(2 downto 0);
    dec_out  : out std_logic_vector(7 downto 0));
end decoder_3_to_8;
        architecture for_decoder3_to_8 of decoder_3_to_8 is
  signal dec_out_aux : std_logic_vector(7 downto 0);
begin  -- for_decoder3_to_8
  process (dec_in)
  begin  -- process
    case dec_in is
      when "000"  => dec_out_aux <= "00000001";
      when "001"  => dec_out_aux <= "00000010";
      when "010"  => dec_out_aux <= "00000100";
      when "011"  => dec_out_aux <= "00001000";
      when "100"  => dec_out_aux <= "00010000";
      when "101"  => dec_out_aux <= "00100000";
      when "110"  => dec_out_aux <= "01000000";
      when "111"  => dec_out_aux <= "10000000";
      when others => null;
    end case;
  end process;
  process (polarity, dec_out_aux)
  begin  -- process
    if polarity = '0' then       dec_out <= dec_out_aux;
    else       dec_out <= not dec_out_aux;
    end if;
  end process;
end for_decoder3_to_8;
```

Listado 1.- Modelo VHDL del decodificador 3 a 8.

A continuación se explican las diferencias entre los distintos modelos de implementación del ejemplo de diseño.

3.1.- Modelo 1. Implementación del diseño con un reloj derivado.

La Figura 9 muestra la implementación del desplazador con un reloj derivado. De acuerdo con la metodología de diseño de circuitos síncronos, todas las entradas de reloj de todos los dispositivos que componen el diseño deber estar conectadas a la misma señal de reloj. En ejemplo del modelo 1, la entrada de reloj de contador módulo 8 es la salida combinacional del *prescaler*. En el ejemplo de la Figura 9, la salida *pre_out* del *prescaler* se conecta con la entrada *clk* y el *clock enable* (ce) se conecta nivel alto. Los relojes generados con lógica combinacional pueden introducir *glitches* que crean problemas funcionales y de retardo. En un diseño síncrono, un *glitch* en la entrada de datos no causa en principio ningún problema ya que es automáticamente eliminado ya que los datos se capturan en el flanco activo de la señal de reloj lo que permite bloquear el *glitch*. Sin embargo, un *glitch* en la entrada de reloj o en la entrada asíncrona de un registro, puede tener consecuencias negativas que afecten al funcionamiento correcto del modelo. Un *glitch* con una anchura pequeña puede violar los requerimientos de anchura mínima de los pulsos de reloj. Asimismo, se pueden violar los tiempos de *setup* y de *hold* si los datos de entrada del registro están cambiando cuando el *glitch* alcanza la entrada de reloj. Incluso si el diseño cumple todas las restricciones temporales, la salida del registro puede cambiar de valor de forma inesperada y causar un fallo de funcionamiento en otras partes del circuito.

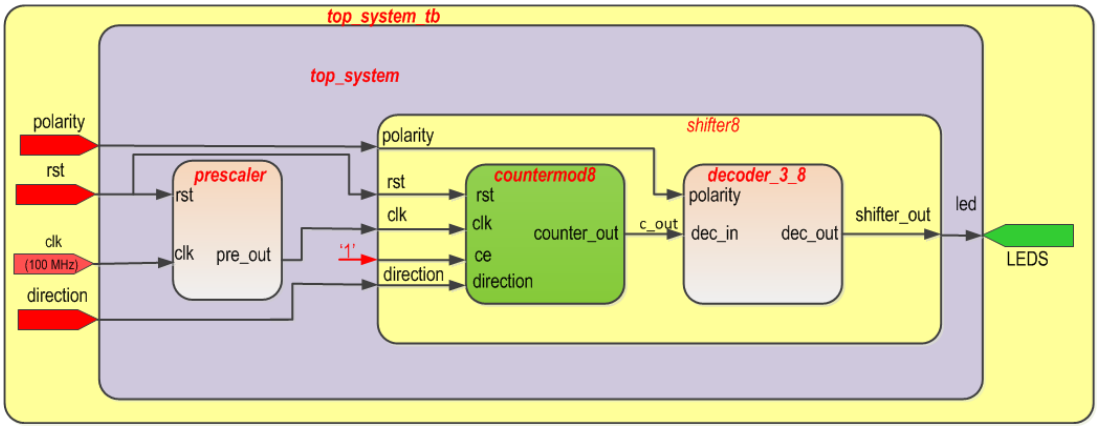


Figura 9.- Modelo 1 del ejemplo de diseño: relojes derivados.

Para evitar este problema se proponen las modificaciones implementadas en el modelo 2.

3.2.- Modelo 2. Diseño síncrono.

La Figura 10 muestra el diseño del desplazador utilizando una estrategia totalmente síncrona sin relojes derivados. En este caso la salida *pre_out* del *prescaler* se conecta a la entrada del *clock enable* (ce) del contador módulo 8 y la entrada de reloj del contador se conecta al reloj del sistema (100 MHz). De esta forma, aunque el contador tiene una entrada de reloj de 100 MHz, solamente cambiará de estado cuando la llegada del flanco activo de la señal de reloj coincida con un nivel alto en el *clock enable*. El diseño es totalmente síncrono ya que todas las entradas de reloj de los dispositivos síncronos que componen el circuito están conectadas al mismo punto.

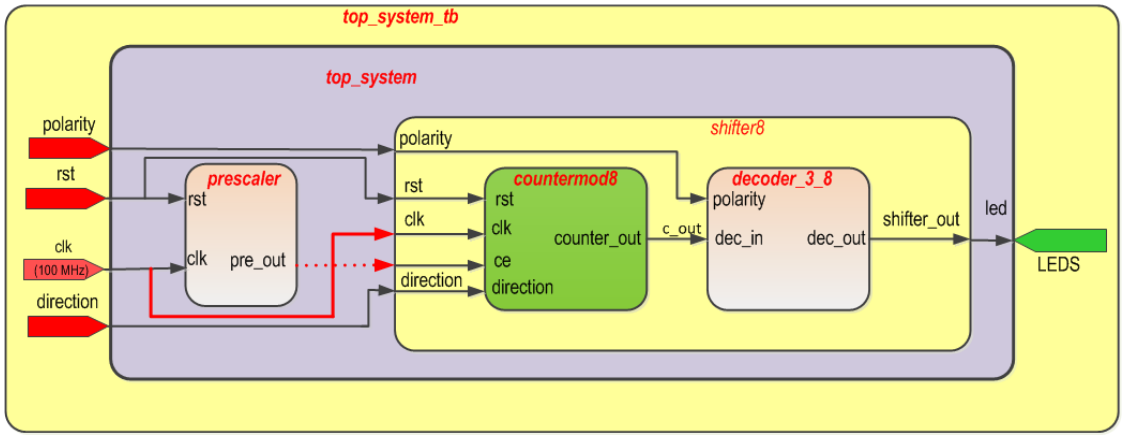


Figura 10.- Modelo síncrono del ejemplo de diseño.

3.3.- Modelo 3: DCM. Digital Clock Manager.

La Figura 11 muestra el tercer modelo del ejemplo de diseño basado en la utilización de un gestor de reloj (DCM). El reloj maestro de la placa se conecta a la entrada de reloj del DCM y la salida de este a las entradas de reloj de los elementos síncronos del circuito. El objetivo de este modelo es eliminar el *skew* en la señal de reloj, es decir, el gestor de reloj compensa el retardo introducido en el desplazamiento de la señal de reloj para que esta llegue a todos los elementos síncronos al mismo tiempo.

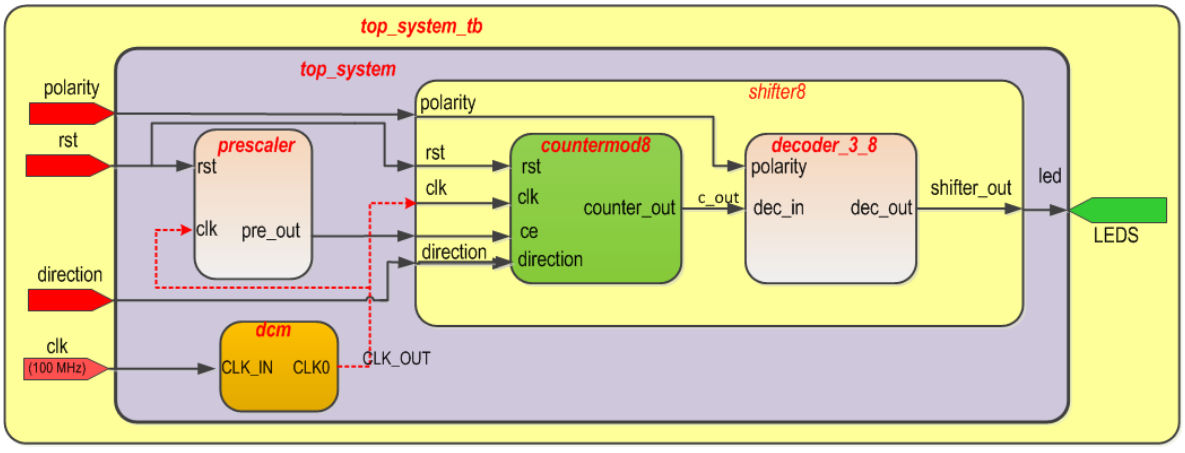


Figura 11.- Modelo 3 del ejemplo de diseño con gestor de reloj (DCM)

3.4.-Modelo 4. Prescaler implementado con el CORE-GENERATOR.

Finalmente, en la Figura 12 se muestra el modelo del desplazador en el cual se ha implementado el *prescaler* utilizado la herramienta *CORE-GENERATOR* la cual viene integrada en el entorno ISE. Esta herramienta acelera considerablemente el proceso de diseño al utilizar *IPs (Intellectual Property)* que se pueden parametrizar y por lo tanto adaptar a las necesidades del circuito a implementar. En el apartado 4.7 donde se trata el funcionamiento de las herramientas se explicará el funcionamiento de este entorno.

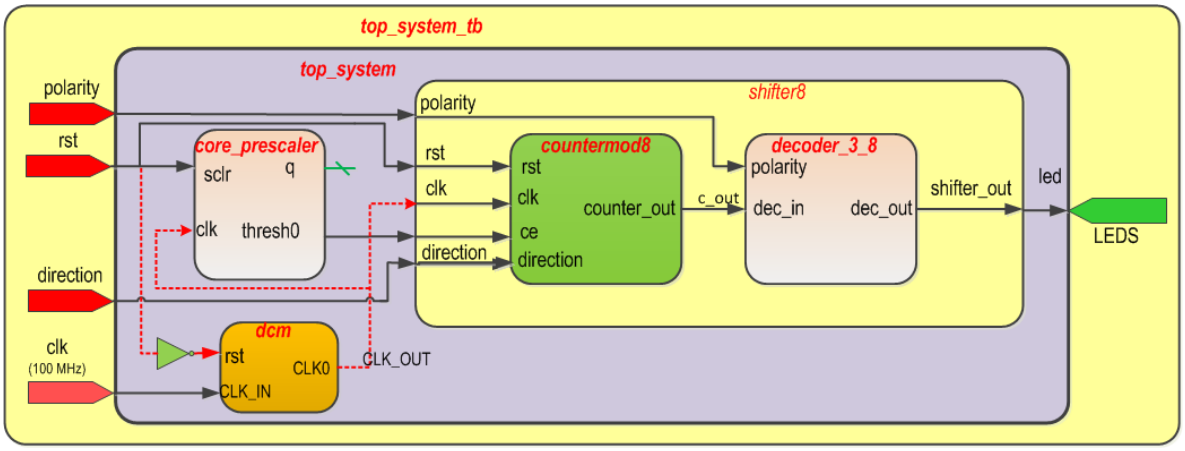


Figura 12.- Modelo 4. *Prescaler* implementado con la herramienta *CORE GENERATOR*.

4.- Flujo de diseño: Especificación del modelo a través de código VHDL.

Desde un punto de vista práctico, antes de afrontar el proceso de diseño se debe llevar a cabo una planificación de las etapas que entrarán en juego durante todo el proceso y del presupuesto disponible para ello. Lógicamente, este aspecto aunque importante, no se trata en este tutorial. La siguiente etapa en el proceso se centra en la especificación del diseño (Figura 13). Para ello se hace uso lenguajes de descripción hardware como VHDL y de las herramientas que ayudarán a su creación, fundamentalmente un editor de texto. Alternativamente, se puede utilizar esquemas para introducir el diseño aunque esto último no se contempla en el tutorial.

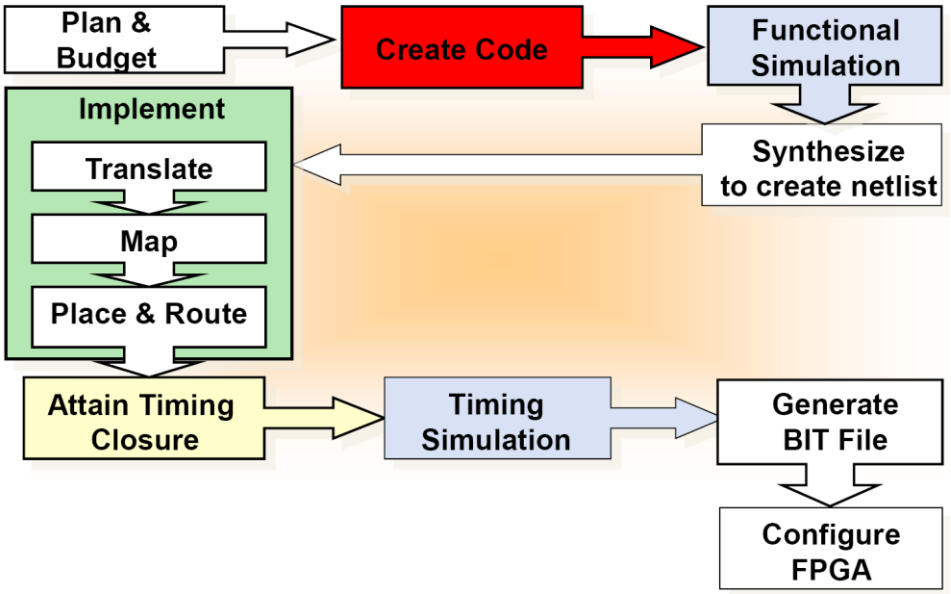


Figura 13.- Flujo de diseño. Especificación del modelo.

4.1.- Estructura de directorios.

Antes de comenzar con la especificación del código, y con el objetivo de organizar los ficheros que se irán generando a lo largo de proceso de diseño, es aconsejable crear la estructura de directorios que se propone a continuación (Figura 14), todos ellos localizados en el directorio TUTORIAL:

- **Sources:** este directorio alojará todos los ficheros VHDL que representan el diseño, junto con los ficheros de comandos del simulador ISIM.
- **Cores:** este directorio se usa para almacenar los ficheros generados por la herramienta *Core Generator*.
- **IMP_nombre_del_proyecto:** este directorio almacenará los ficheros generados por las herramientas de síntesis e implementación. Para el ejemplo del tutorial se puede crear el directorio *IMP_Tutorial*.

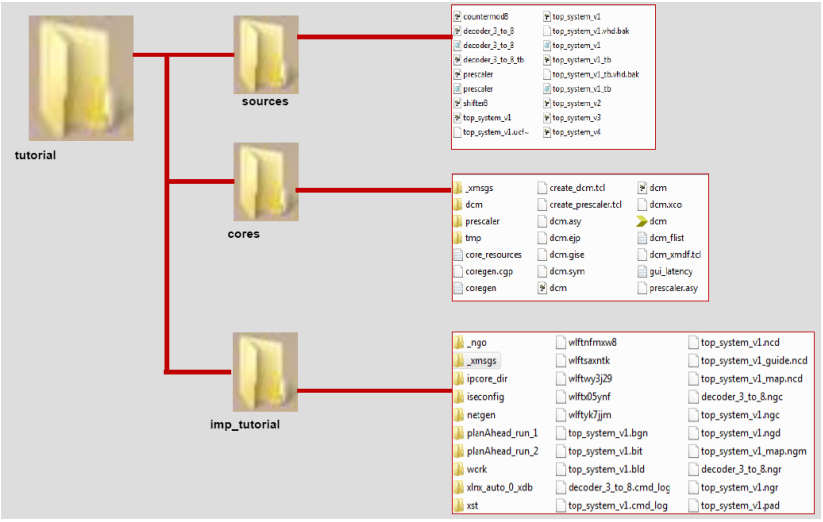


Figura 14.- Estructura de directorios.

4.2.- Herramientas dentro del entorno *ISE Design Suite*.

En entorno *ISE Design Suite* engloba distintas herramientas que se explicarán a continuación y que servirán para la cubrir todas las fases de diseño. Aunque hay distintas versiones, en general todas ellas incluyen las siguientes herramientas:

- *Project Navigator:* se utiliza para gestionar los ficheros generados durante el proceso de diseño y también permite ejecutar los distintos procesos que se llevan a cabo sobre los mismos: desde simulación, síntesis e implementación hasta la generación del fichero de configuración *(bitstream).*
- *XST Synthesis:* es la herramienta de síntesis que permite crear los ficheros de conexiones a partir de código HDL o esquemas.
- *ISE Simulator (ISim):* se trata de la herramienta de simulación que permite validar el comportamiento del diseño.
- *CORE Generator:* como ya se ha comentado, permite acelerar el proceso de diseño al incluir IP parametrizables.

4.3.- Especificación del diseño: *ISE Project Navigator*.

Para arrancar el entorno ISE se busca el icono que aparece en la Figura 15 y se hace doble *click*.

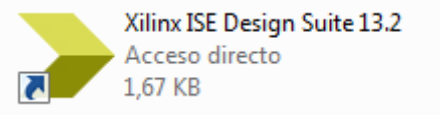


Figura 15.- Icono del entorno *ISE Design Suite*.

A continuación aparecerá la ventana principal de la herramienta *ISE Project Navigator* que se muestra en la Figura 16.

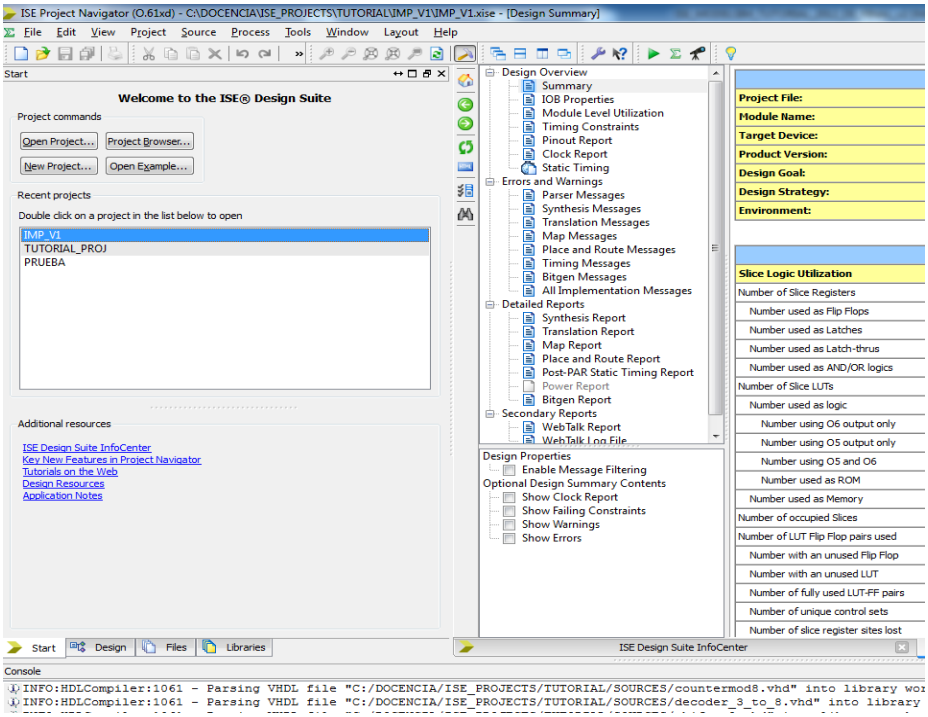


Figura 16.- Ventana principal de la herramienta *ISE Project Navigator*.

La ventana principal del *ISE Project Navigator* está formada a su vez por diferentes subventanas y/o paneles que se pueden visualizar utilizando la barra de pestañas.

La primera de ellas es la ventana de comienzo (pestaña o panel *start*) o de arranque la cual proporciona acceso rápido a los proyectos que ya se hayan creado, recursos a los que se acceden con frecuencia, documentación y tutoriales, entre otros. Por lo tanto, a través de esta ventana, se pueden crear proyectos o abrir proyectos existentes.

El siguiente panel es el de diseño (Figura 17). En él se puede configurar la vista (*view*) de diseño en función de si se va a realizar la implementación o la simulación del mismo. La configuración de la vista del diseño condiciona los fuentes o ficheros del diseño sobre los que se puede trabajar. También se muestra la jerarquía que indica las dependencias entre los ficheros que forman parte del diseño, el nombre del proyecto y el dispositivo destino. Por último, en este panel se proporciona la ventana de procesos donde para cada fichero del diseño y en función de su tipo, se establecen los procesos que se pueden ejecutar sobre los mismos. Por lo tanto, se trata de un panel que es sensible al contexto cambiando en función del tipo de fichero fuente seleccionado en la ventana de la jerarquía.

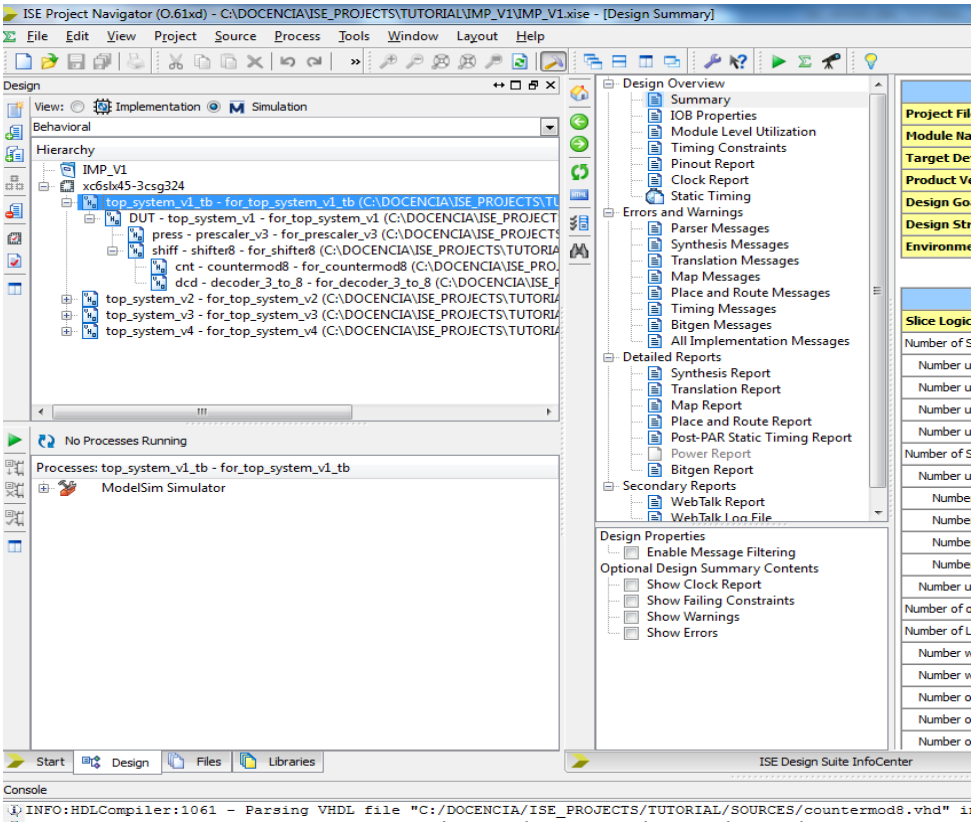


Figura 17.- Ventana o panel de diseño

La ventana o panel de procesos permite el acceso a las siguientes funciones:

- Informes de diseño: proporciona acceso a los informes de diseño, mensajes y resúmenes de los datos generados por los procesos.
- Utilidades de diseño: proporciona acceso a la generación de símbolos, instanciación de platillas, vista de los últimos comandos ejecutados y compilación de las bibliotecas de simulación.
- Restricciones de usuario: permite la especificación de restricciones temporales y de localización.
- Síntesis: proporciona acceso a distintas herramientas para poder, chequear la sintaxis del modelo, llevar a cabo su síntesis, ver el resultado de la síntesis en forma de esquemático en dos versiones, RTL y tecnológica y finalmente poder consultar el informe generado por la herramienta de síntesis.
- Implementación del diseño: proporciona acceso a las herramientas de implementación y de análisis posterior a la implementación.
- Generación de fichero de programación: permite generar el fichero de configuración *bitstream*.
- Configuración del dispositivo destino: proporciona acceso a las herramientas de configuración para crear los ficheros de configuración *(bitstream)* y programarlos en los dispositivos.

La ventana de ficheros (Figura 18) proporciona una lista de todos los ficheros fuentes generados en el proyecto.

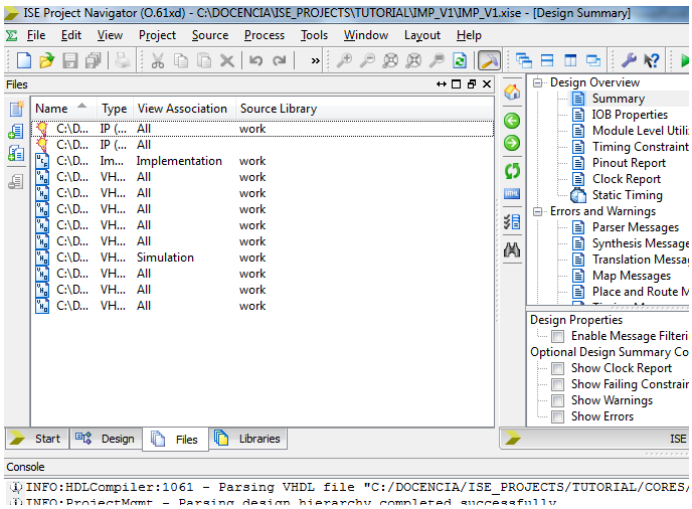


Figura 18.- Ventana de ficheros.

Finalmente, la ventana de bibliotecas (Figura 19) permite gestionar las bibliotecas HDL y los ficheros fuentes asociados a estas bibliotecas.

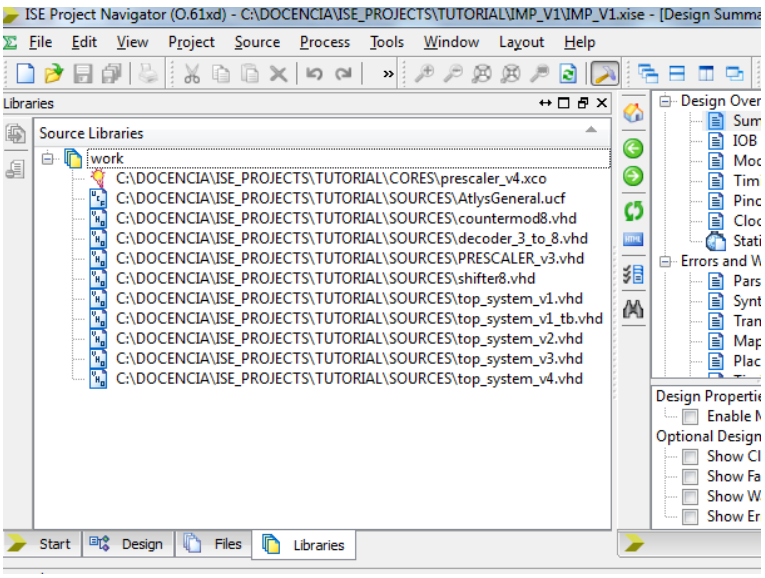
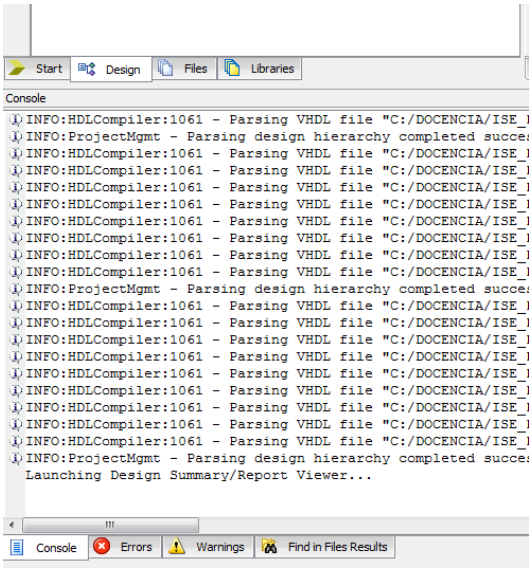


Figura 19.- Ventana de bibliotecas.

Existen otros paneles (Figura 20) que permiten realizar diversas funciones. Estos paneles son:

- Consola: proporciona una salida estándar para los procesos que se ejecutan desde el *Project Navigator*. Permite representar errores, avisos y mensajes informativos en general. Los errores se representan mediante una cruz roja mientras que para los avisos se utiliza un signo de admiración amarillo como se verá en el apartado 8.
- Panel de errores: esta ventana solamente muestra los mensajes de error filtrando el resto.
- Panel de avisos: esta ventana muestra únicamente los avisos filtrando el resto.



```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:ProjectMgmt - Parsing design hierarchy completed succe
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:ProjectMgmt - Parsing design hierarchy completed succe
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/DOCENCIA/ISE_
INFO:ProjectMgmt - Parsing design hierarchy completed succe
Launching Design Summary/Report Viewer...
```

Figura 20.- Otros paneles.

Una vez que se ha introducido de forma breve el conjunto de ventanas que conforman el entorno, el siguiente paso consiste en la creación de un proyecto.

4.4.- Creación de un proyecto nuevo.

Para poder gestionar el diseño de forma cómoda, *ISE Project Navigator* permite crear proyectos, los cuales contienen todos los ficheros relacionados con el diseño. Primero se crea el proyecto y después se deben añadir los ficheros fuentes. Dentro del entorno de trabajo, una vez que se ha creado el proyecto y generado los fuentes que modelan el diseño, se puede ejecutar los procesos asignados a cada uno de ellos. Para facilitar la tarea, el entorno proporciona un asistente denominado *"wizard"* que guía en el proceso de creación de nuevos proyectos.

La creación de un nuevo proyecto se lleva a cabo siguiendo los pasos que se detallan a continuación:

- En la ventana principal del *Project Navigator* se selecciona ***File*** y luego ***New Project*** apareciendo la ventana de ayuda para la creación de proyectos (Figura 21). En esta ventana se debe seleccionar el tipo de proyecto (HDL o esquemático), el nombre del mismo, su localización y el directorio de trabajo. Para pasar a la siguiente ventana se pulsa ***Next***.

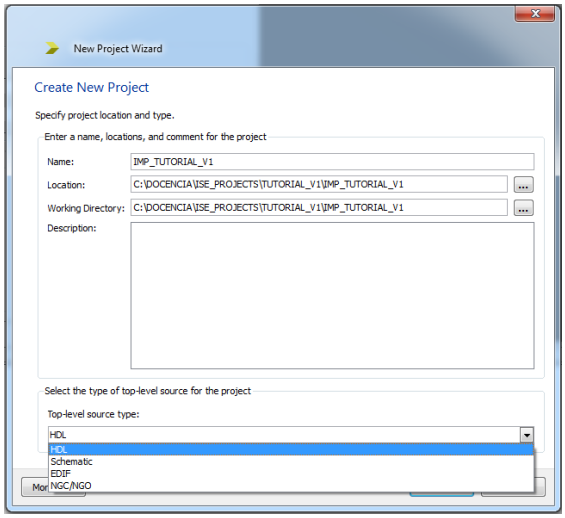


Figura 21.- Asistente para la creación de un nuevo proyecto.

En la siguiente ventana (Figura 22) se selecciona el dispositivo destino que dependerá de la placa usada en el laboratorio. En este tutorial se utiliza la placa *Atlys* diseñada y fabricada por la empresa *Digilent*. Esta placa contiene una FPGA *Spartan 6LX45*. Más información sobre esta placa se puede encontrar en: *http://www.digilentinc.com/*. La Figura 22 muestra cada uno de los campos que se deben completar junto con los valores que se deben introducir. Para la FPGA se selecciona la familia, el dispositivo, el encapsulado y finalmente la velocidad. En cuanto a las herramientas se selecciona como herramienta de síntesis *XST* y como herramienta de simulación *ISim*. Una vez se han completado los diferentes campos se hace *click* sobre ***Next*** apareciendo una ventana que resume los parámetros configurados para el proyecto.

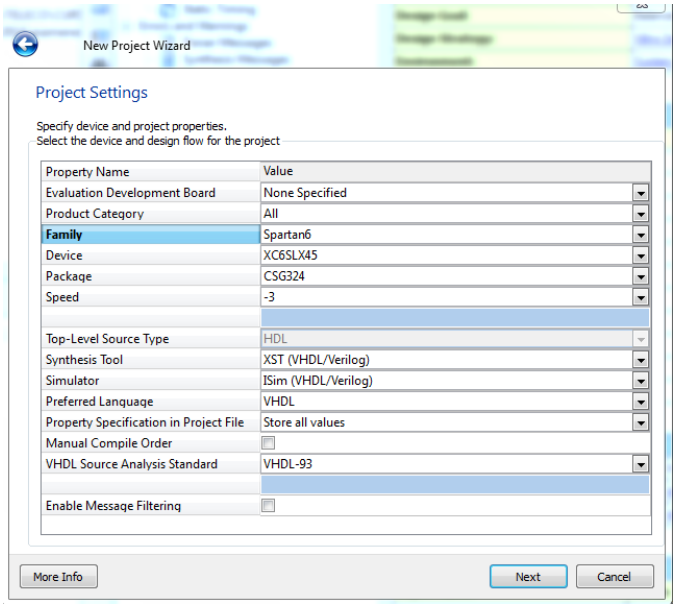


Figura 22.- Selección del dispositivo y herramientas para el proyecto.

4.5.- Creación de un fichero fuente en VHDL.

La creación de fichero fuente VHDL se puede realizar a través de un simple editor de líneas. Sin embargo existen herramientas que facilitan esta tarea. El propósito es doble: por un lado, estas herramientas facilitan la especificación automatizando parcialmente el proceso. Por otro lado, pretenden minimizar el número de errores desde un punto de vista sintáctico que se suelen introducir a la hora de especificar en VHDL.

La herramienta ISE proporciona un asistente que a través de plantillas del lenguaje facilita la tarea de especificación. Sin embargo, en este tutorial se explicará el editor *Emacs*. Es recomendable el uso de *Emacs* debido a que tiene precargadas plantillas con la sintaxis de VHDL que facilita, de forma notable, la creación de los archivos fuentes. En este apartado no se pretende realizar un estudio minucioso de todas las posibilidades que ofrece el editor *Emacs*, estudiando sólo una pequeña parte de ellas.

Con respecto a la creación de un fichero VHDL se puede utilizar el editor del ISE. Para ello se abre un nuevo fichero con los comandos ***File → New → Text File*** y se salva con el nombre que se quiera asignar al modelo VHDL y extensión .vhd. En este punto el editor activo es el del ISE. Para lanzar el *emacs* se puede abrir el archivo con los comandos ***File → Open File*** y luego se selecciona el fichero que desea abrir indicando como editor el *emacs*.

4.5.1.- Especificación de modelos VHDL mediante el editor emacs.

Para iniciar el editor *Emacs* se hace doble clic en el icono de acceso directo de la figura 23 apareciendo la ventana de trabajo de la Figura 24.

Figura 23.- Icono de acceso directo del *Emacs*.

Hay que tener en cuenta que el ***emacs*** es un editor pensado para trabajar en UNIX, por lo que no acepta los comandos típicos de Windows (*Ctr+C, Ctr+V,* etc.). No obstante esto no supone un inconveniente importante.

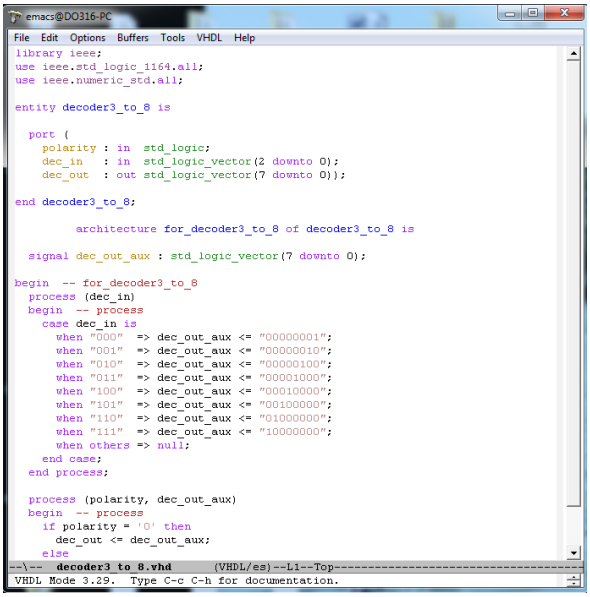


Figura 24.- Ventana principal de trabajo del *emacs.*

La ventana de trabajo de *emacs* (Figura 24) se divide en tres zonas: la superior en la que se encuentra la barra de herramientas, la central o zona de trabajo en la que se escribe las líneas de código, y la inferior o ventana de comandos, desde la que se pueden ejecutar algunos comandos de la herramienta.

En la pestaña *File* (Figura 25) de la barra de herramientas se accede a los comandos de manejo de archivos, de forma similar a otros programas de Windows, con la salvedad de que no aparece el comando ***new***. Un archivo nuevo se crea con el comando ***File → Open File***, y una vez seleccionada la carpeta donde queremos crear el archivo se le asigna un nombre que no exista en dicha carpeta.

Figura 25.- Pestaña *File* del editor *Emacs.*

Cuando se está editando un archivo VHDL (extensión .vhd) aparece un nuevo elemento en la barra de herramientas, de nombre VHDL (Figura 26) en la barra de herramientas. Esto indica que se ha utilizado la extensión correcta para el archivo VHDL.

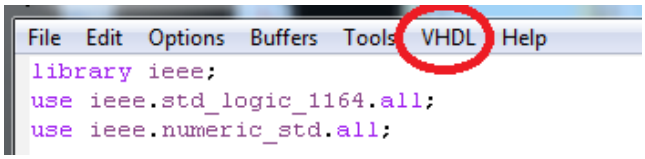


Figura 26.- Pestaña VHDL en la barra de herramientas.

Una vez seleccionado el archivo de entrada, ya se puede a introducir las líneas de texto como si de otro editor de líneas se tratase. Para la operación de guardar archivo se utiliza el comando ***Edit → Save buffer***. En este entorno del *emacs*, el término *"buffer"* equivale a *"archivo"* en un entorno de Windows. En este punto ya se podría crear los archivos VHDL de la práctica, si bien la utilización del editor *emacs* viene justificada por la sencillez en la utilización de las plantillas de construcciones típica de VHDL que se explicarán a continuación.

Para acceder a las plantillas de VHDL se selecciona ***VHDL→ Template***, apareciendo las opciones de la Figura 27. Dentro de las plantillas se pueden introducir datos como:

- Construcciones, con ***VHDL Construct 1*** y ***VHDL Construct 2***. Permite acceder a las diferentes sentencias y construcciones de VHDL.

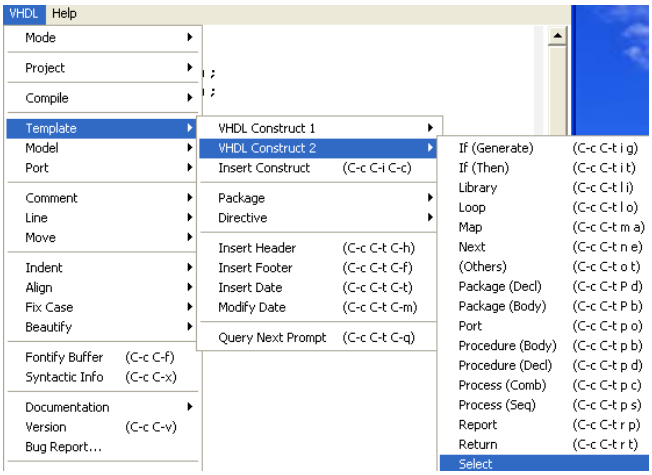


Figura 27.- Añadir construcciones VHDL mediante plantillas.

- Paquetes. Con ***Package*** se puede insertar un paquete VHDL (Figura 28), automáticamente también se añade la biblioteca.

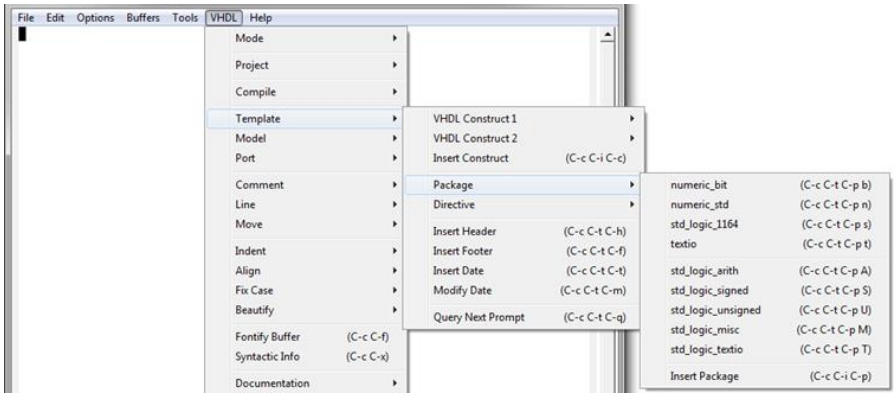


Figura 28.- Ventana para especificar un paquete.

- Directivas. Permite añadir pragmas.
- Otros elementos organizativos como son cabecera (***Insert Header***), pie de página (***Insert Footer***) y fechas.

Dependiendo del elemento seleccionado habrá casos (p.e si se selecciona ***entity***) en los que se deban introducir datos adicionales. Para estos casos, en la ventana de comandos se pregunta por el valor del dato a introducir, validándose con ↵ (*enter*).

Por ejemplo, para crear un archivo VHDL en primer lugar se debe añadir los paquetes que se van a utilizar. En este caso no es necesario añadir ningún dato, a no ser que se vaya a utilizar un paquete creado por el usuario. A continuación se introduce la declaración de la entidad. Para ello se selecciona la construcción VHDL ***entity*** y se introduce el nombre decoder3_to_8 (Figura 29), validando con ↵.

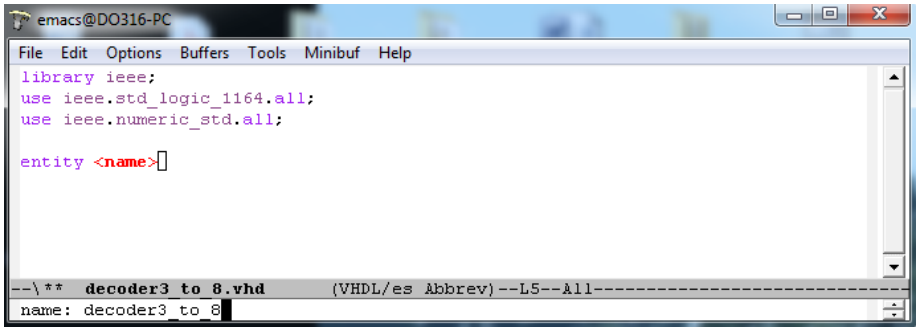


Figura 29.- Declaración de una entidad.

Como la declaración de una entidad está compuesta por la declaración de genéricos y puertos, a continuación se pedirá el nombre de los primeros (***generic***), en el caso de haberlos se introducen uno a uno sus identificadores (Figura 30). Para cada genérico se solicitara el tipo de datos y su valor por defecto. Sí no hay genéricos se pulsa ↵, pasando a introducir el nombre de los puertos.

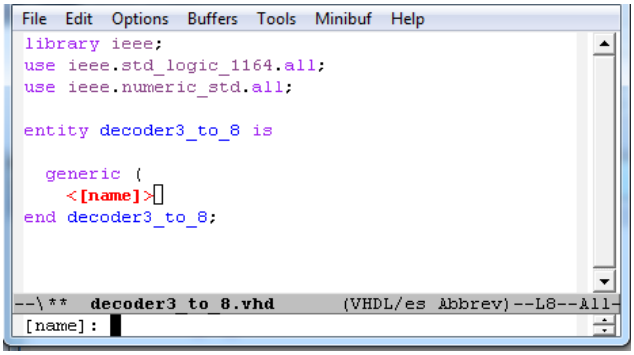


Figura 30.- Especificación de genéricos.

Por cada puerto o conjunto de puertos, una vez introducido su/sus identificadores y validados con ↵ se solicita que se especifique el modo (tipo de acceso, ***in***, ***out*** o ***inout***) del puerto y se valida nuevamente con ↵. Paso seguido se debe introducir el tipo de datos, y a continuación, si se desea, un comentario (Figura 31). El proceso se repite para cada uno de los puertos hasta que no haya más puertos a especificar, pulsando ↵ y se finaliza la plantilla de ***entity***. Independientemente de la construcción que se esté completando, cuando ***emacs*** solicite un parámetro que se quiera ignorar, caso de los genéricos anteriores, se debe pulsar ↵.

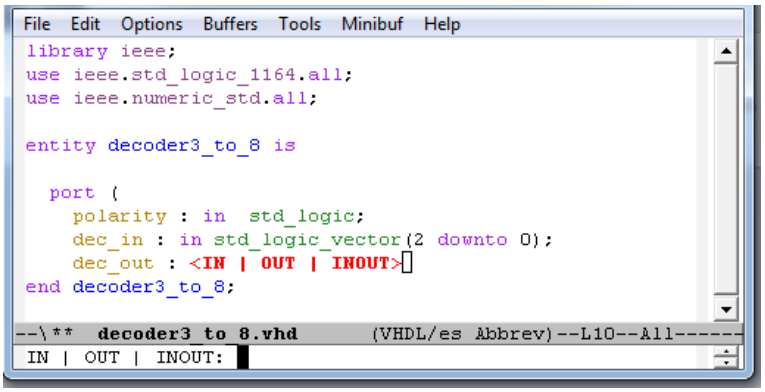


Figura 31.- Especificación de los puertos de la entidad.

Emacs tiene la posibilidad de autocompletar palabras, las cuales puedan ser tanto palabras reservadas del VHDL como un identificador utilizado en el diseño (puertos, señales, etc.). Para ello basta teclear los primeros caracteres del comando y con la tecla ***Tab*** (↹) se puede ir avanzando por las distintas opciones que proporciona el editor. Obviamente, cuantos más caracteres de la palabra reservada se introduzcan más rápido se accederá a la misma. Esta facilidad permite introducir construcciones VHDL de igual forma que se puede hacer desde la pestaña de plantillas. Para ello, una vez completado el nombre de una palabra reservada de VHDL con la tecla ***Tab***, a continuación se pulsa la barra espaciadora para poder acceder al primer campo del comando. De igual forma que para el caso de las plantillas, el valor de cada campo se valida con ↵, pasando a introducir los datos para el siguiente campo, si procede.

Un caso que merece especial atención es la creación de un proceso, ya que dependiendo de su tipo los parámetros a introducir son diferentes. En este caso, independientemente desde donde se acceda a la plantilla en primer lugar se pregunta si el proceso es combinacional o secuencial (Figura 32).

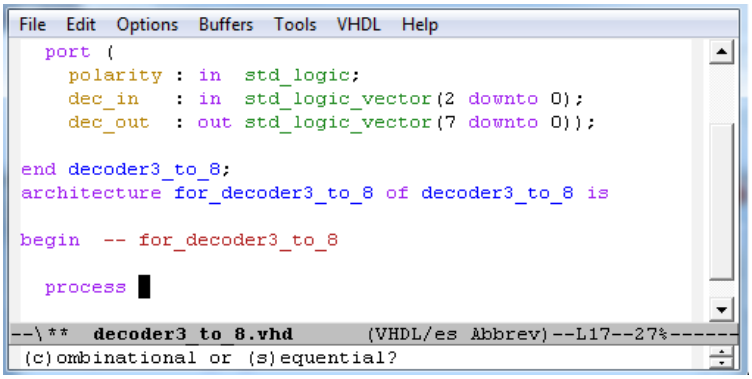


Figura 32.- Especificación del tipo de proceso.

En ambos casos, el primer paso consiste en añadir una etiqueta para el proceso, aunque es opcional y se puede ignorar. A continuación, si el proceso es combinacional, se solicita el nombre de las señales que forman parte de la lista de sensibilidad, y al validar con ↵ se crea una estructura similar a la de la Figura 33.

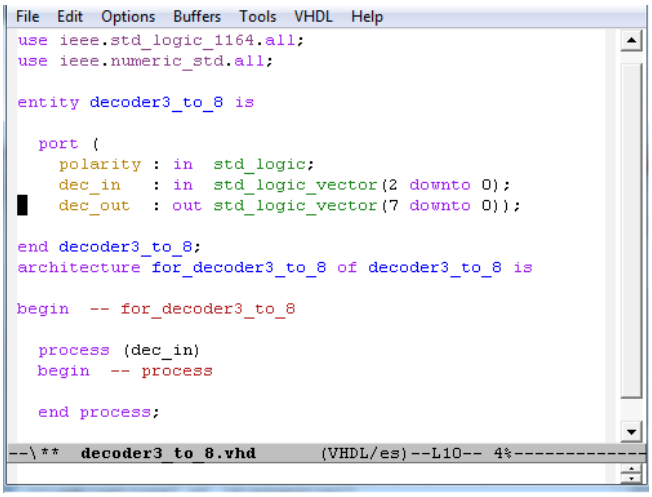


Figura 33.- Estructura de un proceso combinacinal.

Si el proceso es secuencial en primer lugar se solicita el nombre de la señal de sincronismo (reloj) y a continuación una señal asíncrona de inicialización. Una vez introducidos al validar con ↵ se crea una estructura similar a la de la Figura 34. En este punto solo resta introducir el código que modele su funcionamiento.

```
-- purpose: <[description]>
process (clk, rst)
begin  -- process
  if rst = '0' then

  elsif clk'event and clk = '1' then

  end if;
end process;
```

Figura 34.- Estructura de un proceso secuencial.

Es recomendable utilizar la facilidad de autocompletar e introducción de campos ya que se puede generar una gran cantidad de código libre de errores de sintaxis. Téngase en cuenta que en la depuración de código se invierte un tiempo importante en eliminar estos errores.

Tal y como se dijo al principio, *emacs* es un editor creado para trabajar en un entorno UNIX, con lo que no acepta algunos de los comandos típicos de Windows, tal es el caso de ***Crt+c*** para copiar y ***Ctr+v*** para pegar. Para copiar, tan sólo es necesario seleccionar con el cursor la porción de código que se quiere copiar, pasando de forma automática al portapapeles. Para pegar se ejecuta el comando ***Edit*** → ***Paste*** o ***Ctr+y***.

Para poder seleccionar texto con las teclas May(↑) y los cursores, como todas las herramientas de Windows, es necesario teclear: ***Alt+x***, para entrar en la ventana de comandos, y desde aquí se introduce ***pc-selection-mode***.

En la tabla 1 se proporciona una lista de los comandos más utilizados dentro del editor emacs.

Tabla 1.- Lista de los comandos más utilizada en el editor emacs.

COMANDO	TECLA	FUNCION
Deshacer cambios	C-x / C_	Deshacer cambios uno por uno en todo el buffer
	C-x u	Deshacer cambios en una región
División de la ventana principal en subventanas	C-x 0	Borra la ventana activa
	C-x 1	Borra todas las ventanas menos la activa
	C-x 2	Divide horizontalmente la ventana activa en dos ventanas
	C-x 3	Divide verticalmente la ventana activa en dos ventanas
	C-x o	Lleva el cursor a la próxima ventana
	C-x -v	Desplaza hacia abajo el texto de la próxima ventana (como si en ella se hiciera AvPág)
Abrir archivo	C-x C-f	Abrir un archivo, hay que indicar el nombre
	C-x d	Abre un directorio
Guardar archivo	C-x C-s	Guardar archivo
	C-x C-w	Guardar como

Moverse por un archivo	C-x s	Guardar, preguntando si se desea sobrescribir
	C-Inicio / M-<	Principio del buffer
	C-Fin / M->	Final del buffer
	M-Derecha / M-f	Próxima palabra
	M-Izquierda / M-b	Palabra anterior
	C-Arriba	Próximo párrafo
	C-Abajo	Párrafo anterior
	M-r	Línea central
	C-l	Centrar pantalla sobre el cursor
Borrar texto	C-k	Elimina el texto hasta el final de la línea
	M-RETRO	Elimina una palabra hacia atrás
	M-d	Elimina la próxima palabra
	C-w	Elimina el texto seleccionado
	M-z	Elimina hasta la próxima ocurrencia del carácter que se teclee inmediatamente detrás del mandato
	M-w	Añade el texto seleccionado a la memoria intermedia, como si se hubiera eliminado, pero sin eliminarlo
Pegar texto	C-y	Pega texto copiado
	M-y	Pega texto borrado. Antes hay que ejecutar el comando C-y
Buscar	C-s	Busca hacia delante
	C-r	Busca hacia atrás
	C-w	Incorpora una palabra a la búsqueda
	ESC/ ENTER	Finaliza la búsqueda
	M-%	Buscar y Reemplazar
Comentar región	C-c C-c	Comentar/descomentar una región seleccionada

C=Ctrl.; M(meta)= Alt-izquierda o Esc.

4.5.2.- Especificación de un banco de pruebas (test bench).

La especificación de un banco de pruebas para poder realizar la simulación, se puede llevar a cabo de una forma muy sencilla a través del editor *emacs*. Para ello se debe primero seleccionar los puertos de la entidad del modelo bajo test (Figura 35) y luego copiarlos a través de ***VHDL→Port→Copy***

```
entity decoder3_to_8 is

  port (
    polarity : in  std_logic;
    dec_in   : in  std_logic_vector(2 downto 0);
    dec_out  : out std_logic_vector(7 downto 0));

end decoder3_to_8;

architecture for_decoder3_to_8 of decoder3_to_8 is
```

Figura 35.- Seleccionar puertos de la entidad bajo test.

Una vez seleccionados los puertos, el siguiente paso consiste en pegarlos como un *Test Bench* a través de ***VHDL->Port-> Paste as Instance*** (Figura 36)

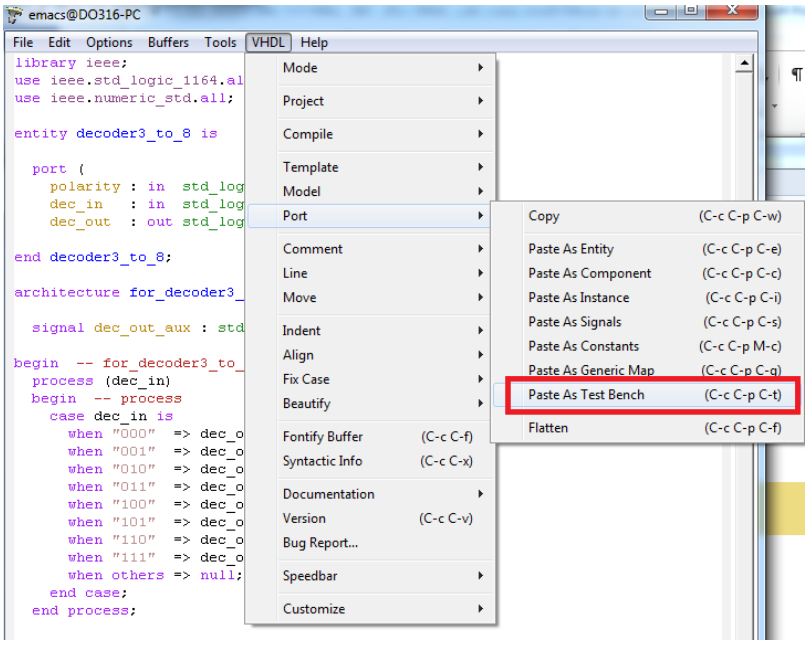


Figura 36.- Pegar los puertos como un *Test Bench.*

A continuación se introduce un nombre para la arquitectura del *Test Bench* (Figura 37) creándose una platilla que se deberá completar en función de los estímulos necesarios para simulador en modelo. El listado 2 muestra el código completo del *Test Bench* para el decodificador.

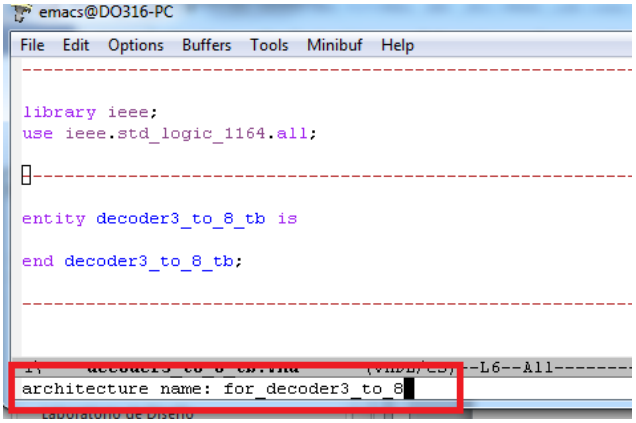


Figura 37.- Añadir el nombre de la arquitectura del *Test Bench*.

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

entity decoder3_to_8_tb is

end decoder3_to_8_tb;

architecture for_decoder3_to_8 of decoder3_to_8_tb is

  component decoder3_to_8
    port (
      polarity : in  std_logic;
      dec_in   : in  std_logic_vector(2 downto 0);
      dec_out  : out std_logic_vector(7 downto 0));
  end component;

  signal polarity_i  : std_logic                    := '0';
  signal dec_in_i    : std_logic_vector(2 downto 0) := "000";
  signal dec_out_o   : std_logic_vector(7 downto 0);
  signal clk, rst    : std_logic                    := '0';
  signal countermod8 : unsigned(2 downto 0);

begin  -- for_decoder3_to_8

  DUT : decoder3_to_8
    port map (
      polarity => polarity_i,
      dec_in   => dec_in_i,
      dec_out  => dec_out_o);

  rst             <= '1', '0' after 100 ns;
  clk             <= not clk  after 5 ns;
  process (clk, rst)
  begin  -- process
    if rst = '1' then
      countermod8 <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge
      if countermod8 = 7 then
        countermod8 <= (others => '0');
      else
        countermod8 <= countermod8+1;
      end if;
    end if;
  end process;

  polarity_i <= '1' after 500 ns;
  dec_in_i   <= std_logic_vector(countermod8);

end for_decoder3_to_8;
```

Listado 2.- *Test Bench* para el decodificador.

4.6.- Chequear la sintaxis del modelo VHDL.

Cuando se configura el entorno ISE para implementación se puede realizar un chequeo sintáctico en el apartado de síntesis. En relación a este punto, aunque realizar la síntesis no es necesario para llevar a cabo la simulación funcional, se recomienda someter al código a un proceso de síntesis con el objetivo de verificar que la especificación del modelo VHDL es sintetizable. De nada sirve que la simulación arroje resultados satisfactorios en relación a la funcionalidad del modelo, si éste no es sintetizable. En la Figura 38 se muestra como hacer el chequeo de la sintaxis del modelo.

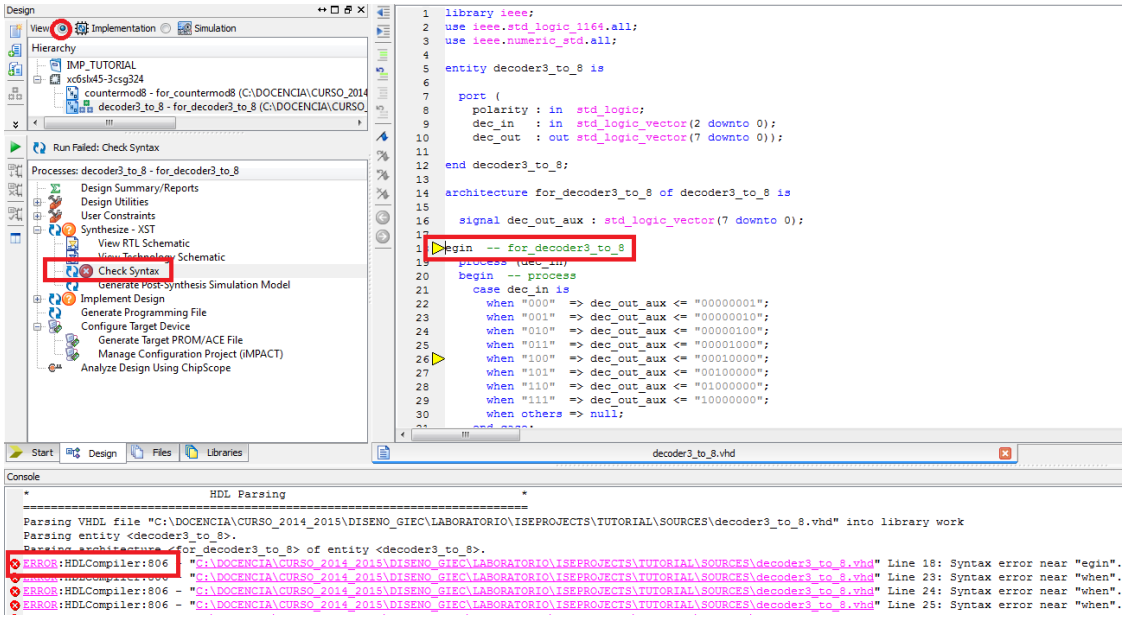


Figura 38.- Comprobación de la sintaxis del modelo VHDL.

Por lo tanto, para poder verificar la sintaxis en el panel View se selecciona *Implementation* y en el panel de procesos se selecciona *Check Syntax*. En la Figura 38 se puede ver cómo se comporta la herramienta cuando se encuentra con un error. En la consola aparece la línea del código VHDL donde se encuentra el error. Seleccionando el error y haciendo doble *click* la herramientas abre el fichero que contiene el error y marca con una flecha amarilla la posición donde se encuentra.

4.7.- Entrada del diseño: cómo añadir ficheros ya creados.

Para añadir ficheros existentes, es decir, creados con anterioridad en la ventana principal del *ISE Project Navigator* se selecciona ***Project → Add Source*** eligiendo a continuación el fichero fuente que se quiere añadir. Hay que resaltar que cada fichero fuente que se incorpora al proyecto tiene asignado una asociación, es decir, si es solamente para simulación, como ocurre con los bancos de pruebas o también se implementará. Es importante que este paso (Figura 39) se especifique claramente ya que en caso contrario el ISE no seleccionará correctamente los ficheros para llevar a cabo la implementación.

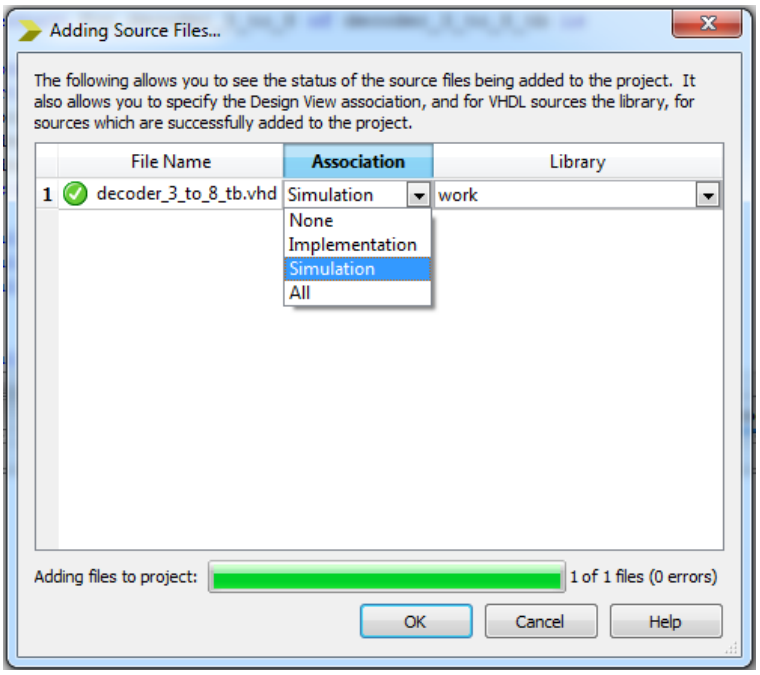


Figura 39.- Definición del tipo de fichero fuente añadido al proyecto.

En general, todos los ficheros VHDL excepto los bancos de pruebas deben tener la asociación ***all***, para poder llevar a cabo tanto la simulación como la síntesis y la implementación.

A veces los ficheros que se añaden al proyecto hacen referencia a otros modelos ya que se especifican mediante una jerarquía de componentes interconectados. En este caso, en el panel del entorno ISE que representa la jerarquía se muestra el modelo con una interrogación. Esto indica que el modelo no existe en el proyecto y, por consiguiente, deberá añadirse. Para ello, se puede seleccionar el modelo que falta, hacer *click* sobre el botón del ratón y luego ***Add Source*** en la ventana que aparece para posteriormente añadir el modelo en cuestión (Figura 40).

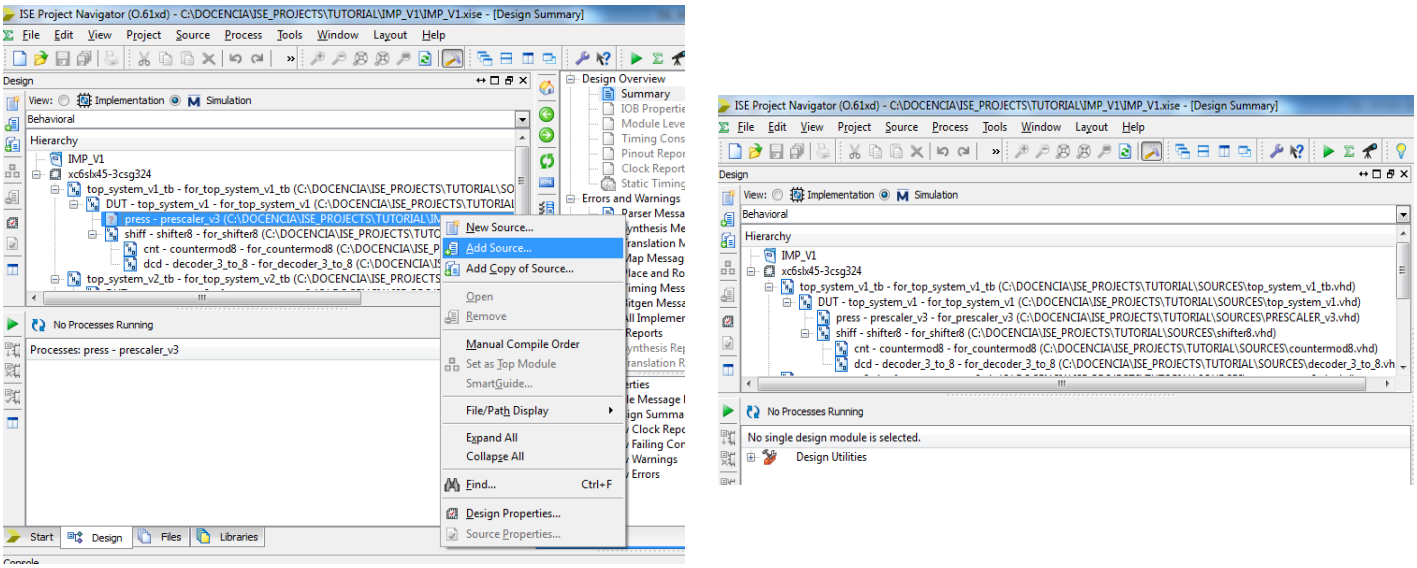


Figura 40.- Añadiendo ficheros fuente al proyecto.

4.8.- Creación de modelos basados en la herramienta CORE Generator Module.

En este tutorial ya se ha hecho referencia a la utilidad que tiene la herramienta *CORE Generator*. Se trata de una herramienta de diseño gráfica e interactiva que permite la creación de modelos de forma automática sin necesidad de escribir líneas de código.

En este apartado se explica cómo, a través de la misma, se especifica el *prescaler* del Modelo 4 del ejemplo de diseño (ver apartado 3.4).

Para crear un módulo *CORE Generator* se selecciona ***Project→New Source*** y después ***IP(Corengen&Architecture Wizard)*** (Figura 41). Claramente se trata de un nuevo fuente basado en el *CORE Generator* que se deberá añadir al proyecto. Para ello se deberá introducir el nombre del fichero (prescaler_v4) y su localización para a continuación pulsar ***Next***.

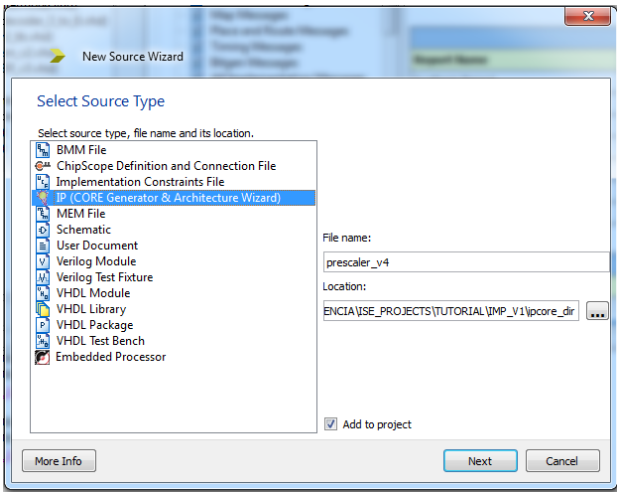


Figura 41.- Creación de un nuevo fuente basado en la herramienta CORE Generator.

El paso anterior lanza un asistente (Figura 42) que facilita considerablemente la tarea. A continuación se debe seleccionar el tipo de IP que para el Tutorial será ***Basic Elements → Counters → Binary Counter.*** Téngase en cuenta que el divisor de frecuencia se basa en un contador binario. Después se pulsa ***Next*** y después ***Finish*** para abrir el asistente del contador binario.

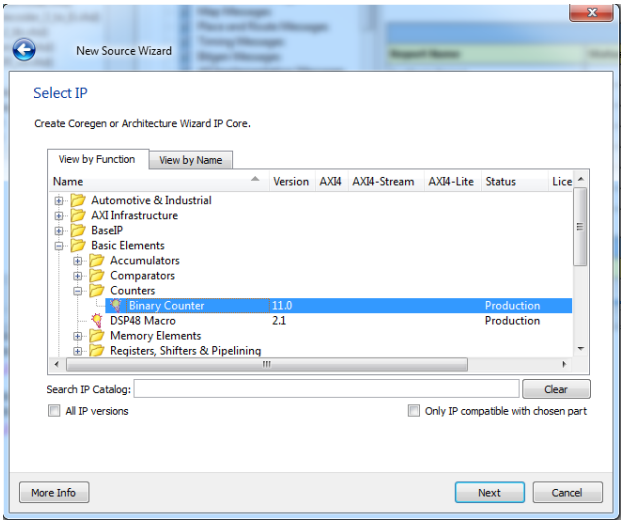


Figura 42.- Selección del IP.

El siguiente paso consiste en parametrizar el contador rellenado los distintos campos que aparecen en el asistente (Figura 43).

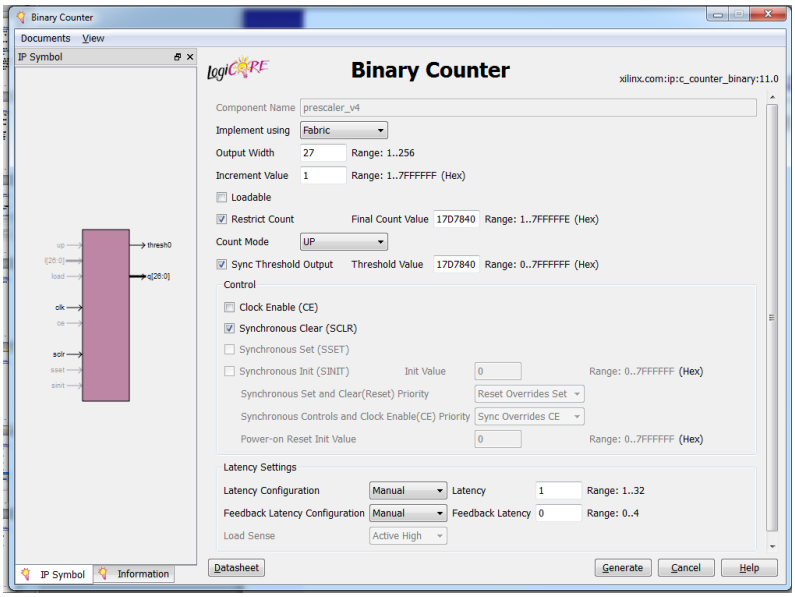


Figura 43.- Parámetros para el contador binario.

Para el contador binario, los campos relevantes para el tutorial son los siguientes:

- *Output Width*: especifica el rango de salida del contador.
- *Increment Value*: especifica en hexadecimal el valor de incremento mínimo del contador.
- *Restrict Count*: cuando se selecciona este parámetro, el contador se incrementará o decrementará hasta el valor especificado en el parámetro ***Final Count Value***. Cuando no

se impone restricción en la cuenta, el contador contará hasta el valor máximo que se pueda representar con el rango de salida especificado.

- *Count Mode:* este parámetro especifica si el contador cuenta de forma ascendente o descendente.
- *Sync Threshold Output:* cuando este valor es true, se genera la salida combinacional THRESH0.
- *Threshold Value:* es el valor en hexadecimal para el cual se activa la salida THRESH0.
- *Synchronous Clear (SCLR):* especifica si el pin de *clear* síncrono deber incluirse en el contador.

Para más información sobre el resto de los pines se puede hacer *click* sobre *Datasheet* y aparecerá un documento donde se explica la funcional del contador binario parametrizable.

Una vez parametrizado el contador se hace *click* sobre *Generate* para general el IP. Este proceso genera una serie de ficheros que se añaden al proyecto. Algunos de estos ficheros son:

- *Prescaler_v4.vho:* contiene las plantillas de declaración de componentes e instanciación que se pueden utilizar para instanciar el componente dentro del modelo VHDL que vaya a utilizar el componente generado.
- *Prescaler_v4.vhd:* se trata de un modelo VHDL del IP generado y se utiliza para realizar simulaciones funcionales.
- *Prescaler_v4.xco*: este fichero almacena la información de configuración del módulo generado y se usa como fichero fuente para el proyecto como se puede ver en la Figura 44.

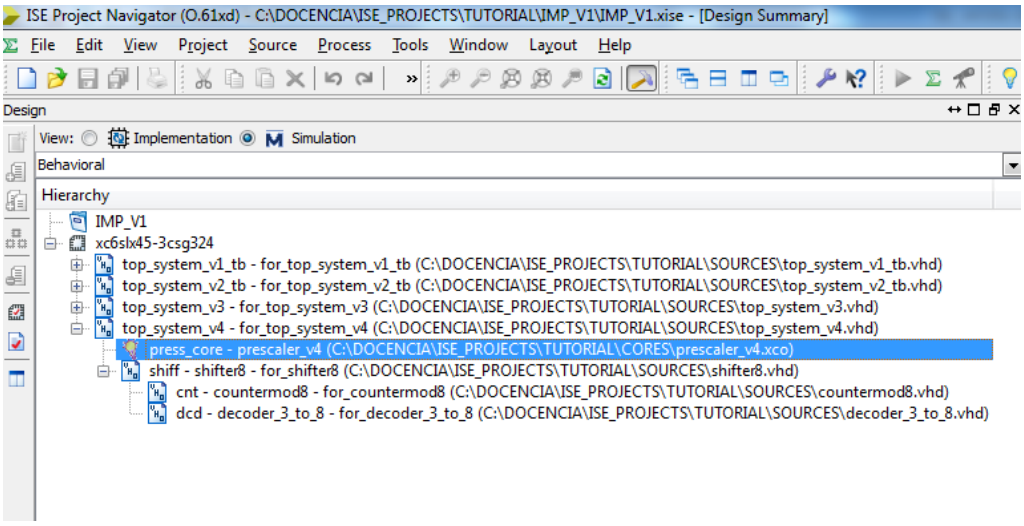


Figura 44.- Prescaler implementado con el CORE Generator.

Con los ficheros anteriores se puede realizar la instanciación del componente prescaler, en este caso para el modelo 4 del ejemplo de diseño. Para ello se abre el fichero que en modelo 4 del ejemplo de diseño podría llamarse *top_system_v4* y se añaden las platillas de declaración e instanciación que se encuentran en el fichero *prescaler_v4.vho* (Figura 45). Éste fichero se abre seleccionando ***File→Open*** y luego el fichero en cuestión el cual se puede encontrar en la carpeta CORES. Después se hace *click* sobre ***Open***. A continuación se copia y pega las mencionadas platillas: la platilla de

declaración del componente se copia en la parte declarativa de la arquitectura y la plantilla de instanciación en el cuerpo de la arquitectura del modelo top_system_v4.vhd.

```
42  ------------ Begin Cut here for COMPONENT Declaration ------ COMP_TAG
43  COMPONENT prescaler_v4
44    PORT (
45      clk : IN STD_LOGIC;
46      sclr : IN STD_LOGIC;
47      thresh0 : OUT STD_LOGIC;
48      q : OUT STD_LOGIC_VECTOR(26 DOWNTO 0)
49    );
50  END COMPONENT;
51  -- COMP_TAG_END ------ End COMPONENT Declaration ------------
52
53  -- The following code must appear in the VHDL architecture
54  -- body. Substitute your own instance name and net names.
55
56  ------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
57  your_instance_name : prescaler_v4
58    PORT MAP (
59      clk => clk,
60      sclr => sclr,
61      thresh0 => thresh0,
62      q => q
63    );
```

Figura 45.- Plantillas de declaración e instanciación del componente *prescaler*.

5.- Flujo de Diseño: Simulación.

Una vez se ha especificado el modelo con su correspondiente banco de pruebas el siguiente paso en el flujo de diseño es la simulación (Figura 46). Para ello se van a utilizar dos ejemplos sencillos: el decodificador como modelo combinacional y el *prescaler* como modelo secuencial. Aunque se trata de ejemplos sencillos el proceso seguido es extrapolable a cualquier circuito. El simulador que se va a utilizar es el *ISE Simulator (ISim)* que viene instalado por defecto con *ISE*.

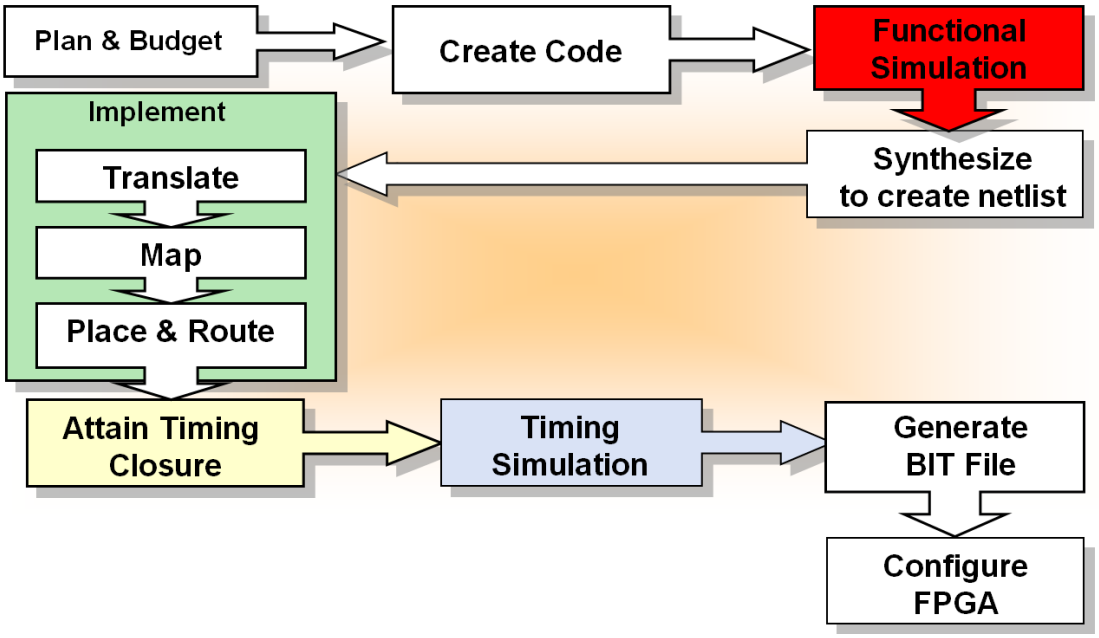
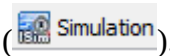


Figura 46.- Flujo de diseño: simulación.

5.1.- Lanzar el simulador ISim. Ventana principal.

Para iniciar este proceso de simulación se debe seleccionar la pestaña *Design* del Panel de Pestañas y en las opciones **View** se selecciona *Simulation* (Figura 47). Al lado de este indicador, se muestra el icono del simulador que se va a utilizar, en este caso *ISim* (

). Además, con el selector de la izquierda, se debe seleccionar el tipo de simulación a realizar, en este caso *Behavioral* (funcional o de comportamiento).

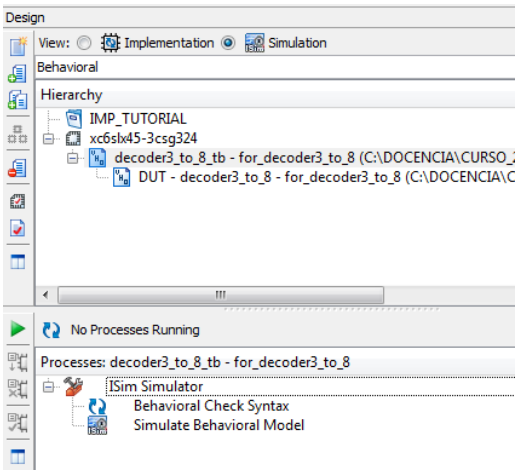


Figura 47.- Selección de simulación desde el panel *Hierarchy*.

En el caso de que apareciese otro simulador se debe hacer doble clic sobre el código del dispositivo para el que se realiza el diseño situado en la ventana *Hierarchy*, mostrándose la ventana de la Figura 48. En el apartado *Simulator* se selecciona *ISim (VHDL/Verilog)* y se valida con **Ok**.

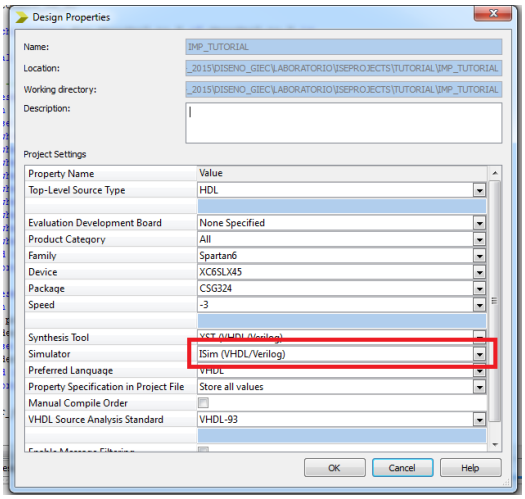


Figura 48.- Selección del simulador ISIM.

Es importante comprobar en las opciones del diseño (Figura 48) que se encuentre seleccionado el valor *VHDL* en el campo *Preferred Language*, de esta forma se puede utilizar, indistintamente, mayúsculas o minúsculas para referirse a una traza. En el caso de que se seleccione *Verilog*, para referirse a una señal se deberán utilizar los mismos caracteres y el mismo tipo (mayúsculas o minúsculas) que el utilizado en el esquema.

Para lanzar la simulación se selecciona el modelo del banco de pruebas *decoder3_to_8* en la ventana *Design* y se hace doble clic en *Simulate Behavioral Model* . En la ventana de mensajes se va mostrando la información generada por las herramientas utilizadas para realizar la simulación. Si no se han producido errores se mostrará la pantalla principal (*ISim Graphical User Interface*, GUI) del simulador ISim (Figura 49).

La ventana principal de ISim muestra la Barra de Herramientas, la Ventana de formas de Onda, la Consola, la Barra de estado y las ventanas de Fuentes y Señales.

Antes de pasar a analizar las diferentes ventanas de la GUI de ISim es conveniente definir dos elementos básicos de toda simulación:

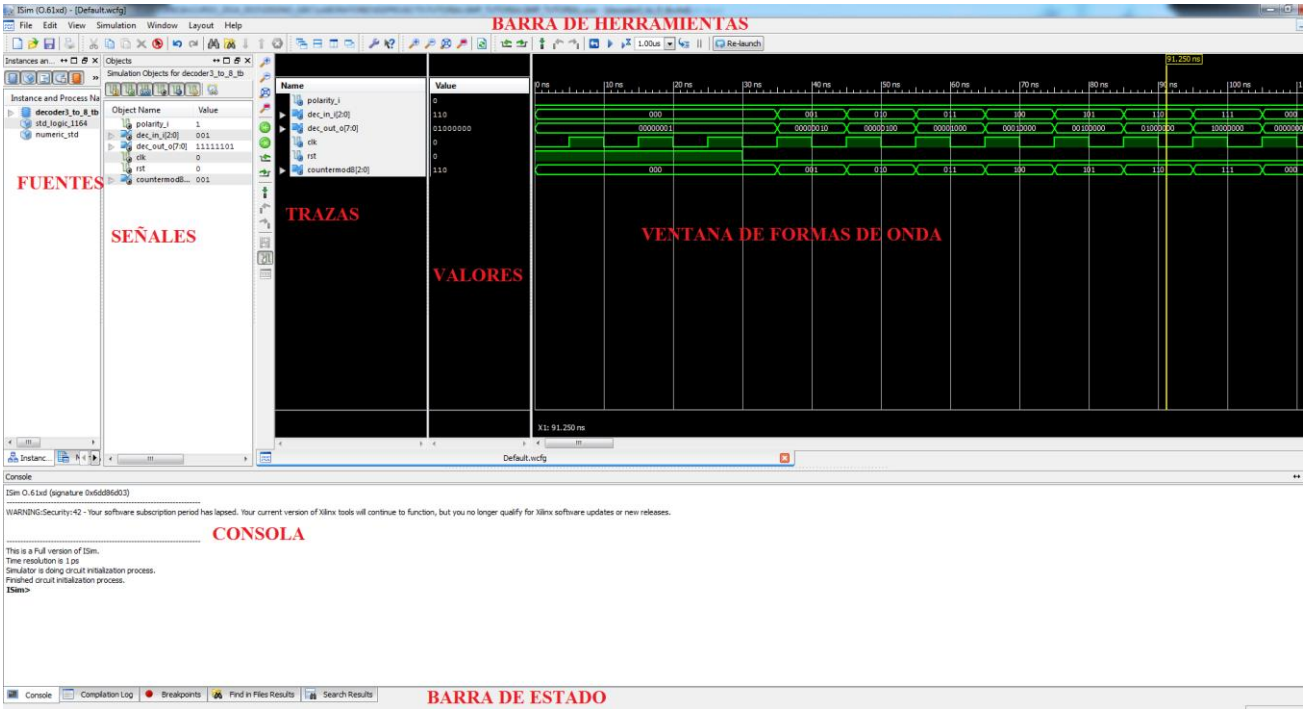


Figura 49.- Pantalla principal del ISim.

- ***Traza***. Son todos aquellos elementos de un diseño de los que se quiere ver su evolución en la simulación.
- ***Estímulo.*** Son los valores que se asignan a los terminales de entradas de un modelo a simular para analizar la respuesta del mismo.
- ***Señales.*** Se trata de los puertos que definen la interface del modelo y de las señales que en el código se especifican en la parte declarativa de la arquitectura.

La Ventana de Formas de Onda es el lugar donde se muestran las señales (trazas) y sus valores. La configuración de esta ventana se puede guardar en un archivo, de extensión *wcfg* (*Wave Configuration File*), para poder ser utilizado en un futuro. Este archivo contiene las señales (trazas) que se visualizan y su configuración, pero no las formas de onda. Por defecto el nombre de archivo de configuración es *Default.wcfg*, pudiéndose cambiar ejecutando los siguientes pasos:

- Desde la barra de herramientas con el menú ***File*** y a continuación ***Save As***.
- Desde la consola ejecutando el comandos *wcfg save <nombre_archivo.wcfg>*.

En algunas ocasiones interesa abrir un archivo de configuración:

- Desde la barra de herramientas con el menú ***File*** y a continuación ***New*** *u* ***Open***, dependiendo de si se quiere crear uno nuevo o cargar uno ya existente.
- Desde la consola ejecutando los comandos *wcfg new* o *wcfg open.*

La Barra de Herramientas (Figura 50) disponible en la ventana principal ISim ofrece un acceso rápido los comandos de uso frecuente:

- Menú de comandos ***File*** y ***Edit***
- Menú de comandos ***Window*** y ***View***
- Menú de comandos de simulación

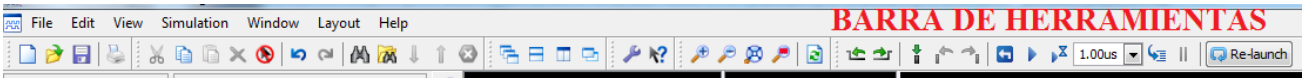


Figura 50.- Barra de herramientas del ISim.

Las ventanas a visualizar en la pantalla principal de ISim se pueden seleccionar utilizando el menú **View** de la barra de herramientas y a continuación ***Panels*** (Figura 51).

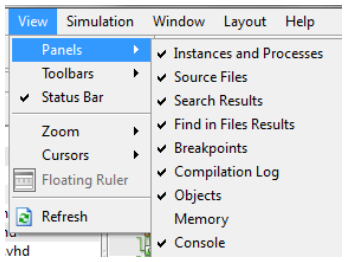


Figura 51.- Selección de las ventanas a visualizar.

- *Ventana Objects*. Muestra todas las señales que se encuentran en un diseño. Esta ventana es de mucha utilidad porque, desde ella se pueden añadir las señales que se quieren visualizar (trazas) en la Ventana de Formas de Onda.

- *Ventana Instances and Processes*. Muestra los procesos que modelan el diseño (Figura 52). Seleccionado uno, en la ventana *Objects* se muestran las señales que hay en él. Con esta ventana se puede navegar por los elementos de un diseño jerárquico para seleccionar como trazas señales de los procesos que se encuentran en diferentes niveles de la jerarquía. Los procesos de un modelo se identifican por su referencia.

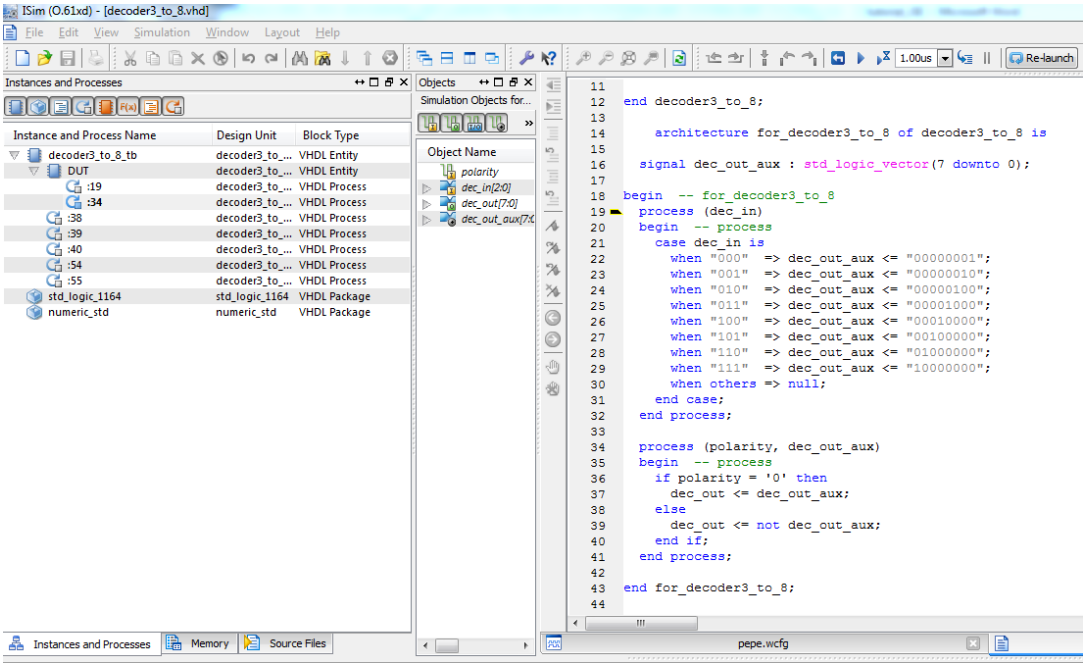


Figura 52.- Ventana de *Instances and Processes.*

- *Ventana Source Files*. Muestra el listado de todos los modelos en VHDL de un diseño (Figura 53).

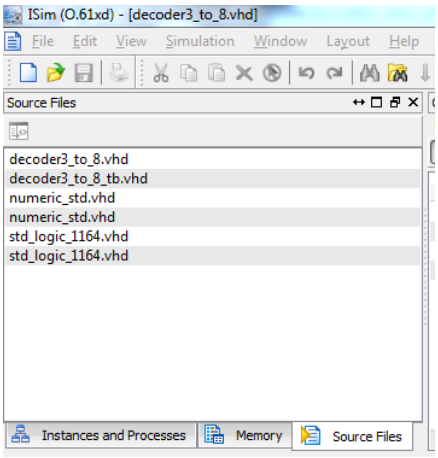


Figura 53.- Ventana de ficheros fuente.

- *Ventana Objects*. Muestra todas las señales (Figura 54) del modelo VHDL seleccionado en la ventana *Instances and Processes.* Para cada señal se muestra su nombre, su tipo y su valor en el instante actual de la simulación. Dependiendo del tipo de señal el icono mostrado al lado de su nombre es diferente:

  Señal de entrada (In) individual.

  Señal de entrada (In) tipo bus.

  Señal de salida (Out) individual.

  Señal de salida (Out) tipo bus.

  Señal bidireccional (in/Out) individual.

Señal bidireccional (in/Out) tipo Bus.

Señal interna (Out) individual.

Señal interna (Out) tipo bus.

El tipo de una señal toma el valor *Array* cuando se trata de un vector, en caso contrario su tipo es *Logical*.

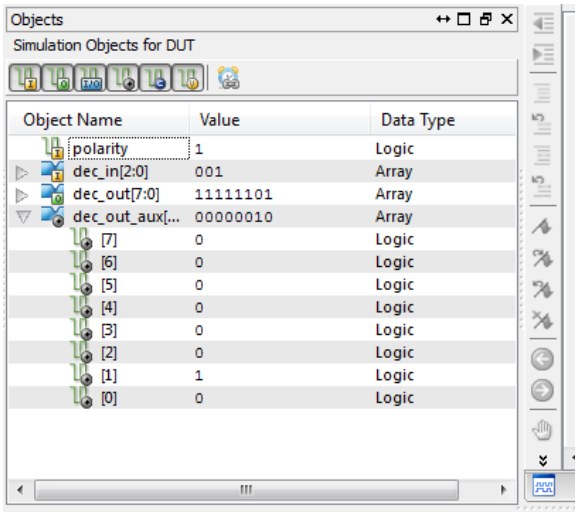


Figura 54.- Ventana de objetos para el decodificador.

La barra  presente en la ventana **Objects** permite seleccionar el tipo de señales que se quieren visualizar en esta ventana. Cada vez que se actúa sobre uno de ellos cambia la visibilidad de las señales de ese tipo.

- La *Ventana Console* (Figura 55) permite ver información generada por ISim, a la vez que se utiliza para introducir comandos que permiten controlar la simulación. Por esto, a esta ventana también se le denomina ventana de comandos.

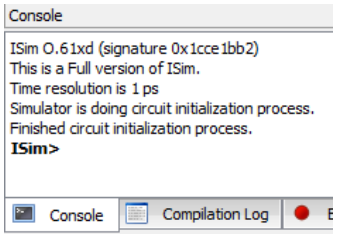


Figura 55.- Consola o ventana de comandos.

5.2- Organización de las trazas.

Por defecto, cuando se lanza ISim desde ISE en la ventana de formas de onda se visualizan todas las señales y/o puertos del diseño. En la mayoría de los casos, algunas de ellas son innecesarias y

dificultan la visualización. Para borrar una señal de la ventana de formas de onda se selecciona y se pulsa *Del*.

En otras ocasiones es necesario añadir trazas a la ventana de formas de ondas, sobre todo cuando se trata de un diseño jerárquico en el que se desea visualizar señales presentes en esquemas que se encuentran en diferentes niveles de la jerarquía. Para añadir una o varias señales a la ventana de formas de ondas se seleccionan en la ventana *Objects* y se hace clic sobre el ratón derecho del ratón, apareciendo la ventana de la Figura 56, en la que se selecciona *Add To Wave Window*.

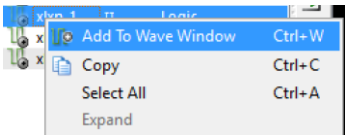


Figura 56.- Añadir trazas.

Las señales a visualizar y su formato se puede seleccionar con el comando *wave add*, si bien es más sencillo añadirlas desde la ventana *Objects*.

Antes de asignar valores a los puertos de entrada del modelo a simular, en primer lugar se debe seleccionar el aspecto (formato) con el que se quieren visualizar cada una de las señales (trazas). El orden que ocupa cada elemento de la ventana *Formas de Onda* se puede variar simplemente arrastrándola a su posición final. Además, las características de cada traza (color, valor, formato, etc.) se pueden modificar seleccionándola y haciendo *clic* con el botón derecho del ratón, apareciendo la ventana de la Figura 57.

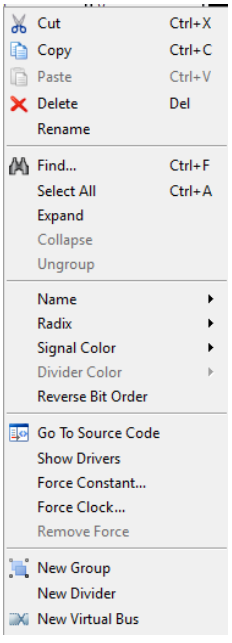


Figura 57.- Opciones de una traza.

Las opciones de la Figura 57 son:

- *Delete*. Borra la traza.
- *Rename*. Permite cambiar el nombre de la traza. No es recomendable.
- *Expand*. Muestra todos los elementos de un bus. Haciendo *clic* sobre el icono ▶ que se encuentra al lado del bus se realiza la misma función.
- *Name*. Especifica el formato con el que se muestra el nombre:
    - Short. Se visualiza sólo el nombre de la traza
    - Long. Se visualiza el nombre y su posición en el árbol de la jerarquía.
- *Radix*. Permite especificar el formato numérico (Figura 58) con el que se muestran los valores que toma la señal. Sólo está disponible para los vectores. Cabe hacer una aclaración con los formatos *Unsigned Decimal* y *Signed Decinal*, los cuales se utilizan para ver los datos binarios en formato decimal sin signo (binario natural) o con signo (complemento a 2), respectivamente.

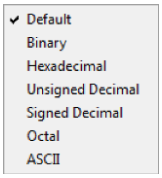


Figura 58.- Base de numeración de las trazas.

- *Signal Color*. Permite cambiar el color con el que se visualiza la señal en la ventana de formas de onda.

- *Reverse Bir Order*. Permite visualizar el vector con los pesos de los bits en orden inverso.

Una vez restructurada la ventana *Formas de Onda* se debe guardar su configuración, tal y como se mencionó anteriormente. Es recomendable que el archivo tenga el mismo nombre que el esquema (*full_add*) y extensión *wcfg*. De esta forma se tienen una trazabilidad por el nombre para saber cuál es la función de un determinado archivo.

5.3.- Inicio de la simulación.

Para realizar la simulación se debe tener en cuenta que en VHDL se especifica el modelo con los estímulos en forma de banco de pruebas. Por ello cuando se lanza el simulador, tanto el valor de los puertos de entrada como el tiempo de simulación están definidos en el banco de pruebas.
Sin embargo, cuando se lanza el simulador desde ISE sin haber especificado un banco de pruebas, por defecto el intervalo de tiempo de simulación es de 1 µs. Además, como a las entradas no se le ha asignado ningún valor, las salidas tampoco tendrán un valor fijo, de ahí que aparezcan todas las trazas con el valor U (*Undefined*).(Figura 59)

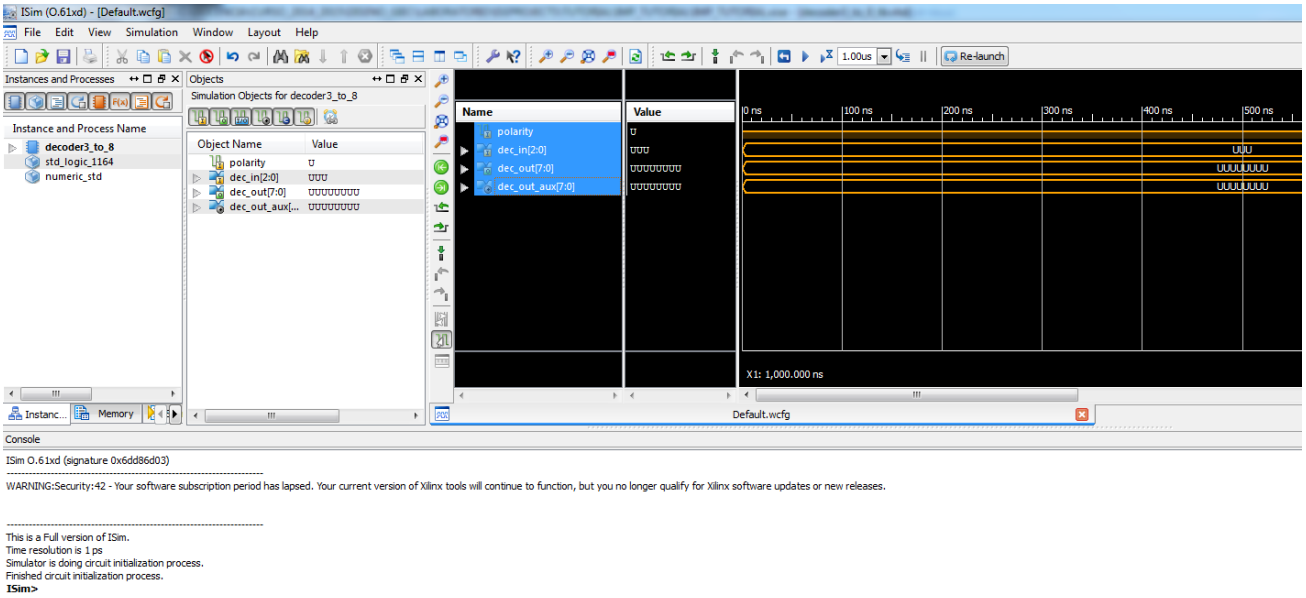


Figura 59.- Ventana de formas de onda donde las trazas tienen un valor no definido.

Para reinicializar la simulación y pasar el tiempo a 0 s se:

- Actúa sobre  en la barra de herramientas.
- Ejecutar ***Restart*** del menú ***Simulation*** de la barra de herramientas.
  Teclear *restart* en la Consola.

La reinicialización de la simulación anula todos los valores que se le hayan asignados a las entradas.

Llegado a este punto, la pantalla formas de onda de la Figura 59 tiene el aspecto mostrado en la Figura 60.

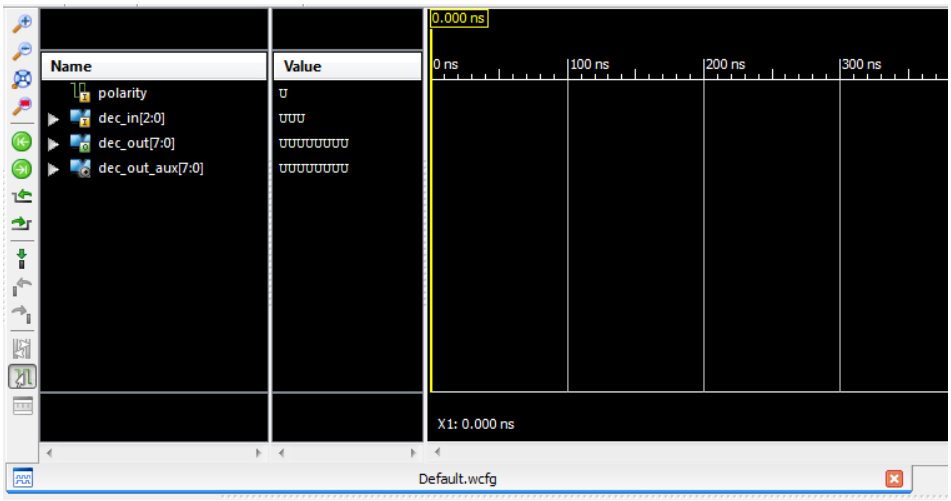


Figura 60.- Inicialización de la simulación.

5.4.- Creación de estímulos.

Para poder realizar la simulación de un esquema es preciso crear los estímulos (valores) que se van a aplicar en las entradas. Los estímulos son valores que se distribuyen en el tiempo, por lo que es preciso tener claro que valores van a tomar las entradas y en que instantes de tiempo. Puede ser interesante realizar sobre un papel una distribución de los estímulos que van a tomar las entradas. Queda por fijar cuanto tiempo se espera entre valor y valor; para ello no conviene trabajar con tiempos muy grades que pueden dar lugar a tiempos de simulación muy largos: una simulación de 1 s tarda en realizarse más de 1 s, dando lugar a excesivos tiempos de espera ante la pantalla del ordenador, lo que hace tediosa la labor de simulación. En general, tiempos del orden de las decenas o centenas de ns (nanosegundos) es lo recomendable. Para el ejemplo del tutorial se pueden definir los estímulos representados en la Figura 61. La señal ***polarity*** cambia de valor cada 100 ns comenzando con el valor inicial cero. Por otro lado, la señal ***dec_in*** cambia de valor cada 10 ns en forma de contador ascendente partiendo de cero.

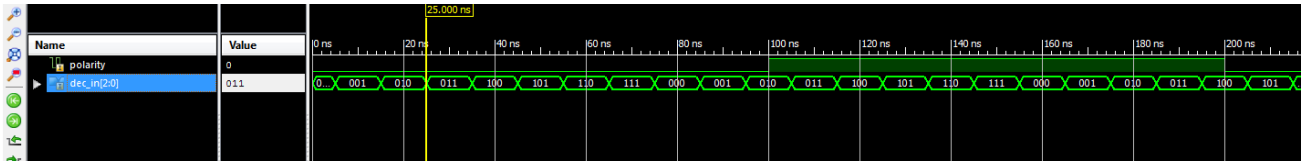


Figura 61.- Estímulos para las entradas del decodificador.

Para asignar un valor a una entrada, se selecciona en la ventana de formas de onda, o en la ventana *Objects*; por ejemplo *polarity*. A continuación se cliquea sobre el botón derecho y se selecciona *Force Constant*, (Figura 62) apareciendo la ventana de la Figura 63. Sus opciones son:

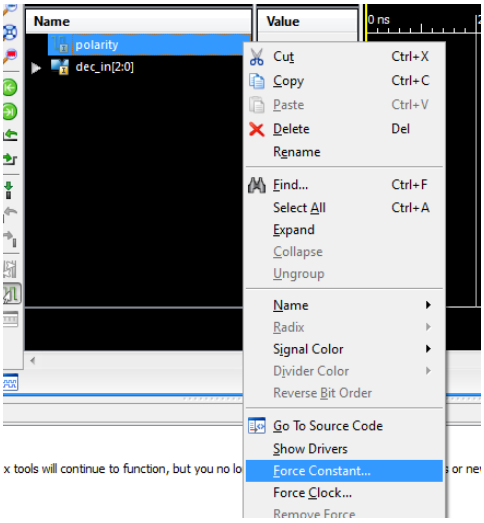


Figura 62.- Definir un valor constante sobre una señal.

- *Signal Name*. Representa el valor de la señal. No se debe modificar.
- *Value Radix*. Representa el formato con el que se va a introducir el valor.
- *Force to Value*. Representa el valor a asignar a la señal con el formato antes indicado. Para el caso de las señales cuyo rango sea un bit (señales que no son vectores), el valor seleccionado en *Value Radix* es irrelevante, al ser el 0 y el 1 el mismo valor en todos ellos. En el caso de

los valores hexadecimal correspondientes a letras (a-f) se pueden utilizar indistintamente mayúsculas o minúsculas.

- *Starting at Time Offset*. Instante de tiempo, contado a partir del tiempo actual de simulación en el que se aplica el valor indicado. Un valor se representa con una cantidad numérica, expresada en decimal (entero con decimales) seguida de una unidad: ps (pico segundos), ns (nano segundos), us (micro segundos), ms (mili segundos) o s (segundos). Por defecto las unidades son ps. Entre el instante actual de la simulación y el especificado a la señal se le asigna el valor U (*Undefined*). Normalmente en este campo se pone 0 (0 ps).
- *Cancel after Time Offset.* Representa el tiempo a partir del cual se cancela el valor asignado, a partir de ese instante, a la señal, se le asigna el valor U.

Al pulsar **Ok** la ventana de la Figura 63, se asigna el valor a la entrada. A su vez en la ventana Consola se muestra el comando ejecutado con la anterior operación.

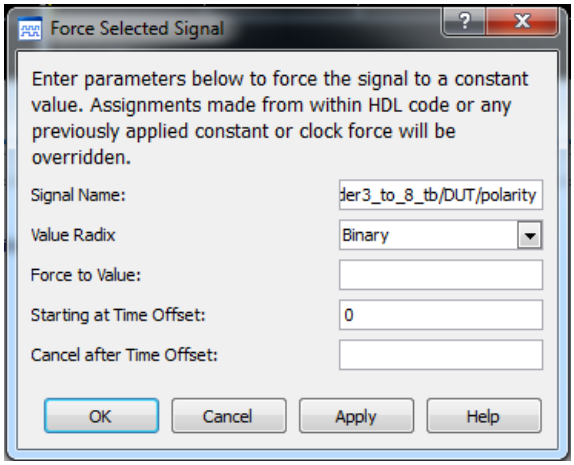


Figura 63.- Ventana para asignar valores a las señales.

Otra alternativa más práctica para introducir el valor que debe tomar una señal consiste en ejecutar el comando *isim force add* desde la ventana Consola. El formato de este comando es:

isim force add <nombre_señal> <valor> [-radix] [-time <tiempo>][-cancel <tiempo>]

Cada uno de los campos se corresponde con los elementos de la ventana de la Figura 63. Los campos marcados con [] son opcionales. Para asignar el valor 0 a la entrada *polarity* se utilizará el comando:

isim force add polarity 0 –radix binary

Hay que indicar que con las teclas de cursor arriba (↑) y abajo (↓) se puede acceder a los comandos anteriores o posteriores introducidos para poder ejecutarlos de nuevo o modificarlos.

Para cada conjunto de valores asignados a las entradas se debe provocar un avance de la simulación. Para ellos se puede:

- Haciendo *clic* sobre el icono de la Barra de Herramientas. La simulación se realiza durante el tiempo especificado en el recuadro 100 ns de la Barra de herramientas.

- Desde el menú ***Simulation*** de la barra de herramientas y a continuación ***run***.
- Desde la Ventana Consola introduciendo el comando *run*. Posibles construcciones de este comando son:

Comando	Descripción
run 100 ns	Avanza la simulación 100 ns
run 100	Avanza la simulación de 100 ps
run	Avanza la simulación un tiempo igual al seleccionado en la Barra de Herramientas

Esta forma de introducir los estímulos es muy engorrosa y obliga a introducir los estímulos cada vez que se lleva a cabo la reinicialización del proceso.

5.5.- Creación de un archivo de comandos *tcl*.

Una vez finalizada una simulación hay que analizar todas las señales para verificar que cumplen con las especificaciones del diseño. Si esto no sucediese, habría que comenzar de nuevo cerrando el ISim, modificando el modelo a simular y finalmente habría que realizar de nuevo todo el proceso de simulación (configurar trazas, asignar estímulos y ejecutar la simulación). En consecuencia, la etapa de simulación puede ser laboriosa. Para simplificar el proceso se puede utilizar un archivo que contiene la secuencia de comandos a ejecutar desde la ventana Consola, de forma que ejecutando este archivo se ejecutan de forma secuencial uno a uno los comandos introducidos en él. Este es el archivo de comandos, al que también se le suele denominar *script*. Este archivo se crea una sola vez y se ejecuta tantas como veces se quiera realizar la simulación del modelo. Se recomienda que el nombre del archivo sea el del modelo y de extensión *tcl*.

En principio, cabe pensar que un archivo de comandos sólo tiene comandos *isim force* y *run*. De ser así, habría que configurar las señales a visualizar cada vez que se lanzara el simulador. Una solución más práctica consiste en que la primera vez que se lanza la simulación se configuren las señales a visualizar y se cree un archivo de configuración (*wcfg*), el cual se añade al archivo de comandos con el comando *wcfg open*.

Con todo ello, el contenido del archivo para simular el decodificador tiene el aspecto mostrado en el listado 3. Nótese que todo el texto que va precedido del carácter # se considera como un comentario y por lo tanto no se considerara un comando de simulación. Los comentarios en el listado que se muestra a continuación tienen como misión explicar la funcionalidad de cada uno de los comandos.

```
# cargar la configuración de la ventana de formas de onda
wcfg open decoder3_to_8_wave.wcfg
# reiniciar la simulación
restart
#generar estímulos
isim force add polarity 0 -radix bin
isim force add dec_in 000 -radix bin
run 5 ns
isim force add dec_in 001 -radix bin
```

```
run 10 ns
isim force add dec_in 010 -radix bin
run 10 ns
isim force add dec_in 011 -radix bin
run 10 ns
isim force add dec_in 100 -radix bin
run 10 ns
isim force add dec_in 101 -radix bin
run 10 ns
isim force add dec_in 110 -radix bin
run 10 ns
isim force add dec_in 111 -radix bin
run 5 ns
isim force add polarity 1 -radix bin
run 5 ns
isim force add dec_in 000 -radix bin
run 10 ns
isim force add dec_in 001 -radix bin
run 10 ns
isim force add dec_in 010 -radix bin
run 10 ns
isim force add dec_in 011 -radix bin
run 10 ns
```

Listado 3.- Contenido del archivo de comandos decoder3_to_8_func.tcl.

El archivo de comandos es un archivo de texto que se puede crea utilizando cualquier editor de textos. ISim tiene un editor que se puede acceder a él desde el menú ***File*** de la Barra de Herramientas y a continuación ***new***, o actuando sobre el icono  de la misma barra. En ambos casos se muestra la ventana de la Figura 64, donde se selecciona *Text File*, accediéndose al editor de ISim. El archivo que se abre no tiene nombre por lo que hay que asignarle uno. Esto se puede hacer desde el menú ***File***.

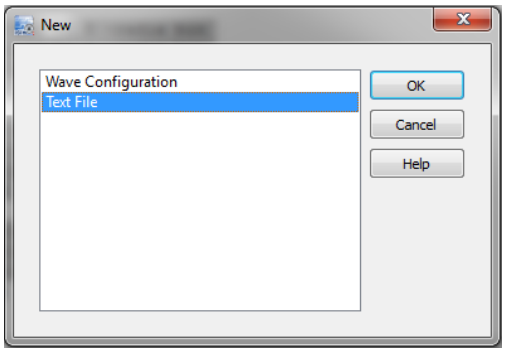


Figura 64.- Creación de un archivo de texto.

Para ejecutar un archivo de comandos se ejecuta en la ventana Consola el comando *source*, cuyo formato es:

source <nombre_archivo.tcl>

Con la ejecución del archivo de comandos del listado 3 (*source decoder3_to_8_func.tcl*) se obtienen el resultado mostrado en la Figura 65.

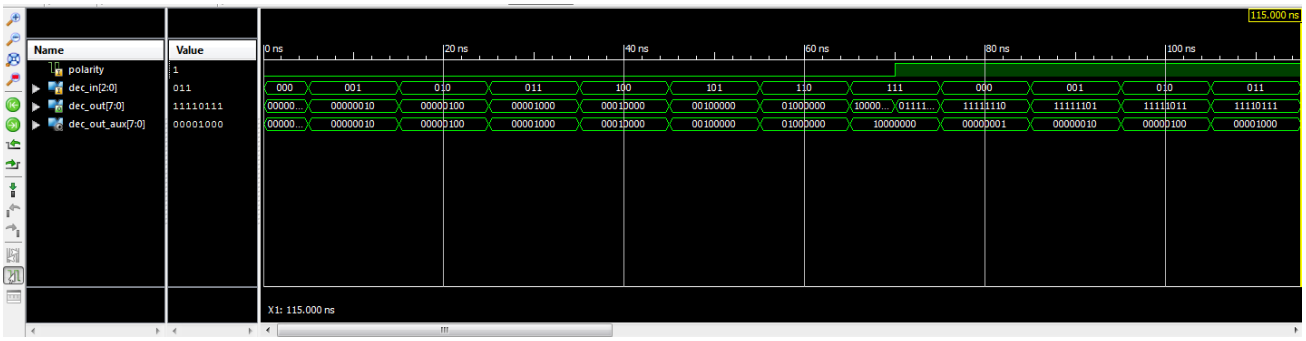


Figura 65.- Formas de onda de la simulación.

Para diseños secuenciales, es decir, aquellos que requieran una señal de reloj, es útil poder definir una señal periódica. Para ello, se puede utilizar el comando *Force Clock* en lugar de *Force Constant*, apareciendo la ventana de la Figura 66.

- *Signal Name*. Representa el valor de la señal. No se debe modificar.
- *Value Radix*. Representa el formato del valor anterior.
- *Leading Edge Value*. Especifica el primer valor de la señal.
- *Trailing Edge Value*. Especifica el otro valor de la señal.
- *Starting at Time Offset*. Instante de tiempo, contado a partir del tiempo actual del simulación en el que se genera la señal periódica.
- *Cancel after Time Offset*. Representa el tiempo a partir del cual se cancela la señal periódica
- *Duty Cycle (%).* Especifica el ciclo de trabajo definido como el cociente entre el tiempo que la señal está a nivel alto y el periodo especificado en tanto por ciento (%).
- *Period*. Especifica el periodo de la señal.

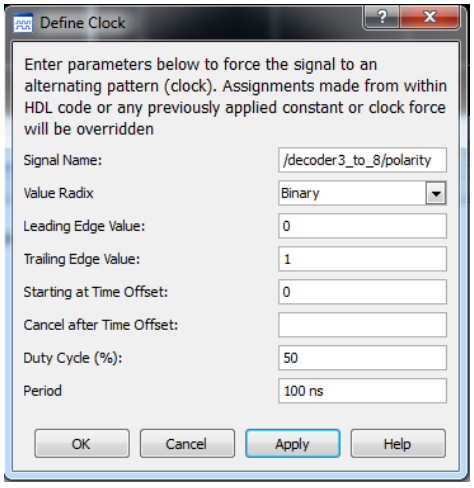


Figura 66.- Especificación de una señal periódica.

Con los datos mostrados en la Figura 66, en la señal *polarity* se aplica una señal periódica de periodo 100 ns y 50% de ciclo de trabajo, está a nivel alto el mismo tiempo (50 ns) que a nivel bajo.

La señal empieza con un nivel bajo. El mismo resultado se puede conseguir utilizando el comando *isim force*:

***isim force** add polarity 0 -radix bin -value 1 -radix bin -time 50 ns -repeat 100 ns*

Las señales periódicas se pueden aplicar a vectores (buses), en este caso el valor del vector conmutará entre dos valores. El primero seleccionado con *Leading Edge Value* y el segundo con *Trailing Edge Value.*

En el siguiente ejemplo se muestra un fichero de comandos (listado 4) donde se incluyen nuevas opciones. Cada comando se comenta utilizando el carácter #.

```
# cargar la configuración de la ventana de formas de onda
wcfg open decoder3_to_8.wcfg
# reiniciar la simulación
restart
#generar estímulos
#se define un estímulo sobre la señal polarity de tal forma que toma el valor 0
#hasta el instante 170 ns y después se pone a 1 hasta el  instante 270 ns momento
#en el que pasa a 0. De esta forma se pueden definir formas de ondas aleatorias
isim force add polarity 0 -radix bin -value 0 -time 0 ns -value 1 -time 170 ns
-value 0 -time 270 ns
#se define un estímulo sobre la señal dec_in para que contemple todos los valores
# posibles. Cada nuevo valor se genera con una diferencia temporal de 20 ns. El
#conjunto de valores asignados se repite cada 100 ns
isim force add dec_in 0 -radix bin -value 000 -time 0 ns -value 001 -time 10 ns
-value 010 -time 20 ns  -value 011 -time 30 ns -value 100 -time 40 ns -value 101
-time 50 ns -repeat 100ns
#se ejecuta la simulación durante 300 ns
run 300 ns
```

Listado 4.- Comandos de simulación.

El resultado de la ejecución de los comandos del listado 4 se muestra en la Figura 67.

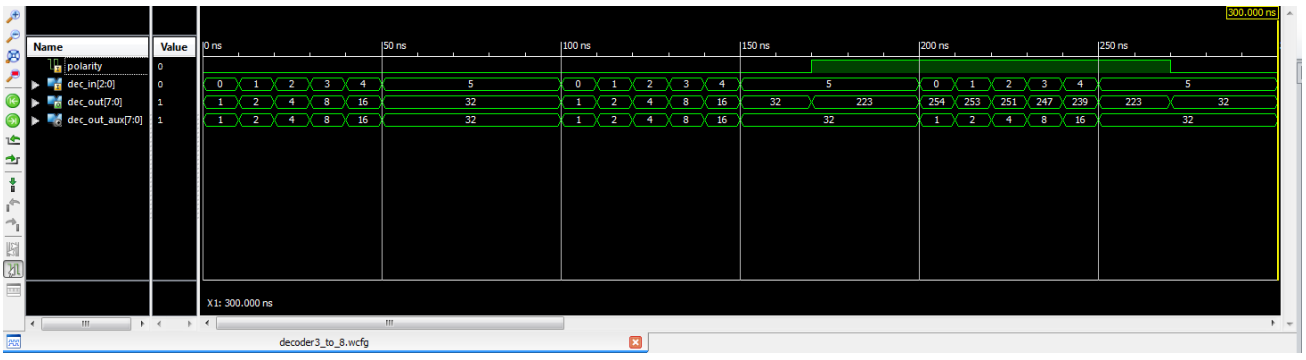


Figura 67.- Resultado de la simulación para los comandos del listado 4.

5.6.-Especificación de los estímulos en un banco de pruebas.

Como se ha analizado en el punto anterior, ISim permite la especificación de estímulos a través de comandos. Por lo tanto, los estímulos que se especifican con ISim, solamente se pueden utilizar para esta herramienta, lo que hace la portabilidad imposible.

Otra forma de definir estímulos es a través de los bancos de pruebas especificados en HDLs. En este caso, el modelo de especificación de estímulos no depende de la herramienta de simulación y por lo tanto, se puede simular con cualquier herramienta.

A continuación en el listado 5 se muestra el código VHDL del banco de pruebas para el decodificador donde se ha especificado un contador de 3 bits para generar los estímulos del puerto de entrada *dec_in*.

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

entity decoder3_to_8_tb is
end decoder3_to_8_tb;

architecture for_decoder3_to_8 of decoder3_to_8_tb is

  component decoder3_to_8
    port (
      polarity : in  std_logic;
      dec_in   : in  std_logic_vector(2 downto 0);
      dec_out  : out std_logic_vector(7 downto 0));
  end component;

  signal polarity_i  : std_logic                    := '0';
  signal dec_in_i    : std_logic_vector(2 downto 0) := "000";
  signal dec_out_o   : std_logic_vector(7 downto 0);
  signal clk, rst    : std_logic                    := '0';
  signal countermod8 : unsigned(2 downto 0);

begin  -- for_decoder3_to_8

  DUT : decoder3_to_8
    port map (
      polarity => polarity_i,
      dec_in   => dec_in_i,
      dec_out  => dec_out_o);

  rst            <= '1', '0' after 4 ns;
  clk            <= not clk  after 5 ns;
  process (clk, rst)
  begin  -- process
    if rst = '1' then
      countermod8 <= (others => '0');

    elsif clk'event and clk = '1' then
      if countermod8 = 7 then
        countermod8 <= (others => '0');
      else
        countermod8 <= countermod8+1;
```

```
      end if;
    end if;
  end process;

  polarity_i <= not polarity_i after 100 ns;
  dec_in_i   <= std_logic_vector(countermod8);

end for_decoder3_to_8;
```

Listado 5.- Banco de pruebas para el decodificador.

Como se puede observar en el código anterior, el contador dispone de una señal de *reset* activa a nivel alto sobre la que se define un estímulo con valor '1' hasta los 4 ns, después se desactiva.

Por otro lado, la señal de reloj que sincroniza el contador tiene un periodo de 10 ns con un ciclo de trabajo del 50 %. La salida del contador se conecta con la entrada *dec_in* y finalmente, el estímulo sobre la señal *polarity* se especifica como una señal de reloj con un periodo de 200 ns y un ciclo de trabajo del 50 %.

Como se puede observar en el listado 5, el banco de pruebas instancia el modelo bajo test. Por ello, a la hora de simular, se debe lanzar el banco de pruebas ya que éste constituye el nivel más alto de la jerarquía en el ámbito de la simulación.

Antes de lanzar el *ISim*, se pueden configurar el tiempo de simulación, accediendo a la ventana que aparece si se sitúa el ratón en el panel *processes* del ISE y se pulsa el botón derecho del ratón y después *Process Properties* (Figura 68) apareciendo la ventana de la Figura 69.

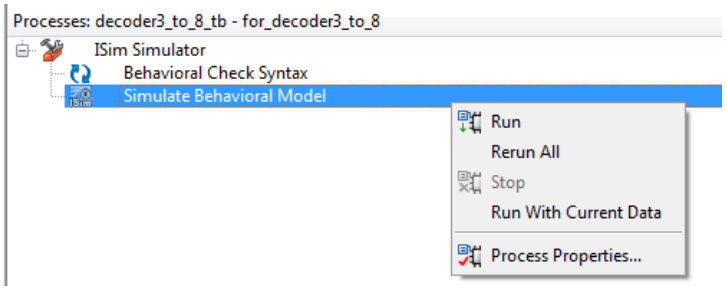


Figura 68.- Definición de las propiedades del simulador.

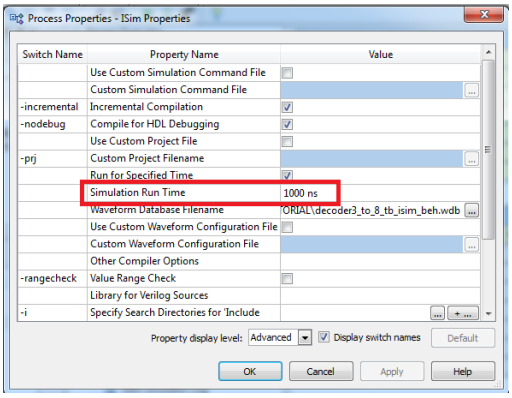


Figura 69.- Ventana de propiedades del ISim.

En la ventana de la Figura 69 se observa que por defecto el tiempo de simulación es de 1000 ns. Este valor se puede modificar en función de los estímulos generados. Por ejemplo, si se han especificado estímulos hasta el instante 400 ns, en principio no tiene sentido ejecutar la simulación durante 1000 ns ya que ralentiza el proceso. Esto último se pone de manifiesto cuando se llevan a cabo simulaciones temporales, donde la cantidad de datos que maneja la herramienta es mayor.

Con el banco de pruebas del listado 5 y para un tiempo de simulación de 250 ns el resultado se muestra en la Figura 70.

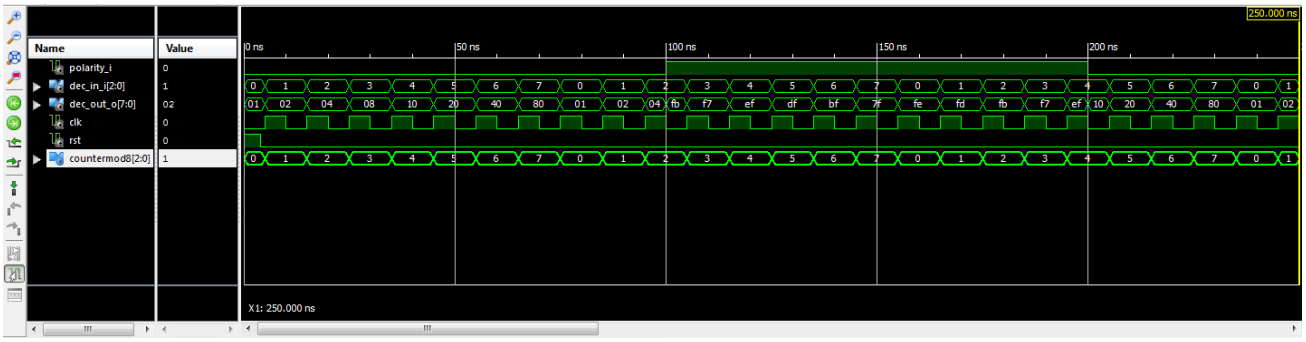


Figura 70.- Resultado de la simulación del decodificador con el banco de pruebas del listado 5.

Nótese que en la figura anterior, los valores del puerto de entrada *dec_in* se representan en decimal siendo binario su tipo por defecto. La base de numeración o *radix* se puede modificar desde la ventana *wave* seleccionando la señal, haciendo click sobre el botón derecho del ratón y seleccionado la opción *radix* y después la base que se quiera utilizar para el puerto o señal seleccionado (Figura 71)

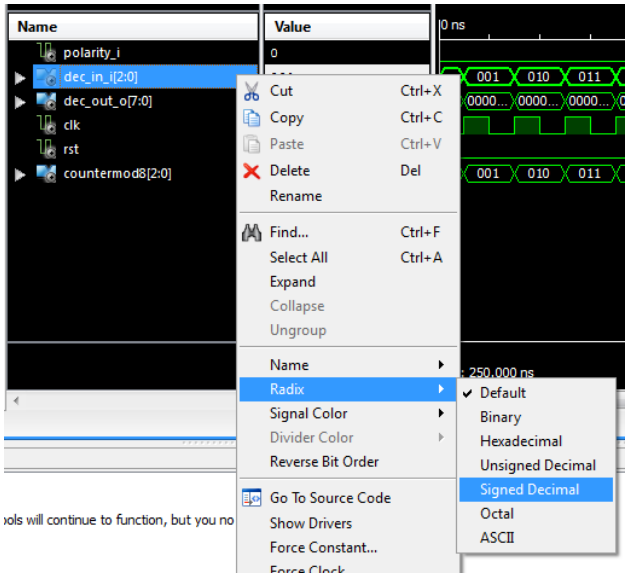


Figura 71.- Selección del *Radix.*

Del mismo modo se puede cambiar el nombre, el color y otros parámetros asociados a la traza que el alumno puede consultar en la documentación disponible para el ISim. Téngase en cuenta que este documento es un tutorial donde se da una introducción a las herramientas ISE y no una guía de usuario detallada.

Aunque la definición de estímulos a través de un banco de pruebas facilita la tarea, es recomendable definir un fichero de comandos con el ISim donde se incluya el formato de la ventana *wave* y la inicialización de la simulación.

5.7.- Análisis de los resultados de la simulación.

Una vez ejecutada la simulación, sobre la ventana de Formas de ondas, se deben analizar los resultados para comprobar si el modelo cumple con las especificaciones de diseño. Para ello ISim dispone de algunos elementos que facilitan esta tarea.

Para ajustar el zoom de la visualización se emplean los elementos que aparecen en la barra de herramientas:

- Zoom Out. Decrementa el tamaño de las trazas. Se visualiza más tiempo de simulación.
- Zoom In. Aumenta el tamaño de las trazas. Se visualiza menos tiempo de simulación.
- Zoom to Full View. Visualiza todo el tiempo de simulación.
- Zoom to Cursor. Muestra el intervalo de tiempo delimitado por dos cursores. Si sólo hay uno, la visualización se centra en la posición que ocupa.

ISim dispone de un cursor (reflejado como una línea vertical amarilla) que se puede mover por todo la ventana de formas de onda para ver el valor que tienen todas las señales en la posición que ocupa. Haciendo click en un punto de dicha ventana se posiciona en él el cursor. Se puede mover de un lado a otro arrastrándolo. También se puede seleccionar una señal determinada pulsando  o  saltando el cursor a la siguiente o anterior transición en la señal, respectivamente. Por transición se entiende un cambio de valor. Actuando sobre  o  se accede al instante inicial o final, respectivamente, de la simulación.

Un elemento muy útil, sobre todo para medir tiempos entre eventos de señales, son los marcadores (*marker*). Un marcador se ubica en la posición que ocupa el cursor cuando se pulsa  . Un marcador se borra seleccionándolo y pulsando *Del*. Las mediciones de tiempo se pueden realizan utilizando un marcador y el cursor y activando la regla flotante (floating ruler)  (Figura 72). La regla flotante se puede activar si esta seleccionado el marcador. Tambien se puede desactivar.

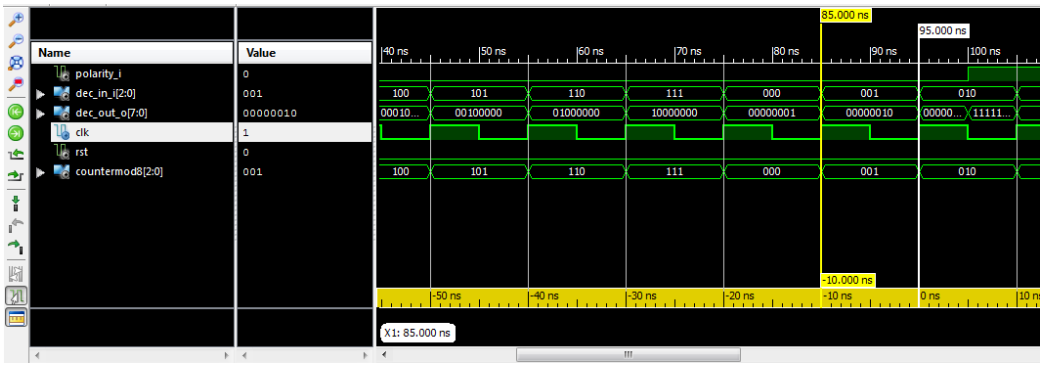


Figura 72.- Medida de tiempos utilizando un marcador.

Si en una simulación se tienen varios marcadores y está activada la regla flotante, el origen (0s) se puede fijar en la posición que ocupa un marcador cliqueando sobre él.

Otra alternativa para medir tiempos consiste en la utilización de un segundo cursor, el cual se ubica haciendo clic con el botón izquierdo a la vez que se mantiene pulsada la tecla *Shift* (↑, mayúsculas). En este caso se muestra la regla flotante y en la posición del segundo cursor se pone el punto 0 de la regla flotante. El segundo cursor se representa como una línea amarilla discontinua. Cliqueando en cualquier zona de la ventana de formas de onda se borra el segundo cursor.

Una vez compobado que la simulación es correcta se cierra debe cerrar ISim y se continua con el diseño.

6.- Flujo de diseño: Síntesis.

Una vez se ha verificado que el diseño está libre de errores, el siguiente paso consiste en la síntesis del mismo (Figura 73).

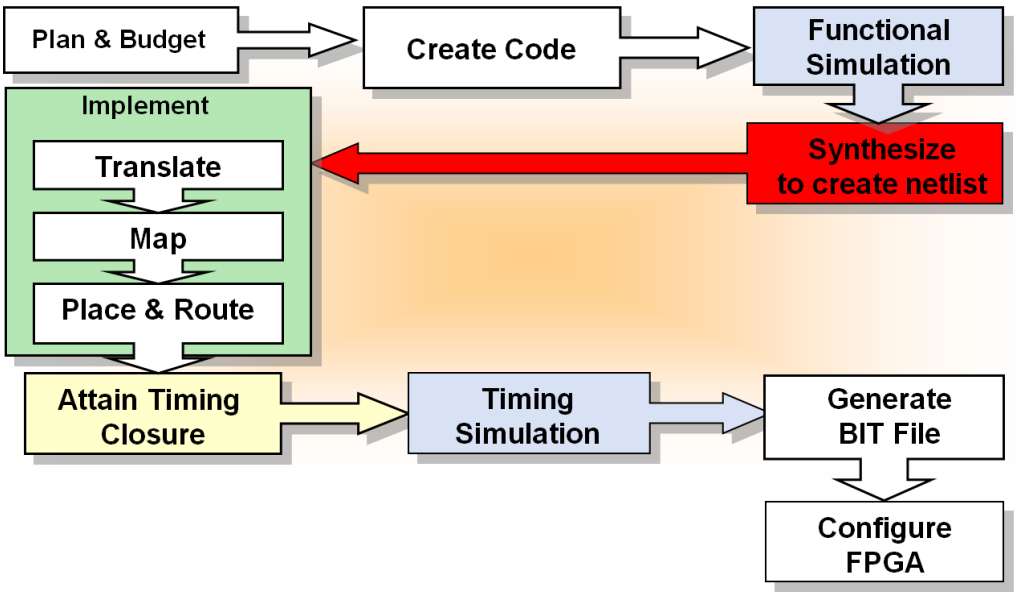


Figura 73.- Flujo de diseño: síntesis.

El diseño se puede sintetizar usando varias herramientas de síntesis, aunque como ya se ha comentado, en el laboratorio se utilizará la herramienta XST (*Xilinx Synthesis Technology*). Para ello en el *ISE Project Navigator* en la ventana *view* se selecciona la option de *Implementation* y después se selección el fichero a sintetizar (Figura 74).

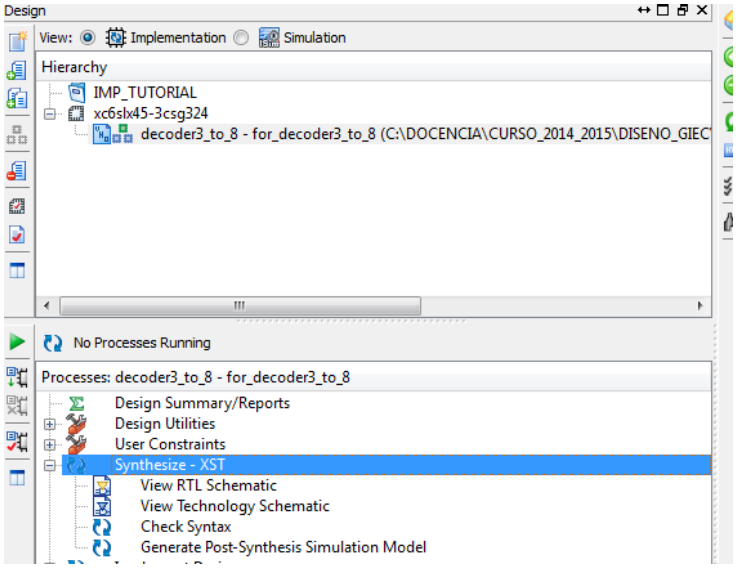


Figura 74.- Configuración del *ISE Project Navigator* para la síntesis.

Como se puede observar en la figura anterior, el banco de pruebas ha desaparecido de la jerarquía ya que se trata de un modelo que sólo se utiliza para la simulación.

Para controlar los resultados del proceso de síntesis, se deben configurar algunas propiedades. Para ello se selecciona en la ventana *Processes* la opción *Synthesize-XST* y se hace *click* en el botón derecho del ratón apareciendo la ventana de la Figura 75.

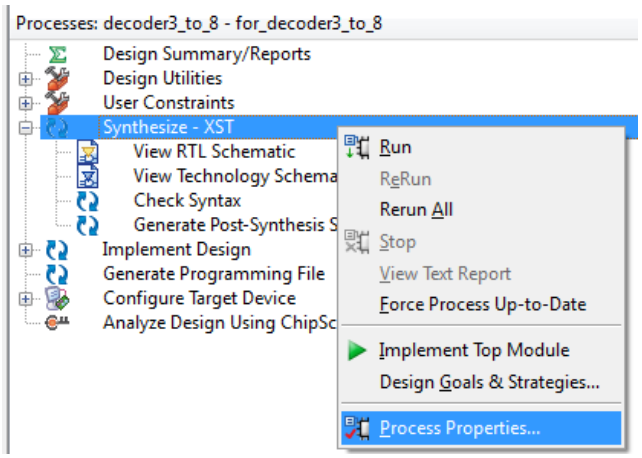


Figura 75.- Lanzando las propiedades del proceso de síntesis.

Pulsando *Process Properties*, se puede configurar las propiedades del proceso de síntesis en la ventana de la Figura 76.

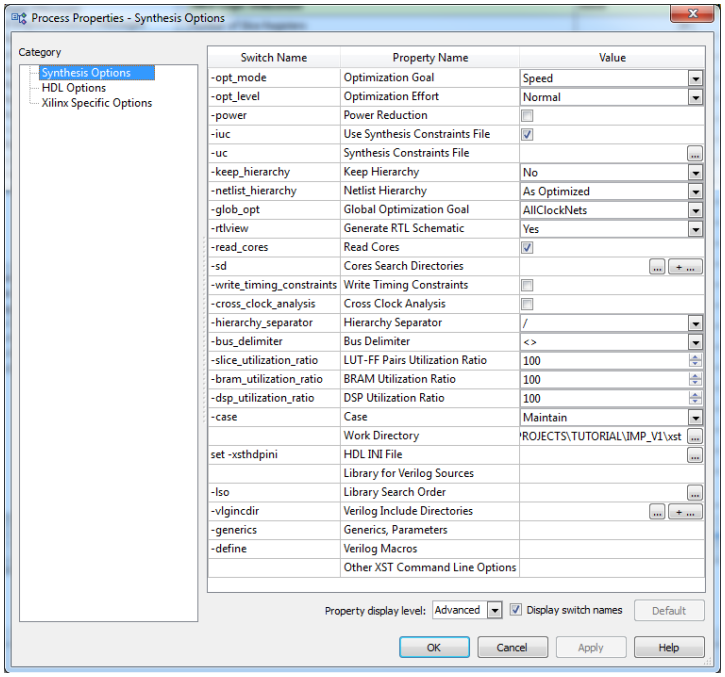


Figura 76.- Propiedades del proceso de síntesis.

Con estas propiedades se puede especificar si la síntesis debe optimizar el área, la velocidad o simplemente el tiempo que la herramienta requiere para sintetizar el modelo. Lo anterior se puede llevar a cabo con la opción *Optimization Goal.*

Para obtener más información de cada una de las propiedades de la ventana anterior se selecciona la opción y se pulsa F1.

En cuando a las opciones HDL (*HDL Options* en la Figura 77), destacar que se puede establecer el tipo de codificación de las máquinas de estado de los circuitos secuenciales.

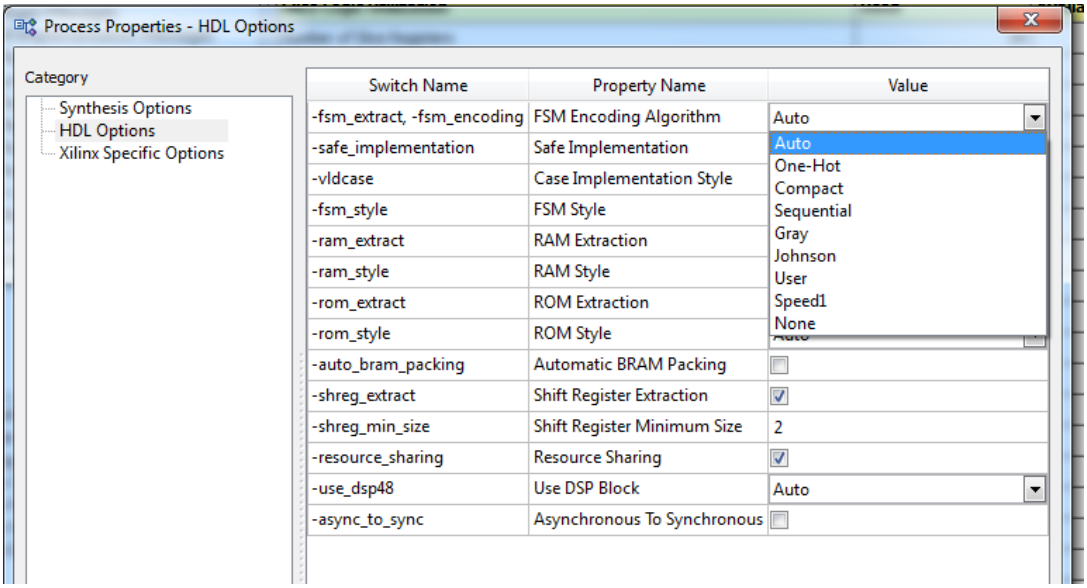


Figura 77.- Definición del tipo de codificación de la máquina de estados.

Información más detallada sobre los tipos de codificación de las máquinas de estado se puede consultar en los apuntes de clase.

7.- Flujo de diseño: Implementación.

Como ya se ha comentado en la introducción de este tutorial, el proceso de implementación consiste en trasladar, mapear, emplazar e interconectar el diseño (Figura 78).

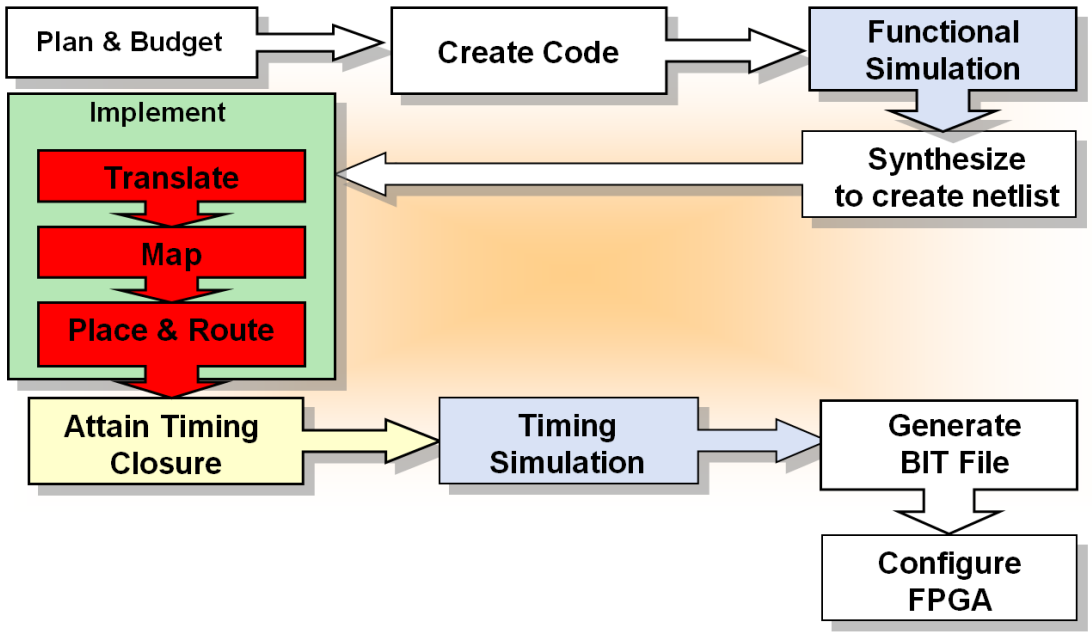


Figura 78.- Flujo de diseño: implementación.

Antes de poder llevar a cabo el proceso de implementación se debe generar un fichero de texto en el que se indica los pines de la FPGA a los que se asignan a las señales de entrada/ salida, asociadas a los puertos de la entidad de nivel superior de la jerarquía. Este archivo se conoce con el nombre de archivo de restricciones (*User Constraints File*). Se recomienda que este archivo tenga como nombre el de la entidad de nivel superior de la jerarquía y de extensión *ucf* y además en minúsculas. Una vez creado, se deberá añadirse al proyecto antes de realizar la implementación del mismo.

Para crear el archivo de restricciones se puede utilizar el editor de texto de *ISim*, tal y como se vio en apartados anteriores. Por cada puerto de la entidad del nivel más alto en la jerarquía se introduce una línea con el siguiente formato:

NET "nombre_del_ puerto " LOC = "numero del pin";

Para ello se deben tener en cuenta las siguientes consideraciones:

- Se pueden utilizar indistintamente mayúsculas y minúsculas
- Cada asignación comienza por la palabra NET (o net)
- El nombre del puerto (entrada/salida) se pone entre “ “
- El número de pin va precedido de LOC= (o loc) y se coloca, también, entre ” ”
- Cada asignación se realiza en líneas independientes terminadas en ;
- Los comentarios van precedidos por #

Así, por ejemplo para asignar la entrada *polarity* al pin N4 de la FPGA se pondría la línea:

```
NET "polarity" LOC = N4; # entrada polarity
```

Para el caso de los vectores (buses), se debe asignar uno a uno los pines a cada uno de sus elementos, poniendo los índices de los elementos del bus entre < >. Así, con las asignaciones:

```
#  Leds
NET "LED<0>" LOC = "U18";
NET "LED<1>" LOC = "M14";
NET "LED<2>" LOC = "N14";
NET "LED<3>" LOC = "L14";
NET "LED<4>" LOC = "M13";
NET "LED<5>" LOC = "D4";
NET "LED<6>" LOC = "P16";
NET "LED<7>" LOC = "N12";
```

El elemento LED<0> del vector LED se asocia al pin U18 de la FPGA, el LED<1> al M14, y así sucesivamente.
El diseño realizado se va a programar en la FPGA de la placa Atlys 2. La correspondencia entre los pines y los elementos de la placa debe se muestra en la Figura 79.

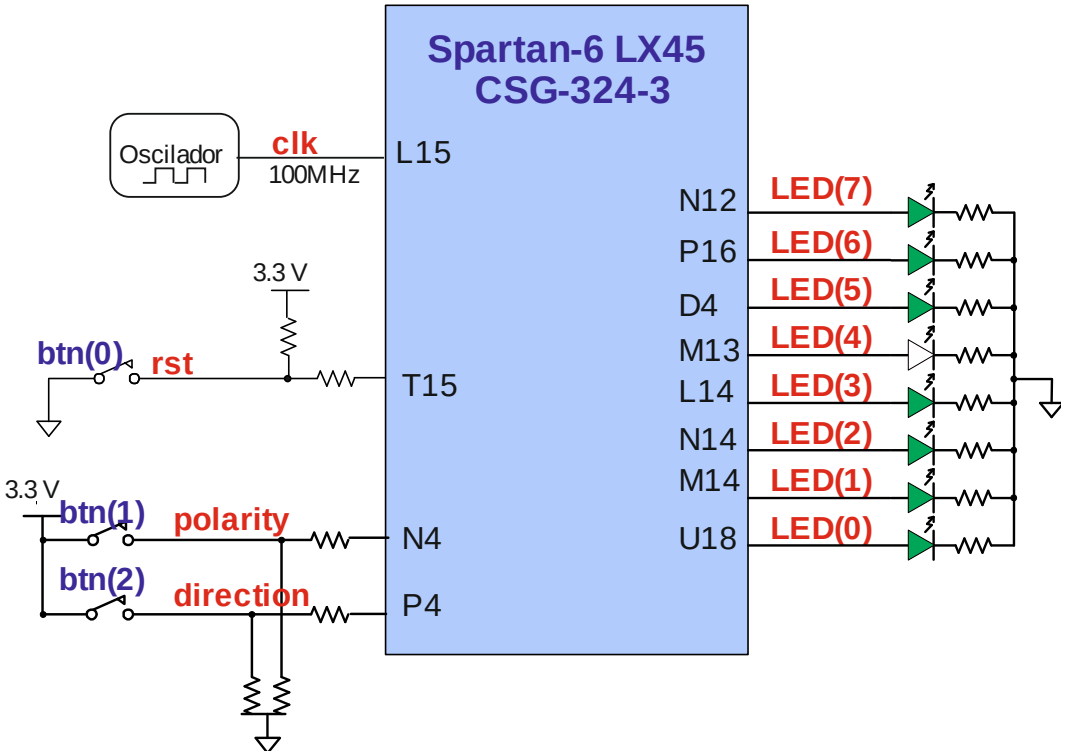


Figura 79.- Correspondencia de pines de la FPGA con los puertos del modelo del desplazador.

De acuerdo a la Figura 79, el archivo de restricciones (*top_system.ucf*) tendrá el contenido mostrado en el listado 6.

```
# clock
NET "clk" LOC = L15;
#  Leds
 NET "LED<0>" LOC = "U18";
 NET "LED<1>" LOC = "M14";
 NET "LED<2>" LOC = "N14";
 NET "LED<3>" LOC = "L14";
 NET "LED<4>" LOC = "M13";
 NET "LED<5>" LOC = "D4";
 NET "LED<6>" LOC = "P16";
 NET "LED<7>" LOC = "N12";
#  PUSH BUTTONS
NET "rst" LOC = T15;
NET "polarity" LOC = N4;
NET "direction" LOC = P4;
```

Listado 6.- Fichero de restricciones de usuario.

Una vez añadido el archivo de restricciones ya se puede implementar el diseño. En primer lugar se debe comprobar que el archivo de mayor nivel jerárquico (top_system) es el que, en la ventana de archivos, tiene la marca . En caso de no ser así, se selecciona el archivo de nivel superior, se actúa sobre el botón derecho del ratón y en la ventana de la Figura 80 se selecciona *Set as Top Module*.

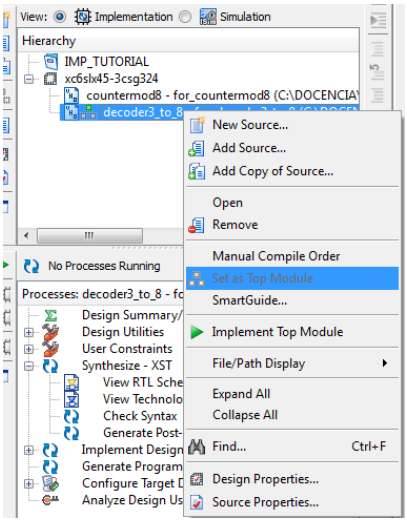


Figura 80.- Asignar el nivel superior de la jerarquía a un archivo.

Antes de lanzar las herramientas que realizarán la implementación se pueden configurar las propiedades de cada uno de los procesos, seleccionándolos individualmente y haciendo *click* con el botón derecho del ratón como ya se explicó para el proceso de síntesis.

Como ejemplo se muestra en la figura 81 la ventana de propiedades del proceso *translate*.

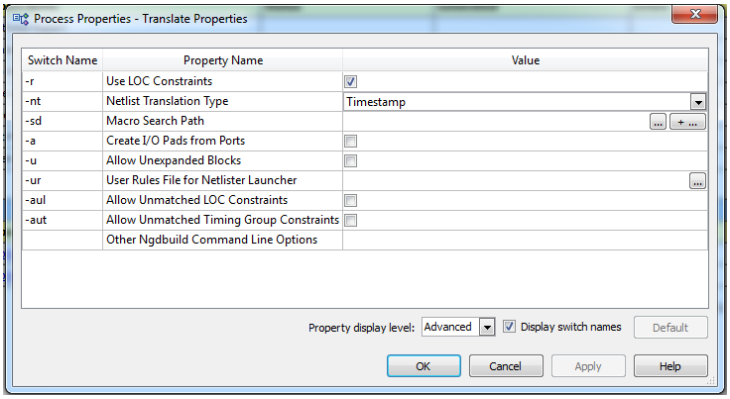


Figura 81.- Propiedades del proceso *Translate.*

Para este tutorial y para el resto de las prácticas de laboratorio se dejarán las opciones por defecto para cada uno de los procesos en la implementación. Está fuera del alcance de este tutorial, la explicación de cada una de las propiedades. En cualquier caso el alumno siempre puede consultar los manuales del fabricante.

La implementación de diseño se puede realizar paso a paso, es decir, lanzando cada uno de los procesos (*Translate*, *Map*, *Place&Route)* que entran en juego en la implementación localizados en la ventana *Processes* (Figura 82).

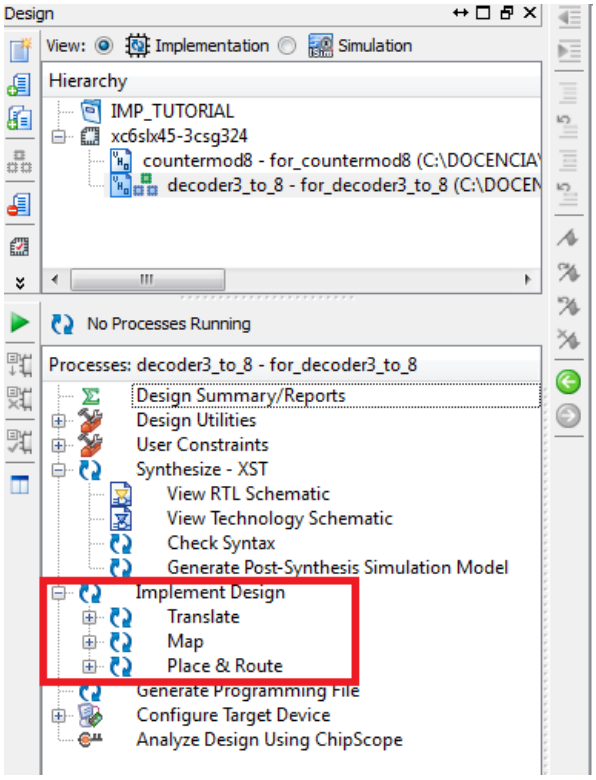


Figura 82.- Pasos en el proceso de implementación.

Existe la posibilidad de ejecutar todos los procesos lanzando el último necesario para la implementación. En este caso sería necesario únicamente lanzar el proceso *Place&Route*. Con ello la herramienta ejecuta automáticamente los procesos anteriores. Asimismo, existe la posibilidad de lanzar todos los procesos incluido el de síntesis, aunque este último ya se haya realizado. Para ello se selecciona el último proceso que se quiere ejecutar, por ejemplo *Place&Route Place&Route* , se hace *click* en el botón derecho del ratón y se selección *Rerun All* (Figura 83).

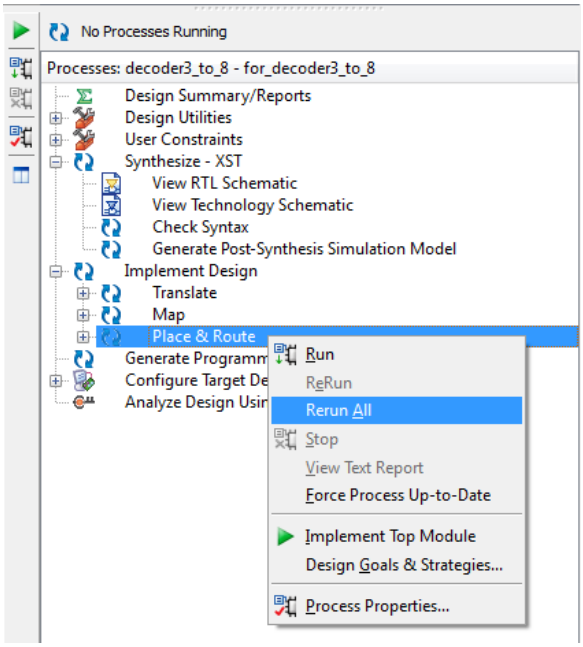


Figura 83.- Ejecutar todos los procesos de nuevo.

8.- Flujo de diseño: simulación temporal.

Después de la implementación ya se dispone de información temporal para poder realizar una simulación más precisa de comportamiento del diseño (Figura 84)

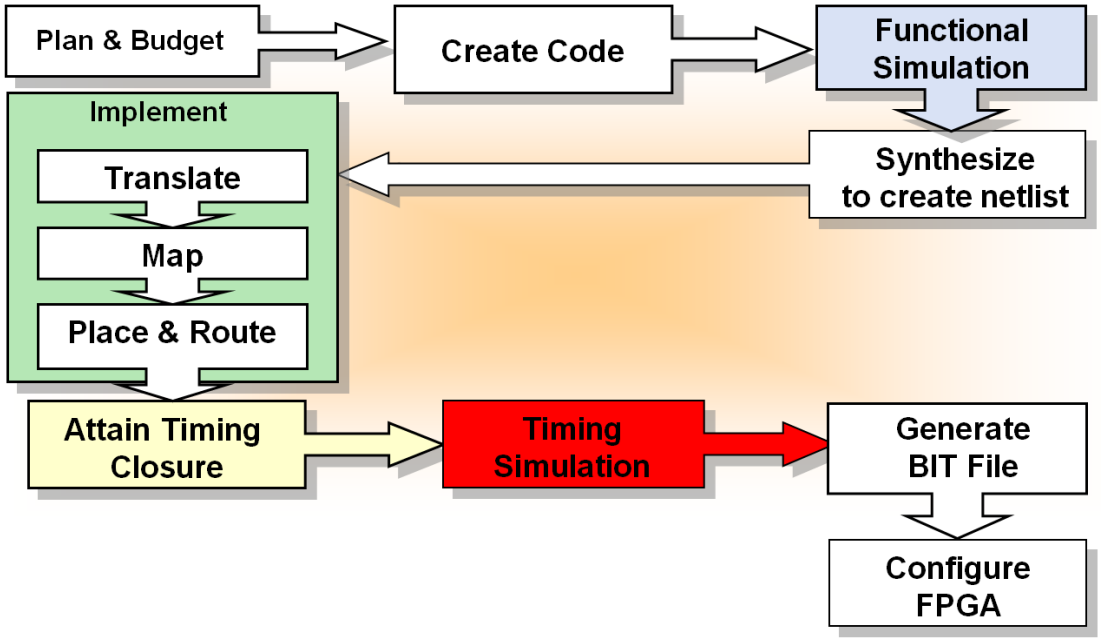


Figura 84.- Flujo de diseño. Simulación temporal.

La simulación temporal usa los datos de retardos extraídos de los modelos de simulación de los componentes y los retardos procedentes del proceso de interconexión. A este respecto es necesario ejecutar un proceso adicional para poder generar un fichero donde se anoten los retardos. Este proceso se encuentra dentro del proceso *Place&Route* (Figura 85).

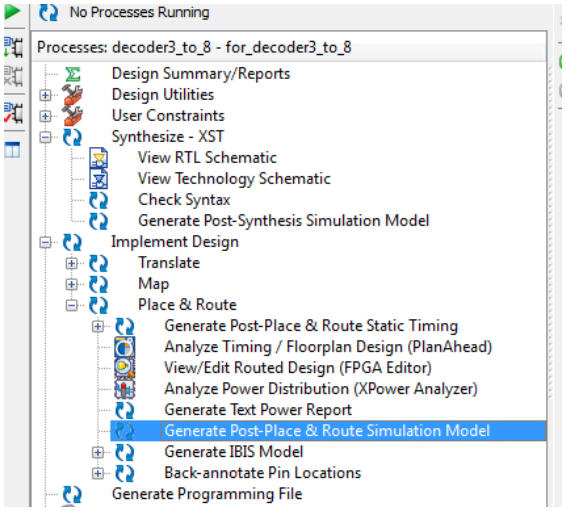


Figura 85.- Generar el modelo de simulación temporal.

La ejecución de los distintos procesos genera informes que se pueden consultar especialmente para el caso en el que existen errores que haya que corregir. Cuando se trata de *warnings* aparece un icono amarillo con el signo de exclamación al lado del proceso . Este tipo de errores, aunque no paran el proceso de síntesis o implementación, se deben tener en cuenta. Especialmente críticos son los *warnings* que aparecen el proceso de síntesis cuando se modelan *latches* indeseados debido fundamentalmente a especificaciones no conformes con las directrices de síntesis. Por otro lado, cuando hay algún error de sintaxis o se utilizan construcciones no sintetizables aparece el símbolo para indicar que hay un error que debe ser corregido antes de poder continuar con el resto de los procesos. Finalmente si no hay ningún error, aparecerá el símbolo

Para lanzar la simulación temporal se deber seleccionar en la ventana *View* la opción *Simulation* y después seleccionar la opción *Post-Route* en la ventana desplegable (Figura 86)

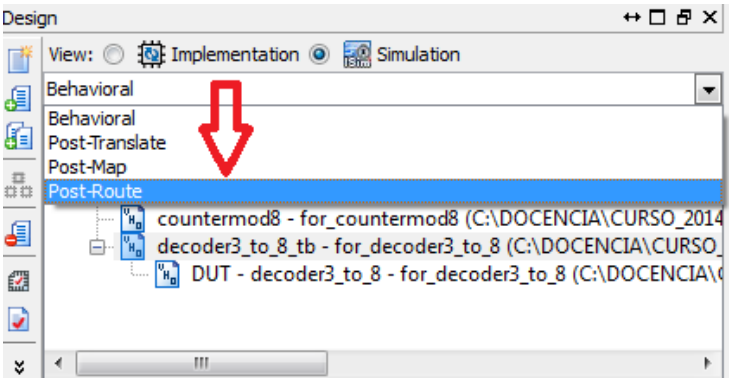


Figura 86.- Configurar el entorno ISE para la simulación temporal.

A continuación, como se hizo para la simulación temporal se lanza al simulador que en este caso se referencia como *Simulate Post-Place & Route Model* (Figura 87).

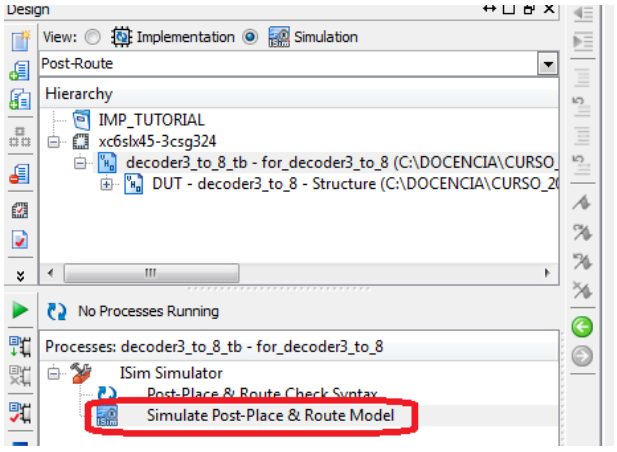


Figura 87.- Lanzar la simulación con el modelo temporal.

Como en el caso de la simulación funcional, en este paso es recomendable especificar un fichero de comando para el ISim. Sirva como ejemplo el del listado 7.

```
# cargar la configuración de la ventana de formas de onda
wcfg open wave_for_temporal.wcfg
# reiniciar la simulación
restart
#generar estímulos
#se define un estimulo sobre la señal polarity de tal forma que toma el valor 0
hasta el
#intante 170 ns y después se pone a 1 hasta el  instante 270 ns momento en el que
pasa a 0
#De esta forma se pueden definir formas de ondas aleatorias
isim force add polarity_i 0 -radix bin -value 0 -time 0 ns -value 1 -time 170 ns
-value 0 -time 270 ns
#se define un estimulo sobre la señal dec_in para que contemple todos los valores
posibles. Cada nuevo valor
#se genera con una diferencia temporal de 20 ns. El conjunto de valores asignados
se repite cada 100 ns
isim force add dec_in_i 0 -radix bin -value 000 -time 0 ns -value 001 -time 10 ns
-value 010 -time 20 ns  -value 011 -time 30 ns -value 100 -time 40 ns -value 101
-time 50 ns -repeat 100ns
#se ejecuta la simulación durante 300 ns

run 150 ns
```

Listado 7.- Comandos de ISim para la simulación temporal

Como se puede observar, los comandos para la definición de estímulos son los mismos variando únicamente la ventana de representación.

En la Figura 88 se muestra el resultado de la simulación temporal. Como se puede observar en la figura, hay un retado de 6,549 ns desde que cambia la entrada del decodificador hasta que la salida toma el valor esperado para la entrada anterior. Asimismo se producen transiciones intermedias o espúreos en la salida *dec_out_o* debido a que no todos los bits que conforman el bus de salida conmutan a la vez.

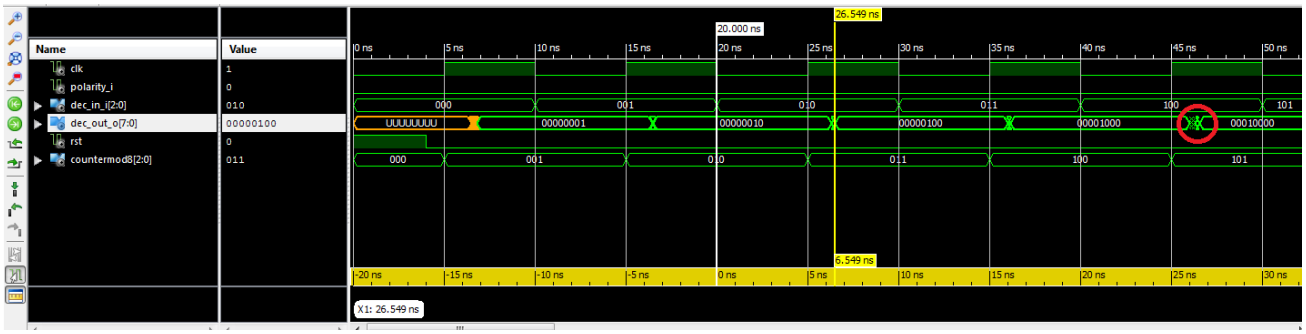


Figura 88.- Resultado de la simulación temporal.

9.- Flujo de diseño: generación del fichero de configuración.

La última etapa en el flujo de diseño es la generación del fichero de configuración (Figura 89) que será descargado en la FPGA.

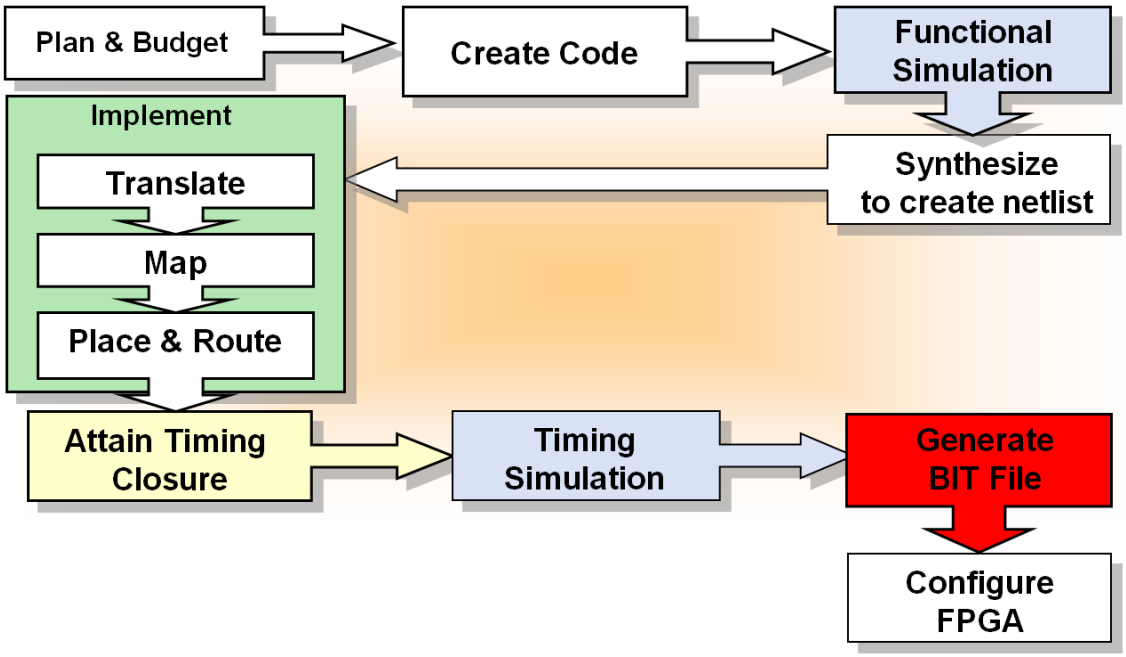


Figura 89.- Generación del fichero de configuración de la FPGA.

Para ello se debe ejecutar *Generate Programming File* desde la ventana procesos (Figura 90). Para poder acceder a este proceso se debe seleccionar en la ventana *View* la opción *Implementation*.

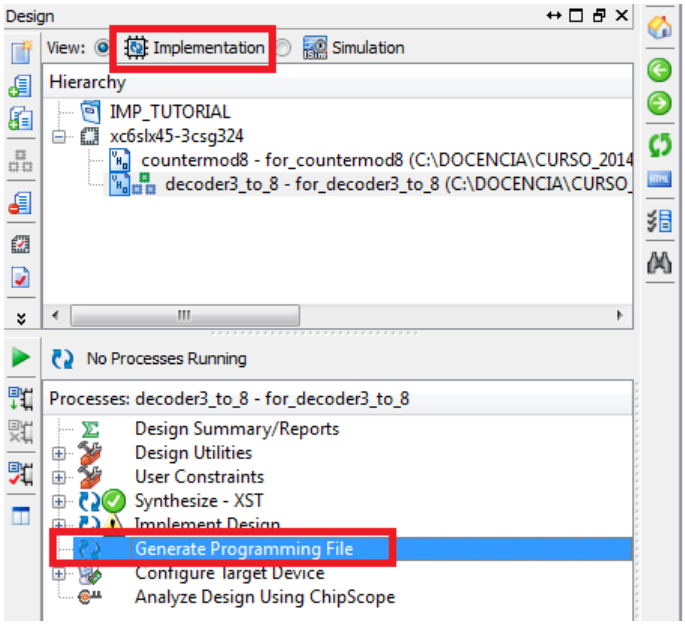


Figura 90.- Generar el fichero de configuración.

Antes de lanzar el proceso es aconsejable cambiar las propiedades del proceso *Generate Programming File*. Para ello, una vez seleccionado actuando sobre el botón derecho del ratón se selecciona *Process Properties*, mostrándose la ventana de la Figura 91.

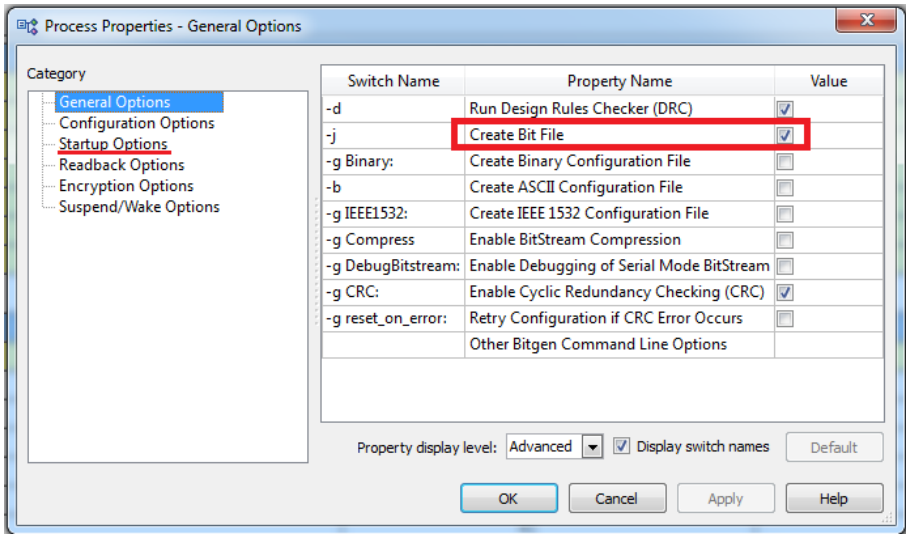


Figura 91.- Propiedades para el proceso de generación del fichero de configuración.

En ella se debe comprobar que la opción *Create Bit File* está activada. Seguidamente, y sin cerrar la ventana de opciones, en el apartado *Category* se selecciona la categoría *Startup Options* y se debe cambiar el valor del campo *FPGA Start-Up Clock de CCLK* a *JTAG-Clock* (Figura 92).

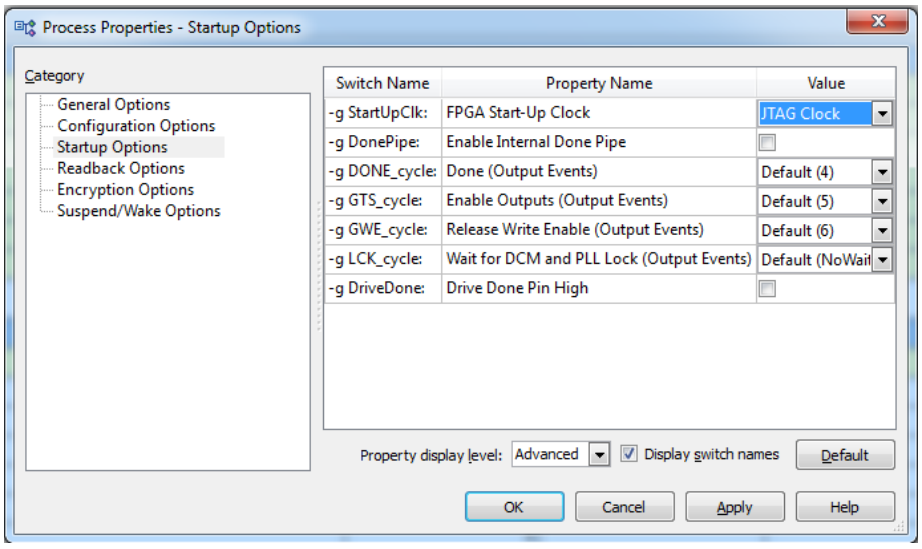


Figura 92.- Especificación de las opciones de arranque.

Tras realizar los cambios mencionados, se deben validan con ***OK***. Seguidamente, se hará doble clic sobre *Generate Programming File* para lanzar la creación del archivo de configuración.

Finalizado el proceso, en la carpeta del proyecto se crea un archivo con el nombre del modelo que se ha implementado que normalmente se corresponde con el mal alto en la jerarquía. El fichero generado por el proceso *Generate Programming File* tiene extensión *.bit*. Este es el archivo que se debe utilizar para programar la FPGA.

10.- Programaciòn de la FPGA.

Para cargar el archivo de configuración obtenidos en el paso anterior se podría realizar ejecutando el proceso *Configure Target Device* (Figura 93) de la ventana de *Processes*. Para ello se requiere un cable especial que comercializa la firma comercial *Xilinx*.

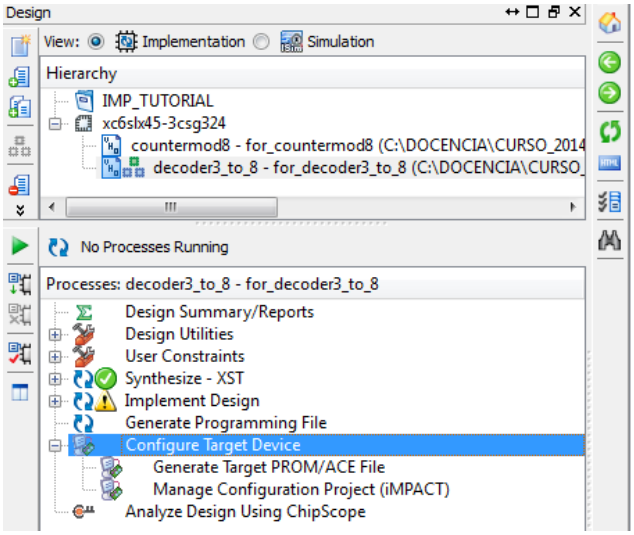


Figura 93.- Configurar el dispositivo.

No obstante, junto con la tarjeta *Atlys* de la firma comercial *Digilent* se proporciona un cable para realizar, entre otras cosas, la programación de la FPGA. Este cable se conecta al puerto USB del ordenador y a la tarjeta. Además se proporciona el programa Adept (Figura 94) para realizar la tarea se programación. Previa a la ejecución se debe conectar la tarjeta *Atlys* al ordenador.

Figura 94.- Icono de acceso al programa Adept.

Cuando se ejecuta el programa *Adept* se muestra la ventana de la Figura 95. El propio software se encarga de identificar la FPGA que está conectada al ordenador. Además, identifica la existencia en la placa de una memoria PROM XCF04S. Esta memoria se puede configurar para que proporcione una configuración inicial a la FPGA cuando se conecte la alimentación. Esta opción no se va a tratar en este tutorial.

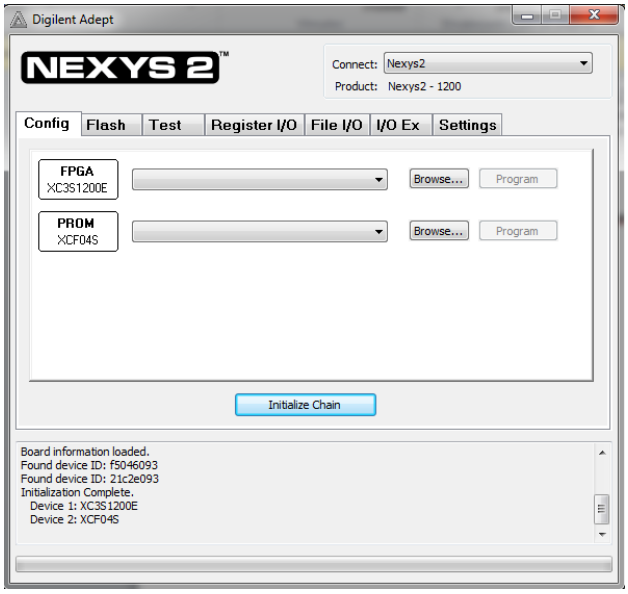


Figura 95.- Ventana principal de Adept para programar la FPGA.

Actuando sobe *Browse* se accede a un explorador para buscar el archivo de configuración de la FPGA (*top_system.bit*). Una vez seleccionado sólo hay que pulsar *Program*. Transcurrido un pequeño intervalo de tiempo la FPGA ya ha sido programada. Ahora, hay que verificar sobre la placa que se cumplen las especificaciones del diseño.