

1. INTRODUCCIÓN

Este documento contiene los enunciados de las **prácticas voluntarias** correspondientes a la asignatura *Estructura y Tecnología de Computadores III*. El software necesario para la realización de la práctica, el simulador de VHDL de Veribest (fichero *vhdl_sim.zip*, 21,1 Mbytes), lo tiene disponible en tres sitios:

- En el apartado *Material Complementario* del curso virtual de la asignatura.
- En el CD-ROM que viene con el texto base de la asignatura.
- En el servidor WWW de la Escuela Técnica Superior de Ingeniería Informática de la UNED: <http://www.ii.uned.es/cdrom/2006-07/material-docente/segundo/etciii/>

Por lo tanto, todo el material que necesita para la realización de las prácticas es el siguiente:

- Enunciado de la práctica (este documento).
- Simulador de VHDL (funciona sobre Windows).
- Texto base de la asignatura o bibliografía complementaria.

Las actividades descritas en esta memoria se encuentran divididas en dos grupos. Las primeras tienen por finalidad que practique con el entorno de simulación VHDL para que aprenda los conceptos básicos del mismo. El segundo tipo de actividades están orientadas a la aplicación y puesta en práctica de diferentes aspectos del lenguaje VHDL.

2. INSTALACIÓN DEL SIMULADOR VHDL DE VERIBEST

La instalación del software de simulación es bastante sencilla. Una vez que se ha descargado el software *VHDL_sim.zip*, se descomprime en una carpeta y se procede a su instalación haciendo doble clic sobre el fichero *setup.exe*.

Tras la instalación, en el grupo de programas de Windows dispondrá de un nuevo grupo denominado *Veribest VB99.0* que corresponde al simulador que acaba de instalar.

3. ACTIVIDAD MANEJO DEL SIMULADOR (I)

El objetivo de estas actividades es que se inicie en el manejo del simulador para que adquiera los conocimientos mínimos para poder abordar la realización de las actividades prácticas más complejas. En esta actividad se limitará a seguir las indicaciones que se dan en cada paso.

Paso 1

Arranque el simulador VHDL de Veribest. Para ello, seleccione el icono Veribest VHDL en el grupo de programas *Inicio -> Programas -> Veribest VB99.0 -> Veribest VHDL Simulator*. Una vez que haya arrancado el simulador, lo primero que debe hacer es crear un espacio de trabajo. El espacio de trabajo será el área en el que se almacenen todos los ficheros VHDL que vaya a utilizar durante el desarrollo de una actividad práctica concreta.

Para crear el espacio de trabajo despliegue el menú *Workspace* y seleccione la opción *New...* También puede optar por seleccionar el comando *New...* del menú *File*; en este caso obtendrá una pequeña ventana en la que tiene que seleccionar la opción *VHDL Workspace* ya que también tiene la opción de crear un fichero VHDL (*VHDL Source File*). Con independencia del método por el que opte para la creación definitiva del espacio de trabajo, visualizará una ventana (Figura 1) en la que tiene que teclear el nombre del espacio de trabajo, que en este caso, es *actividad1*.

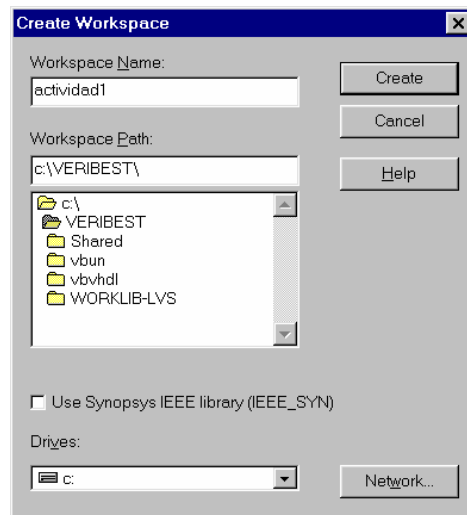


Figura 1: Creación de un espacio de trabajo.

Tras realizar esta acción aparecerá en el lateral izquierdo de la ventana principal del simulador una nueva ventana con el título *actividad1.vpd*. Inicialmente, la ventana sólo muestra una carpeta vacía, denominada *actividad1 source*, que es la que contendrá todos los ficheros VHDL que vaya desarrollando a medida que avance en la actividad (ver Figura 2).

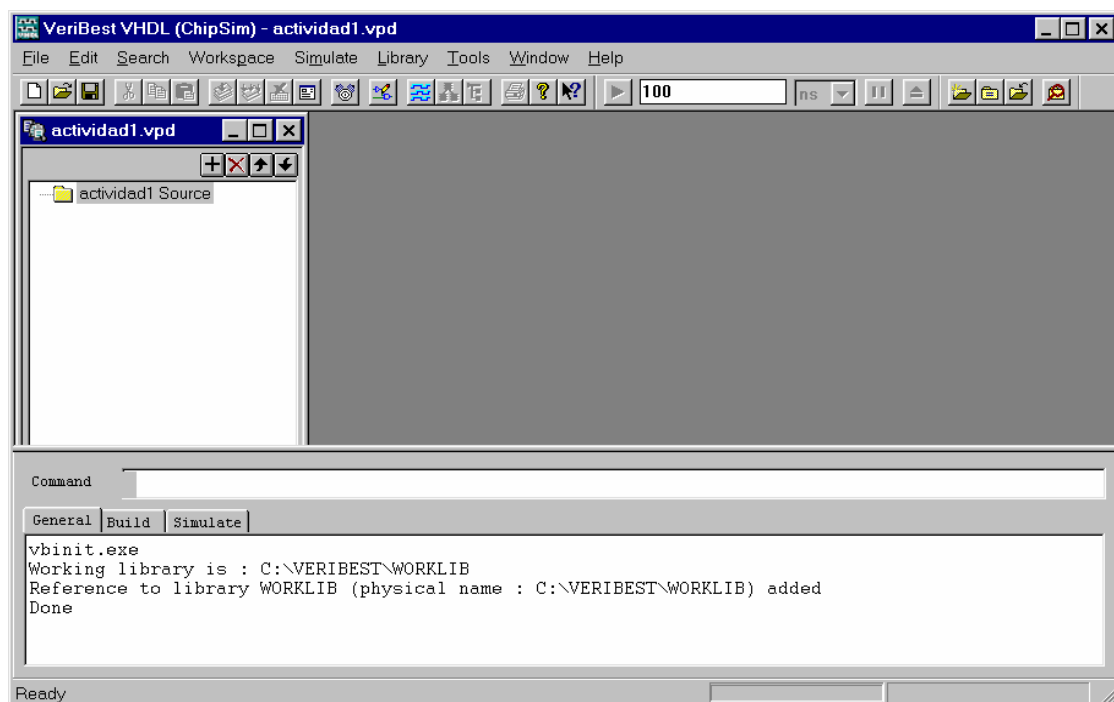


Figura 2: Entorno de simulación con el espacio de trabajo actividad1 ya creado.

Paso 2

En primer lugar tiene que crear el código que declara una entidad de una puerta lógica NOT (un inversor). El código que debe teclear en un fichero *inversor.vhd* es el siguiente:

```
ENTITY inversor IS
    PORT(x:IN BIT; y:OUT BIT);
END inversor;
```

Fichero 1: *inversor.vhd*.

Para introducir el código de la entidad de la puerta NOT, cree un nuevo fichero utilizando el comando *New...* del menú *File*. Obtendrá una ventana en blanco en la que puede teclear el código VHDL. Una vez haya concluido, guárdelo con el nombre *inversor.vhd* mediante el comando *Save* situado en el menú *File*. Esto produce la creación del fichero *inversor.vhd* en el disco duro de su ordenador.

Paso 3

El siguiente paso es la inclusión del fichero *inversor.vhd* en el espacio de trabajo *actividad1*. Para ello haga clic con el puntero del ratón sobre el botón “+” situado en la esquina superior derecha de la ventana del espacio de trabajo *actividad1.vpd*. También puede utilizar el comando *Add Files into Workspace...* situado en el menú *Workspace*. Con independencia de la opción que seleccione obtendrá una ventana para la selección de un fichero. Seleccione el fichero *inversor.vhd*. La ventana principal del simulador tiene que quedar tal y como se muestra en la Figura 3.

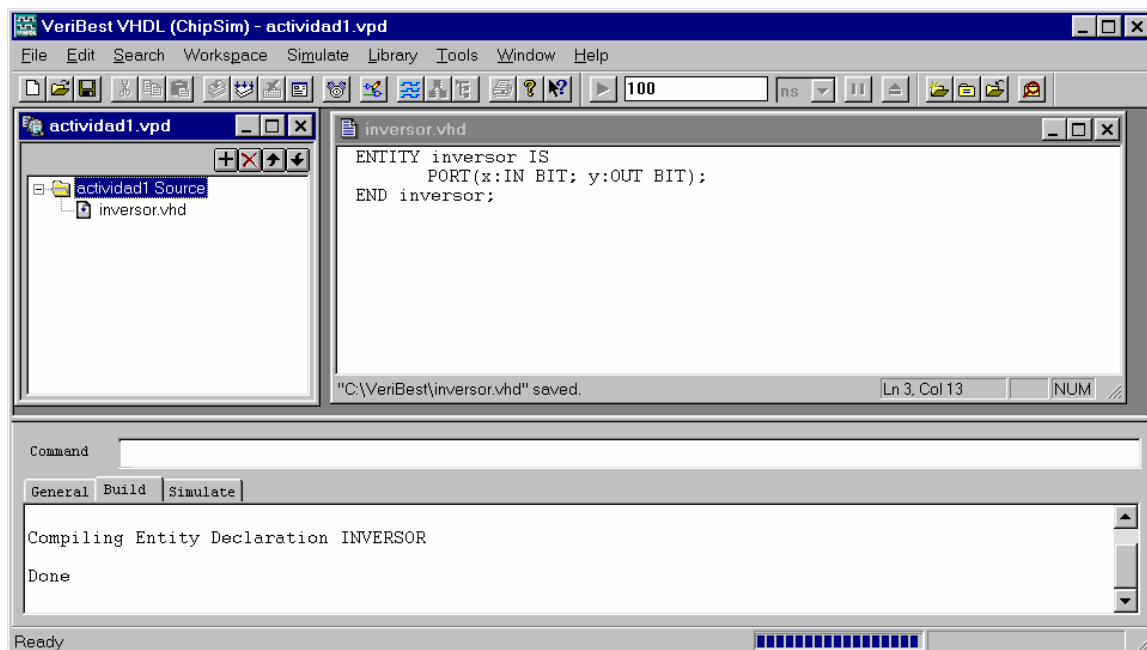


Figura 3: Área de trabajo con el fichero *inversor.vhd*.

Para saber si el código que ha introducido es correcto, compile el fichero recurriendo al comando *Compile* situado en el menú *Workspace*. Tras unos instantes, obtendrá el resultado de la compilación en la ventana de información situada en la parte inferior de la ventana principal. Realice prácticas de compilación introduciendo algunos errores en el código, como por ejemplo, suprimiendo algún punto y coma.

Es importante recalcar que el compilador no es sensible a las mayúsculas por lo que le es indiferente si las palabras reservadas están en minúsculas o en mayúsculas. Como buena norma de programación se recomienda que las palabras reservadas del lenguaje se escriban en mayúsculas ya que facilita notoriamente la legibilidad del código VHDL.

Otro aspecto a destacar es que la ventana de información contiene tres pestañas: *General*, *Build* y *Simulate*. *General* proporciona información sobre cualquier acción que se realice en la herramienta. *Build* sólo recoge la información referente al proceso de compilación de ficheros: si el fichero es correcto, o si ha habido errores de qué tipo de errores se trata y las líneas en que aparecen. *Simulate* proporciona información sobre la fase de simulación.

Paso 4

Una vez que se haya compilado el código de la entidad con éxito, es decir, sin errores, tiene que crear el código VHDL correspondiente a la arquitectura del inverter. Este código se detalla en el fichero *inversor_arq1.vhd*.

```

ARCHITECTURE arquitectural1 OF inversor IS
BEGIN
PROCESS
BEGIN
    y <= NOT x;
    WAIT ON x;
END PROCESS
END arquitectural1;

```

Fichero 2: *inversor_arq1.vhd*.

Realice los mismos pasos que en el caso de la entidad: salve el fichero en el disco duro de su ordenador con el nombre *inversor_arq1.vhd*, inclúyalo en el espacio de trabajo y compílelo para eliminar los errores que haya podido introducir.

Aunque la entidad y las arquitecturas de un objeto pueden ir en un único fichero, por claridad, la norma que se seguirá en la realización de las prácticas es utilizar ficheros diferentes: uno para la entidad y otro por cada arquitectura asociada a la entidad.

Paso 5

Una vez que ha definido la entidad y la arquitectura de la puerta lógica NOT, el último paso para poder simularla es escribir una prueba. Esta prueba debe estar compuesta de dos elementos: una entidad y su arquitectura. Por lo tanto, debe crear dos ficheros, uno denominado *prueba_inversor.vhd* y otro *prueba_inversor_arq1.vhd* utilizando el código VHDL que figura en los Ficheros 3 y 4.

```

ENTITY prueba_inversor IS
END prueba_inversor;

```

Fichero 3: *prueba_inversor.vhd*.

```

ARCHITECTURE arquitectural1 OF prueba_inversor IS
COMPONENT puerta
    PORT(x:IN BIT; y:OUT BIT);
END COMPONENT;

FOR puerta1 : puerta USE ENTITY WORK.inversor(arquitectural1);

SIGNAL entrada, salida: BIT;

BEGIN
    puerta1 : puerta PORT MAP (entrada, salida);
    entrada <=
        '1' AFTER 3 ns,
        '0' AFTER 6 ns,
        '1' AFTER 9 ns,
        '0' AFTER 11 ns,
        '1' AFTER 13 ns,
        '0' AFTER 15 ns,
        '1' AFTER 17 ns,
        '0' AFTER 20 ns,
        '1' AFTER 23 ns,
        '0' AFTER 26 ns;
END arquitectural1;


```

Fichero 4: *prueba_inversor_arq1.vhd*.

No olvide incluir los dos ficheros en el espacio de trabajo y compilarlos para asegurarse de que ambos están libres de errores.

Observe que la prueba consiste en instanciar un inversor e introducirle una señal binaria denominada *entrada*. El resultado debe ser la señal, denominada *salida*, invertida y sin retardos ya que, por el momento, no se han introducido retardos en la puerta, salvo los retardo δ .

Paso 6

Creada la prueba hay que fijar las condiciones de la simulación. Para ello debe seleccionar la opción *Settings...* del menú *Workspace*, o recurrir al icono  situado en la barra de herramientas.

Tras esto, obtendrá una ventana en la que debe seleccionar la pestaña *Simulate* para acceder al cuadro de diálogo con las opciones de simulación (Figura 4). Lo primero que tiene que realizar es desplegar la carpeta *WORK* para visualizar los elementos sobre los que se puede practicar una simulación. Obtendrá las entidades y arquitecturas que ha programado en los pasos anteriores. Para realizar la prueba de la puerta lógica debe seleccionar la arquitectura *ARQUITECTURA1* (que es la que contiene la prueba) y la entidad que lleva asociada esa arquitectura (*PRUEBA_INVERSOR*). Para ello, seleccione la arquitectura *ARQUITECTURA1* de la entidad *PRUEBA_INVERSOR* con el cursor del ratón (haga clic sobre ella) y pulse el botón *Set*. Tras esta acción, los campos *Entity* y *Arch* se rellenarán automáticamente con los nombres de la entidad y de la arquitectura.

Sin abandonar la ventana, lo siguiente es activar la opción *Trace On*. La activación de esta opción no es necesaria para realizar la simulación pero sí imprescindible si se pretende visualizar la evolución de las señales tras ejecutar la simulación de la puerta. Fijadas las condiciones de simulación, pulse el botón *Aceptar*.

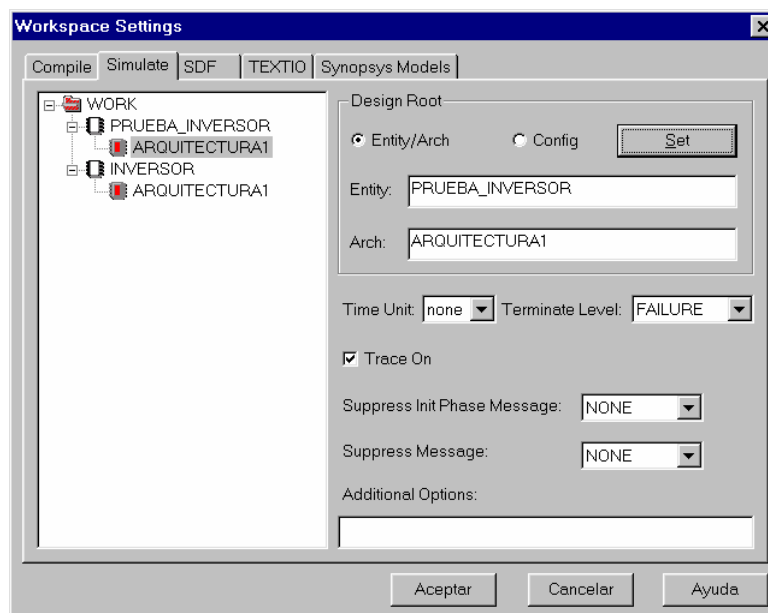



Figura 4: Opciones de simulación.

Paso 7

Llegados a este paso ya se está en condiciones de comenzar a simular. En primer lugar hay que arrancar el simulador, y esto se puede realizar de dos formas: mediante la opción *Execute Simulator* del menú *Workspace*, o mediante el icono  situado en la barra de herramientas. Tras cualquiera de las dos opciones, el simulador se activa y en la ventana *Simulate*, situada en la parte inferior del espacio de trabajo, aparece el texto que se muestra en la Figura 5.

```
vbyhdl4ive -d PRUEBA_INVERSOR ARQUITECTURA1 -tr on -i -dbg
VeriBest VHDL Simulator Version 15.00.00.25.
Starting Simulation ... Fri Dec 20 12:51:31 2002

License error: Error reading license file
Cannot find license file
The license files (or server network addresses) attempted are
listed below. Use LM_LICENSE_FILE to use a different license file,
or contact your software provider for a license file.
Feature:          VBVHDL_CHIP_NT
Filename:         C:\flexlm\license.dat
License path:     C:\flexlm\license.dat
FLEXlm error:    -1,359. System Error: 2 "No such file or directory"
For further information, refer to the FLEXlm End User Manual,
available at "www.globetrotter.com".
```

```

License file is C:\flexlm\license.dat
Feature is VBVHDL_CHIP_NT.
-----
No license was found.
The simulator will run in reduced capacity mode.

If you would like a license for full capacity operation,
please contact us at sales@veribest.com
or in the USA call us at 1-888 482-3322.
http://www.veribest.com/vhdl.html
-----
NOTE: Number of component (cell) instances in the design: 1

Checkpointing at simulation time 0 ns.
Checkpoint completed.
Ready to simulate ... Fri Dec 20 12:51:34 2002

-- Message Summary:
Total: 0 error(s), 0 warning(s), 0 note(s)


```


Figura 5: Mensajes del simulador.

Lo más importante de este largo mensaje se encuentra en la última línea. En ella se indica que no hay errores y que se puede proceder a realizar una simulación. Durante la aparición de este mensaje habrá obtenido una ventana de aviso (Figura 6) en la que se indica que al no encontrar una licencia se está utilizando el simulador en modo limitado. Pulse *Aceptar* para continuar trabajando.



Figura 6: Mensaje de aviso sobre la ausencia de licencia.

Si en algún momento desea abandonar el simulador tiene tres opciones: el botón  de la barra de herramientas, el comando *Quit* en el menú *Simulate*, o tecleando el comando *quit* en la línea de comandos.

Una vez activado el simulador ya puede comenzar a realizar simulaciones. Para ello dispone de varias opciones: pulsar el botón *Play* , activar la orden *Run* en el menú *Simulate*, o teclear *run* en la línea de comandos. El tiempo y las unidades de simulación se fijan en un cuadro situado en la barra de herramientas junto al botón *Play* (Figura 7). Por defecto, el tiempo de simulación está establecido en *100 ns.*, pero dado que la prueba que se ha programado sólo alcanza hasta los 30 ns. fije el tiempo de simulación en 40 ns.

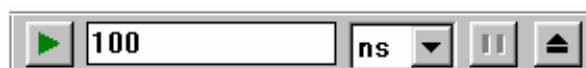



Figura 7: Cuadro para fijación de la duración de la simulación.

Tras ejecutar la simulación de la puerta, tiene que visualizar los resultados recurriendo a la herramienta *Waveform Viewer*. Para abrirla, pulse sobre el botón  o ejecute el comando *New WaveForm Window* situado en el menú *Tools*. Obtendrá una ventana (Figura 8) con una escala de tiempos en la que, por el momento, no se visualiza ninguna señal.

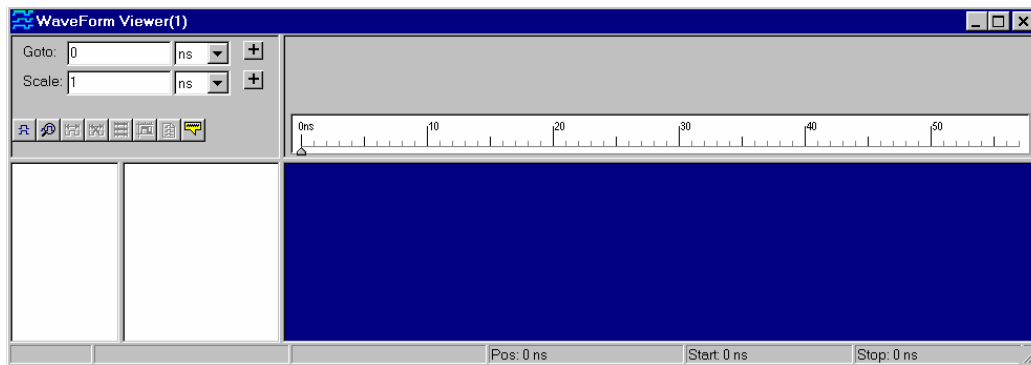



Figura 8: Ventana de visualización de señales.

Para seleccionar las señales pulse el botón  en la ventana de visualización de señales (lateral izquierdo de la Figura 8). Obtendrá una nueva ventana (Figura 9) en la que puede seleccionar las señales existentes tanto en la arquitectura del test de prueba (*ENTRADA* y *SALIDA*) como en la arquitectura del inversor (*x* e *y*). Seleccione las señales que corresponden a la arquitectura de la prueba pulsando sobre el botón *Add All* y pulse *Close* para cerrar la ventana.

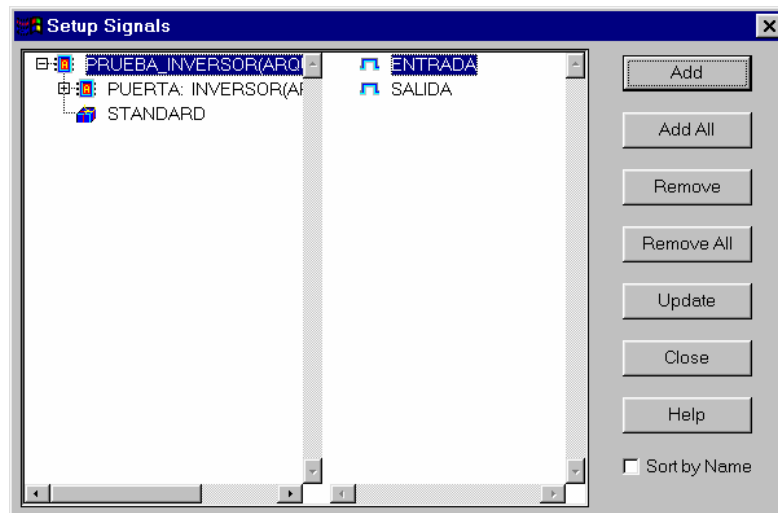


Figura 9: Selección de señales a visualizar.

Tras seleccionar las señales, la ventana de visualización mostrará el resultado de la simulación del inversor. El resultado que tiene que obtener debe coincidir con el mostrado en la Figura 10. Obviamente, la señal de salida debe ser la señal de entrada invertida.

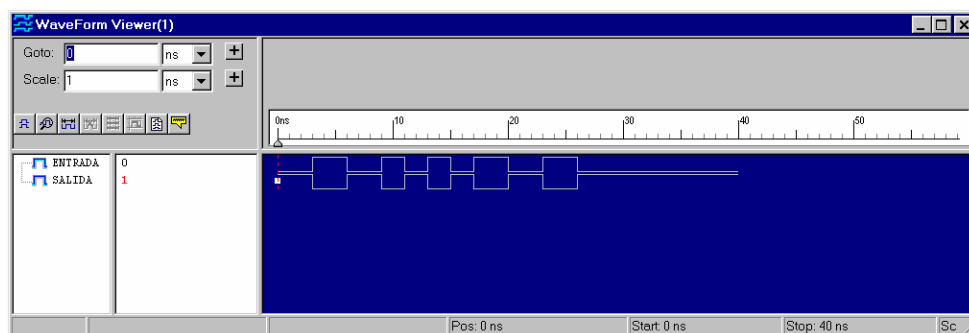


Figura 10: Resultado de la simulación.

Actividades complementarias

1. Trabaje con esta herramienta de visualización analizando las opciones que presenta: almacenamiento de resultados en ficheros de texto, visualización detallada de las señales, creación de cursores auxiliares de señalización, etc.
2. Modifique el código VHDL del fichero *prueba_inversor_arq1.vhd* con el objeto de que no sea necesario incluir la especificación de la configuración del inversor.
3. Copie todos los ficheros VHDL (extensión *.vhd*) que haya creado en esta actividad en una carpeta del disco duro de su ordenador denominada *Actividad1*. Incluya en esta carpeta el fichero *actividad1.vpd* que define el contenido del espacio de trabajo.

4. ACTIVIDAD MANEJO DEL SIMULADOR (II)

Esta segunda actividad práctica es muy similar a la anterior pero se introducen algunas variaciones como, por ejemplo, retardos en el inversor y unidades de configuración.

Paso 1

Arranque el simulador y cree un nuevo espacio de trabajo denominado *actividad2*. Si ha optado por crear el espacio de trabajo en el mismo directorio en que creó *actividad1* obtendrá un mensaje en el que se le indica si desea reinicializar la biblioteca de trabajo ya existente (Figura 11). Seleccione *Sí* para continuar trabajando.

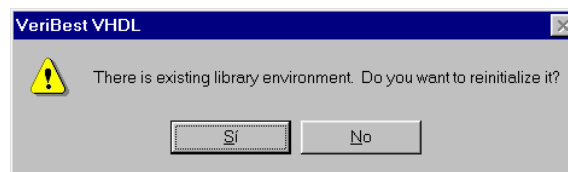


Figura 11: Reinicialización de la librería.

Añada en el nuevo espacio de trabajo el fichero *inversor.vhd* que define la entidad asociada a la puerta NOT (este fichero ya lo creó en la actividad previa) y cree un nuevo fichero *inversor_arq2.vhd* en el cual tiene que añadir a la puerta un retardo inercial de 1 ns. Añada el fichero al espacio de trabajo y compílelo con el objeto de comprobar si existen errores.

```
ARCHITECTURE arquitectura2 OF inversor IS
BEGIN
PROCESS
BEGIN
    y <= NOT x AFTER 1 ns;
    WAIT ON x;
END PROCESS;
END arquitectura2;
```

Fichero 5: *inversor_arq2.vhd*.

Paso 2

Desarrolle una prueba para la nueva arquitectura que acaba de crear. Para ello añada el fichero *prueba_inversor.vhd* creado en la primera actividad al espacio de trabajo de esta actividad, y escriba un fichero *prueba_inversor_arq2.vhd* como el que se muestra en el Fichero 6. Compile todos los ficheros para eliminar errores de sintaxis.


```

ARCHITECTURE arquitectura2 OF prueba_inversor IS

COMPONENT puerta
    PORT(x:IN BIT; y:OUT BIT);
END COMPONENT;

SIGNAL entrada, salida: BIT;

BEGIN

    puertal : puerta PORT MAP (entrada, salida);
    entrada <=
        '1' AFTER 3 ns,
        '0' AFTER 6 ns,
        '1' AFTER 9 ns,
        '0' AFTER 11 ns,
        '1' AFTER 13 ns,
        '0' AFTER 15 ns,
        '1' AFTER 17 ns,
        '0' AFTER 20 ns,
        '1' AFTER 23 ns,
        '0' AFTER 26 ns;

END arquitectura2;

```

Fichero 6: prueba_inversor_arq2.vhd.

Paso 3

Observe que en esta nueva arquitectura no se ha realizado la especificación de configuración en el fichero ya que se va a recurrir para ello a una *declaración de configuración*. El fichero que debe crear se llama *prueba_config1.vhd* (Fichero 7). Añádalo al espacio de trabajo y compílelo con el objeto de comprobar si es correcto.

```

CONFIGURATION config1 OF prueba_inversor IS
    FOR arquitectura2
        FOR puertal: puerta USE ENTITY WORK.inversor(arquitectura2);
        END FOR;
    END FOR;
END config1;

```

Fichero 7: prueba_config1.vhd.

En este instante el espacio de trabajo debe contener cinco ficheros tal y como se muestra en la Figura 12.

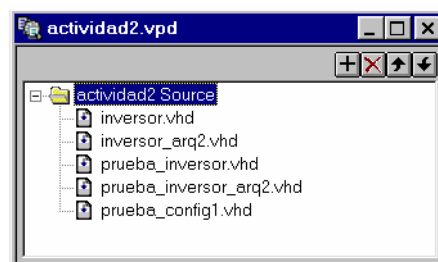


Figura 12: Espacio de trabajo de la actividad 2.

Paso 4

En este paso se va a realizar la simulación de la puerta con el retardo inercial de 1 ns. Al igual que se realizó en la actividad 1, antes de proceder a simular debe fijar las condiciones de simulación. Abra la ventana con las opciones del espacio de trabajo (Figura 13), seleccione la pestaña *Simulate* y despliegue los elementos que hay en la carpeta *WORK*. Observe que ahora aparece un nuevo elemento además de las entidades y las arquitecturas: la *configuración*. Para introducir el elemento raíz del que se parte para realizar la simulación seleccione el objeto *CONFIG1* y pulse el botón *Set*. El nombre del elemento se introduce automáticamente en el campo *Config*. No olvide activar la opción *Trace On*. Para finalizar pulse el botón *Aceptar*.

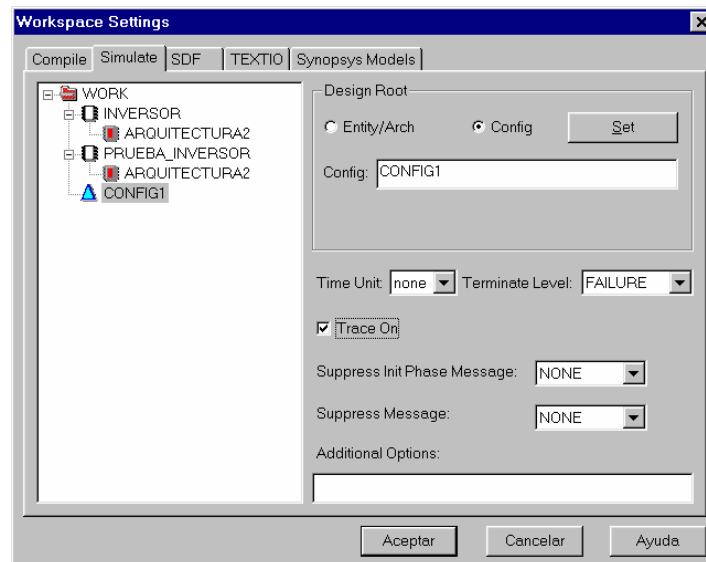


Figura 13: Condiciones de simulación de la actividad 2.

Paso 5

Arranque el simulador tal y como hizo en la actividad previa, fije el tiempo de simulación a 40 ns. y realice una simulación. Si visualiza las señales de entrada y salida en una ventana (Figura 14) podrá apreciar que la señal de salida *SALIDA* está desplazada 1 ns. con respecto a la señal de entrada.

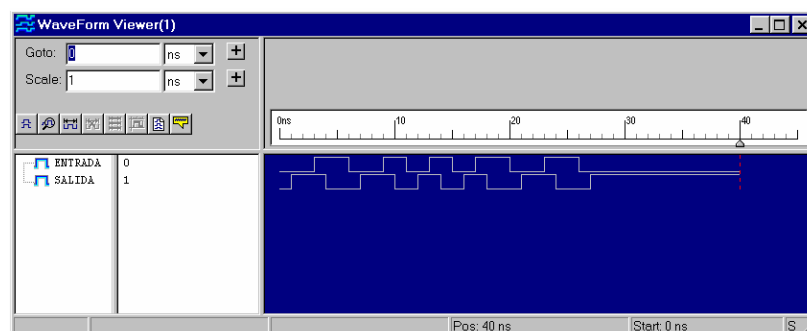


Figura 14: Resultado de la simulación con un retardo de 1 ns.

Paso 6

El siguiente paso consiste en modificar el tipo de retardo de la puerta lógica. En lugar de un retardo de 1 ns. debe incluir un *retardo inercial* de 5 ns. Compile los ficheros de nuevo y realice una simulación de 40 ns. La señal de respuesta que se obtiene debe ser análoga a la que refleja la Figura 15.

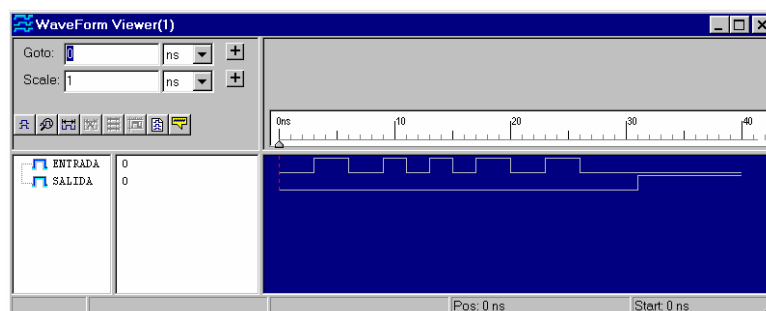


Figura 15: Resultado de la simulación con un retardo inercial de 5 ns.

Paso 7

Vuelva a realizar la misma simulación pero esta vez el retardo tipo *TRANSPORT* de la puerta debe ser de 5 ns. El resultado obtenido debe ser análogo al de la Figura 16.

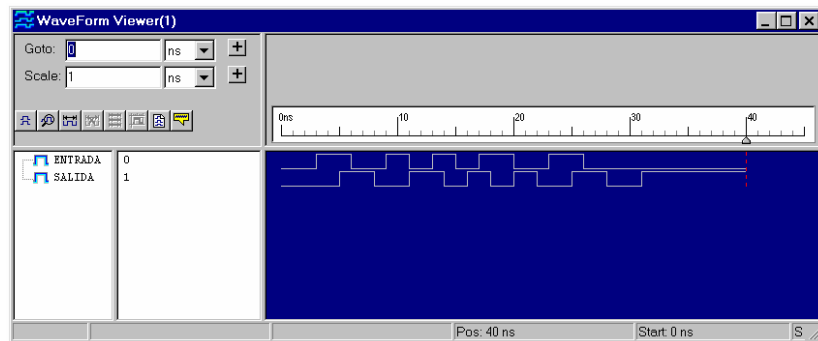


Figura 16: Resultado de la simulación con un retardo de transporte de 5 ns.

Actividades complementarias

1. ¿Qué ocurre si se elimina la sentencia *WAIT ON* x del inversor?
2. ¿Cuál es la diferencia entre el retardo de tipo *INERTIAL* y el *TRANSPORT*?
3. ¿Qué otro comando existe en la herramienta que sea análogo a la opción *Trace On* que aparece en la ventana en la que se fijan las condiciones de la simulación?
4. Sin modificar la señal de entrada que está utilizando como prueba, ¿qué sucede en la señal de salida si la puerta lógica dispone de un retardo del tipo *INERTIAL* de 4 ns.?
5. Reinicie la biblioteca sobre la que está trabajando mediante el comando *Reinitialize Lib environment* situado en el menú *Workspace*. A continuación, y sobre la ventana que contiene los ficheros del espacio de trabajo, sitúe el fichero con la arquitectura del inversor delante de la entidad, es decir, sitúe el fichero *inversor_arq2.vhd* delante del fichero *inversor.vhd*. Compile todos los ficheros y obtendrá el siguiente error:

```
Compiling Architecture ARQUITECTURA2 of INVERSOR
-----
1: ARCHITECTURE arquitectura2 OF inversor IS
   ^^^^^^^
[Failure] Entity declaration INVERSOR not found in library WORK
Done
```

¿Por qué sucede este error si los ficheros son correctos? ¿Cuál es la solución?

6. Copie todos los ficheros VHDL (extensión *.vhd*) que haya creado en esta actividad en una carpeta denominada *Actividad2*. Incluya en esta carpeta el fichero *actividad2.vpd* que define el contenido del espacio de trabajo.

5. ACTIVIDAD MANEJO DEL SIMULADOR (III)

La última actividad relacionada con el manejo del simulador consistirá en la utilización de la sentencia de generación *GENERATE*. El objetivo que se persigue es que construya un nuevo elemento compuesto por 8 inversores mediante la replicación de la puerta lógica que ha creado en las actividades previas.

Paso 1

El primer paso consistirá en crearse un nuevo espacio de trabajo *actividad3* en el que debe incluir la puerta lógica y su arquitectura inicial (sin ningún tipo de retardo). Es decir, debe incluir en el espacio de trabajo los ficheros *inversor.vhd* e *inversor_arq1.vhd*.

Paso 2

El segundo paso es la creación de la entidad que define la etapa. El fichero que necesita se muestra en el cuadro Fichero 8. Escríbalo, inclúyalo en el espacio de trabajo y compílelo.

```
ENTITY etapa_inversores IS
  PORT (entradas : IN bit_vector (7 downto 0);
        salidas : IN bit_vector (7 downto 0));
END etapa_inversores;
```

Fichero 8: etapa.vhd.

Tras la creación de la entidad de la etapa debe crear la arquitectura (fichero *etapa_arq1.vhd*), incluirla en el espacio de trabajo y compilarla para conocer si está libre de errores.

```
ARCHITECTURE arquitectural1 OF etapa_inversores IS
  COMPONENT inversor
    PORT(x:IN BIT; y:OUT BIT);
  END COMPONENT;

BEGIN

  gen : FOR I IN 0 TO 7 GENERATE
    puerta : inversor PORT MAP (entradas(I), salidas(I));
  END GENERATE gen;

END arquitectural1;
```

Fichero 9: etapa_arq1.vhd.

Paso 3

En este paso tiene que programar la prueba de la etapa de 8 inversores que se acaba de crear. Para ello cree los ficheros *prueba_etapa.vhd* y *prueba_etapa_arq1.vhd*, tal y como se muestra en los ficheros 10 y 11, y añádalos al espacio de trabajo. No olvide compilarlos para comprobar que están libres de errores.

```
ENTITY prueba_etapa IS
END prueba_etapa;
```

Fichero 10: prueba_etapa.vhd.

```
ARCHITECTURE arquitectural1 OF prueba_etapa IS
  COMPONENT circuito
    PORT (entradas : IN bit_vector (7 downto 0);
          salidas : OUT bit_vector (7 downto 0));
  END COMPONENT;

  FOR ALL : circuito USE ENTITY WORK.etapa_inversores(arquitectural1);

  SIGNAL entradas, salidas: bit_vector (7 downto 0);

BEGIN

  circuito1 : circuito PORT MAP (entradas, salidas);

  entradas <= "10101010" AFTER 5 ns,
             "01010101" AFTER 10 ns,
             "11110000" AFTER 15 ns,
             "11110000" AFTER 20 ns,
```

```

"11110000" AFTER 25 ns,
"00001111" AFTER 30 ns,
"10000001" AFTER 35 ns;

END arquitectural;

```

Fichero 11: prueba_etapa_arq1.vhd.

Paso 6

Una vez que ha creado los seis ficheros (dos para la puerta, dos para la etapa y dos para la prueba) ya está en condiciones de poder realizar la simulación. Recuerde fijar las condiciones de simulación antes de arrancar el simulador y fijar el tiempo de la prueba a 40 ns. Una vez que haya realizado la simulación, el aspecto de las señales de entrada y salida debe ser similar a las señales que se visualizan en la Figura 17.

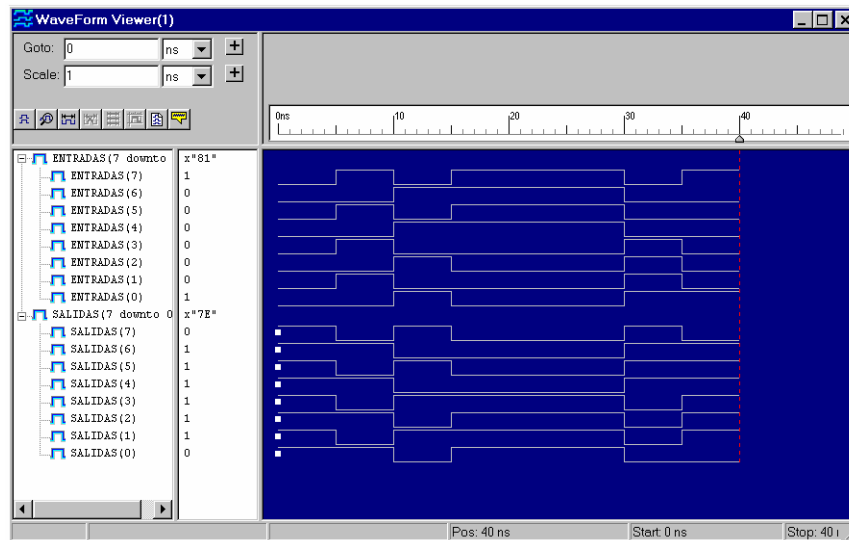


Figura 17: Resultado de la simulación de la etapa de 8 inversores.

Actividades complementarias

Responda a las siguientes cuestiones:

1. ¿Cómo se almacenan los resultados de la simulación en un fichero ASCII?
2. Cree un fichero con una declaración de configuración de forma que no tenga que especificar la configuración del objeto *etapa* en la arquitectura de la prueba. Para ello cree un fichero con la nueva arquitectura *prueba_etapa_arq2.vhd* y otro *prueba_conf1.vhd* que contenga la configuración. Ejecute la simulación y compruebe que es correcta.
3. ¿Por qué no es necesario especificar la configuración del inversor en el fichero que define la arquitectura de la etapa (fichero *etapa_arq1.vhd*)?
4. ¿Qué sucede si el nombre del componente en lugar de ser *inversor* es *puerta*?
5. Cree un fichero con una nueva arquitectura de la prueba en la cual la generación de las 8 señales de entrada se realice mediante una sentencia `FOR...GENERATE`. El fichero debe llamarse *prueba_etapa_arq3.vhd*.
6. Copie todos los ficheros VHDL (extensión *.vhd*) que haya creado en esta actividad en una carpeta denominada *Actividad3*. Incluya en esta carpeta el fichero *actividad3.vpd* que define el contenido del espacio de trabajo.

6. ACTIVIDAD MANEJO DEL SIMULADOR (IV)

Esta actividad está orientada exclusivamente a aprender las nociones básicas sobre el empleo del depurador con objeto de aprender a encontrar errores en aquellos casos en que la complejidad del diseño lo requiera. Como ficheros prueba cree un espacio de trabajo denominado *actividad4* e incluya en él todos los ficheros VHDL que se crearon en la actividad anterior, la tercera.

Paso 1

En primer lugar, y para posteriormente poder hacer uso del depurador, es necesario activar la opción *Debug* en las opciones de compilación. Para ello, visualice la ventana de opciones mediante el comando del menú *Workspace* y seleccione la pestaña *Compile* (Figura 18).

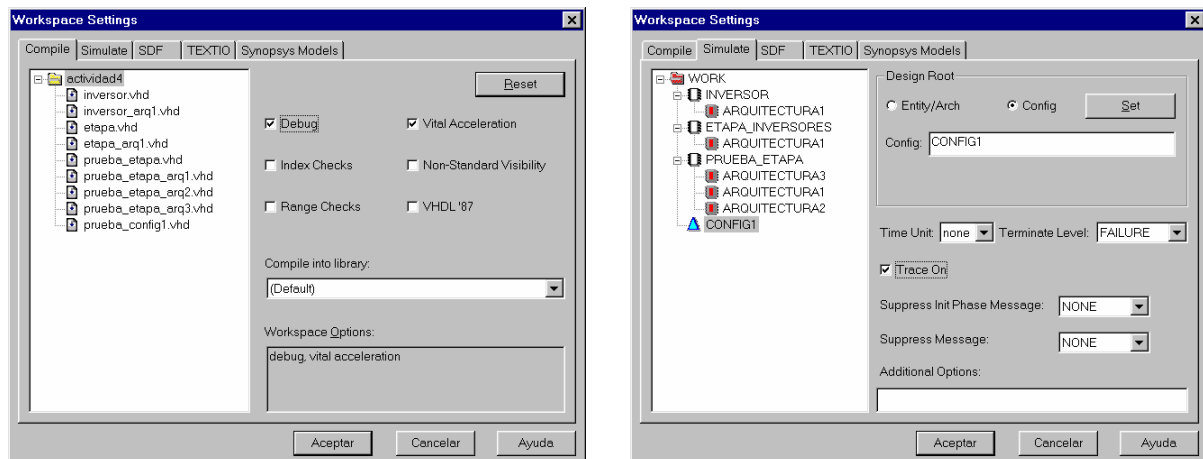


Figura 18: Opciones de compilación y de simulación.

Aunque solo interesa por el momento la opción *Debug*, a continuación se describen brevemente otras opciones de compilación:

- *Debug*. Activando esta opción el archivo seleccionado será compilado con la opción de depuración. Con esta opción activa antes de una compilación, es posible utilizar todos los comandos del menú *Debug* durante la fase de simulación. Si se desea que todos los archivos del espacio de trabajo sean compilados con esta opción basta con seleccionar la carpeta del espacio de trabajo en la ventana y activar la opción.
- *Index Checks*. Si esta opción se encuentra activa, el compilador comprueba la existencia de errores en los índices de los arrays.
- *Range Checks*. Si esta opción se encuentra activa, el compilador comprueba la existencia de errores en los rangos de los subtipos que se hayan definido.

Asegúrese de seleccionar la carpeta *actividad4* y activar la opción de depuración. Como elemento raíz para la simulación seleccione la configuración *CONFIG1* (Figura 18). Tras fijar las condiciones de compilación y simulación, pulse *Aceptar* para cerrar la ventana.


No olvide realizar una compilación de todos los ficheros ya que ahora tiene la opción de depuración activada.

Paso 2

Active el simulador y observe que la botonera de depuración (Figura 19) presenta activo el botón para la inclusión de puntos de ruptura (botón con punto rojo).



Figura 19: Botonera para depuración.

Para añadir puntos de ruptura, es decir, puntos en los que se podrá detener la ejecución de la simulación, debe abrir los ficheros que está utilizando la simulación e incluir los puntos. Por ejemplo, abra el fichero *inversor_arq1.vhd*, coloque el cursor en la línea 6 y pulse el botón . El aspecto que tendrá la ventana de edición será similar al de la Figura 20. Observe que tras esta acción en la ventana de información *Simulate* aparece un mensaje en el que se indica la línea en que se ha colocado el punto de ruptura.

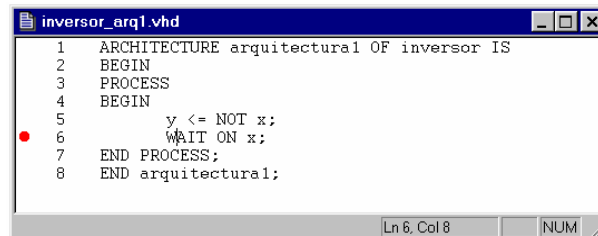


Figura 20: Inclusión de puntos de ruptura.





Paso 3

Antes de iniciar la simulación, abra una ventana de visualización de las señales para que pueda observar cómo la simulación se detiene cuando alcanza el punto de ruptura que se ha fijado en la línea 6 del fichero *inversor_arq1.vhd*. Proceda a ejecutar la simulación tal y como lo ha hecho en las actividades previas.

Observe en la ventana de visualización que la simulación se detiene en el instante de tiempo 5 ns. que es el instante en el que el inversor queda a la espera de que se produzca un cambio en el valor de la señal *x*. Observe también que en la ventana de información de la simulación ha aparecido el mensaje:

dbg NOTE : Debugger interrupt at time 5 ns, cycle 0

Los comandos que dispone para continuar ejecutando la simulación son los que aparecen en el menú *Debug*. Los comandos más interesantes de depuración son los siguientes:

- *Continue*. Continúa la ejecución de la simulación hasta el siguiente punto de ruptura o hasta que concluya el tiempo de simulación. Botón .
- *Step*. Avanza la simulación una línea del código fuente VHDL. Botón .
- *Stop Debugging*. Elimina todos los puntos de ruptura y continúa la ejecución de la simulación hasta el final del tiempo establecido. Botón .
- *Show Breakpoints*. Abre una ventana en la que se muestran todos los puntos de ruptura existentes. Botón .

Observe que en este ejemplo, al ejecutar una única vez el comando *Step* o *Continue* el tiempo de simulación no avanzará. Ello es debido a que el punto de ruptura se ha colocado en la sentencia *WAIT* del inversor pero la etapa que se está simulando consta de 8 inversores. Por ello, observe que se pasa varias veces por el punto de ruptura pero sin que se produzca un avance en el tiempo de simulación. Para que el tiempo avance es necesario pasar por el punto de ruptura tantas veces como la señal cambia de nivel con el objeto de activar la sentencia *WAIT ON* de todos los inversores implicados. Así, para que se alcance el instante 4 ns. es necesario pasar 4 veces por el punto de ruptura debido a que las señales de entrada que cambian son cuatro ("00000000" -> "10101010"), mientras que para alcanzar el instante de tiempo 10 ns. será necesario pasar 8 veces por el punto de ruptura ya que varían las 8 señales de entrada ("10101010" -> "01010101").

Paso 4

Mediante el uso del depurador, también es posible observar la evolución de las señales de forma numérica en una nueva ventana. Para ello debe acudir al comando *New Watch Window* situado en el menú *Debug*. Obtendrá una ventana (Figura 21) en la que dispone de hasta 4 paneles de observación independientes (*Watch1* a *Watch4*).

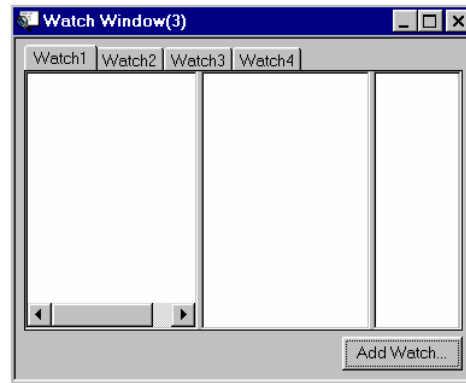


Figura 21: Ventana de observación.

Para añadir a uno de los cuatro paneles los elementos a observar pulse el botón *Add Watch...* y obtendrá una ventana (Figura 22) en la que puede seleccionar tanto las señales de forma independiente (se selecciona la señal y se pulsa el botón *Watch*) como todas las señales de un elemento (se selecciona el componente o bloque en la parte izquierda de la ventana y se pulsa el botón *Watch block*).

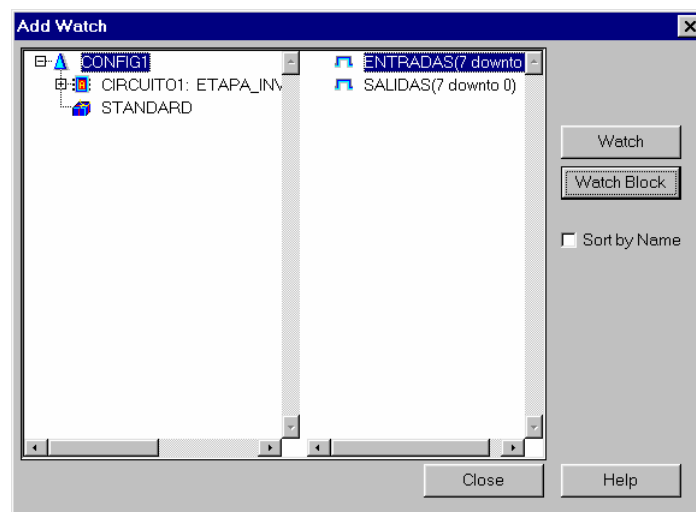



Figura 22: Ventana con la selección de todas las señales del bloque CONFIG1.

Paso 5

Otra función que proporciona el depurador es la visualización de los puntos de ruptura que se han insertado en el código. Para ello debe ejecutar el comando *Show breakpoints...* situado en el menú *Debug* o pulsar el botón  ubicado en la botonera de depuración. En ambos casos obtendrá una ventana, denominada *Breakpoints*, similar a la de la Figura 23 en la que se muestran todos los puntos de ruptura que se han insertado en el código VHDL.

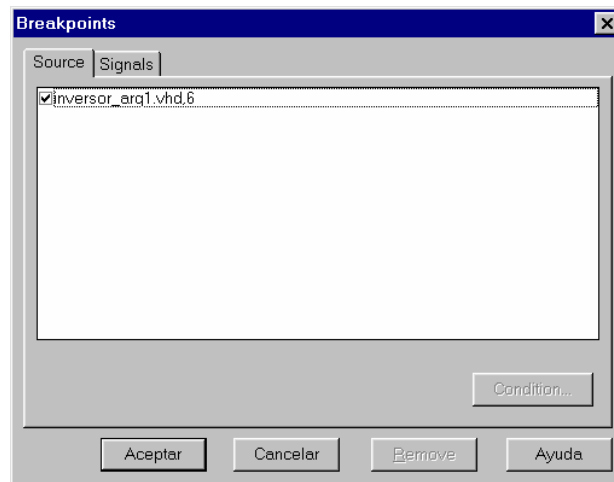


Figura 23: Visualización de los puntos de ruptura.

Seleccionando en la ventana de visualización los puntos de ruptura podrá eliminarlos del código VHDL mediante el botón *Remove*. Otra posibilidad es deshabilitarlos, para lo cual debe suprimir la selección del punto o puntos de ruptura en la ventana *Breakpoints* (Figura 23) y pulsar el botón *Aceptar*. Tras esta acción, si observa el fichero VHDL en el que ha insertado el punto de ruptura que ha deshabilitado, el círculo rojo se habrá transformado en una circunferencia de color rojo indicando que hay un punto de ruptura pero que se encuentra deshabilitado.

Continuando con la manipulación de los puntos de ruptura, observe en la ventana *Breakpoints* la existencia de un botón denominado *Condition...* Si pulsa este botón obtendrá una ventana (Figura 24) en la que puede establecer condiciones de activación para un punto de ruptura determinado.

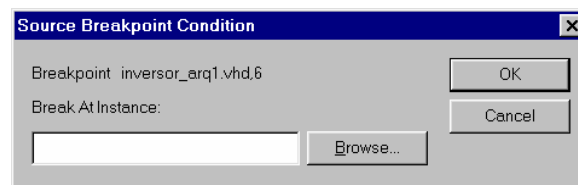




Figura 24: Ventana de establecimiento de condiciones en los puntos de ruptura.

Observe que la Figura 24 muestra la ventana de establecimiento de condiciones para el punto de ruptura que se ha insertado previamente en el Paso 2 (*Breakpoint inversor_arq1.vhd,6*). Desde esta ventana, denominada *Source Breakpoint Condition*, podrá indicar que el punto de ruptura permanezca activo únicamente para una instancia o proceso específico. De esta forma, cuando se establecen condiciones, el simulador se detiene únicamente en el punto de ruptura que hay dentro del proceso especificado, no en todos los procesos en los que está el punto de ruptura.

Por último, si desea eliminar todos los puntos de ruptura que ha insertado, puede recurrir al comando *Clear All Breakpoints*, situado en el menú *Debug*, o pulsar el botón . De esta forma, todos los puntos de ruptura que ha insertado con anterioridad serán eliminados.

Paso 6

Otra opción que posee el depurador es *Call Stack* o pila de llamadas. Mediante ésta es posible conocer la próxima línea de código VHDL que se ejecutará cuando se continúe con la simulación. Para visualizar la *Call Stack*, seleccione el comando *Call Stack* en el menú *Debug* o pulse el botón . Obtendrá una ventana (Figura 24) en la que se muestra en la parte superior la próxima línea a ejecutar. Observe en la Figura 25 la sintaxis que se emplea para indicar la próxima línea que se va a ejecutar.

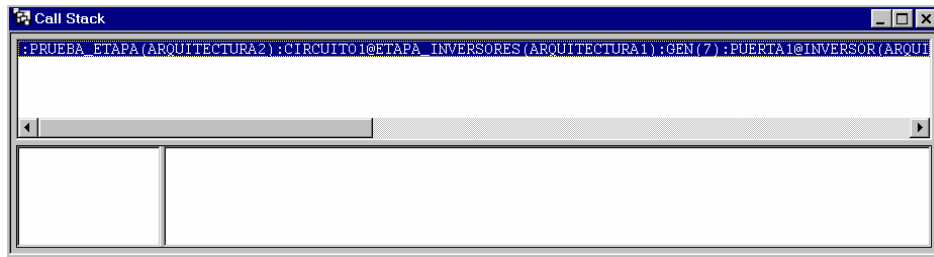


Figura 25: Ventana de la pila de llamadas.

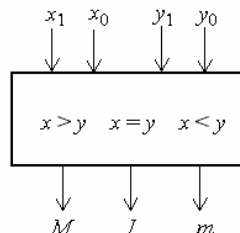
Si se realiza doble clic sobre una de las entradas que aparecen en la *Call Stack*, obtendrá una nueva ventana en la que se muestra el código fuente VHDL que se indica en la entrada que ha seleccionado.

En el caso de que el diseño que está ejecutando contenga subprogramas anidados, en el momento en que se entre a ejecutar un subprograma, la ventana *Call Stack* se incrementará con una nueva entrada correspondiente a la llamada al subprograma. Cuando concluya la ejecución del subprograma, la correspondiente entrada en la ventana será eliminada.

7. ACTIVIDAD MANEJO LENGUAJE (I): CIRCUITO COMBINACIONAL BÁSICO. COMPARADOR DE 2 BITS

El objetivo de esta práctica es simular el funcionamiento de un circuito combinacional utilizando los tres estilos de descripción: algorítmico, flujo de datos y estructural. El circuito que se debe programar y simular es un comparador de 2 bit.

De forma genérica, un comparador de n bits y tres salidas es un circuito lógico que tiene dos entradas (x e y), de n bits cada una, que representan las magnitudes de dos números enteros, y tres salidas M (mayor), I (igual) y m (menor), que indican la relación que existe entre dichas magnitudes (Figura 26).

Figura 26: Esquema de un comparador de 2 bits con tres salidas (M , I , m).

Es posible, sin embargo, simplificar el diseño si se tiene en cuenta que el resultado de la comparación de dos enteros positivos $X = x_{n-1}x_{n-2}...x_0$ e $Y = y_{n-1}y_{n-2}...y_0$ se puede codificar únicamente con dos salidas M y m . De esta forma cuando $M = 1$ entonces $X > Y$ y cuando $M = 0$ entonces $X \leq Y$; al mismo tiempo, si $m = 1$ entonces $X < Y$, y si $m = 0$, entonces $X \geq Y$. Así pues, se sabe que $X \neq Y$ siempre que $M = 1$ ó $m = 1$, e inversamente, que $X = Y$ siempre que $M = 0$ y $m = 0$.

Para determinar estos resultados para enteros X e Y de n bits, el comparador inicia su comparación con los bits menos significativos, calculando M_i y m_i para cada sufijo de X e Y , donde los sufijos i de X e Y se definen como los enteros $x_i x_{i-1}...x_0$ e $y_i y_{i-1}...y_0$ tales que $i \leq n-1$. Estos valores se pueden resumir en las expresiones siguientes:

$$M_i = (x_i > y_i) \text{ ó } ((x_i = y_i) \text{ y } (M_{i-1}))$$

$$m_i = (x_i < y_i) \text{ ó } ((x_i = y_i) \text{ y } (m_{i-1}))$$

Como puede verse, se ha reducido la comparación completa a una comparación de números de dos bits (x_i M_{i-1}) e (y_i m_{i-1}). Se puede diseñar así un comparador básico de dos enteros de dos bits (a_1 a_0) y (b_1 b_0) y utilizarlo para construir comparadores de un número mayor de n bits. En la Figura 27 se muestra la tabla de verdad para un comparador de dos bits cuyas entradas son los dos enteros de dos bits (a_1 a_0) y (b_1 b_0) y sus salidas son M y m . El circuito lógico del comparador se muestra en la Figura 28.

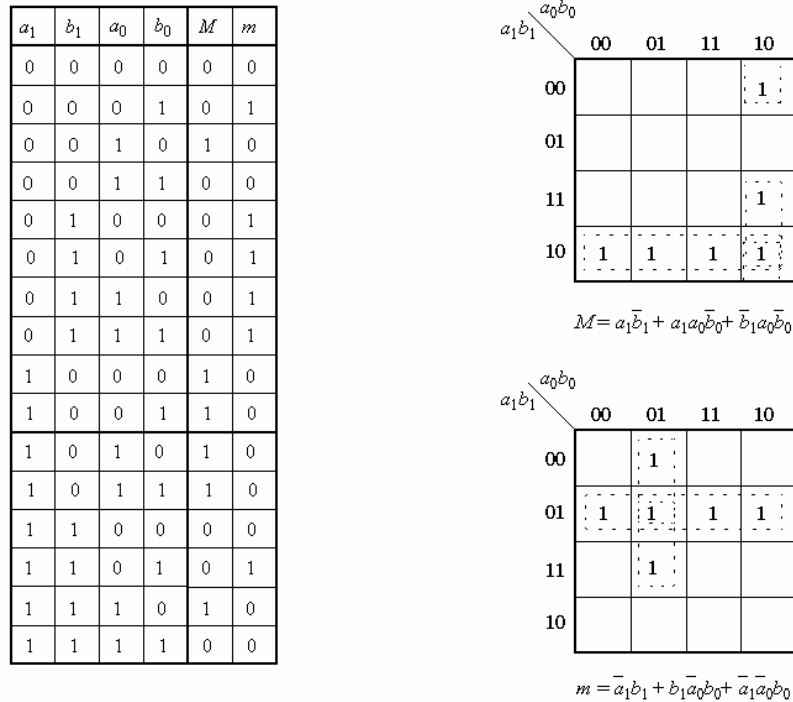


Figura 27: Tabla de verdad y mapas de Karnaugh de un comparador de 2 bits.

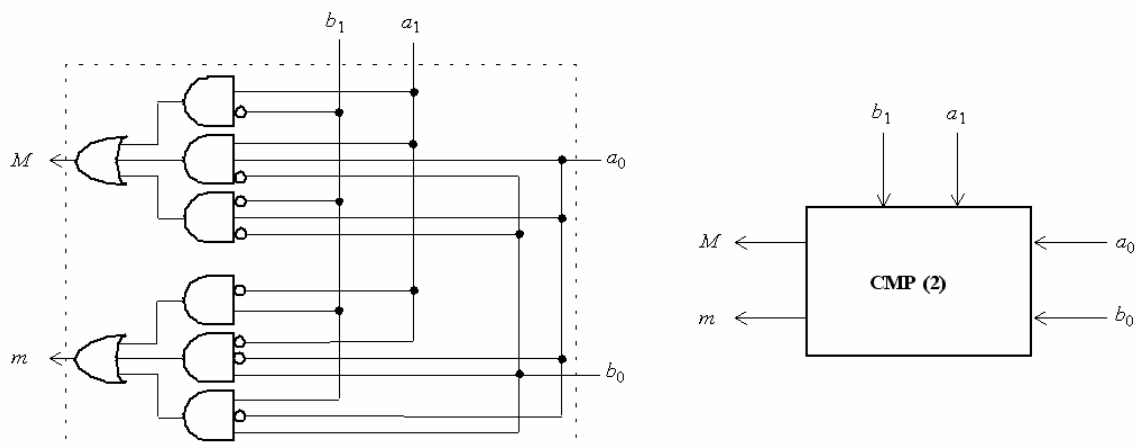


Figura 28: Circuito lógico del comparador de 2 bits CMP (2) con salidas M y m y representación esquemática.

Para facilitar la realización del ejercicio cree una carpeta y un espacio de trabajo que se llamen *practica1* en el que almacene todos los ficheros VHDL que cree para realizar esta actividad práctica.

Paso 1: Definición de la entidad

Escriba una interfaz que represente las 4 entradas y las 2 salidas del comparador de 2 bits y 2 salidas definido anteriormente. El nombre del fichero VHDL debe ser *comp2.vhd*.

Paso 2: Modelo algorítmico

Escriba una arquitectura que refleje el modelo de comportamiento algorítmico del comparador anterior. El nombre del fichero debe ser *comp2_arq1.vhd*.

Escriba también una prueba para comprobar el correcto funcionamiento del comparador con la arquitectura algorítmica. Para ello cree una entidad, una arquitectura y una unidad de configuración en la que se especifique que hay que utilizar el modelo de comportamiento del comparador. Los nombres de los ficheros de la prueba deben ser *testcomp2.vhd* y *testcomp2_arq.vhd* y *testcomp2_conf1.vhd*.

Las combinaciones de X e Y que tiene que introducir en la prueba (fichero *testcomp2_arq.vhd*) deben cubrir todos los casos posibles que se le puedan plantear al comparador, es decir, las señales de salida M y m deben adquirir las cuatro posibles combinaciones. Consulte la tabla de verdad del comparador para plantear los casos de prueba.

Paso 3: Modelo de flujo de datos

Escriba una arquitectura que refleje el modelo de comportamiento de flujo de datos de un comparador de 2 bits. El nombre del fichero debe ser *comp2_arq2.vhd*.

Con el objeto de comprobar que el modelo de comportamiento de flujo de datos es correcto, utilice la prueba definida en la actividad anterior pero con una nueva unidad de configuración. En esta nueva unidad debe especificar que hay que utilizar el modelo de comportamiento de flujo de datos para el comparador. El nombre del fichero VHDL para nueva la unidad de configuración debe ser *testcomp2_conf2.vhd*.

Paso 4: Modelo estructural

Escriba una arquitectura que refleje el modelo estructural de un comparador de 2 bits. El nombre del fichero debe ser *comp2_arq3.vhd*.

Defina para cada puerta, dos ficheros VHDL: uno para la entidad y otro para la arquitectura. Para cada puerta lógica deje abierta la posibilidad de introducir un retardo recurriendo a un parámetro *retardo* de tipo genérico en su declaración de entidad. De acuerdo con esto, y como ejemplo, la declaración de la entidad de un inversor podría ser la que se recoge en el Fichero 12.

```
ENTITY inversor IS
  GENERIC (retardo: TIME);
  PORT(x:IN BIT; y:OUT BIT);
END inversor;
```

Fichero 12: Inversor con retardo genérico.

Con el objeto de comprobar que el modelo de comportamiento de flujo de datos es correcto, utilice la prueba definida en la actividad anterior pero con una nueva unidad de configuración e introduciendo retardos de 2 ns. a todas las puertas lógicas. En esta nueva unidad debe especificar que hay que utilizar el modelo de comportamiento estructural para el comparador. El nombre del fichero VHDL para la nueva unidad de configuración debe ser *testcomp2_conf3.vhd*.

8. ACTIVIDAD MANEJO LENGUAJE (II): CIRCUITO COMBINACIONAL BÁSICO. COMPARADOR DE 8 BITS

El comparador que se ha diseñado en el ejercicio anterior puede utilizarse para implementaciones serie y paralelo de comparadores de n bits que pueden comparar cualquier par de enteros sin signo X e Y . Así, en una implementación serie se utiliza un comparador de 2 bits para cada pareja de bits, representada por x_i e y_i . Para cada comparador de 2 bits, los valores de entrada de x_i e y_i se conectan a las entradas a_i y b_i , mientras que los valores M_{i-1} y m_{i-1} obtenidos de la comparación del sufijo ($i-1$) se conectan a las entradas a_0 y b_0 . De este modo, cada comparador de 2 bits produce las salidas M y m que representan las funciones M_i y m_i .

En la Figura 29 se muestra la implementación de un comparador de 8 bits utilizando los comparadores de 2 bits. Como puede apreciarse, este comparador de 8 bits necesita solo siete comparadores de dos bits ya que se ha utilizado sólo un comparador de dos bits para comparar x_1x_0 e y_1y_0 .

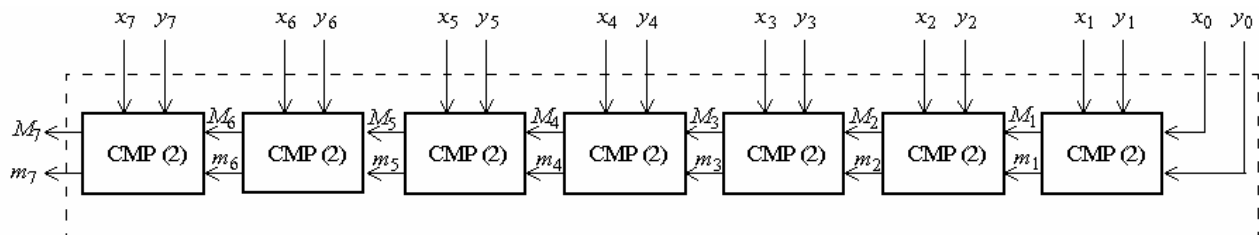


Figura 29: Implementación serie de un comparador de 8 bits con comparadores de 2 bits.

El objetivo de esta segunda práctica es simular el funcionamiento de un comparador de n bits utilizando para ellos los comparadores de 2 bits que diseñó en el ejercicio práctico anterior y la sentencia *GENERATE*. En el Fichero 13 se recoge el esquema de la entidad que debe tener el comparador genérico de n bits.

```
ENTITY comparador_serie IS
  GENERIC (magnitud: natural := 8);
  PORT(x, y: IN bit_vector(magnitud-1 DOWNTO 0);
        M, n: OUT bit);
END comparador_serie;
```

Fichero 13: Entidad de un comparador de 8 bits (fichero comparador_n.vhd).

Observe que, por defecto, el comparador genérico es de 8 bits salvo que se especifique lo contrario al realizar la instanciación del componente mediante el parámetro genérico `magnitud`. El nombre del fichero en el que se realiza la declaración de la entidad del comparador debe ser *comparador_n.vhd*. El nombre del fichero en el que debe introducir la arquitectura estructural del comparador genérico, recurriendo a la sentencia *GENERATE*, y realizar las conexiones adecuadas debe ser *comparador_n_arq.vhd*.

Escriba los correspondientes ficheros de prueba para comprobar que todo es correcto. Los nombres de estos ficheros deben ser, respectivamente, *testcompn.vhd* y *testcompn_arq.vhd*.

Para facilitar la realización del ejercicio cree una carpeta y un espacio de trabajo que se llamen *practica2* en el que almacene todos los ficheros VHDL que desarrolle para realizar esta actividad práctica.

9. ACTIVIDAD MANEJO LENGUAJE (III): GENERADOR PROGRAMABLE DE PULSOS

En esta práctica va a diseñar un generador programable de pulsos. Un generador programable de pulsos permite especificar el retardo que hay entre dos pulsos y la duración o longitud del pulso (Figura 30). Ambas variables, longitud y retardo, se especifican mediante una cantidad numérica que representa ciclos de reloj.

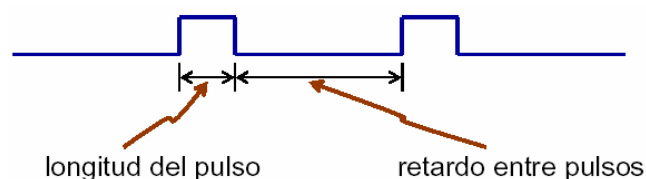


Figura 30: Ejemplo de dos pulsos.

En la Figura 31 se recoge el esquema del circuito generador. La activación de las entradas *LoadDelay* y *LoadLength* permiten indicar, respectivamente, que el valor binario que se coloca en las entradas *Data* debe ser considerado como el retardo entre dos pulsos o la longitud de un pulso. Ya que *Data* es una entrada de 8 bits, la máxima cantidad que se puede especificar es 255. La señal de entrada *reset* permite colocar todos los valores a 0 de forma asíncrona, es decir, con independencia de la señal de reloj. *Clk* es la señal de reloj que establece el sincronismo en todo el circuito y mediante la cual se puede medir la duración del retardo y la longitud del pulso.

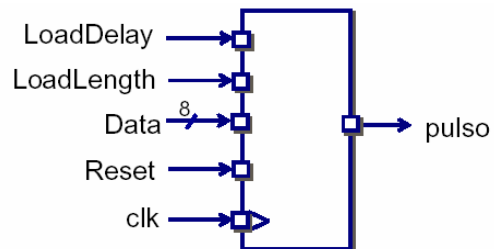


Figura 31: Esquema del generador programable de pulsos.

En la Figura 32 se ilustra con mayor detalle la estructura de bloques del generador de pulsos. Se puede apreciar que consta de dos partes: el *Camino de datos* y el *Control*. El *Camino de datos* es la parte del generador que, en función del valor especificado a través de la entrada *data*, se encarga de emitir el pulso de las características especificadas de acuerdo con las órdenes que le envía el circuito *Control* a través de *ci* y *pi*.

El *Control* es un circuito secuencial que activa las líneas *ci* o *pi* según tenga el *Camino de datos* que dejar pasar el tiempo correspondiente al retardo entre dos ciclos o a la duración del pulso respectivamente. *Z* es una señal que emite un contador del *Camino de datos* y es la que permite determinar cuándo *Control* debe activar la señal *pi* o *ci*.

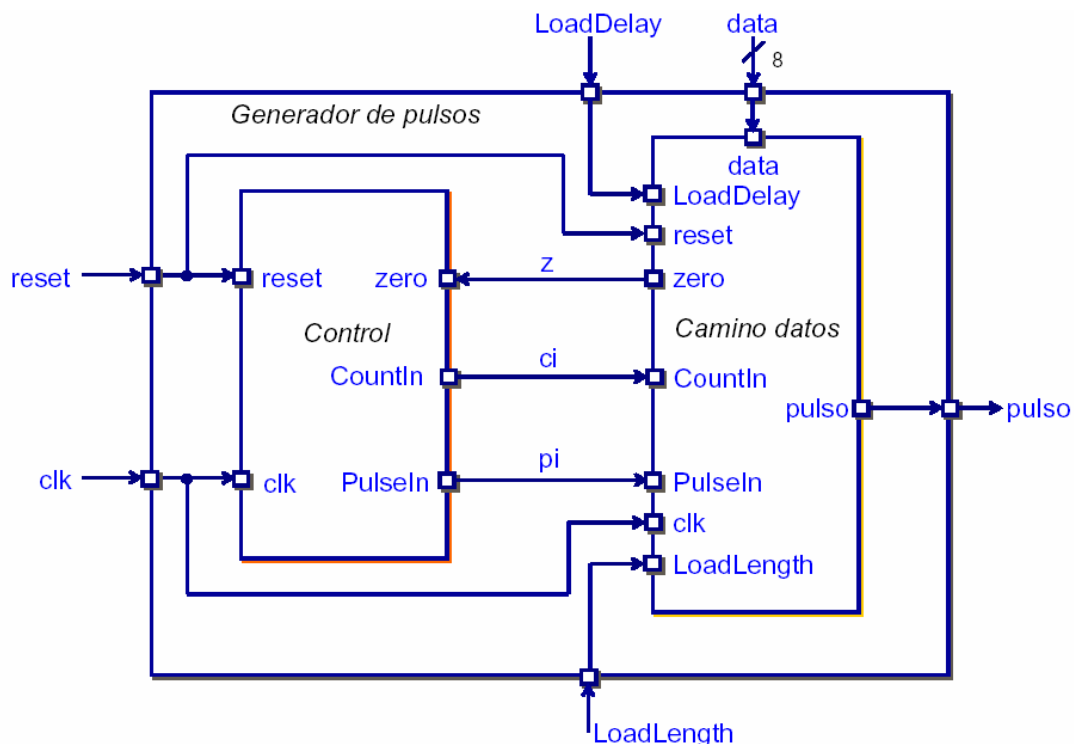


Figura 32: Diagrama de bloques del generador programable de pulsos.

En la Figura 33 se recoge el diagrama de bloques del *Camino de datos*. El dato que indica el valor del retardo o la duración del pulso se almacena en el *Registro 1* o en el *Registro 2* en función de que se active la línea *loadDelay* o *loadLength*. La salida de los registros está conectada a un *multiplexor* de dos entradas que se ocupa de seleccionar con qué valor hay que iniciar el *contador*, que es el que realmente cuenta el tiempo de retardo entre pulsos o la duración del pulso. El *contador* cuenta tantos ciclos de reloj como se le haya indicado a través de la entrada *data*, de forma que una vez que ha concluido la cuenta (ha alcanzado el valor 0) emite un 1 por la salida *zero*.

Obsérvese que la selección de la salida del *multiplexor* conectado al *contador* se realiza en función del valor de la señal *ci* que proviene de *Control*. Es decir, es el circuito *Control* el que determina cuándo hay que empezar a contar el tiempo correspondiente al retardo o a la duración del pulso.

Para que el *Camino de datos* conmute el valor de *pulso* a través del *biestable tipo D* es necesario que se active la señal *pi* con el objeto de seleccionar la entrada a 1 del segundo *multiplexor*. La señal *pi* se activa en función de la información que se envía al *Control* a través de la salida *zero* del *contador*.

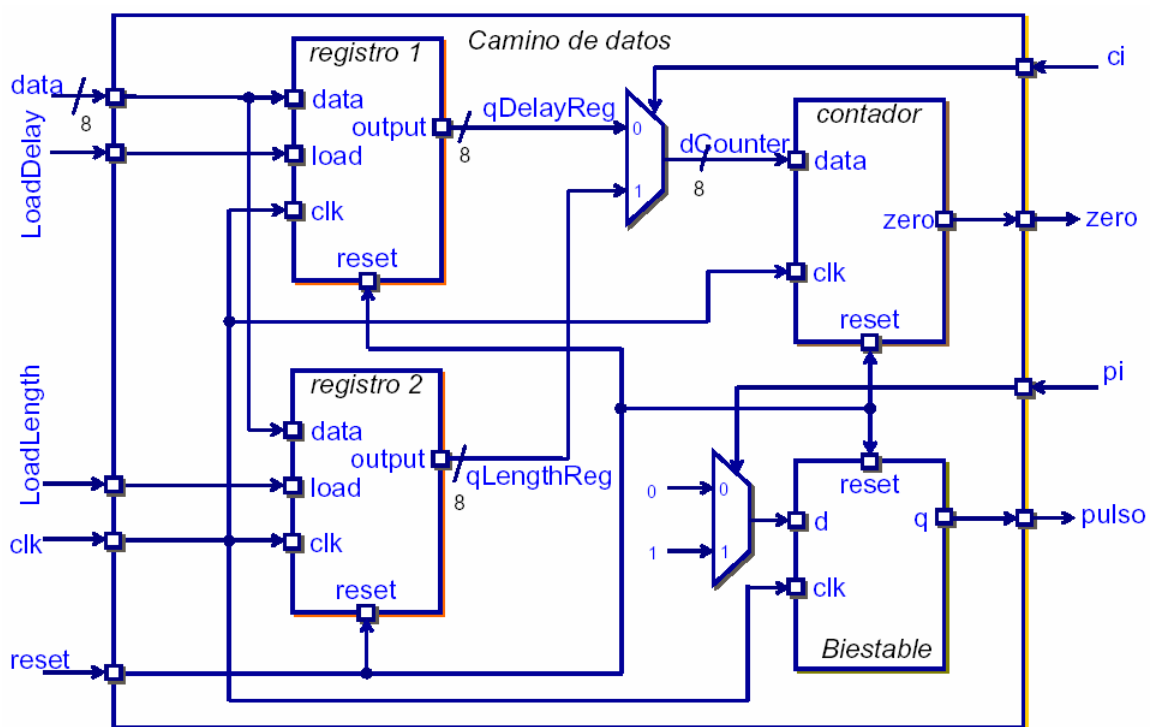


Figura 33: Diagrama de bloques del Camino de datos.

La Figura 34 muestra la máquina de estados finita correspondiente al circuito secuencial *Control*. Obsérvese que el diagrama consta de cuatro estados (S0, S1, S2, S3), aunque es posible realizar una minimización del diagrama y reducirlo a dos estados. El motivo de mantener los cuatro estados es introducir un poco de claridad en el comportamiento del sistema secuencial que se comenta a continuación:

S0: Coloca *ci* a 0 con el objeto de indicar al *contador* que tome como valor a contar la longitud del retardo del pulso. Por ello, el valor de la salida *pi* se mantiene a 0.

S1: Es el estado al que se pasa desde el estado S0 una vez que el *contador* ha comenzado a contar el número de ciclos de retardo entre dos pulsos. Se permanece en este estado hasta que el *contador* coloque la señal *zero* a 0. En ese instante se pasa al estado S2.

S2: Es un estado análogo al S0 pero indica al *contador* que el valor con que debe iniciarse es la longitud del pulso, es decir, se activa la señal *ci* a 1 para que el *multiplexor* envíe el valor de la

duración del pulso al *contador*. Para crear el pulso se activa la señal *pi* a 1 de forma que el *multiplexor* seleccione como entrada del *biestable* la entrada con la señal activa.

S3: Estado análogo al S2. Se permanece en él mientras que el *contador* no haya finalizado de contar el número de ciclos de duración del pulso. En el instante en que el *contador* concluye, se activa la señal *zero* y se pasa de nuevo al estado S0.

Las transiciones directas entre los estados S0 y S2 se producen cuando la duración y retardo del pulso se fijan a 0, lo que produce que el *contador* no inicie nunca la cuenta. De esta forma, la señal de salida del generador de pulsos permanecerá siempre a 0.

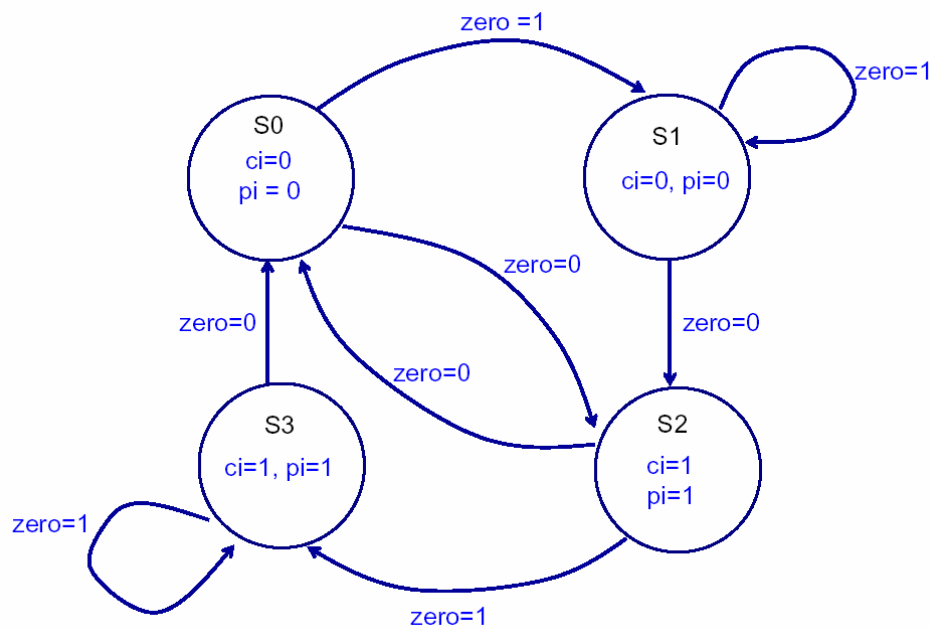


Figura 34: Autómata de Moore correspondiente al circuito secuencial de Control.

La Figura 35 recoge la tabla de estados del circuito secuencial y la Figura 36 la codificación de los estados.

Estado presente	Estado siguiente		Salida	
	<i>zero=0</i>	<i>zero=1</i>	<i>ci</i>	<i>pi</i>
S0	S2	S1	0	0
S1	S2	S1	0	0
S2	S0	S3	1	1
S3	S0	S3	1	1

Figura 35: Tabla de estados del circuito secuencial.

Estado	Código ($y_1 y_0$)
S0	0 0
S1	0 1
S2	1 0
S3	1 1

Figura 36: Codificación de estados.

Estado presente $y_1 y_0$	Estado siguiente		Salida $ci \ pi$
	$zero=0$ $y_1 y_0$	$zero=1$ $y_1 y_0$	
0 0	1 0	0 1	0 0
0 1	1 0	0 1	0 0
1 0	0 0	1 1	1 1
1 1	0 0	1 1	1 1

Figura 37: Tabla de estados del circuito secuencial en binario.

A partir de la tabla de la Figura 37 obtenemos las siguientes ecuaciones lógicas de transición de estado y de salida:

$$\begin{aligned}
 Y_0 &= \text{zero} \\
 Y_1 &= \overline{\text{zero}} \cdot y_1 + \text{zero} \cdot y_1 \\
 ci &= Y_1 \\
 pi &= Y_1
 \end{aligned}$$

Existen varios estilos de descripción explícita de máquinas de estados a nivel algorítmico. Estos estilos se pueden resumir en tres:

Estilo 1: La máquina de estados se describe mediante dos procesos: un proceso secuencial para describir la función del próximo estado y las salidas, y otro proceso secuencial para describir las asignaciones sobre el registro de estados en la transición activa de la señal de reloj (Fichero 14).

Estilo 2: Se recurre a un único proceso secuencial con una sentencia de espera del reloj. En este estilo descriptivo todas las acciones tienen lugar en sincronía con la señal de reloj (Fichero 15).

Estilo 3: La máquina de estados se describe utilizando un único proceso secuencial con las señales de reloj, reset, registros estado y entradas en la lista de sensibilidad del proceso (Fichero 16).

```

ENTITY FSM IS
    PORT (reloj, reset : IN BIT;      -- señales de reset y reloj
          x1, ..., xm : IN BIT;      -- señales de entrada
          z1, ..., zn : OUT BIT);    -- señales de salida
END FSM;

ARCHITECTURE estilo1 OF FSM IS
    TYPE estados IS (s0, s1, ..., sp);
    SIGNAL estado, siguiente_estado: estados;
BEGIN
    secuencial: PROCESS (estado, x1, xm)
    BEGIN
        siguiente_estado <= estado;    -- asignación por defecto
        CASE estado IS
            WHEN s0=>                -- estado s0
                -- acciones del estado s0
            WHEN s1=>                -- estado s1
                -- acciones del estado s1

            WHEN sp=>                -- estado sp
                -- acciones del estado sp
        END CASE;
    END PROCESS secuencial;

    sincronia: PROCESS (reset, reloj)
    BEGIN
        IF (reset = '0') THEN
            estado <= s0;
        ELSIF (reloj'event and reloj='1') THEN
            estado <= siguiente_estado;
        END IF;
    END PROCESS sincronia;
END estilo1;

```

Fichero 14: Descripción genérica de una máquina de estados correspondiente al estilo 1.

```

ENTITY FSM IS
    PORT (reloj, reset : IN BIT;      -- señales de reset y reloj
          x1, ..., xm : IN BIT;      -- señales de entrada
          z1, ..., zn : OUT BIT);    -- señales de salida
END FSM;

ARCHITECTURE estilo2 OF FSM IS
BEGIN
    secuencial: PROCESS
        TYPE estados IS (s0,...,sp);
        VARIABLE estado: estados;
    BEGIN
        WAIT UNTIL reloj='1';
        IF (reset='0') then
            estado:= s0;
        ELSE
            CASE estado IS
                WHEN s0=>  -- estado s0
                           -- acciones del estado s0
                WHEN s1=>  -- estado s1
                           -- acciones del estado s1

                WHEN sp=>  -- estado sp
                           -- acciones del estado sp
            END CASE;
        END IF;
    END PROCESS secuencial;
END estilo2;

```

Fichero 15: Descripción genérica de una máquina de estados correspondiente al estilo 2.

```

ENTITY FSM IS
    PORT (reloj, reset : IN BIT;
          x1, ..., xm : IN BIT;
          z1, ..., zn : OUT BIT);
END FSM;

ARCHITECTURE estilo3 OF FSM IS
    TYPE estados IS (s0,sp);
    SIGNAL estado: estados;
BEGIN
    secuencial: PROCESS (reset, reloj, estado, x1, xm)
    BEGIN
        --
        -- lógica combinacional 1
        --
        IF (reset='0') then
            estado <= s0;
        ELSIF (reloj'event AND reloj='1') THEN
            CASE estado IS
                WHEN s0=>  -- estado s0
                           -- acciones del estado s0
                WHEN s1=>  -- estado s1
                           -- acciones del estado s1

                WHEN sp=>  -- estado sp
                           -- acciones del estado sp
            END CASE;
        END IF;
        --
        -- lógica combinacional 2
        --
    END PROCESS secuencial;
END estilo3;

```

Fichero 16: Descripción genérica de una máquina de estados correspondiente al estilo 3.

El trabajo a realizar en esta práctica es modelar y simular el generador de pulsos siguiendo un modelo estructural análogo al que se recoge en el diagrama de bloques de la Figura 32. Para facilitar la realización del ejercicio cree una carpeta y un espacio de trabajo que se llamen *practica3* en el que almacenará todos los ficheros VHDL que desarrolle para realizar esta última actividad práctica.

A continuación se enumeran algunas especificaciones para el desarrollo del generador de pulsos en lenguaje VHDL:

- Para el *Camino de datos* debe utilizar un modelo de comportamiento estructural similar al de la Figura 33.
- El *contador* se puede construir recurriendo a un modelo de comportamiento algorítmico.
- El *biestable tipo D* se puede modelar recurriendo a un modelo de comportamiento algorítmico.
- El circuito *Control* puede construirse mediante uno de los tres estilos que se han indicado previamente para la construcción de máquinas de estados finitos.
- La señal debe ser *reset* es asíncrona y activa a nivel bajo. Las restantes señales son activas a nivel alto.
- Cuando se realiza un *reset* la máquina de estados se inicia en el estado S1 y los registros a 0.
- No es necesario considerar retardos en los circuitos.

De acuerdo con lo indicado, los ficheros VHDL que deberá desarrollar en la práctica serán, como mínimo, los siguientes:

- Entidad y arquitectura de comportamiento algorítmico un multiplexor.
- Entidad y arquitectura comportamiento algorítmico de un registro.
- Entidad y arquitectura de comportamiento algorítmico un biestable tipo D.
- Entidad y arquitectura de comportamiento algorítmico de un contador.
- Entidad y arquitectura de comportamiento algorítmico del circuito *Control*.
- Entidad y arquitectura estructural del circuito *Camino de datos*.
- Entidad y arquitectura estructural del circuito *Generador de pulsos*.

A continuación se muestran la declaración de la entidad para el generador de pulsos que se debe utilizar, así como la entidad y la arquitectura para realizar las pruebas de todo el conjunto.

```
ENTITY generador_pulsos IS
  PORT(loadLength, loadDelay, reset, clock :IN BIT;
        data: IN bit_vector(7 DOWNTO 0);
        pulso: OUT BIT);
END generador_pulsos;
```

Fichero 17: Entidad para el generador de pulsos.

```
ENTITY prueba_generador IS
END prueba_generador;

ARCHITECTURE arquitectura OF prueba_generador IS

  COMPONENT generador
    PORT (loadLength, loadDelay, reset, clock: IN BIT;
```

```
        data: IN bit_vector (7 DOWNTO 0);
        pulso: OUT BIT);
END COMPONENT;

FOR ALL : generador USE ENTITY WORK.generador_pulsos(estructural);

CONSTANT duracion_ciclo: TIME:= 20 ns;
SIGNAL CLK_s: BIT:='0';
SIGNAL longitud_s, duracion_s: BIT := '0';
SIGNAL datos: bit_vector (7 DOWNTO 0) := "00000000";
SIGNAL pulso: BIT;
SIGNAL reset_s: BIT := '1';

BEGIN

    generador1: generador PORT MAP (longitud_s, duracion_s, reset_s, CLK_s, datos, pulso);

    reloj: PROCESS
    BEGIN
        WAIT FOR duracion_ciclo;
        CLK_s <= '1';
        WAIT FOR duracion_ciclo;
        CLK_s <= '0';
    END PROCESS reloj;

    pruebas: PROCESS
    BEGIN
        --
        -- reset asíncrono
        -- asignar data
        -- asignar duración del retardo entre pulsos
        -- asignar longitud del pulso
        -- esperar un tiempo para cambiar valores de entrada
        -- nuevos valores y pruebas
        --
        WAIT;
    END PROCESS pruebas;
END arquitectura;
```

Fichero 18: Entidad y arquitectura de prueba para el generador.