

1. INTRODUCCIÓN

Este documento contiene los enunciados de las **prácticas obligatorias** 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 (archivo *vhdl_sim.zip*, 21,1 Mbytes), lo tiene disponible en tres sitios:

- En el CD-ROM *Asignaturas* que se le proporciona junto con la Guía de Curso de la ETSI Informática.
- En el servidor WWW de la Escuela: <http://www.ii.uned.es/cdrom/2005-06/mat-doc-1-ciclo/asig/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.

La evaluación de la asignatura se efectúa tras la realización de dos pruebas obligatorias, una presencial de tipo teórico y otra de carácter práctico (la que está leyendo en este instante). Para superar la asignatura deberán aprobarse cada una de las partes. La prueba teórica se califica de 0 a 10, mientras que la prueba práctica se calificará únicamente como APTA o NO APTA.

En caso de superar ambas pruebas, la nota final de la asignatura será la obtenida en la prueba presencial. Si se ha obtenido la calificación de NO APTO en la parte práctica, la nota final de la asignatura será NO APTO (4.0) con independencia de la calificación obtenida en la prueba teórica.

A los alumnos que tengan superada la práctica realizada en la prueba presencial de junio se les conservará la calificación hasta la prueba presencial de septiembre del curso actual, nunca para cursos posteriores. Análogamente sucederá con la nota de la prueba presencial.

Es requisito imprescindible entregar la práctica para poder optar a realizar el examen.

Las prácticas se realizan individualmente, no se aceptan grupos de trabajo. La detección de una práctica copiada obligará al equipo docente a ponerlo en conocimiento del Servicio de Inspección de la UNED para la apertura de un expediente informativo.

Existe un grupo de actividades de carácter voluntario y otro grupo de actividades obligatorias. Las primeras tienen por finalidad que practique con el entorno de simulación VHDL para que aprenda los conceptos básicos del mismo. Las actividades de carácter voluntario no tienen que ser enviadas para su evaluación. Las actividades obligatorias, como su nombre da a entender, son de obligada realización y entrega.

Hay que entregar TODAS las actividades de carácter obligatorio para proceder a la evaluación de la práctica. La falta de realización de una de las actividades obligatorias implica la calificación de NO APTO. Todas los programa VHDL deben entregarse compilados y probados en el entorno de simulación que se recomienda en la asignatura.

Se puede utilizar el entorno de simulación VHDL que se desee, ya sea Windows o Linux. En cualquier caso, los programas se probarán en el simulador VHDL referenciado en esta memoria. Por ello, la práctica deberá estar compilada y probada en ese entorno.

Lea detalladamente el resto de la memoria ya que se le indica claramente la forma de realizar la entrega de la práctica y los plazos de que dispone para ello. Para cualquier aclaración, no dude en ponerse en contacto con los miembros del equipo docente de la asignatura utilizando los medios que se indican en la Guía de Curso o en la página WWW.

2. FORMATO DE LA MEMORIA

La memoria debe constar de los siguientes apartados:

- Portada con nombre, dirección, número de DNI, teléfono y correo electrónico.
- Memoria descriptiva en la que se debe explicar el trabajo realizado en cada una de las actividades, bien sean de carácter voluntario u obligatorio. La descripción de cada una de las actividades debe incluir el listado de los ficheros VHDL y, si procede, pantallas del simulador en el que se visualicen las pruebas realizadas con los resultados. No olvide redactar las conclusiones, opiniones y mejoras relacionadas con la práctica.
- Acompañando a la memoria debe ir un disquete o CD-ROM en el que se encuentren los ficheros con el código VHDL organizados tal y como se indica en cada ejercicio con el objeto de facilitar la evaluación por parte del equipo docente.

Las hojas deben ser DIN-A4, con número de página centrado en el pie de página, y grapadas en la esquina superior izquierda. No se aceptarán memorias manuscritas con el fin de evitar problemas de legibilidad y comprensión.

3. FORMAS DE ENTREGA

Enviar por correo electrónico a la siguiente dirección:

etc3@dia.uned.es

Se deberá enviar un único archivo comprimido en formato ZIP y libre de virus. Si se procede a esta forma de entrega, se recomienda que el documento relativo a la memoria se encuentre en formato PDF (extensión .pdf).

También se puede remitir todo el material por correo postal a la dirección

José Sánchez Moreno (Práctica E.T.C. III Curso 2003-04)
Departamento de Informática y Automática
Escuela Técnica Superior de Ingeniería en Informática
UNED
C/. Juan del Rosal nº 16, 5ª planta
28040 MADRID

4. PLAZOS DE PRESENTACIÓN

Prueba presencial de junio: Hasta el día 21 de mayo.

Prueba presencial de septiembre: Desde el 1 de julio hasta el 1 de septiembre.

5. 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.

6. ACTIVIDAD VOLUNTARIA 1: INTRODUCCIÓN AL 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*.

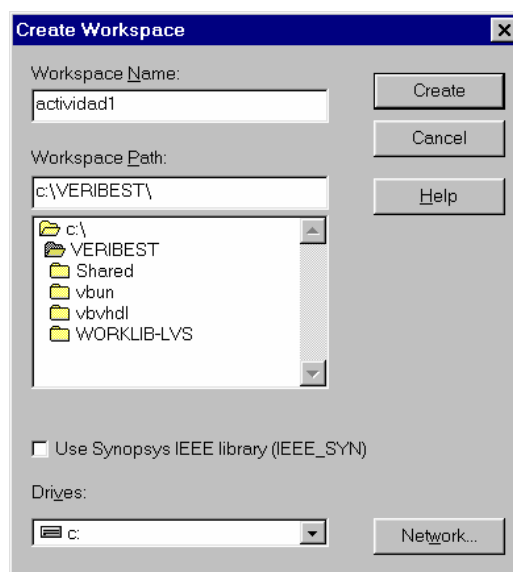


Figura 1: Creación de un espacio de trabajo.

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).

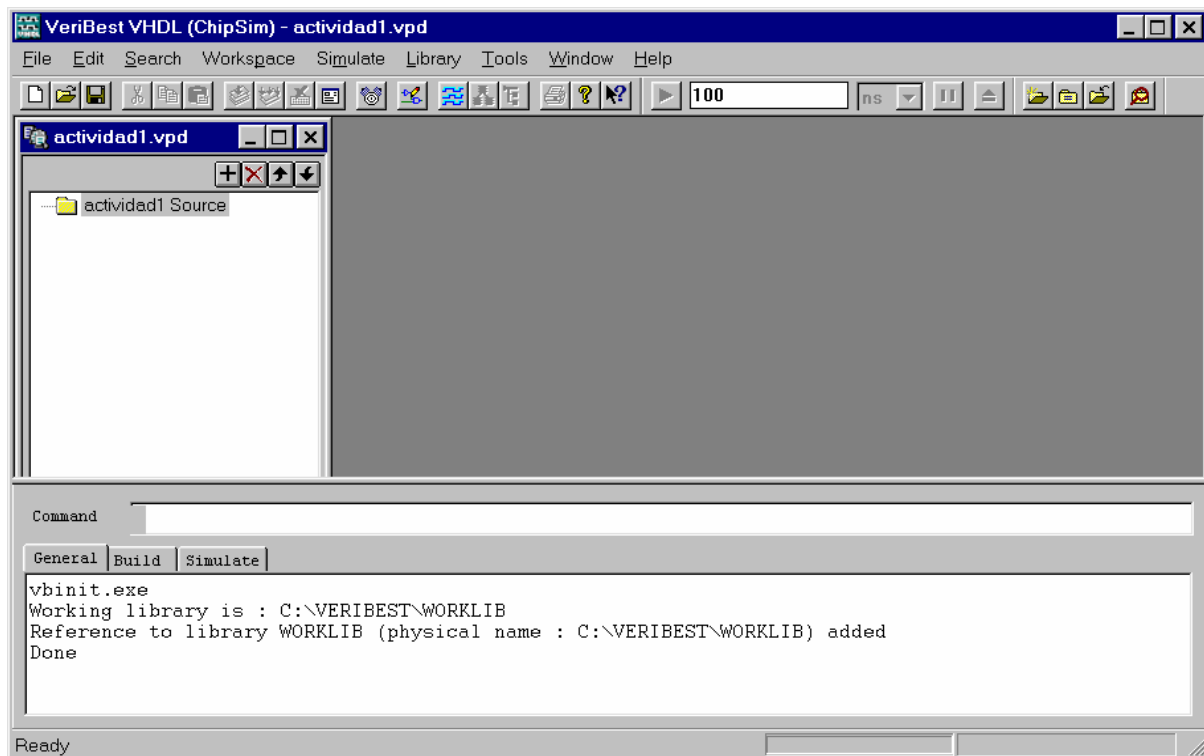


Figura 2: Entorno de simulación con el espacio de trabajo actividad1 ya creado.

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.

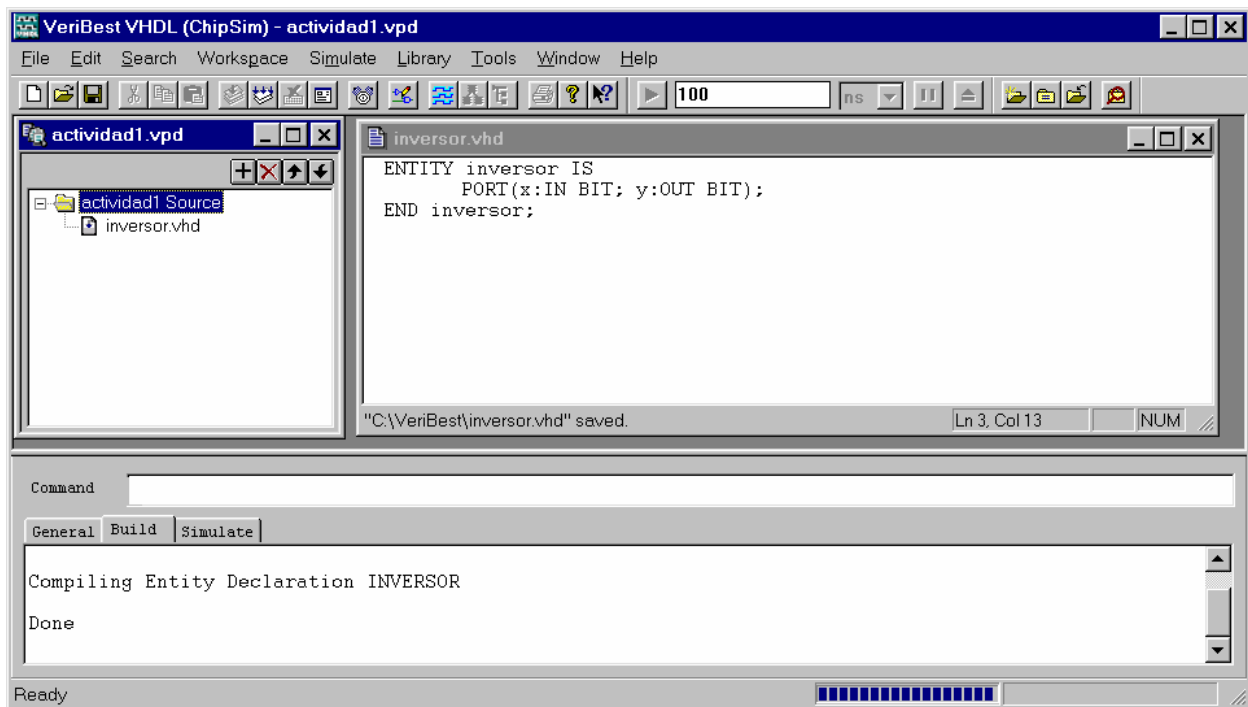


Figura 3: Área de trabajo con el fichero inversor.vhd.

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 inversor. Este código se detalla en el fichero *inversor_arq1.vhd*.

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

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 arquitectural OF prueba_inversor IS

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

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

  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 arquitectural;
```

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*.

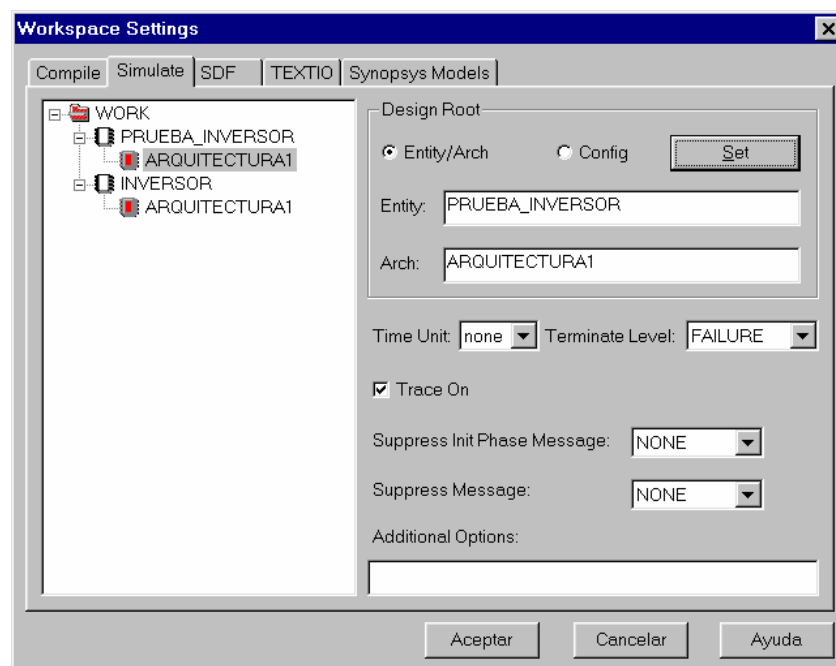



Figura 4: Opciones de simulación.

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.

```
vbvhdl4ive -d PRUEBA_INVERSOR ARQUITECTURA1 -tr on -i -dbg
VeriBest VHDL Simulator Version 15.00.00.25.
Starting Simulation ... Tue Feb 10 11:15:08 2004

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 ... Tue Feb 10 11:15:12 2004

-- 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.

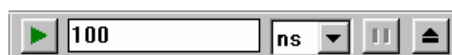



Figura 7: Cuadro para fijación de la duración de la simulación.

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.

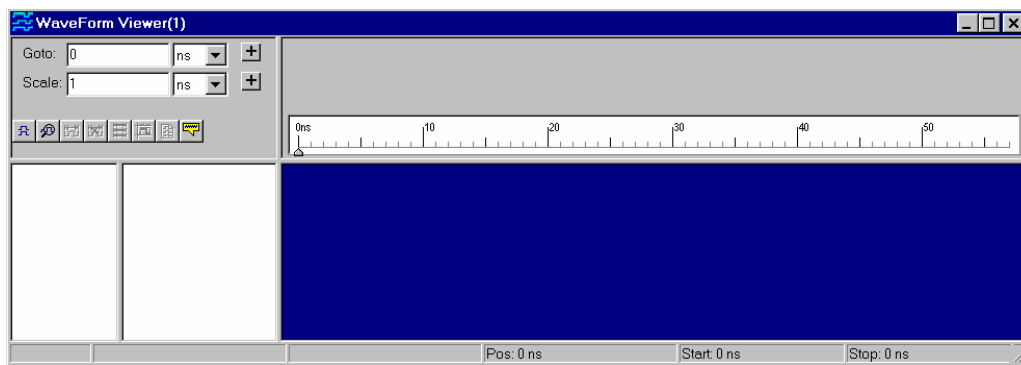



Figura 8: Ventana de visualización de señales.

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.

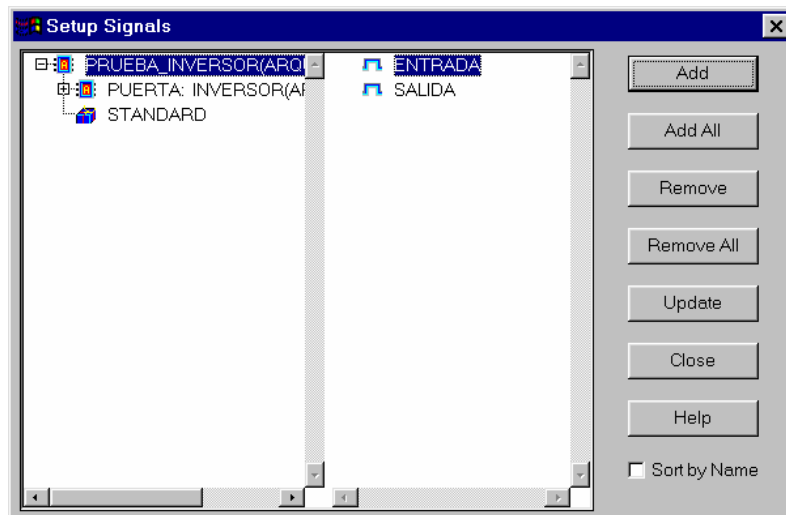


Figura 9: Selección de señales a visualizar.

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.

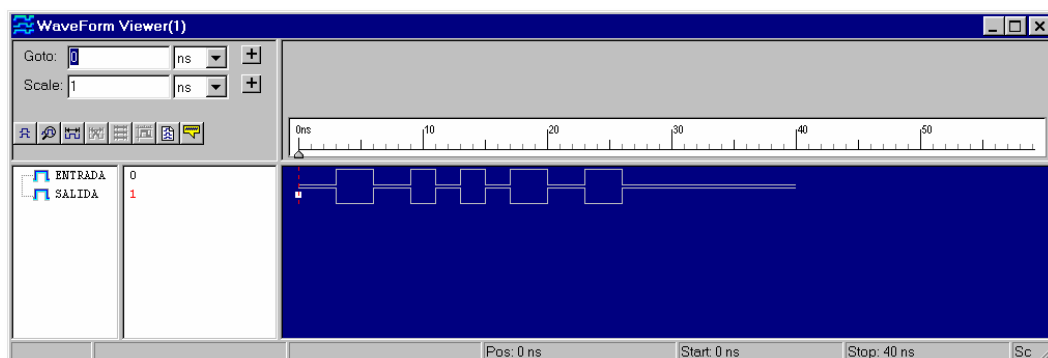


Figura 10: Resultado de la simulación.

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.

7. ACTIVIDAD VOLUNTARIA 2: INTRODUCCIÓN AL 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.

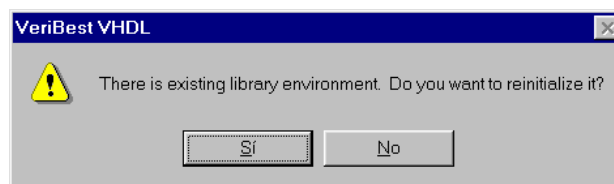


Figura 11: Reinicialización de la librería.

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.

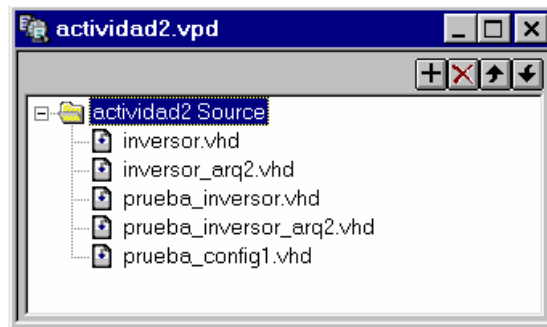


Figura 12: Espacio de trabajo de la actividad 2.

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*.

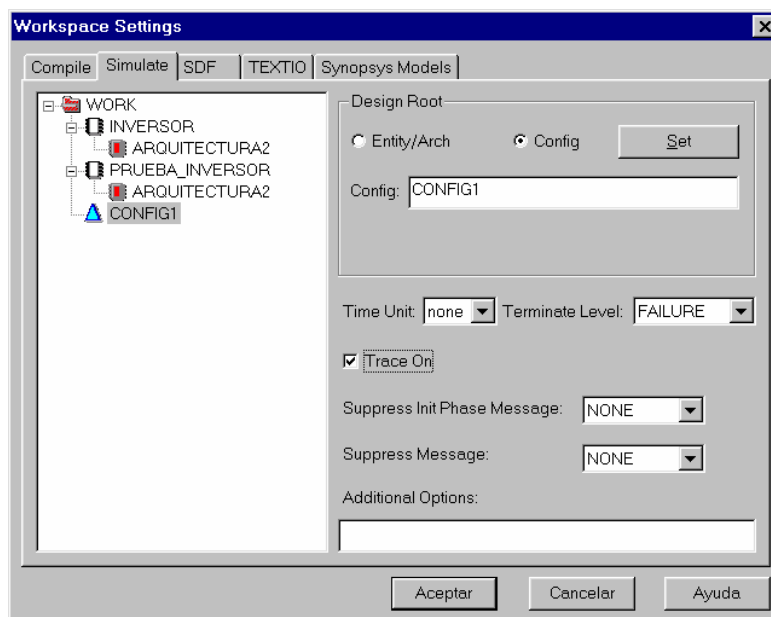


Figura 13: Condiciones de simulación de la actividad 2.

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.

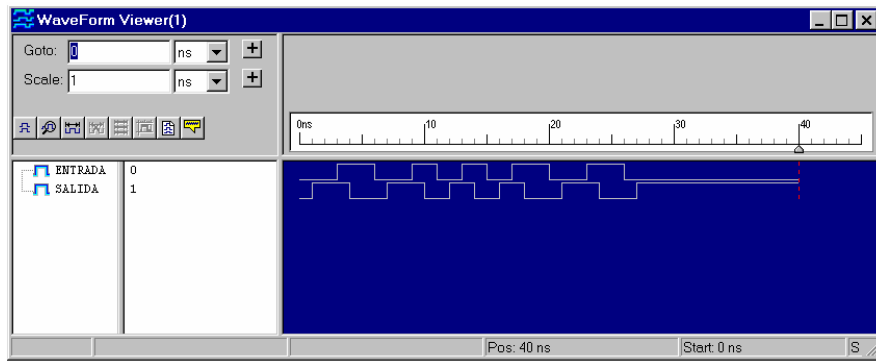


Figura 14: Resultado de la simulación con un retardo de 1 ns.

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.

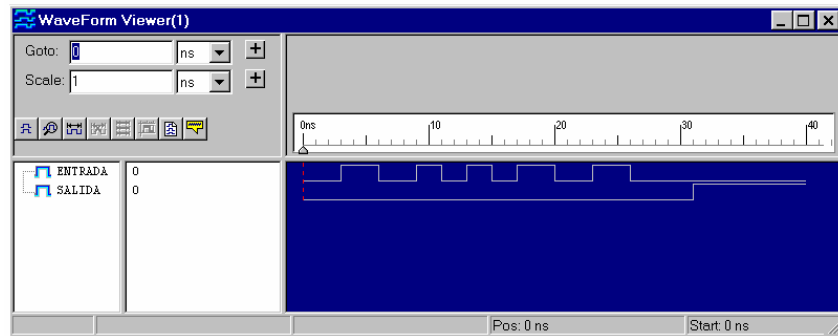


Figura 15: Resultado de la simulación con un retardo inercial de 5 ns.

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.

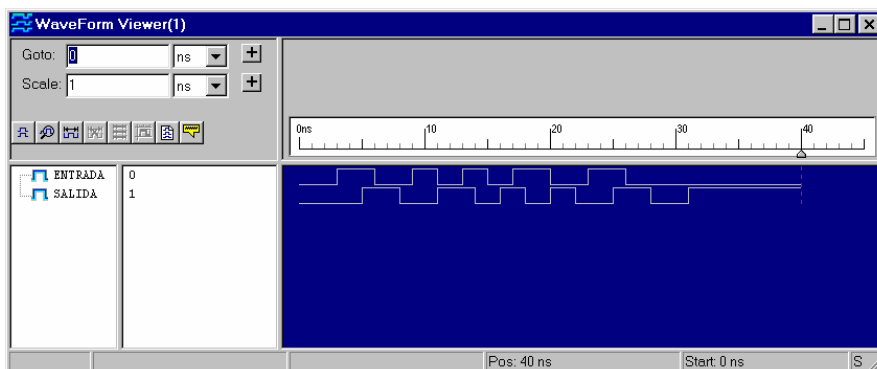


Figura 16: Resultado de la simulación con un retardo de transporte de 5 ns.

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. Reinicialice 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.

8. ACTIVIDAD VOLUNTARIA 3: INTRODUCCIÓN AL 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  : OUT 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 arquitectural 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(arquitectural);

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.

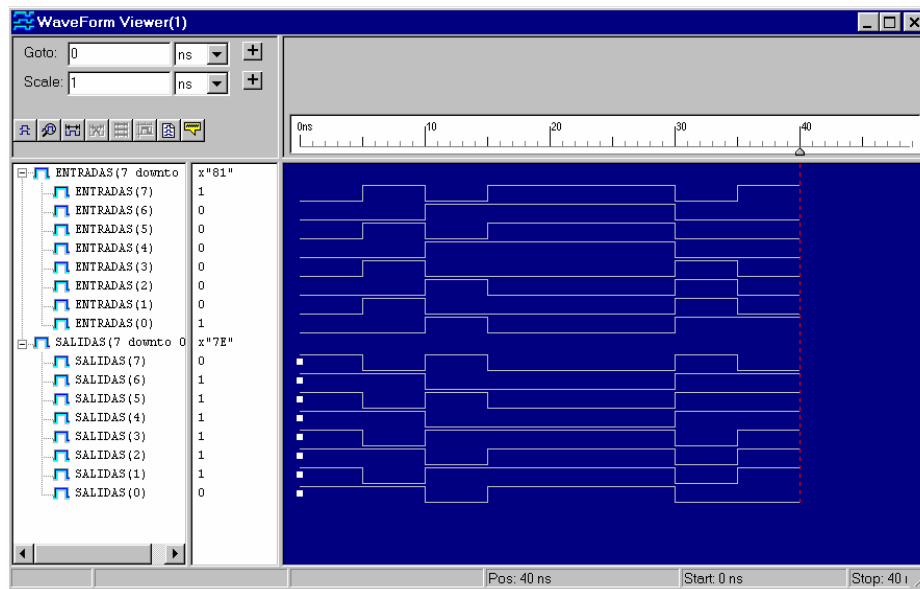


Figura 17: Resultado de la simulación de la etapa de 8 inversores.

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.

9. ACTIVIDAD VOLUNTARIA 4: Uso del depurador

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).

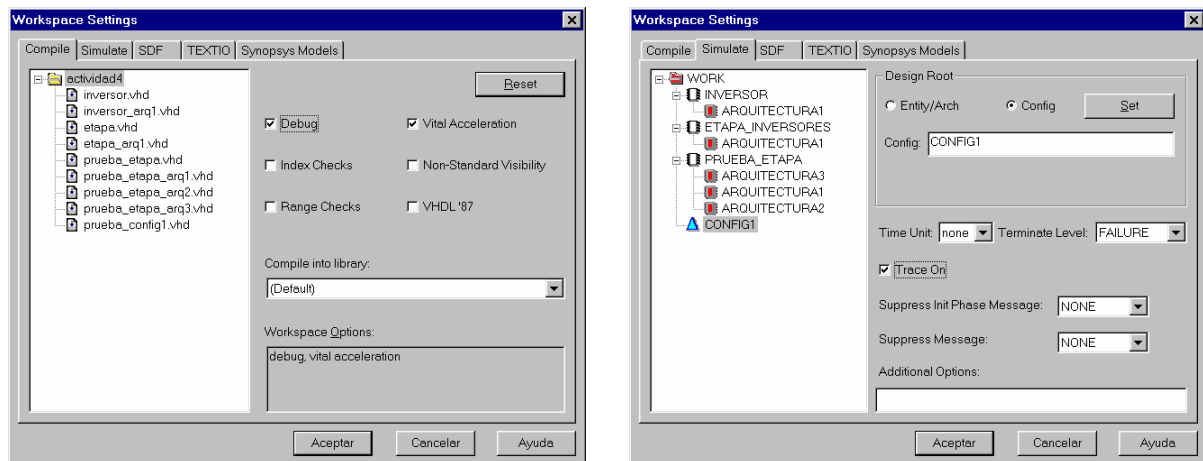


Figura 18: Opciones de compilación y de simulación.

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.

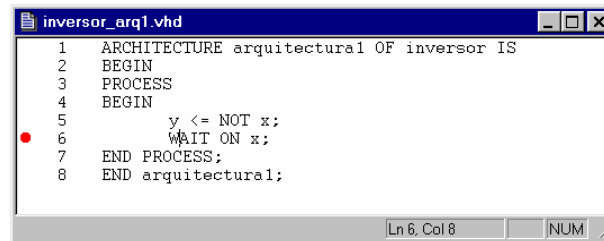


Figura 20: Inclusión de puntos de ruptura.





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*).

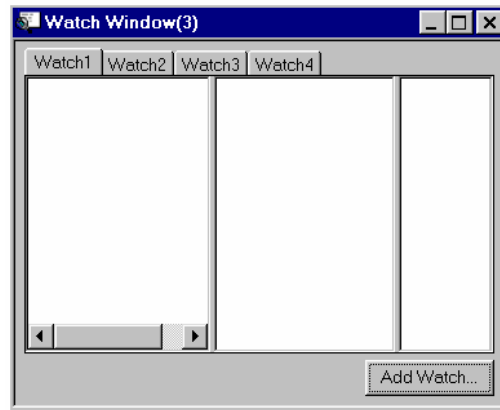


Figura 21: Ventana de observación.

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*).

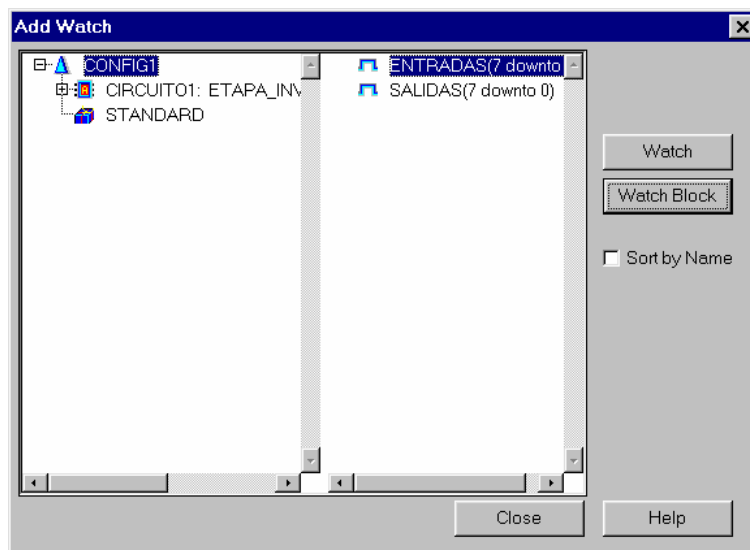



Figura 22: Ventana con la selección de todas las señales del bloque CONFIG1.

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.

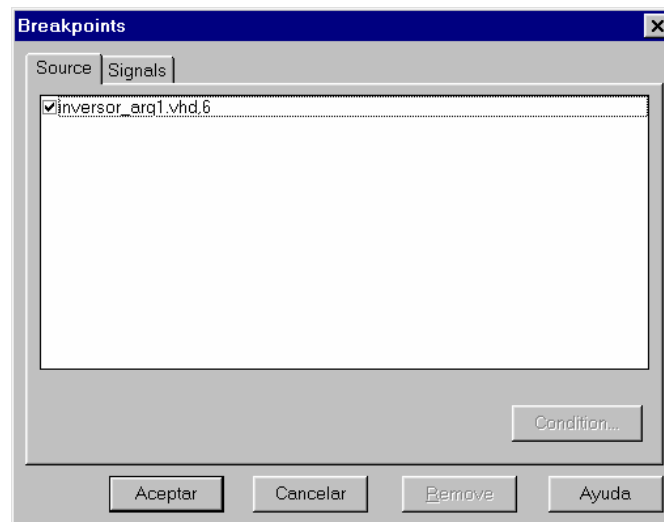


Figura 23: Visualización de los puntos de ruptura.

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.

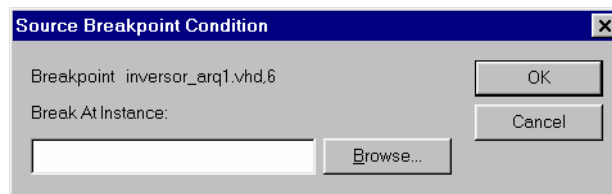




Figura 24: Ventana de establecimiento de condiciones en los puntos de ruptura.

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.

10. PRÁCTICA OBLIGATORIA 1: DISEÑO ALGORÍTMICO DE UN BIESTABLE J-K

El objetivo de esta primera práctica es simular en VHDL el funcionamiento de un biestable J-K utilizando la información que le proporciona la Figura 26. Para la programación del biestable recurra a una arquitectura de comportamiento algorítmico. Recuerde que la activación de la señal *CLR* lleva al biestable al estado RESET, es decir, $Q = 0$ y $\bar{Q} = 1$ de forma asíncrona, es decir, con independencia de la señal de reloj.

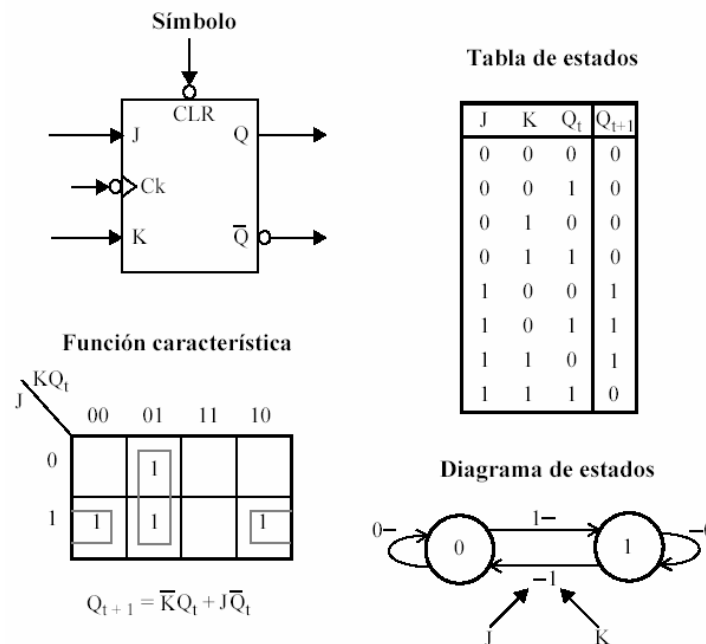


Figura 26: Biestable J-K.

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

Paso 1: Definición de la entidad

Escriba una interfaz que represente las entradas y salidas del biestable J-K descrito anteriormente. El tipo de datos tiene que ser `std_logic` tanto para las entradas (*J*, *K*, *CLR*, *Clk*) como para las salidas (*Qs*).

El nombre del fichero VHDL debe ser *biestableJK.vhd*.

Paso 2: Modelo algorítmico

Escriba una arquitectura que refleje el modelo de comportamiento algorítmico del biestable J-K. El nombre del fichero debe ser *bioestableJK_arq.vhd*.

Paso 3: Prueba del circuito

Desarrolle una prueba para comprobar el correcto funcionamiento de la arquitectura algorítmica del biestable. La prueba debe contemplar las entradas que se presentan en la tabla de estados de la Figura 26 y la entrada *CLR*. Para ello cree una entidad y una arquitectura, siendo el nombre del fichero de la prueba *testBiestableJK.vhd*.

En la Figura 27 se muestra un ejemplo de fichero VHDL que puede utilizar para la elaboración de una prueba para circuitos de tipo secuencial.

```

1 ENTITY prueba_generica IS
2   END prueba_generica;
3
4 ARCHITECTURE arquitectura OF prueba_generica IS
5   -- Declaración del componente a probar.
6   -- Especificación de configuración, salvo que recurra a declaración de configur.
7
8   CONSTANT duracion_ciclo: TIME:= 1 us;
9   SIGNAL CLK_s: STD_LOGIC:='0';
10
11   -- Declaración de señales de entrada y salida del componente a probar.
12
13 BEGIN
14   -- Instanciación del componente a probar.
15
16   reloj: PROCESS
17   BEGIN
18     WAIT FOR duracion_ciclo;
19     CLK_s <= '1';
20     WAIT FOR duracion_ciclo;
21     CLK_s <= '0';
22   END PROCESS reloj;
23
24   pruebas: PROCESS
25   BEGIN
26     --
27     -- Asignación de valores a las señales de entrada del componente a probar.
28     --
29     WAIT;
30   END PROCESS pruebas;
31 END arquitectura;

```

Figura 27: Ejemplo de entidad y arquitectura de prueba para un circuito secuencial.

11. PRÁCTICA OBLIGATORIA 2: DISEÑO ESTRUCTURAL DE UN CONTADOR ASÍNCRONO

El objetivo de esta segunda práctica es simular en VHDL el funcionamiento de un contador asíncrono de 3 bits módulo-5 cuyo diseño estructural se presenta en la Figura 28. Este contador se caracteriza por disponer únicamente de 5 estados distintos (000, 001, 010, 011 y 100). Para su construcción debe utilizar *obligatoriamente* el biestable J-K que diseñó en la práctica anterior.

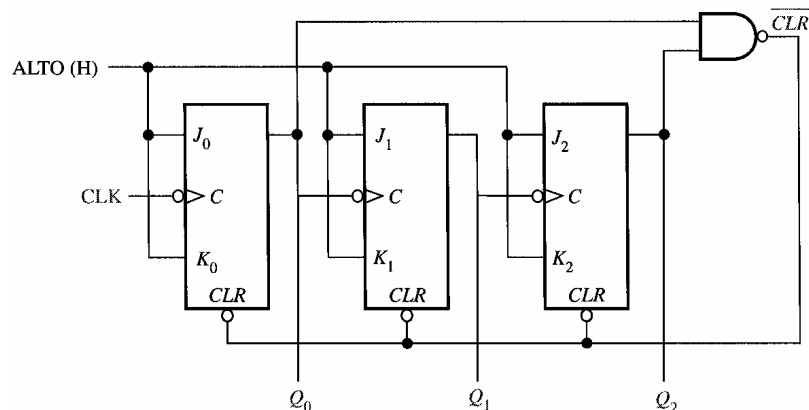


Figura 28: Contador asíncrono de 3 bits modulo-5.

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

Paso 1: Definición de la entidad

Escriba una interfaz VHDL para el contador asíncrono de 3 bits módulo-5. Los tipos de los datos a utilizar son `std_logic` para la única señal de entrada *CLK* y `std_logic_vector` para el vector de salida *Q*. El nombre del fichero VHDL debe ser *contador_mod5.vhd*.

Paso 2: Modelo estructural

Escriba una arquitectura que refleje el modelo estructural del contador asíncrono de 3 bits de acuerdo con el esquema de la Figura 28. Recuerde que tiene que utilizar el biestable J-K del apartado anterior y diseñar la puerta NAND necesaria para completar la arquitectura.

El nombre del fichero debe ser *contador_mod5_arq1.vhd*.

Paso 3: Prueba del circuito

Escriba el correspondiente fichero de prueba para comprobar que todo es correcto. El nombre del fichero debe ser *testcontador_mod5.vhd*.

12. PRÁCTICA OBLIGATORIA 3: DISEÑO ESTRUCTURAL DE UN DETECTOR DE ERROR EN SERIE

En esta tercera práctica deberá diseñar y simular en VHDL el funcionamiento de un detector de error en serie utilizando los elementos que ha desarrollado en los dos ejercicios previos.

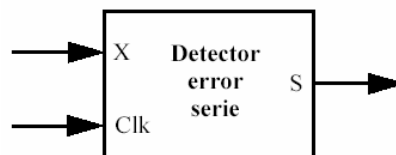


Figura 29: Circuito detector de error en serie.

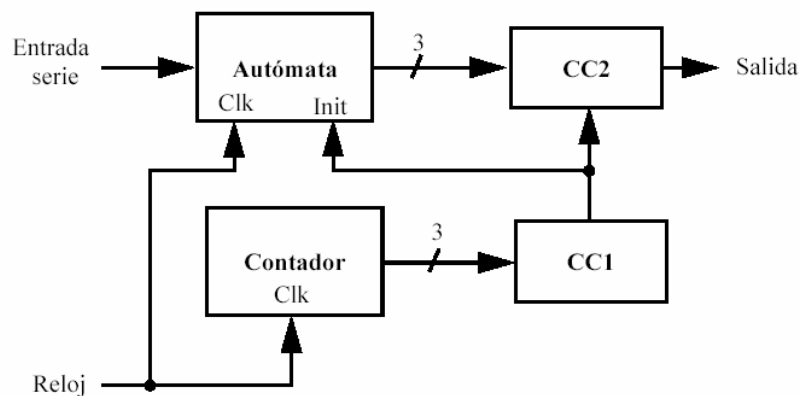


Figura 30: Diagrama lógico simplificado del detector de error en serie.

Las figuras 29 y 30 representan el circuito detector serie y su diagrama lógico simplificado. Los bits que componen una palabra entran por *X* al autómata. Al finalizar una palabra, la salida *S* dará como resultado 1 si hay un error ó 0 si no lo hay. Las palabras de 4 bits que entran en serie por *X* serán erróneas cuando el número de 1s que contengan sea distinto de dos.

Para facilitar la corrección del ejercicio cree una carpeta y un espacio de trabajo que se llamen *practica3* en el que almacene todos los ficheros VHDL que desarrolle en esta actividad.

Paso 1: Definición de la entidad

Escriba la interfaz VHDL para el circuito detector de error en serie. Los tipos de la señal de entrada y de reloj deben ser `std_logic`. La salida debe ser de tipo `std_logic`. El nombre del fichero VHDL debe ser `detector_serie.vhd`.

1	ENTITY detector_serie IS
2	PORT(X, clock : IN STD_LOGIC;
3	S : OUT STD_LOGIC;
4	END detector_serie;

Figura 31: Entidad para el detector.

Paso 2: Modelo estructural

Realice la arquitectura del circuito para el detector serie recurriendo a un diseño estructural. Como consecuencia de utilizar este tipo de diseño, deberá diseñar y crear previamente los cuatro componentes principales que aparecen en la Figura 30: autómata, contador, CC1 y CC2.

A continuación se enumeran algunas especificaciones que debe tener en consideración para el desarrollo del detector de error en serie:

- La función del autómata es detectar el número de 1s que contiene cada palabra, dando como resultado un número binario comprendido entre 0 y 4.
- El número de estados del autómata de Mealy es cuatro. Debe contar con una entrada de inicialización asíncrona para llevar al autómata al estado E0 cuando ha concluido el número de bits de cada palabra (al final de los cuatro bits). Al inicializar cada palabra el autómata se encontrará en el estado E0.
- Debe diseñar el autómata de dos formas:
 1. Mediante un modelo de comportamiento algorítmico como el que se muestra en la Figura 33.
 2. Mediante un diseño estructural formado por biestables tipo D y un circuito combinacional (información en la pág. 162 del texto base de la asignatura). El biestable tipo D debe construirlo reutilizando el biestable J-K que creó previamente. La Figura 32 le muestra cómo realizarlo.

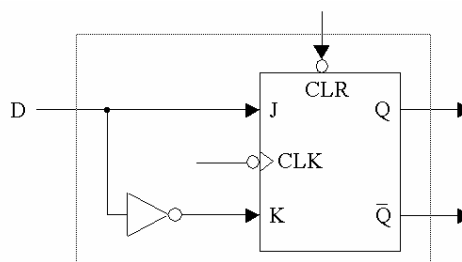


Figura 32: Entidad para el detector.

- El contador proporciona la señal de inicialización asíncrona cuando se han contado 4 pulsos de reloj y, por lo tanto, se han analizado los 4 bits de la palabra de entrada.
- El CC1 (Circuito Combinacional 1) sirve para la inicialización asíncrona del autómata (salida CC1 = 0) y para validar la salida del CC2. Cuando la salida de CC1 es 1, el CC2 queda bloqueado.

- El CC2 produce como salida S=1 exclusivamente al final de cada palabra y sólo si ésta es errónea.
- Las salidas del autómata y del contador son de tipo `std_logic_vector`.
- El nombre del fichero VHDL que debe utilizar para la arquitectura del detector es *detector_serie_arq.vhd*.
- Los nombres de los ficheros VHDL que debe utilizar para las entidades y arquitecturas del autómata y los dos circuitos secuenciales son los siguientes:
 1. Para el autómata: *automata.vhd* y *automata_arq.vhd*. Incluya en el mismo fichero el código VHDL correspondiente a las dos arquitecturas con los nombres *arq_algoritmica* (construido siguiendo el prototipo de la Figura 33) y *arq_estructural*.
 2. Para el circuito combinacional 1: *cc1.vhd* y *cc1_arq.vhd*.
 3. Para el circuito combinacional 2: *cc2.vhd* y *cc2_arq.vhd*.
- No es necesario considerar retardos en los circuitos.
- Cualquier consideración que necesite ser realizada debe especificarse en la memoria. Todo es aceptado siempre que se razone adecuadamente y sea consecuente con el problema que se plantea en la práctica.

```

ENTITY Mealy IS
  PORT (reloj, reset : IN bit;      -- señales de reset y reloj
        entrada  : IN tipo_in;    -- señales de entrada
        salida   : OUT tipo_out);  -- señales de salida
END Mealy;

ARCHITECTURE arq_algoritmica OF Mealy IS
  TYPE estados IS (E0, E1, ..., En);
  SIGNAL estado_actual: estados := E0;
BEGIN

  estado: PROCESS (reset, reloj)
  BEGIN
    IF (reset = '1') THEN estado_actual <= E0;
    ELSIF reloj='1' AND reloj'event THEN
      CASE estado_actual IS
        WHEN E0=>      -- estado E0
                      -- acciones del estado E0
        WHEN E1=>      -- estado E1
                      -- acciones del estado E1
        .....
        WHEN En=>      -- estado En
                      -- acciones del estado En
      END CASE;
    END IF;
  END PROCESS estado;

  salida: PROCESS (entrada, estado_actual)
  BEGIN
    CASE estado_actual IS
      WHEN E0=>      -- estado E0
                    -- salida del estado E0
      WHEN E1=>      -- estado E1
                    -- salida del estado E1
      .....
      WHEN En=>      -- estado En
                    -- salida del estado En
    END CASE;
  END PROCESS salida;
END arq_algoritmica;

```

Figura 33: Descripción genérica de una máquina de estados síncrona de Mealy.

Paso 3: Prueba del circuito

Escriba el correspondiente fichero de prueba para comprobar que el circuito es correcto con independencia del tipo de arquitectura que se utilice para simular el autómata. El nombre del fichero debe ser *test_detector.vhd*.

A continuación se muestran la entidad y la arquitectura para realizar las pruebas de todo el conjunto.

```
1  ENTITY prueba_detector IS
2  END prueba_detector;
3
4
5  ARCHITECTURE arquitectura OF prueba_detector IS
6
7      COMPONENT detector
8          PORT(X, clock : IN std_logic;
9               S : OUT std_logic;
10         END COMPONENT;
11
12      FOR ALL : detector USE ENTITY WORK.detector_serie(estructural);
13
14      CONSTANT duracion_ciclo: time:= 1 us;
15      SIGNAL CLK_s: std_logic:='0';
16      SIGNAL entrada_s: std_logic:='0';
17      SIGNAL salida_s: std_logic;
18
19  BEGIN
20
21      detector1: detector PORT MAP (entrada_s, CLK_s, salida_s);
22
23      reloj: PROCESS
24      BEGIN
25          WAIT FOR duracion_ciclo;
26          CLK_s <= '1';
27          WAIT FOR duracion_ciclo;
28          CLK_s <= '0';
29      END PROCESS reloj;
30
31      pruebas: PROCESS
32      BEGIN
33          --
34          -- asignar valores a la entrada teniendo en cuenta la sincronía
35          -- con la señal de reloj
36          -- nuevos valores y pruebas
37          --
38          WAIT;
39      END PROCESS pruebas;
40  END arquitectura;
```

Figura 34: Entidad y arquitectura de prueba para el detector de error serie.