E.T.S.I. Telecomunicación
Departamento de Ingeniería Electrónica

Diseño de Sistemas Electrónicos Digitales (DSED)

ENUNCIADO DE LA PRACTICA I
(Ejercicios 1 a 3)

Capítulo 1

Introducción al VHDL y a las herramientas de simulación y síntesis

Los objetivos de esta práctica, en la que se realizarán seis ejemplos sencillos, son dos:

1. Descubrir las construcciones básicas del lenguaje de descripción *hardware* VHDL.
2. Conocer y manejar las utilidades básicas de las herramientas de simulación (*ISim*) y síntesis e implementación (*ISE*).

1.1. Ejemplo 1: Lógica combinacional

Como ejemplo de especificación de lógica combinacional vamos a realizar un circuito muy sencillo: un multiplexor de 2 señales (`a`, `b`) a una (`z`). Se selecciona qué entrada pasa a la salida mediante la señal (`sel`). En la figura 1.1 se representa el mismo.

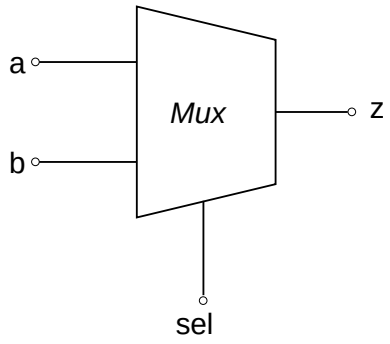

Figura 1.1: Multiplexor 2 a 1

La `entity` correspondiente será:

```
ENTITY mi_mux IS
  PORT(a: in std_logic;
       b: in std_logic;
       sel: in std_logic;
       z: out std_logic);
END;
```

La descripción de la funcionalidad del multiplexor mediante VHDL se puede realizar a diferentes niveles de abstracción. Para ello, utilizaremos varias arquitecturas que reflejen los distintos estilos de descripción.

En primer lugar, vamos a describir el multiplexor a nivel de *flujo de datos*:

```
ARCHITECTURE dataflow OF mi_mux IS
BEGIN
    z <= a WHEN sel = '0' ELSE b;
END dataflow;
```

Todo el código que vamos a utilizar está dentro del directorio `VHDL/PracticaI`. En los ficheros `mux_dataflow.vhd` y `tb_mux.vhd` aparecen la descripción del multiplexor y unos estímulos que se le pueden aplicar, respectivamente. Seguiremos el convenio de llamar al fichero de estímulos (*testbench*) comenzando con el sufijo `tb_`.

1.1.1. Ejercicio 1.1

En el presente ejercicio vamos a trabajar con el simulador ISim, con objeto de familiarizarnos con su flujo de diseño y posibilidades. Comienza iniciando el entorno de desarrollo Xilinx ISE 12.4 con el icono correspondiente y observa la ventana que aparece.

Crea un nuevo proyecto mediante el botón `New Project` o la opción de menú `File` → `New Project`. Rellena el formulario que aparece indicando un nombre de proyecto, por ejemplo `mux`, y un directorio donde contenerlo. Haz click en `Next`.

En el siguiente formulario es necesario escoger el dispositivo destino para el desarrollo que se va a realizar. Rellena como se indica a continuación:

Product Category: General Purpose

Family: Spartan6

Device: XC6SLX25

Package: FTG256

Speed: -3

Después, se muestran varios parámetros sobre el sistema de desarrollo. Comprueba los siguientes valores:

Simulator: ISim

Preferred Language: VHDL

Deja los demás como están, haz click en `Next` y después en `Finish` si la información indicada es correcta.

Una vez creado el proyecto, puedes incorporar los archivos fuente con el botón derecho en el panel `Hierarchy` o mediante la opción `Project` del menú. En ambos casos, selecciona `Add Copy of Source` para copiar los ficheros al directorio del proyecto. Busca entonces la carpeta `PracticaI` y selecciona `mux_dataflow.vhd` y `tb_mux.vhd`.

En la ventana que aparece a continuación se indican las *asociaciones* de los ficheros fuente. Observa las posibilidades y elige `Simulation` para el testbench. Después, pulsa `OK`.

Ahora, encima del panel `Hierarchy`, elige `View: Simulation` y, en el desplegable que aparece debajo, observa las posibilidades y escoge `Behavioral`.

En `Hierarchy` se muestran las entidades que constituyen el proyecto en forma de jerarquía, que puede desplegarse pinchando en los signos `+`. Por otro lado, haciendo doble click en las entidades se abren en un editor sus correspondientes ficheros fuente. Hazlo ahora para familiarizarte con el código. Recuerda que en VHDL no se hace diferencia entre mayúsculas y minúsculas.

De forma general, seleccionando con un click los objetos del panel `Hierarchy`, en `Processes` se muestran las acciones disponibles para cada uno. Pincha en el testbench y después en el `+` de `ISim Simulator`. Ahora, un doble click (o también botón derecho seguido de `Run`) en `Simulate Behavioral Model` realiza una compilación de los fuentes a simular mostrando información sobre lo que está sucediendo en el panel `Console`. Después se abre la ventana del simulador ISim (si se produce algún aviso del antivirus espera a que termine).

Como puedes apreciar, consta de varios paneles. En particular, el *panel de ondas* muestra de forma gráfica los cambios en señales preseleccionadas, que por defecto son todas las señales disponibles en la entidad donde se ha iniciado la simulación (el testbench). Por otro lado, el panel `Console`, similar al del ISE, proporciona información y permite también introducir comandos para accionar al simulador.

Por defecto, el ISim ha realizado ya una simulación de 1 us, aunque esto puede modificarse previamente en el ISE, panel `Processes`, botón derecho sobre `Simulate Behavioral Model`, y después `Process Properties`. Para ver mejor las formas de onda obtenidas pincha en el icono `Zoom to Full View`. Después, puedes arrastrar el cursor amarillo de la derecha hasta los 0 ns y pinchar en el icono `Zoom In` varias veces.

Pregunta 1.1.1 *Analiza las formas de onda. ¿Se comporta el multiplexor como esperabas?*

Cierra el ISim y vuelve al ISE. Edita el fichero de estímulos para comentar los `primeros estímulos` y descomentar los `segundos estímulos`. Puedes seleccionar texto con el ratón y emplear el botón derecho para comentar o descomentar grupos de líneas. También existen aceleradores de teclado para estas tareas tan habituales.

Estudia el fichero modificado para comprender lo que hace y después lanza de nuevo el simulador y emplea los iconos del panel de ondas para visualizar los primeros 100 ns.

Pregunta 1.1.2 *Comprueba que los resultados corresponden al testbench modificado. Indica cuál es el cambio más significativo respecto a la simulación anterior.*

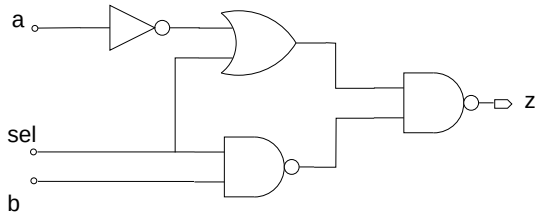

Figura 1.2: Multiplexor 2 a 1: implementación estructural mediante puertas

1.1.2. Ejercicio 1.2

Una segunda aproximación para construir el multiplexor consiste en definir un conjunto de puertas básicas y utilizarlas para una descripción *estructural*. En la figura 1.2 se ve un posible circuito. La arquitectura correspondiente en VHDL se encuentra en el fichero `mux_estructural.vhd`, que emplearemos enseguida.

Vuelve al ISE (recuerda cerrar el ISim) y crea un nuevo proyecto, de nombre `mux2`, en la misma carpeta que el anterior (`mux`). Después, añade con copia el fichero `mux_estructural.vhd` desde `PracticaI`. A continuación, añade, esta vez sin copia (`Add Source`) y desde la carpeta del proyecto, porque ya está allí del ejercicio anterior, el testbench `tb_mux.vhd`. Acuérdate de cambiar su asociación a `Simulation`. Finalmente, selecciona `View: Simulation`, debajo `Behavioral`, y ejecuta una simulación (acuérdate de marcar previamente el testbench en el panel `Hierarchy`).

Pregunta 1.2.1 *Se producen errores. ¿Cuál crees que es la causa? (Cuando hay varios errores debes fíjarte siempre en el primero de ellos.)*

Como hemos visto hasta ahora, en el lado izquierdo del ISE aparecen los paneles `Hierarchy` y `Processes`, para lo cual es necesario que la pestaña seleccionada debajo del último sea `Design`. Por su parte, la pestaña `Files` muestra una lista con todos los ficheros disponibles en el proyecto, en tanto que `Libraries` permite navegar por las librerías disponibles.

Por defecto se crea siempre la librería `work`, que es donde se depositan los componentes del proyecto en desarrollo. Echa un vistazo a su contenido. Sin embargo, en muchas ocasiones es conveniente agrupar componentes en librerías adicionales para facilitar el desarrollo.

Para construir la librería que necesitamos en este momento, selecciona Project → New VHDL Library en el menú del ISE. Aparece una ventana con dos columnas que permite introducir varias librerías a la vez. En este caso sólo necesitamos una, de modo que, en la primera fila escribe a la izquierda el nombre de la misma, `teach_logic_lib`, y a la derecha busca el directorio donde están sus ficheros fuente, `VHDL/teach_logic_lib`. Pulsa `OK`. La siguiente ventana, ya conocida, permite seleccionar asociaciones de manera individual para los ficheros de la librería. Ahora, prueba de nuevo a simular el multiplexor.

Pregunta 1.2.2 *¿Se obtiene el mismo resultado que en 1.1.2?*

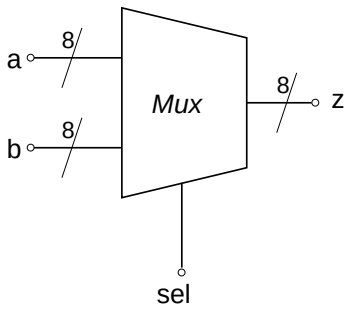

Figura 1.3: Multiplexor 2 a 1 de 8 entradas

1.1.3. Ejercicio 1.3

Ahora vamos a describir un multiplexor 2 a 1 con entradas de 8 bits, como el de la figura 1.3. Para ello, utilizaremos los vectores de bits predefinidos en la librería `ieee.std_logic_1164`.

Crea un nuevo proyecto para este ejercicio, por ejemplo `mux8`. Puede estar en el mismo directorio que los anteriores. Incluye los ficheros `mux2a1_de8.vhd` y `tb_mux2a1_de8.vhd` y echa un vistazo al código de ambos. Contienen lo que se denomina descripción a nivel de *comportamiento* o *behavioral*. Simula este nuevo multiplexor.

Para ver con más claridad los resultados del panel de ondas haz click con el botón derecho del ratón en la pestaña inferior y selecciona `Float`. Después maximiza la ventana y amplia los primeros 50 ns.

Pregunta 1.3.1 *¿Qué ha cambiado en esta simulación respecto a las anteriores? Comenta las diferencias que observes.*

Pregunta 1.3.2 *¿Se te ocurre cómo construir un multiplexor como éste tomando como base el multiplexor anterior de 1 bit? Escribe el código y pruébalo en simulación.*

Pregunta 1.3.3 *Realiza el código y los estímulos de un multiplexor 4 a 1 de líneas de 8 bits. Haz la simulación para comprobar su funcionamiento.*

1.2. Ejemplo 2: Decodificador 3 a 8

Un ejemplo un poco más complejo de lógica combinacional es un decodificador de 3 a 8, cuyo código está contenido en el fichero `decoder.vhd`.

1.2.1. Ejercicio 2.1

Volviendo al ISE, crea un proyecto nuevo, ahora en una carpeta diferente a la del multiplexor para mantener las cosas en orden. Incluye el decodificador y su testbench.

Selecciona `View: Implementation` y después `dec3to8` dentro de `Hierarchy`. Los tres cuadraditos que tiene a la izquierda la entidad señalada indican que es el *top* del diseño. El top no puede ser nunca un testbench, de modo que, si lo fuera, habríamos cometido un error al insertar los archivos fuente en el proyecto y sería necesario revisar las propiedades del testbench (botón derecho). Por otro lado, es posible cambiar la entidad designada como top señalando una nueva y empleando también el botón derecho (`Set as Top Module`).

Ejecuta el proceso `Synthesize-XST` → `Check Syntax`. El resultado obtenido debería ser correcto. Después, inicia la síntesis del diseño con doble click en `Synthesize-XST`.

Pregunta 2.1.1 *¿Qué sucede? ¿Cuál es la causa?*

Prueba a añadir como última línea dentro del `CASE` la siguiente:

```
WHEN others => z <= "XXXXXXXX";
```

Repite la síntesis.

Pregunta 2.1.2 *¿Es correcto ahora? No deja de ser curioso el mensaje que entrega el ISE en este caso. ¿Por qué crees que se hace esta indicación, aparentemente contradictoria repecto al error de la pregunta anterior?*

Quizá la condición errónea de la Pregunta 2.1.1 debería ser mejor advertida por el análisis sintáctico y no por la síntesis.

A continuación, realiza una simulación de comportamiento. En el panel de ondas, es posible cambiar la representación de las señales haciendo click en su nombre con el botón derecho. Por ejemplo, cambia la rotulación de la señal de entrada `sel` seleccionando `Radix` y después `Unsigned Decimal`. Así es más sencillo comprobar si el bit activado en la salida `z` es el correcto.

Pregunta 2.1.3 *¿Funciona el decodificador? Modifica los estímulos para que se prueben todos los casos posibles.*

1.2.2. Ejercicio 2.2

En este apartado vamos a sintetizar el decodificador que acabamos de simular. Para ello, selecciona `View: Implementation` en el ISE, después la entidad `dec3to8` y finalmente el proceso `Synthesize-XST`. Consulta las propiedades de este último (botón derecho) para familiarizarte con ellas. Selecciona `Speed` en `Optimization Goal`, `Normal` en `Opmization Effort` y mantén el resto de opciones por defecto. En la parte inferior se muestra `Property display level`, con valor `Standard`. Si se cambia a `Advanced` se tiene acceso a muchas propiedades más para todos los procesos del ISE. La columna `Switch Name` muestra los modificadores que deben emplearse en los comandos correspondientes cuando el ISE se emplea desde consola. Esto es útil para automatizar ejecuciones en *batch*.

Como la síntesis está recién realizada (observa el signo *tic* en verde o pincha con el botón derecho para comprobar que la acción `Run` está en gris), el ISE no responde si intentas

repetirla.

Para forzar su ejecución en caso necesario debes emplear la acción `ReRun`. De hecho, todos los procesos disponen de las tres acciones: `Run`, `ReRun` y `Rerun All`.

Para averiguar la función de cada una de ellas, observa la información que aparece en la barra de estado (en la parte inferior de la ventana del ISE) cuando el ratón pasa por encima. Estas opciones de ejecución hacen uso del sistema de control de dependencias entre archivos incluido en ISE, que permite minimizar el número de procesos que deben ejecutarse según qué partes del flujo de diseño están actualizadas y cuáles no.

Repite entonces si quieres la síntesis y lanza después el proceso `View RTL Schematic` para acceder a la vista estructural del diseño sintetizado. Elige la segunda opción `Start with a schematic of the top-level block` en la ventana que aparece y haz click en `OK`. Se muestra entonces un esquema de la entidad sintetizada. Haciendo doble click en él y mediante los iconos de zoom del menú superior (similares a los de ISim) se puede navegar por el esquemático. En el panel inferior del ISE aparece también una nueva pestaña (`View by Category`) donde es posible seleccionar componentes, pines y señales del diseño individualmente.

Pregunta 2.2.1 *Analiza el esquema circuital obtenido tras la síntesis. ¿Es lo que esperabas?*

En el panel principal del ISE, cierra la pestaña correspondiente al esquemático. Como ya se ha puesto de manifiesto varias veces, ese panel es donde se abre la edición de los fuentes que es necesario ver o modificar. Otra pestaña del mismo que resulta de interés es `Design Summary`. Habitualmente está siempre presente pero, si no fuera así, es posible volver a abrirla mediante el menú superior, con `Project` → `Design Summary/Reports`.

Pues bien, pincha en la pestaña `Design Summary` para visualizarla. Después, en el cuadro superior izquierdo, dentro de `Detailed Reports`, señala `Synthesis Report`. El informe mostrado en el panel proporciona información detallada sobre el proceso de síntesis, incluyendo estimaciones sobre el uso de recursos en la FPGA destino y la frecuencia alcanzable (circuitos secuenciales) o el retardo introducido (combinacionales).

Pregunta 2.2.2 *Indica los recursos empleados en la FPGA (sección* `Device utilization summary` *del informe)*

Pregunta 2.2.3 *¿Es el circuito combinacional, secuencial o de los dos tipos? Justifica la respuesta.*

Vamos a finalizar la implementación del circuito. Para ello expande `Implement Design`, lo que permite ver los tres procesos en que se divide. Haz doble click en `Place & Route` (o botón derecho y `Run`). Comprobarás como, de forma automática, previamente se ejecutan las operaciones `Translate` y `Map`.

Una vez que se ha realizado la implementación del circuito hay que efectuar la verificación final, lo que se logra mediante una simulación *post-PAR* (*Place-And-Route*) empleando el mismo testbench que en la *behavioral*

Para ello, selecciona `View: Simulation`, debajo `Post-Route`, y realiza la simulación (como siempre, acuérdate de señalar el testbench).

Pregunta 2.2.5 *¿Se comporta el decodificador igual que en 2.1.3? Explica de forma clara qué ha cambiado algo y a qué se debe.*

Para realizar la simulación post-PAR, el ISE ha generado en VHDL un modelo de nuestro diseño que incluye todos los elementos circuitales empleados en la FPGA junto con sus interconexiones. Este modelo está en el fichero `netgen/par/dec3to8_timesim.vhd`. Edítalo y echa una ojeada al código que contiene.

1.2.3. Ejercicio 2.3

Pregunta 2.3.1 *Implementa un decodificador de 4 a 16 a semejanza del descrito anteriormente. Tienes que hacer tanto la descripción de la entidad y arquitectura como su conjunto de estímulos para comprobar que funciona correctamente. Luego, realiza la simulación de comportamiento, la síntesis e implementación y, finalmente, la simulación post-PAR.*

1.3. Ejemplo 3: Lógica secuencial

Hasta ahora hemos hecho dos ejemplos muy sencillos con lógica combinacional. Para empezar con los circuitos secuenciales, vamos a ver primero cómo se especifican registros simples en VHDL. En el caso más habitual de un registro o *flip-flop* tipo D tenemos en `reg1_dataflow.vhd` una posible implementación.

Crea un proyecto en el ISE e incluye el fichero anterior junto a su correspondiente testbench. Recuerda una vez más cambiar la asociación de este último a `Simulation`.

Abre con el editor el código del registro. Como puedes ver, su tamaño es seleccionable mediante un parámetro `generic` llamado `reg_size`.

Presta mucha atención a la arquitectura, porque en ella aparece la estructura básica que debe seguirse siempre que sea necesario especificar un elemento registrado en VHDL. Todos los elementos de un circuito que deban ser síncronos con el reloj deben ir insertados en una condición del tipo:

```
IF clk'event and clk = '1' THEN
    q <= d;
END IF;
```

De este modo se indica a la herramienta de simulación que la señal `q` debe actualizarse de manera síncrona con el flanco de subida del reloj `clk` y tomar en ese instante el valor de la señal `d`. En el caso de la síntesis, se inferirá un flip-flop tipo D con entrada `d` y salida `q`.

1.3.1. Ejercicio 3.1

Vamos a realizar una simulación del registro, pero primero modificaremos el tiempo de simulación tal como ya se describió en el Ejercicio 1.1. Para ello, selecciona `tb_regs` en

`Hierarchy`, haz click con el botón derecho sobre `Simulate Behavioral Model` en `Processes` y después escoge `Process Properties`. Cambia `Simulation Run Time` a `800 ns`.

Realiza la simulación. Puedes cambiar las señales `d_in` y `q1` para que aparezcan rotuladas en hexadecimal con la opción `Radix` (botón derecho). Así, los resultados se interpretan con más facilidad.

En `reg2_dataflow.vhd` se modifica la arquitectura, cambiando la forma de emplear la señal `rst`, pero la condición de sincronismo permanece idéntica. Incluye `reg2_dataflow.vhd` en el proyecto. Observa en el panel `Hierarchy` que se queda en paralelo con el testbench, y no colgando del mismo. Esto sucede porque ISE comprueba que la entidad correspondiente no está instanciada en el testbench. Analiza las diferencias entre las dos arquitecturas del registro.

Edita el testbench y, en primer lugar, analízalo con calma. Observa cómo se declara el `COMPONENT reg1` para luego instanciarlo más abajo con nombre `TOP1` y conectar todos sus terminales. Fíjate también en el empleo de los `generic` como medio para pasar parámetros a una entidad. Después, descomenta todas las líneas comentadas. Hacen referencia a la nueva implementación del registro, que hemos llamado `reg2`, y a sus señales asociadas. Observa entonces que aparece una nueva instancia `TOP2` colgando de `tb_regs`.

Inicia la simulación, cambia los formatos de `d_in`, `q1` y `q2` y analiza las formas de onda.

Pregunta 3.1.1 *¿Qué diferencia hay entre los comportamientos de las dos arquitecturas?*