

Grado en Ingeniería de Computadores

Tecnología de Computadores

Práctica 1



Universidad
Rey Juan Carlos

1. Objetivo

En esta primera sesión de prácticas se pretende que el alumno se familiarice con Vivado, el software de la empresa Xilinx que se utilizará a lo largo de la asignatura para diseñar, sintetizar y simular códigos VHDL.

Una vez terminada la práctica el alumno será capaz de:

- Crear proyectos de VHDL en Vivado
- Usar el editor de código VHDL de Vivado para corregir errores de sintaxis
- Obtener informes de utilización de recursos, consumo energético y tiempos en Vivado
- Simular circuitos sencillos a partir de un testbench dado
- Implementar diseños sencillos en una FPGA y comprobar su funcionamiento

2. Desarrollo de la práctica

Crear un nuevo proyecto en Vivado.

Primero comenzaremos por abrir Vivado y crear un nuevo proyecto. Para ello lanzaremos Xilinx Vivado desde myApps y ejecutaremos Vivado 2020.2 una vez se haya cargado el escritorio de la máquina virtual. Pulsamos a continuación en *Create Project*. Se abrirá el asistente de Vivado que nos guiará paso a paso durante la creación del proyecto.

- Pulsamos *Next* e introducimos el nombre del proyecto y su ubicación. Es recomendable usar una carpeta del directorio R:\ para evitar perder el proyecto al cerrar sesión en myApps.
- En el apartado *Project Type* seleccionamos la opción *RTL Project*. Pulsamos *Next*.
- En el apartado *Add Sources* elegimos VHDL en la opción *Target language*. Pulsamos *Next*.
- Dejamos como está el apartado de *Add Constraints*. Pulsamos *Next*.
- Seleccionamos *Nexys4 DDR* en la opción *Boards*. Pulsamos *Next*. Pulsamos *Finish*.

El editor de código VHDL de Vivado.

Una vez creado el proyecto, pulsamos en *Add Sources > Add or create design sources > Create File*. Llamamos al fichero VHDL *Inversor* y pulsamos *OK* y *Finish*. Pulsamos *OK* y *Yes*. Si lo hemos hecho bien, veremos en la ventana *Sources* el fichero que acabamos de crear dentro de la carpeta *Design Sources*.



Ahora hacemos doble click sobre el fichero *Inversor* para editarlo. Como vemos, el fichero ya contiene una estructura que Vivado ha generado automáticamente. Podemos usar esta estructura como plantilla o bien borrar el contenido y escribir nuestro código desde cero.

Vamos a comenzar editando la entidad del *Inversor*. Primero vamos a crear la entrada de nuestro inversor (a la que llamaremos X) y la salida del inversor (a la que llamaremos Y). Estas señales serán de 1 bit cada una, por lo que tendremos que añadir las siguientes líneas de código.

X : in STD_LOGIC;
Y : out STD_LOGIC

Quedando la entidad como sigue:

```
entity Inversor is
  Port ( X : in  STD_LOGIC;
        Y : out STD_LOGIC
        );
end Inversor;
```

Esto define las entradas y salidas de nuestro componente a modo de “caja negra”. Ahora necesitamos definir en la arquitectura del código VHDL qué es lo que va a hacer este componente denominado *Inversor*.

El comportamiento del componente Inversor se puede definir con la siguiente línea de código VHDL:

```
Y <= not (X);
```

Quedando la arquitectura como sigue:

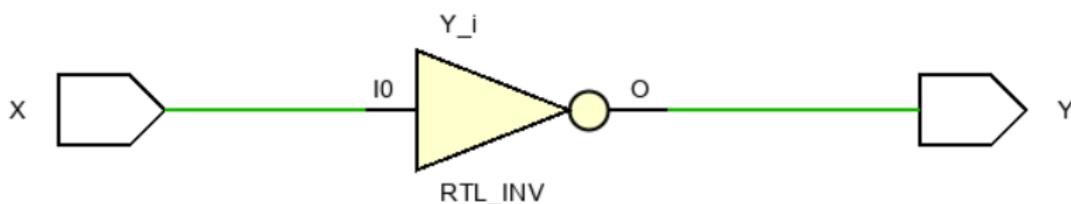
```
architecture Behavioral of Inversor is
begin
  Y <= not(X);
end Behavioral;
```

Con esto ya tendríamos nuestro primer componente en VHDL diseñado.

Generar el diseño elaborado por Vivado.

Una primera comprobación que se puede realizar para verificar que el circuito diseñado es correcto consiste en utilizar la funcionalidad de Vivado para generar automáticamente el diseño elaborado a partir de código VHDL. Vamos a ver cómo se usa.

Para ello, una vez escrito el código VHDL de nuestro diseño, pulsamos en la opción *Open Elaborated Design* dentro de la categoría *RTL Analysis* del menú de la izquierda. Aceptamos los mensajes y esperamos unos instantes a que Vivado genere un esquemático de nuestro circuito. Si todo ha ido bien, deberemos obtener un circuito como el mostrado en la siguiente imagen:



Como vemos, Vivado ha inferido correctamente un inversor a partir de nuestro código VHDL. Cabe mencionar que, en este caso, se trata de un circuito muy sencillo pero, como veremos en prácticas posteriores, en diseños más complejos es una funcionalidad de gran utilidad para revisar que las conexiones entre componentes y la declaración de señales está bien realizada.

Simulación de código VHDL.

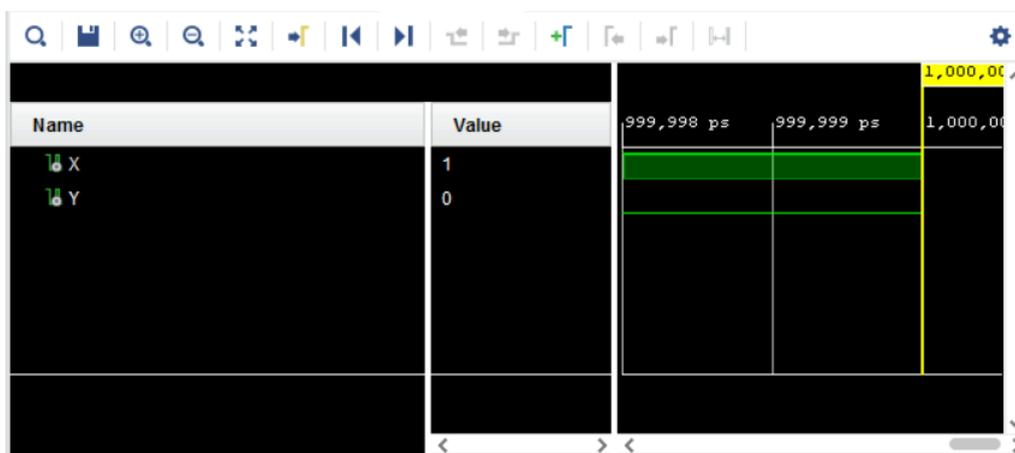
Otra etapa fundamental en el flujo de diseño hardware es la simulación del circuito mediante un banco de pruebas (testbench). Como veremos posteriormente en la asignatura, un testbench no es más que otro código VHDL que contiene valores para las señales de entrada de un circuito, de tal forma que Vivado, a partir de este código VHDL, pueda generar las salidas del circuito. Este proceso de simulación permite analizar de forma rápida si el circuito se comporta según lo esperado.

En esta práctica y posteriores, y hasta que se explique en las clases de teoría cómo se diseña un testbench, se proporcionará el código VHDL de los testbenches necesarios para simular los distintos circuitos.

Siguiendo con el ejemplo del *Inversor*, vamos a utilizar el código VHDL del testbench denominado “*tb_Inversor.vhd*” proporcionado por el profesor. Para simular el inversor primero cerramos el diseño elaborado que generamos previamente y luego pulsamos en la opción *Add Sources > Add or create simulation sources*.

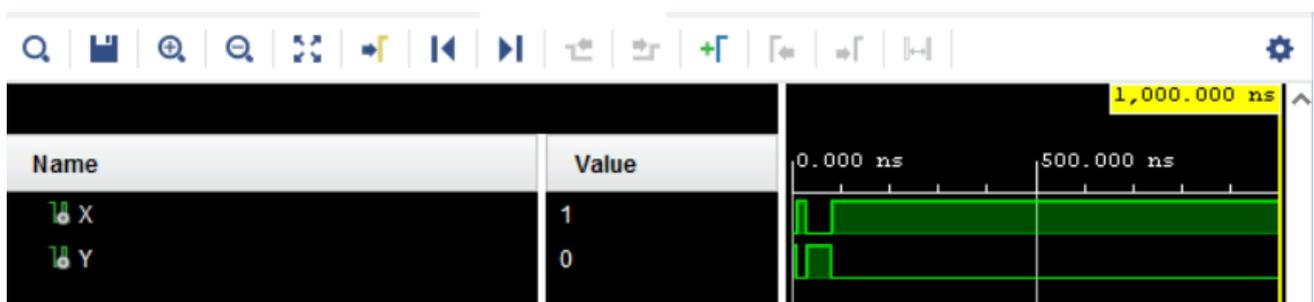
Pulsamos en *Add Files* y buscamos el fichero VHDL mencionado anteriormente. Pulsamos *OK*. Marcamos la casilla *Copy sources into project* y pulsamos en *Finish*. Si lo hemos hecho bien, en la ventana de *Sources*, dentro de la carpeta *Simulation Sources > sim_1* nos aparecerá el fichero VHDL del testbench.

Para ejecutar la simulación pulsamos en la opción *Run Simulation* dentro de la categoría *Simulation* del menú de la izquierda. Luego pulsamos en *Run Behavioral Simulation* y se lanzará la simulación. Si todo ha ido bien se debería abrir una ventana como la siguiente:



Esta es la ventana con las formas de onda de las señales de entrada y de salida de nuestro circuito. Si observamos con detenimiento, en la primera columna aparecen listadas todas las señales de nuestro circuito (la entrada X y la salida Y). En la segunda columna aparecen los valores que tienen esas señales en el instante de tiempo actual de la simulación (el marcado por el cursor amarillo de la tercera columna). Finalmente, en la tercera columna tenemos dibujadas en verde las formas de onda de la simulación del circuito.

En esta tercera columna únicamente se observa que X está valiendo ‘1’ todo el rato, mientras que Y (la salida) toma el valor ‘0’. Esto es consistente con la definición de inversor que hemos hecho en el código VHDL. Sin embargo, esto que se observa no es toda la simulación, sólo la parte final. Para ver toda la simulación realizada pulsamos en el botón *Zoom Fit*  y obtenemos la siguiente forma de onda:



Se puede “jugar” con el resto de botones de la ventana de simulación para hacer zoom a zonas concretas de la simulación y para avanzar o retroceder en el tiempo. Se recomienda al alumno que dedique unos minutos a investigar el uso de todos los botones hasta familiarizarse con ellos y con la navegación en esta ventana.

Obtener informes de utilización de recursos, consumo de potencia y tiempos.

Vivado también permite la obtención de informes para analizar el comportamiento físico de los circuitos.

Primero vamos a analizar la utilización de recursos del *Inversor*. Para ello pulsamos en *Run Implementation*, dentro del menú de *Implementation*. Esperamos a que termine y luego elegimos la opción *Open Implemented Design*. Veremos que nos aparece una imagen negra con cuadrados de colores. Esta imagen es una representación de la distribución interna de la FPGA. Si agrandamos la imagen haciendo zoom, podremos encontrar nuestro circuito, que será un pequeño cuadrado azul, dado que se trata de un circuito muy sencillo.

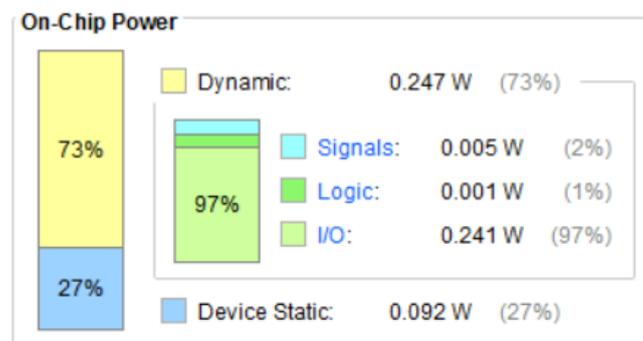


Volviendo al tema del informe de utilización, si pulsamos en *Report Utilization* y pulsamos *OK*, podremos generar un informe que resume el número de recursos usados por nuestro circuito como el que se presenta a continuación:

Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
N Inversor	1	1	1	2

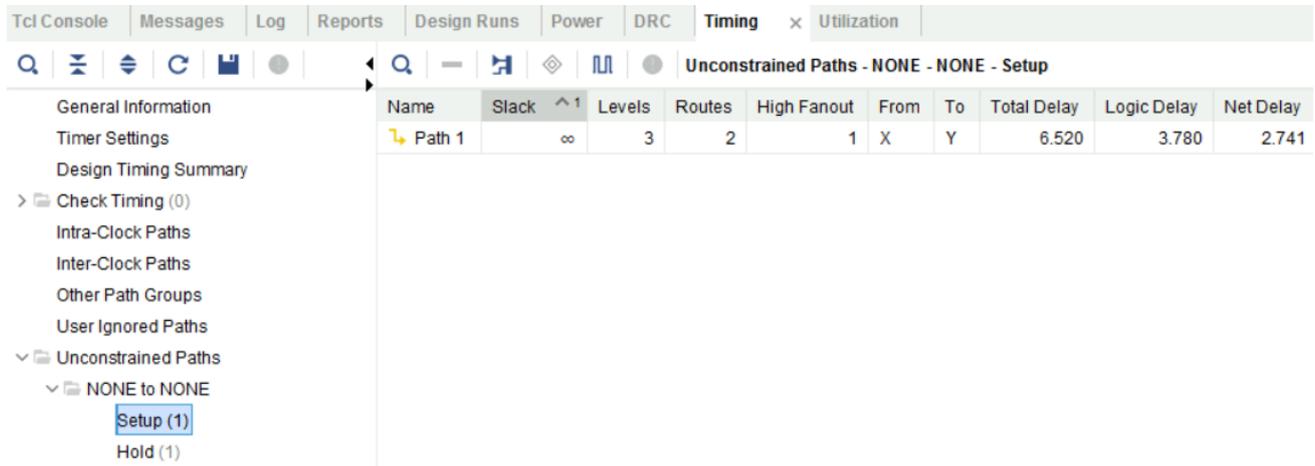
Como vemos, nuestro circuito utiliza una LUT (*look-up table*) que contiene la tabla de verdad del inversor, y dos IOB (*input-output blocks*) que son la entrada X y la salida Y de nuestro circuito. Esta LUT y estos dos IOB se corresponden con los pequeños cuadrados azules vistos en la imagen anterior.

Ahora vamos a pasar a analizar el consumo de nuestro circuito. Para ello pulsamos la opción *Report Power*. Nos saldrá una ventana en la que podemos cambiar muchos parámetros, pero por ahora, nosotros los dejamos tal cual y pulsamos *OK*. Obtendremos, entre otras cosas, la siguiente gráfica de barras apiladas:



Aquí podremos analizar el consumo tanto estático como dinámico de nuestro circuito y el desglose con las partes del diseño que más consumen. En este caso vemos que el consumo dinámico es mayor y que es debido principalmente a las entradas y salidas del circuito.

Por último, vamos a generar un informe de tiempos. Para ello pulsamos en *Report Timing Summary* y pulsamos *OK*. Si navegamos hasta la opción mostrada en la siguiente imagen, podremos ver el tiempo de establecimiento (Setup) y el tiempo de mantenimiento (Hold) del camino que va desde la señal de entrada X hasta la señal de salida Y. Este tiempo está expresado en nanosegundos. Sumando ambos obtendríamos el tiempo total.



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	3	2	1	X	Y	6.520	3.780	2.741

Implementar el diseño en la FPGA.

Finalmente, una vez verificado, simulado y obtenidos los informes de nuestro circuito, vamos a implementarlo en la FPGA para comprobar su funcionamiento en el hardware físico.

Primero, para poder programar nuestro diseño en la FPGA, debemos añadir otro fichero que especifica las conexiones físicas de las entradas y salidas de nuestro diseño. Es decir, un fichero que indica dónde se conectarán la señal X de entrada y la señal Y de salida.

Dado que se trata de un inversor sencillo, lo que podemos hacer es conectar la señal X a un interruptor y la señal Y a un LED. De este modo, al tener el interruptor apagado el LED se encenderá y viceversa.

El fichero en cuestión se denomina fichero de constraints y se añade desde el menú *Add Sources > Add or create constraints*. Creamos un fichero con el nombre *constraints* y le damos a *Finish*. El fichero se encuentra dentro de la carpeta *Constraints*. Si le damos doble click lo podremos editar. Añadiremos las siguientes líneas al fichero:

```
# Conectamos la señal de entrada X al interruptor
set_property PACKAGE_PIN J15 [get_ports X]
set_property IOSTANDARD LVCMOS33 [get_ports X]
```

```
# Conectamos la señal de salida Y al LED
set_property PACKAGE_PIN H17 [get_ports Y]
set_property IOSTANDARD LVCMOS33 [get_ports Y]
```

Con estas líneas lo que estamos indicando es que conectaremos la señal X al pin J15, el cual es el nombre del interruptor situado abajo a la derecha de la placa (SW0). Y también conectaremos la señal Y al pin H17, que es el LED justo encima del interruptor anterior (LED0). Las líneas *IOSTANDARD LVCMOS33* lo que indican es el nivel de tensión que se le asigna al pin, en este caso 3.3V.

Ahora ya podemos sintetizar el circuito. Para ello pulsamos primero en *Run Synthesis* para hacer la síntesis del diseño (en la pestaña *Log* podremos ver el progreso de la síntesis). Con esta fase de **síntesis** lo que conseguimos es crear el circuito digital a partir del código VHDL.

Cuando termine la síntesis del diseño marcamos *Run Implementation* y le damos a *OK*. La fase de **implementación** permite mapear nuestro circuito digital a la FPGA. Es decir, asigna los recursos de la FPGA que se van a usar para el diseño obtenido de la fase de síntesis. En esta fase se usa el fichero de constraints con la ubicación de la entrada y la salida física de nuestro circuito.

Una vez implementado, seleccionamos *Generate Bitstream*. La fase de **generación del bitstream** permite convertir el circuito en un fichero de unos y ceros con el que se programará la FPGA. Si todo ha ido bien ya podremos programar la FPGA. Para ello:

- Conectamos la Nexys4 DDR al ordenador con un cable USB y la encendemos.
- En Vivado, pulsamos en *Open Hardware Manager > Open Target > Auto Connect*
- Por último pulsamos en *Program Device > Program*

Cuando finalice la programación de la FPGA podremos ver cómo se enciende el LED0 de la placa al bajar el interruptor SW0 y viceversa. Comportándose justamente como un inversor.

3. Tareas a realizar

Una vez familiarizados con el uso de Vivado, se propone una serie de ejercicios para practicar lo aprendido.

1. Cree un nuevo fichero VHDL y copie el siguiente código:

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity Circuito is  
  Port ( X1 : in std_logic;  
        X2 : in std_logic;  
        Y : out std_logic;  
  );  
end Circuito;
```

```
architecture Behavioral of Circuit is  
begin  
  process (X1, X2) begin  
    if ((X1='1') and (X2='1')) then  
      Y = '1';  
    else  
      Y = '0';  
    end if;  
  end process;  
end ehavioral;
```

2. Identifique y corrija los errores de sintaxis ayudándose del editor de texto de Vivado.
3. Determine el componente al cual representa este código VHDL utilizando el análisis RTL.
4. Simule el circuito utilizando el testbench proporcionado. ¿Cuánto vale la salida Y transcurridos 20ns?
5. Modifique el fichero de constraints para poder implementar el circuito en la FPGA utilizando los interruptores SW0 y SW1 como entradas y el LED0 como salida. Verifique su funcionamiento en la FPGA.
6. Determine el número de LUTs e IOBs utilizados por el circuito.
7. Determine el consumo dinámico del circuito.
8. Determine el *Total Delay* de *Setup* del camino que une la entrada X2 con la salida Y.