

A large, light gray, stylized profile of a man with a cap and curly hair, facing right, serves as a background for the upper half of the page.

# Tecnología de Computadores

## Práctica 1



UNIVERSIDAD  
**NEBRIJA**

## 1. INTRODUCCIÓN

En esta primera sesión de laboratorio se pretende que el alumno se familiarice con Vivado, un software para diseño, simulación y síntesis de códigos VHDL de la empresa Xilinx.

Una vez terminada la práctica el alumno será capaz de:

- Crear proyectos de VHDL en Vivado.
- Usar el editor de código VHDL para comprobar errores de sintaxis.
- Implementar diseños sencillos en FPGAs.
- Obtener informes de utilización, consumo y delay.

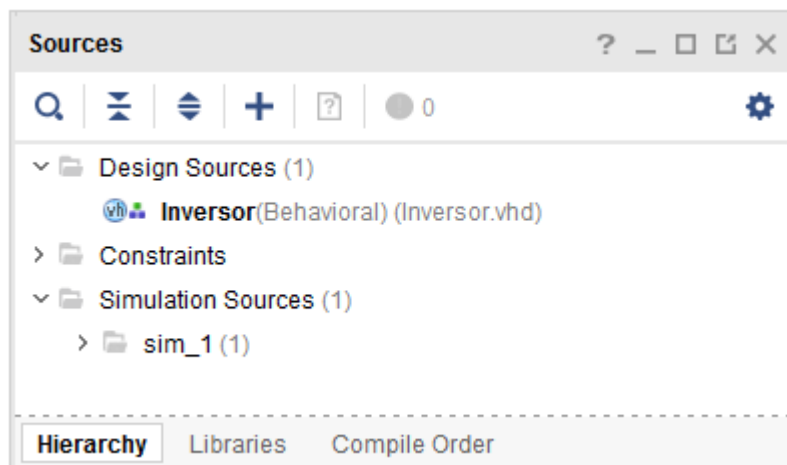
## 2. CREAR UN NUEVO PROYECTO EN VIVADO

Primero comenzaremos por abrir Vivado y crear un nuevo proyecto pulsando en *Create Project*. Se nos abrirá el asistente de Vivado que nos guiará paso a paso durante la creación del proyecto.

- Introducimos el nombre del proyecto y su ubicación. Se recomienda que tanto el nombre como la ruta del proyecto no tengan espacios. Pulsamos *Next*.
- 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 *Display Name*. Pulsamos *Next*. Pulsamos *Finish*.

## 3. 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 *Sources* el fichero que acabamos de crear.



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

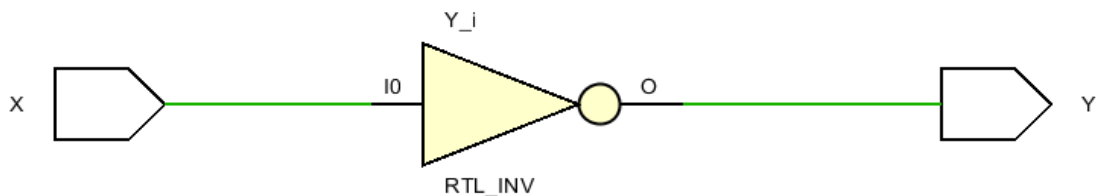
Dentro de la entidad *Inversor* añadimos las siguientes líneas de código para crear dos señales de tipo *std\_logic*, la entrada X de nuestro inversor y su salida Y.

```
port( X : in std_logic;
      Y : out std_logic
    );
```

Ahora, nos queda definir el comportamiento de nuestro componente *Inversor*. Dado que lo que queremos es que nos invierta el valor de la señal de entrada, deberemos añadir la siguiente línea dentro de *architecture*, justo después de *begin*.

```
Y <= not X;
```

Con esto ya tendríamos nuestro inversor definido. Podemos comprobar que Vivado lo interpretará correctamente utilizando el *RTL Analysis*. Si pulsamos en *Open Elaborated Design* deberíamos obtener un circuito similar al siguiente:



Ahora, 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.

#### 4. IMPLEMENTAR EL DISEÑO EN LA FPGA

Si hemos seguido los pasos anteriores sin problemas, entonces podremos cargar nuestro diseño del *Inversor* en la FPGA. 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.

#### 5. OBTENER INFORMES DE UTILIZACIÓN, CONSUMO Y DELAY

Volviendo a Vivado, cerramos el *Hardware Manager* y nos disponemos a obtener una serie de informes que nos permitirán analizar el comportamiento físico de nuestro diseño.

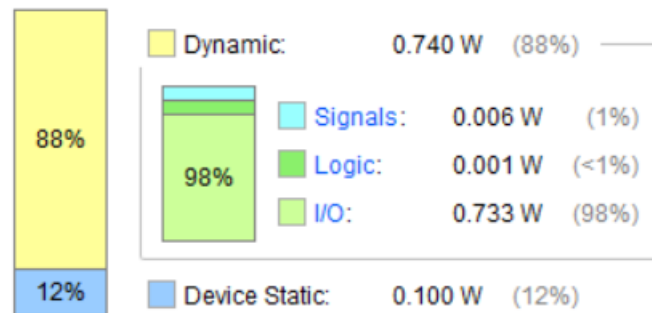
Primero vamos a analizar la utilización de recursos del Inversor. Para ello pulsamos en *Open Implemented Design*, dentro del menú de *Implementation*. Veremos que nos aparece una imagen negra con cuadrados de colores. Esta imagen es una representación de la distribución interna de la FPGA. Si agrandamos 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 ^1	Slice LUTs (63400)	Slice (1585 0)	LUT as Logic (63400)	Bonded IOB (210)
 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.

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*. Como vemos, al tener un circuito combinacional tan simple nos sale un tiempo de 0ns. Si tuviéramos más componentes y/o fuera un circuito con componentes secuenciales, este tiempo sería distinto de cero pues habría un retraso en la propagación de la señal a lo largo de la FPGA.

## 6. TAREAS A REALIZAR

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

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

- Identifique y corrija los errores de sintaxis ayudándose del editor de texto de Vivado.
- Determine el componente al cual representa este código VHDL utilizando el análisis RTL.
- Modifique el fichero de constraints para poder implementar el circuito en la FPGA y verifique su correcto funcionamiento usando los interruptores de la placa.

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

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

architecture Behavioral of Component 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) Determine el número de LUTs utilizadas por un multiplexor 4:1.
- 3) Implemente un decodificador de 7 segmentos mediante un bucle case-when ayudándose de la siguiente tabla:

B3 B2 B1 B0	A B C D E F G
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100

Y sabiendo que los pines para los segmentos de los displays son:

- Segmento A: Pin T10
- Segmento B: Pin R10
- Segmento C: Pin K16
- Segmento D: Pin K13
- Segmento E: Pin P15
- Segmento F: Pin T11
- Segmento G: Pin L18

