



POLITÉCNICA

escuela técnica superior de  
**ingeniería**  
**y diseño**  
**industrial**

**SISTEMAS ELÉCTRÓNICOS DIGITALES**  
**GUIÓN DE PRÁCTICAS DE LABORATORIO**  
Versión 2021

# PRÁCTICA 0

## INTRODUCCIÓN AL ENTORNO VIVADO DE XILINX

Versión: 2021\_v1.1

Escuela Técnica Superior de Ingeniería y diseño Industrial  
Departamento de Electricidad, Electrónica, Automática y Física Aplicada.

## 1 OBJETIVO DE LA PRÁCTICA

El objetivo de esta práctica es que el alumno se familiarice con el entorno de desarrollo (VIVADO de Xilinx) que se utilizará durante el curso. Esta práctica se realizará individualmente por el alumno antes de la primera práctica presencial. **Es obligatorio realizar esta práctica antes de la práctica 1 y entregar un zip con los archivo del proyecto.**

## 2 INTRODUCCIÓN

Este tutorial le guía a través del flujo de diseño utilizando el software Xilinx Vivado para crear un circuito digital simple usando VHDL. Un flujo de diseño típico consiste en crear modelos, crear archivos de restricciones, crear un proyecto de Vivado, importar los modelos creados, asignar archivos de restricciones creados, ejecutar simulaciones de comportamiento, sintetizar el diseño e implementar el diseño, generando el flujo de bits, y finalmente verificando la funcionalidad en el hardware seleccionado (en este caso la Nexus 4 DDR y la FPGA Artix-7) descargando el archivo de flujo de bits generado. El flujo de diseño típico se muestra a continuación. El número en un círculo indica el paso correspondiente en este tutorial.

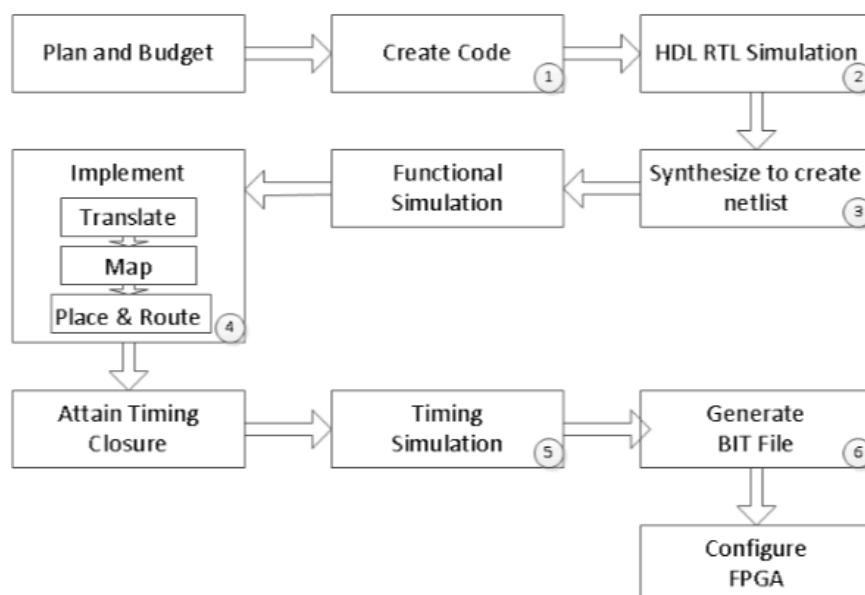


Figura 1. Flujo de diseño típico

Es importante tener en cuenta que las herramientas comerciales tardan un cierto tiempo en adaptarse a las últimas revisiones, por lo que si es importante la compatibilidad entre herramientas conviene ceñirse a revisiones bien establecidas del estándar. En la actualidad, casi todas las herramientas soportan VHDL2008.

Es importante resaltar que VHDL y otros lenguajes HDL tienen capacidad de expresar más cosas de las que las herramientas actuales son capaces de sintetizar. Incluso existen construcciones que tienen sentido para la verificación de un diseño o su descripción a muy alto nivel, pero que no tienen sentido en un chip, como el acceso a ficheros en disco

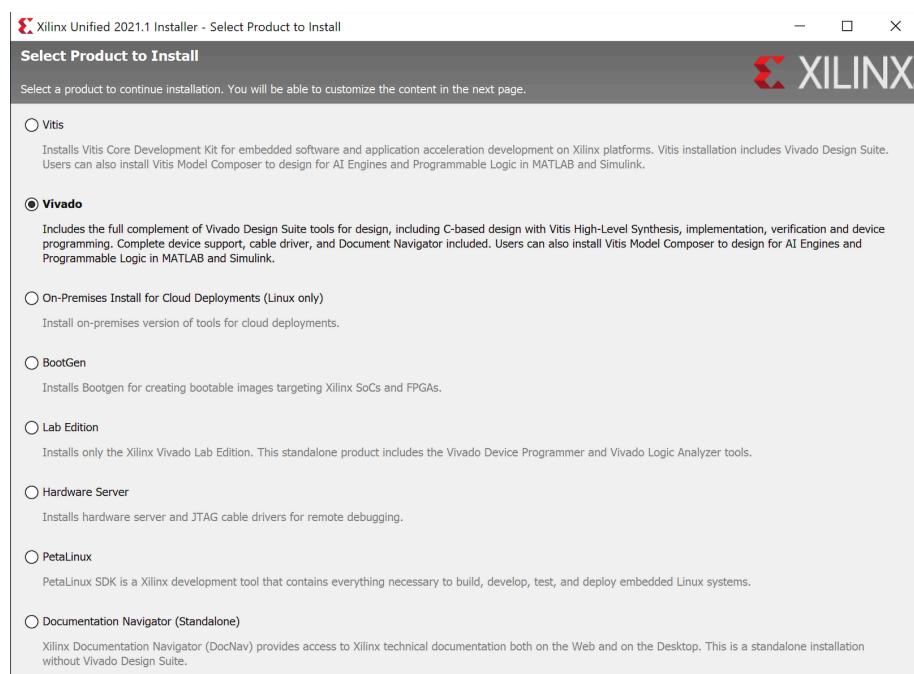
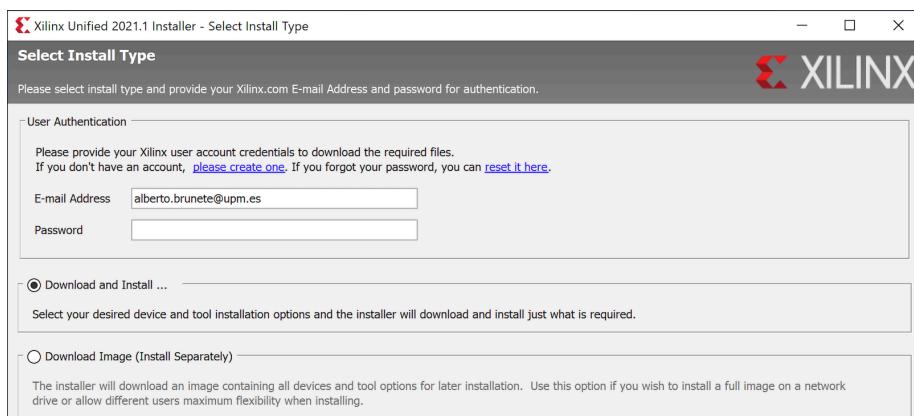
o los bucles for. Por tanto, el hecho de que un diseño pueda simularse no significa que pueda sintetizarse. De hecho, según se avanza en el diseño y éste se va refinando, hay que restringirse a un subconjunto del lenguaje y a unas ciertas reglas de codificación que la herramienta de síntesis pueda comprender. Es lo que se denomina código sintetizable.

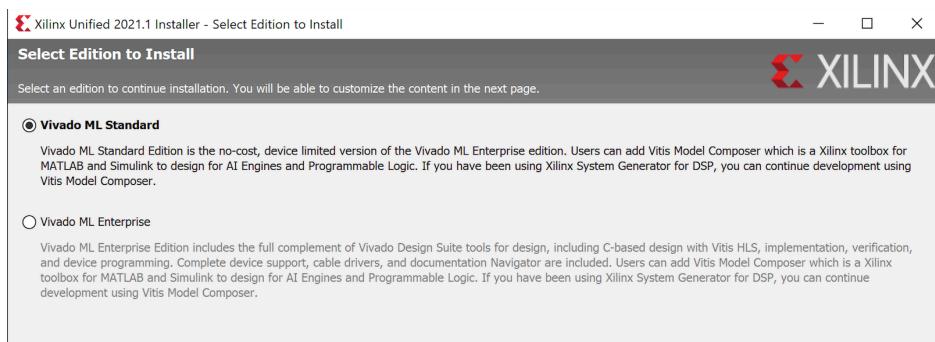
Como entorno de desarrollo emplearemos Vivado Design Suite, ya que trabajaremos con dispositivos programables de este fabricante. Este entorno puede descargarse de la página web de Xilinx:

<https://www.xilinx.com/support/download.html>

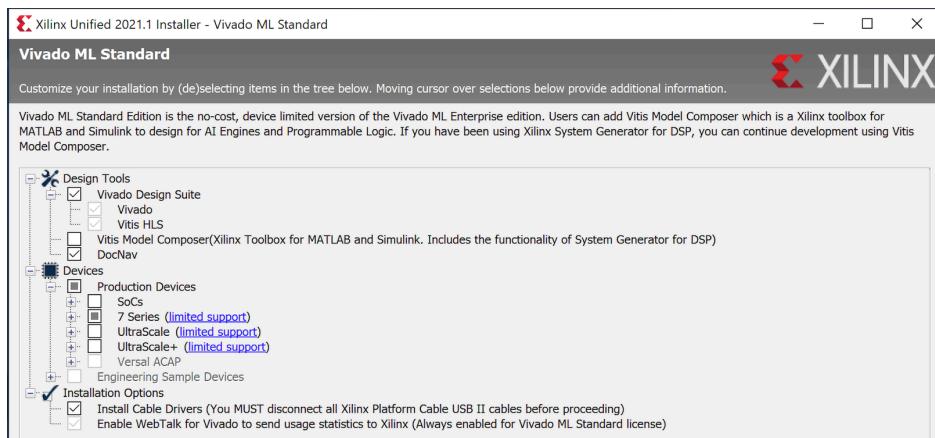
En la pestaña “Vivado” seleccionar la versión correspondiente, “Vivado ML Edition-2021.1 Full Product Installation” (cuidado, no la que pone “Product Update” ni la versión Lab).

Durante la instalación elija “Download and install”, “Vivado” y “Vivado ML Standard”, como muestran las imágenes siguientes.





Para que ocupe menos espacio seleccionar sólo lo imprescindible (aún así son 22GB).



Aunque casi todos los fabricantes ofrecen su propio paquete de herramientas, que suelen subvencionar de algún modo, y que, como era de esperar, sólo es válido para sus productos, también existen paquetes independientes del fabricante, desarrollados por empresas de software de CAD/CAE como Mentor, Cadence o Synopsys.

Esta guía está basada en los tutoriales de Xilinx:

<https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start>

### 3 DESARROLLO DE LA PRÁCTICA

De cara a ejecutar el software en su ordenador y seguir el tutorial siguiente, es necesario que instale los ficheros “boards” en su ordenador (en el laboratorio ya están instalados). Esto le permitirá configurar su placa de forma inmediata. Puede seguir el siguiente tutorial:

<https://reference.digilentinc.com/reference/software/vivado/board-files?redirect=1>

El diseño que vamos a realizar en esta práctica consiste en conectar entradas (conmutadores o switches) a algunas salidas (LEDs), bien directamente o a través de lógica combinacional. Un diagrama del diseño a realizar se muestra en la figura siguiente:

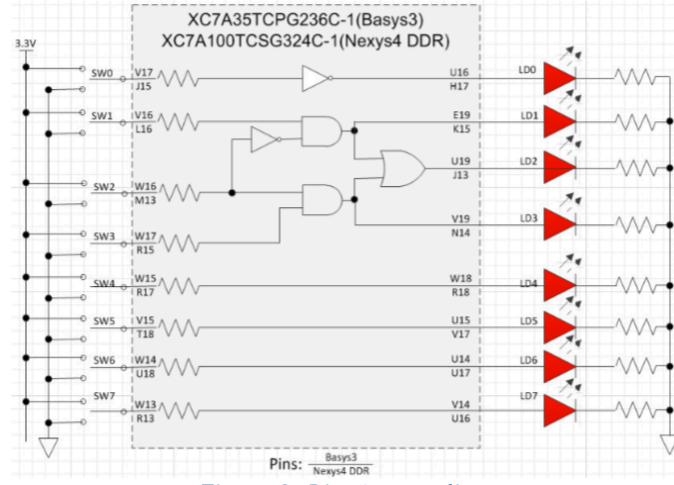


Figura 2. Diseño a realizar

### 3.1 CREACIÓN DE UN NUEVO PROYECTO

Para crear un proyecto hay que seguir los pasos siguientes.

1. Abra Vivado y haga clic en **Crear nuevo proyecto** (o de manera alternativa Fichero -> Nuevo proyecto) para abrir el Asistente para proyectos nuevos de Vivado.

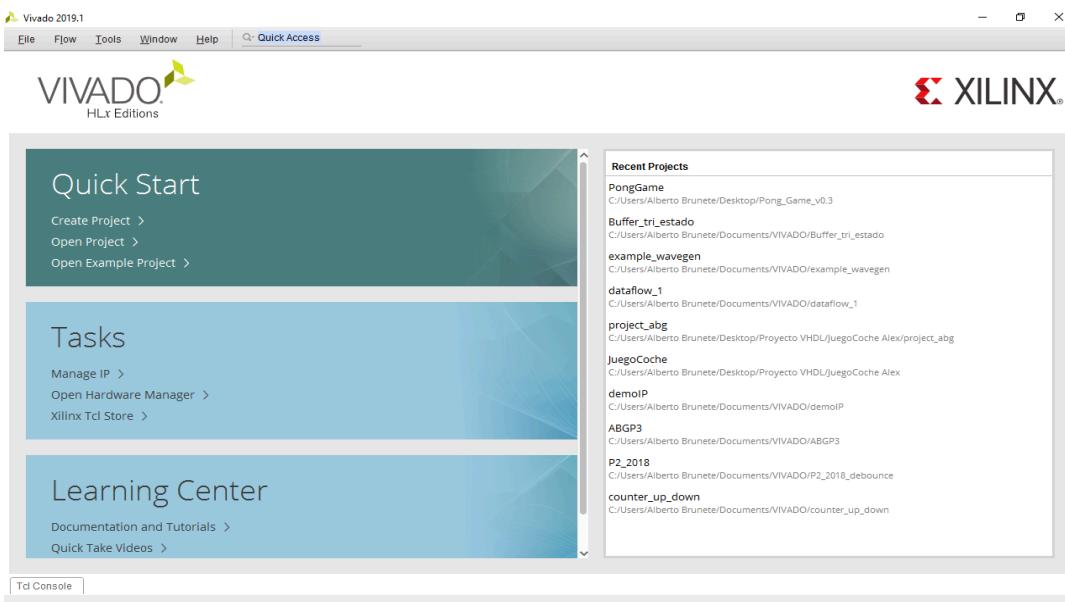


Figura 3. Abrir nuevo proyecto

2. Se abrirá una nueva ventana, haga clic en "Siguiente" y verá la pantalla que se muestra a continuación. Nombre su proyecto (¡sin espacios!) y elija el directorio para guardar el proyecto antes de hacer clic en Siguiente.

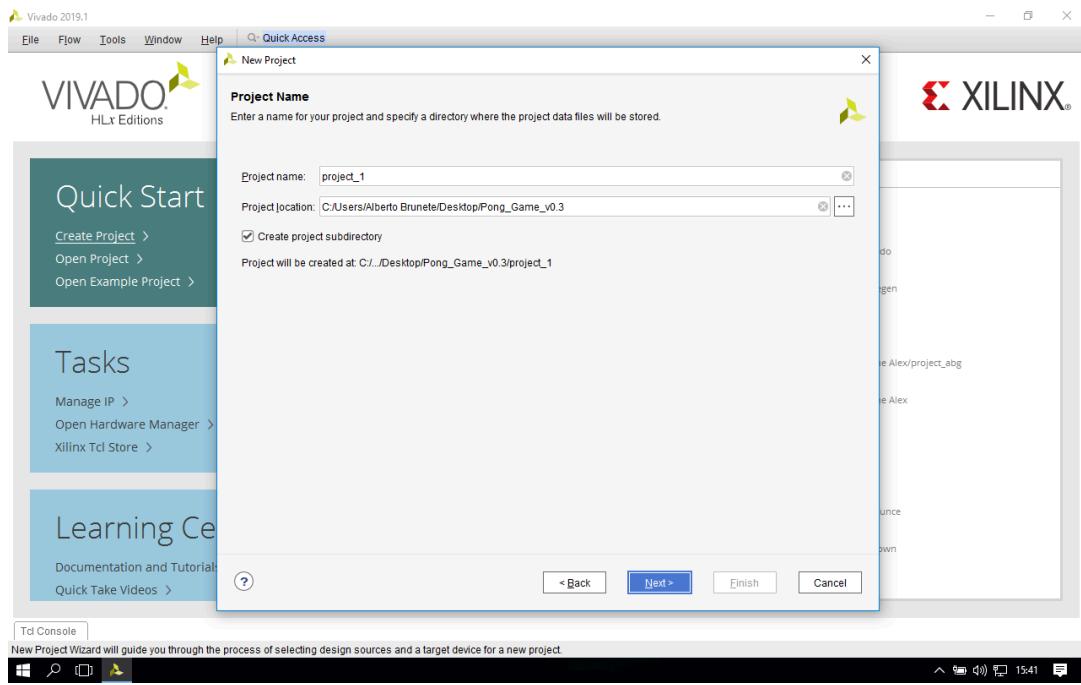


Figura 4. Datos del nuevo proyecto

3. Se va a construir un proyecto desde cero, añadiendo nuestras propias fuentes, así que queremos crear un proyecto de RTL. Seleccione **Proyecto RTL** y haga clic en Siguiente.

4.

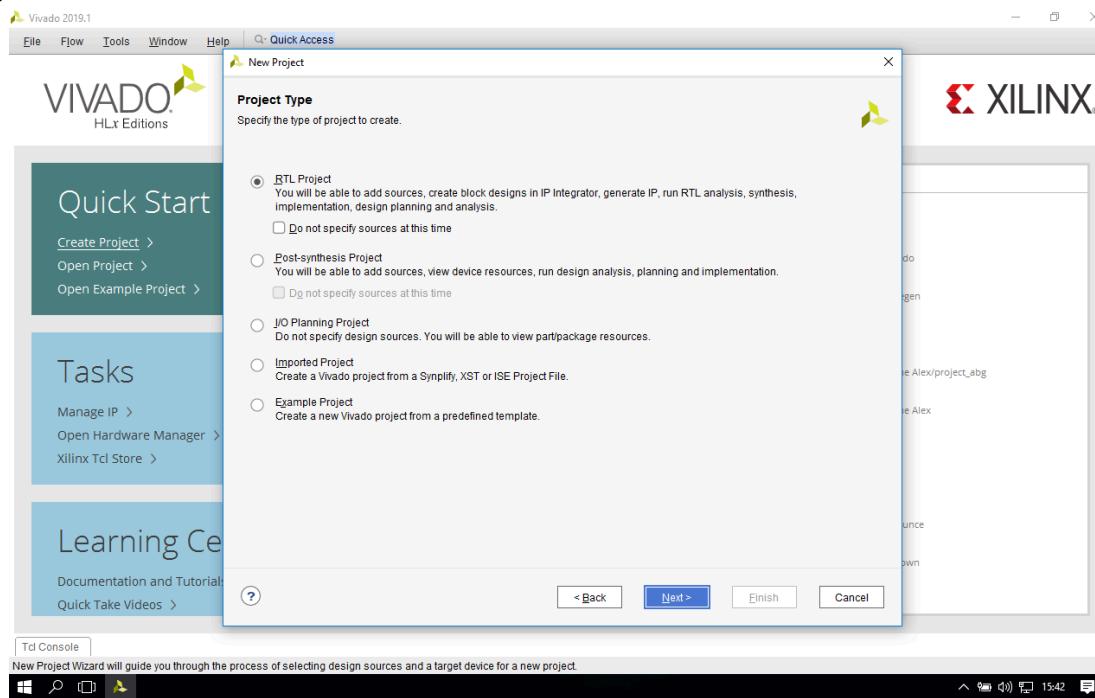


Figura 5. Tipo de proyecto

5. En esta ventana, puede seleccionar cualquier archivo o directorio de origen que desee utilizar en sus proyectos. También podemos seleccionar el idioma en el que estaremos programando, en nuestro caso VHDL. En primer lugar, crearemos un archivo fuente. Haga clic en '**Crear fichero**', elija VHDL como tipo de fichero y póngale un nombre.

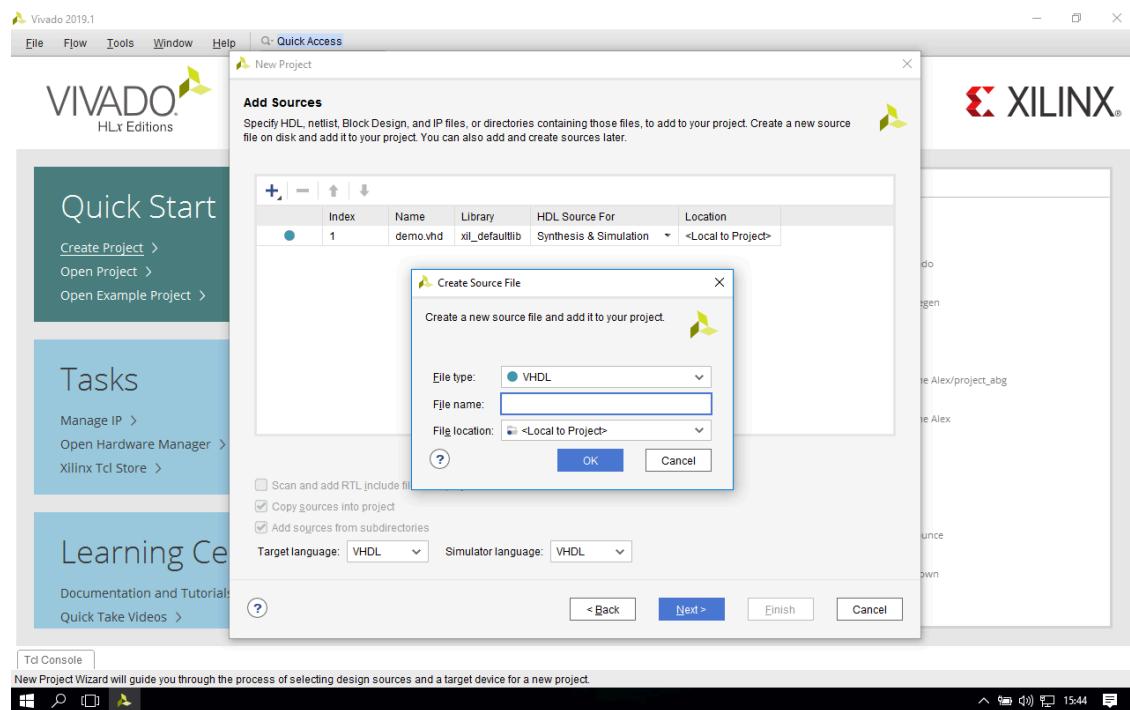


Figura 6. Añadir fuentes al proyecto

Si la casilla de verificación “**Copiar fuentes en el proyecto**” no está marcada, Vivado no creará copias separadas de sus fuentes y las colocará dentro del directorio del proyecto. Haga clic en “Siguiente” para continuar. También puede ir a Setings->Source File y activar opción “Copy sources into project”.

A continuación, podemos añadir restricciones a nuestro proyecto. En nuestro caso las restricciones serán las de la placa que estamos usando, la nexyx4. El objetivo es indicar dónde están conectadas las entradas y salidas. El fichero de restricciones (.XDC) lo podemos obtener del fabricante (<https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start>)

Por tanto, importaremos nuestro archivo Xilinx Design Constraints (XDC) para poder asignar las señales VHDL a los pines Artix-7. Haga clic en ‘+’ en el centro de la pantalla para agregar archivos, navegue hasta donde guardó su archivo **Nexys-4-DDR-Master.xdc**, selecciónelo y haga clic en Siguiente.

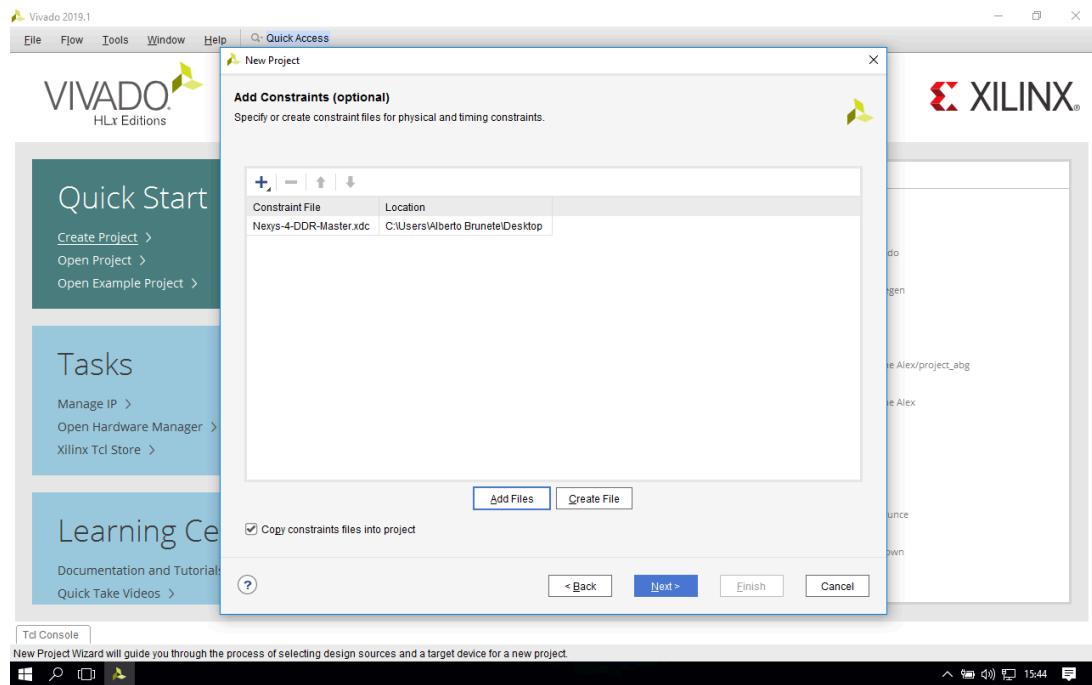


Figura 7. Añadir restricciones (Constraints) al proyecto

6. En este punto, Vivado abrirá una ventana de selección de partes y placas. Seleccione la pestaña “Boards”. Si instaló los archivos de la tarjeta correctamente, debería ver una lista de las tarjetas Digilent. Seleccione Nexys4 DDR y haga clic en Siguiente.

Esto creará su proyecto y lo llevará al administrador del proyecto Vivado. De este modo ha importado correctamente sus archivos de programa y ha configurado su proyecto para comunicarse correctamente con el Nexys4-DDR.

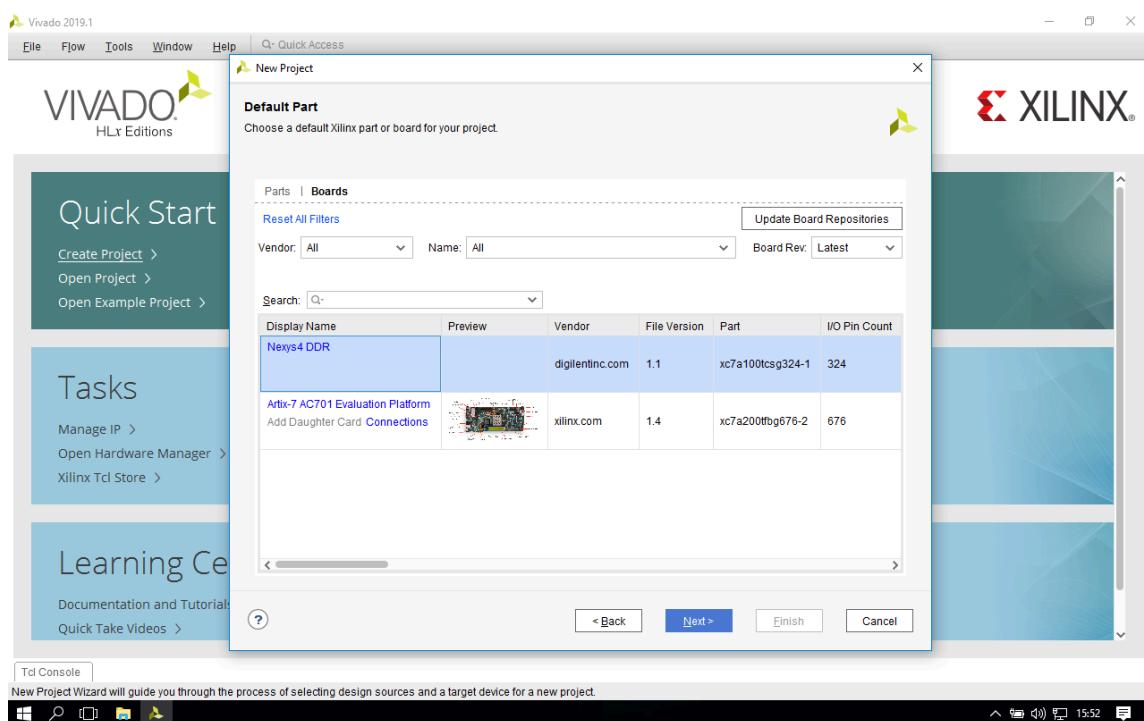


Figura 8. Elegir placa por defecto

7. Una vez creado el proyecto, se abre un asistente que nos permite definir una entidad (en el ejemplo “demo\_entity” o “p1”), su arquitectura y sus E/S. A modo de ejemplo podemos seleccionar un vector de entrada que sean los interruptores de la placa (Switch - SW), y un vector de salida que sean los LEDs de la placa.

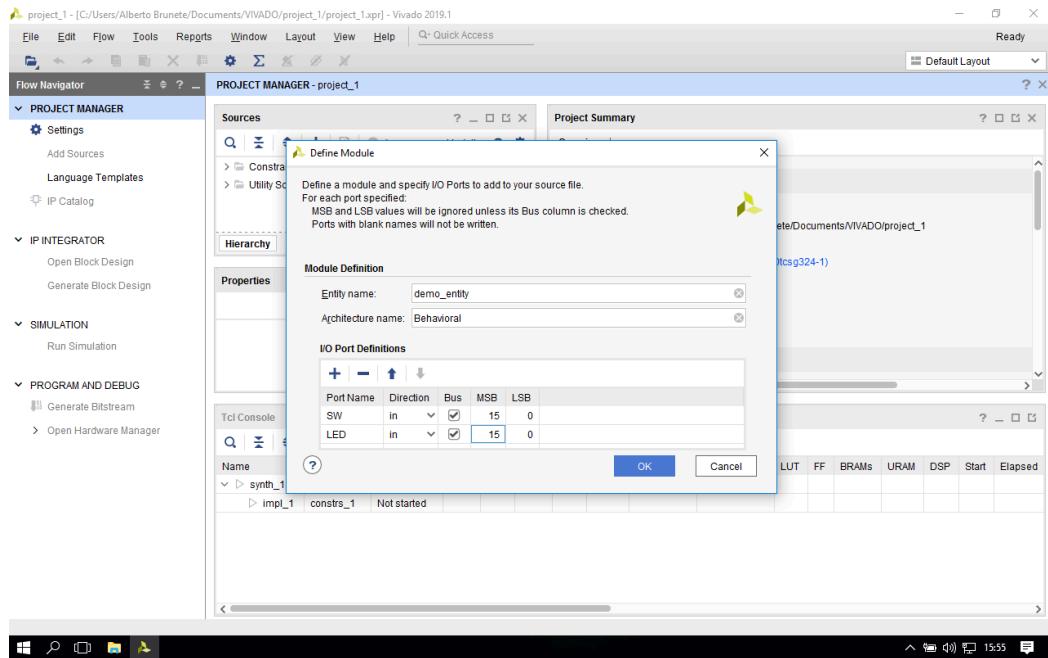


Figura 9. Definición de la entidad

Y obtenemos el proyecto configurado:

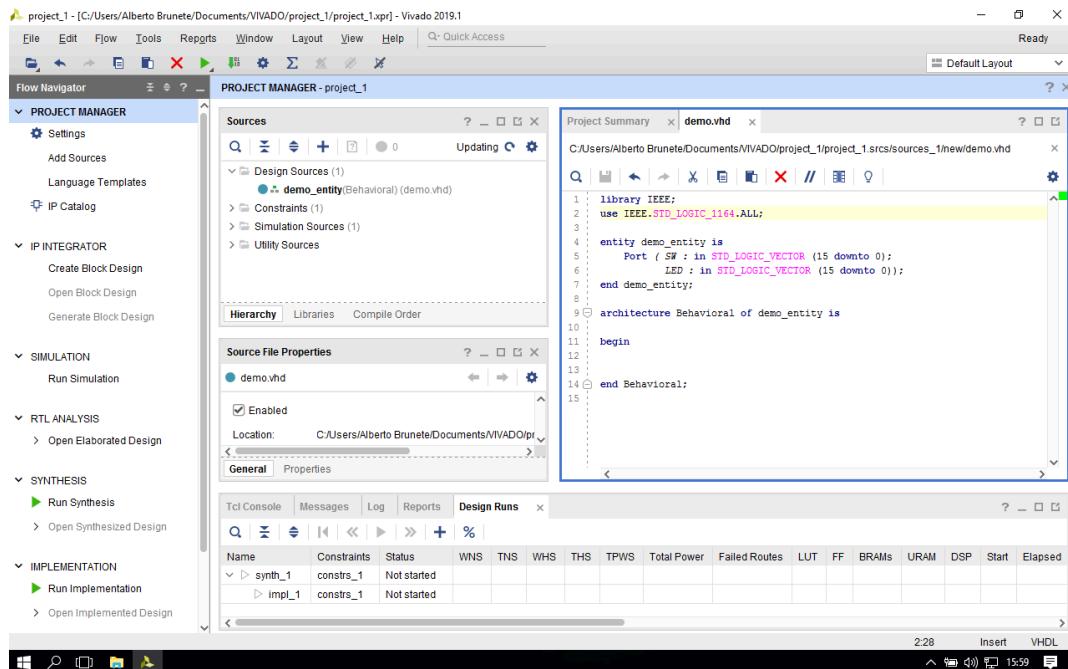


Figura 10. Proyecto configurado

8. Los archivos se habrán generado en el directorio indicado. Tome nota de la ruta, pues la necesitará posteriormente. Tendrá el aspecto siguiente:

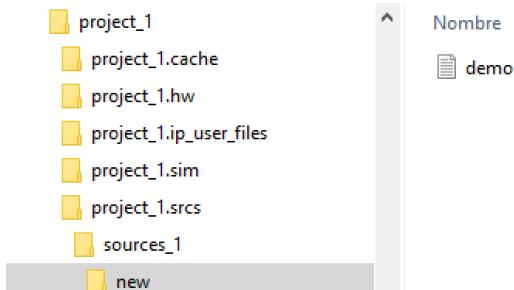


Figura 11. Estructura de directorios generada

### 3.2 EDICIÓN DE CÓDIGO VHDL

- Para incluir nuestro código VHDL tenemos dos opciones: añadir archivos ya existentes, o crear nuevos archivos .vhd. Como ya hemos creado uno al crear el proyecto, trabajaremos sobre ese fichero. En “Fuentes (sources)”, podemos hacer doble clic en nuestro archivo .vhd para editarlo (en el ejemplo “tutorial.vhd” o “demox.vhd”). Añadimos el siguiente código y lo guardamos:

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY UNISIM;
USE UNISIM.VComponents.all;
ENTITY p1 IS
  PORT (
    SW : in STD_LOGIC_VECTOR(15 downto 0);
    LED : out STD_LOGIC_VECTOR(15 downto 0)
  );
end p1;

ARCHITECTURE behavior OF p1 IS
Signal led_int : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
BEGIN
  LED <= led_int;
  led_int(0) <= not(SW(0));
  led_int(1) <= SW(1) and not(SW(2));
  led_int(3) <= SW(2) and SW(3);
  led_int(2) <= led_int(1) or led_int(3);
  led_int(7 downto 4) <= SW(7 downto 4);
END behavior;

```

- Una vez escrito el código podemos realizar un análisis RTL en el archivo fuente (parecido a compilar el código), para verificar que no haya errores. En el panel “Fuentes” (Sources), seleccione la entrada tutorial.vhd y haga clic en “Esquema” (Schematic) (puede que tenga que expandir la entrada “Abrir diseño elaborado” -Open Elaborated Design) en las tareas de Análisis RTL del panel de Flow Navigator. Se muestra una vista lógica del diseño.

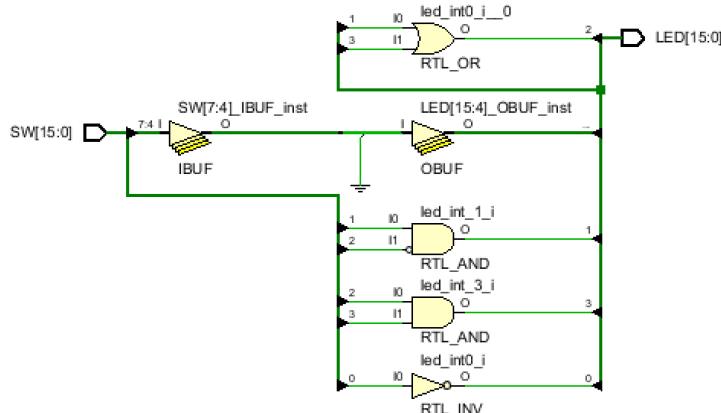


Figura 12. Una vista lógica del diseño

Tenga en cuenta que algunas de las entradas del interruptor pasan por las puertas antes de enviarse a los LEDs y el resto pasa directamente a los LEDs como se muestra en el modelo.

Una vez que se realiza el análisis RTL, se presenta otra vista llamada “I/O Planning Layout”. Para ello hay que hacer clic en el botón desplegable arriba a la derecha y selecciónelo. Pero eso lo veremos más adelante.

### 3.3 SIMULACIÓN

Tanto si hemos diseñado un circuito programando directamente en VHDL o utilizando esquemáticos, la simulación resulta un paso imprescindible para saber si los resultados son los esperados o por el contrario debemos revisar el diseño. Es lo que se llama una **verificación funcional**.

Para simular debemos crear un banco de pruebas (*testbench*) que generará las entradas con que excitar nuestro diseño. El banco de pruebas también puede contener código para comprobar que las salidas evolucionan de la forma correcta. Aunque esto último no es imprescindible, sí es muy recomendable. Permite automatizar las pruebas, acelerando el ciclo de diseño, y es la única forma razonable de comprobar simulaciones de cientos de miles de ciclos de reloj o de dispositivos complejos como un microprocesador con un elevado número de señales.

11. Pulsar el botón derecho del ratón con el cursor dentro del panel Sources en **Simulation Source** y seleccionar en el desplegable **Add Source**:

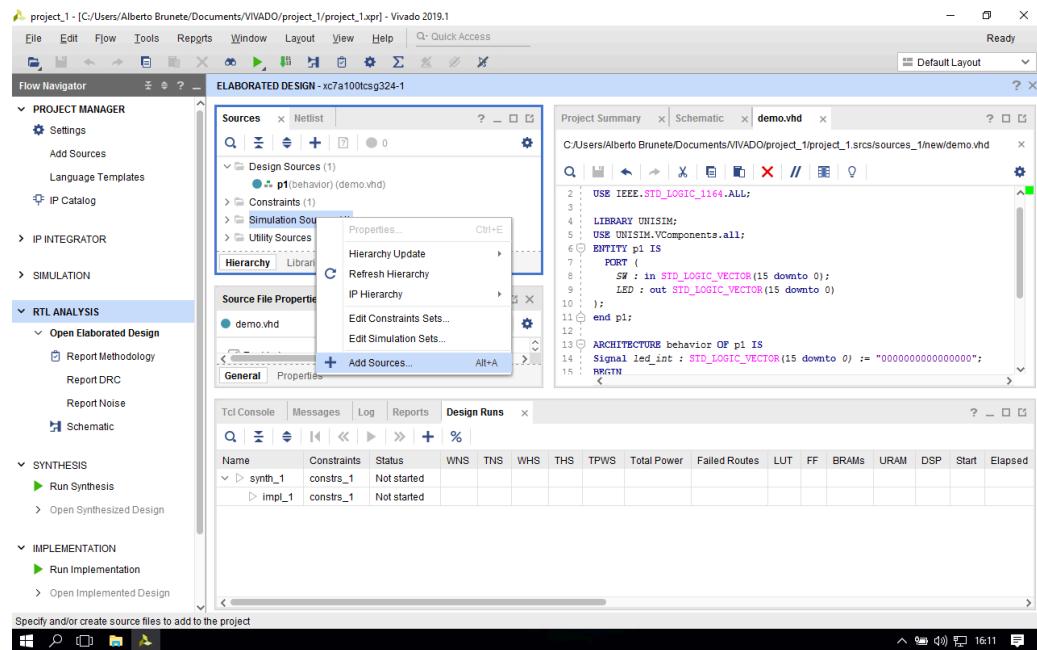


Figura 13. Añadir fuentes

12. En el diálogo de selección del tipo de módulo, elegir **Add or create simulation sources**.

13. Pulsamos **Create File**. Es una buena idea ponerle el de la entidad que se va a verificar con el sufijo “\_tb”. Pulsar **Next >**. No añadir señales.

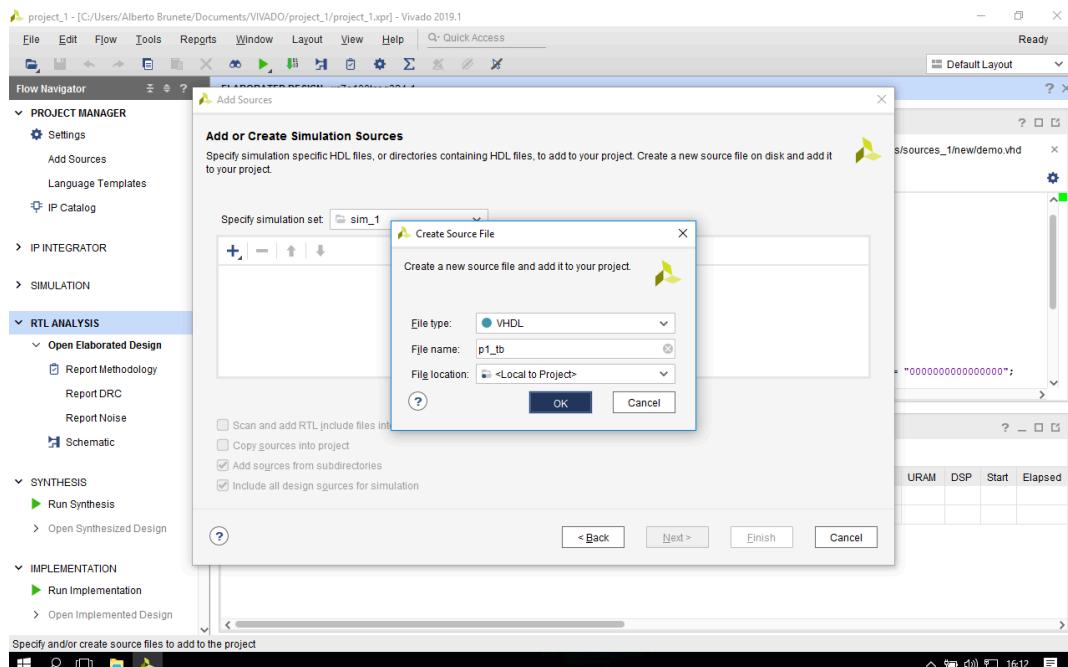


Figura 14: Generación del testbench

Se generará automáticamente un archivo de VHDL con una plantilla del *tesbench*.

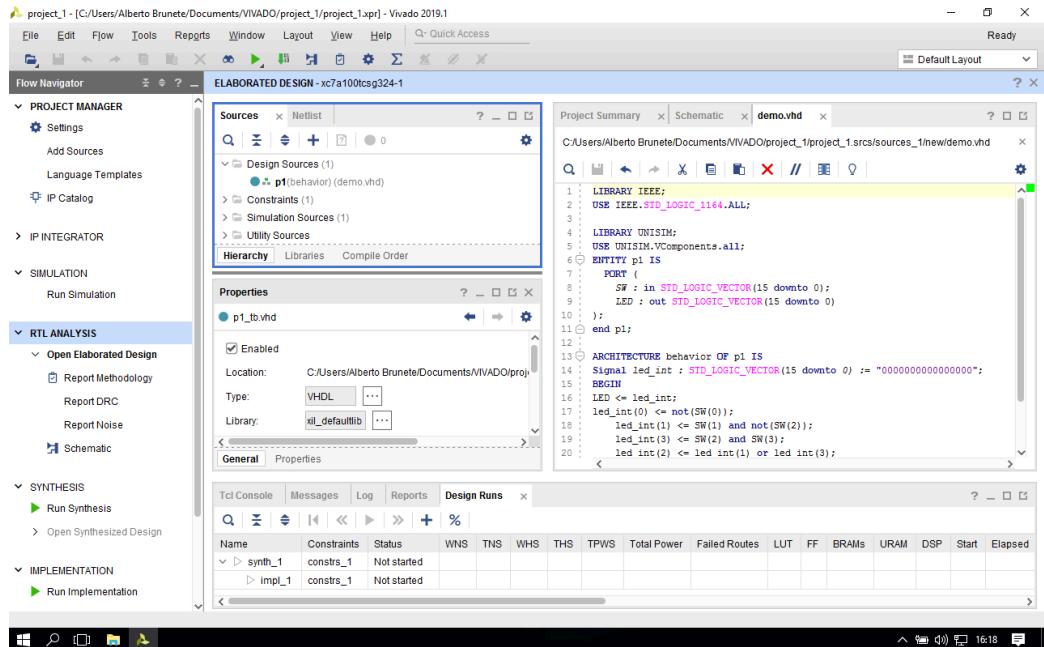


Figura 15: Testbench generado. Modo de visualización Implementation.

14. El *testbench* aparece en el gestor de proyectos dentro de la sección simulation sources.

15. Introducir en la arquitectura el código que se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use STD.textio.all;
use IEEE.std_logic_textio.all;

library UNISIM;
use UNISIM.VComponents.all;

Entity p1_tb Is
end p1_tb;

Architecture behavior of p1_tb Is
    Component p1
        port (
            SW : in STD_LOGIC_VECTOR(15 downto 0);
            LED : out STD_LOGIC_VECTOR(15 downto 0)
        );
    End Component;

    Signal switch : STD_LOGIC_VECTOR(15 downto 0) := X"0000";
    Signal led_out : STD_LOGIC_VECTOR(15 downto 0) := X"0000";
    Signal led_exp_out : STD_LOGIC_VECTOR(15 downto 0) := X"0000";

    Signal count_int_2 : STD_LOGIC_VECTOR(15 downto 0) := X"0000";

    procedure expected_led (
        swt_in : in std_logic_vector(15 downto 0);
        led_expected : out std_logic_vector(15 downto 0)
    )

```

```

) is

    Variable led_expected_int : std_logic_vector(15 downto
0):="0000000000000000";

begin
    led_expected_int(0) := not(swt_in(0));
    led_expected_int(1) := swt_in(1) and not(swt_in(2));
    led_expected_int(2) := swt_in(2) and swt_in(3);
    led_expected_int(3) := led_expected_int(1) or led_expected_int(3);
    led_expected_int(7 downto 4) := swt_in(7 downto 4);

    led_expected := led_expected_int;
end expected_led;

begin
uut: p1 PORT MAP (
    sw => switch,
    led => led_out
);

process
variable s : line;
variable i : integer := 0;
variable count : integer := 0;
variable proc_out : STD_LOGIC_VECTOR(15 downto 0);

begin
    for i in 0 to 127 loop
        count := count + 1;

        wait for 50 ns;
        switch <= count_int_2;

        wait for 10 ns;
        expected_led (switch, proc_out);
        led_exp_out <= proc_out;

        -- If the outputs match, then announce it to the simulator console.
        if (led_out = proc_out) then
            write (s, string'("LED output MATCHED at ")); write (s, count
); write (s, string'(". Expected: ")); write (s, proc_out); write (s,
string'(" Actual: ")); write (s, led_out);
            writeline (output, s);
        else
            write (s, string'("LED output mis-matched at ")); write (s,
count); write (s, string'(". Expected: ")); write (s, proc_out); write (s,
string'(" Actual: ")); write (s, led_out);
            writeline (output, s);
        end if;

        -- Increment the switch value counters.
        count_int_2 <= count_int_2 + 2;
    end loop;
end process;
end behavior;

```

Con el Explorador de Windows, verifique que el directorio sim\_1 se haya creado en el mismo nivel que el directorio sources\_1 en el directorio tutorial.srcts (ruta vista en

página 8), y que una copia de tutorial\_tb.vhd se encuentre en tutorial.srccs> sim\_1> new> sources.

16. Vamos a “Settings -> Simulation” y nos aseguramos de que en **simulation top module name** pone el nombre de nuestro testbench. Pinchamos en la pestaña “Simulation” y verificamos que el tiempo de simulación es **1000ns** (o lo que consideremos, luego se puede modificar).
17. Pulsamos sobre “Simulation -> Run simulation” en **Navigation Flow** y luego en el desplegable sobre “Run Behavioural Simulation”, lo que nos permite simular el código sin tener que sintetizarlo antes. Por tanto, no incluye retardos. Útil para verificar el comportamiento lógico (funcional) del diseño. Siempre se depura el diseño en este modo antes de pasar a simulaciones más detalladas.
18. Al finalizar la simulación se nos debería de abrir una ventana como la mostrada en la figura siguiente, la cual nos muestra las formas de onda de las señales a nivel del *testbench*. Debido al tamaño limitado de la ventana del simulador no se muestra la simulación completa. Para encoger la escala y visualizarla por completo pulsaremos el botón **Zoom Fit** en la barra de herramientas.

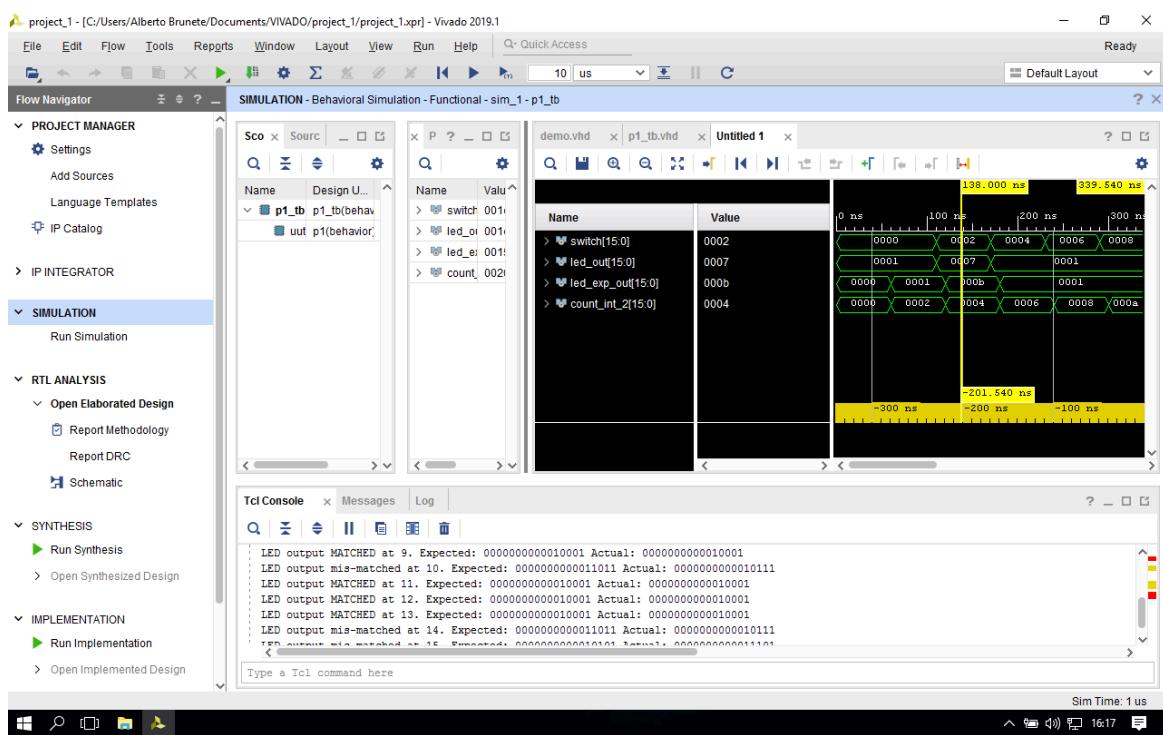


Figura 16: Simulación del testbench.

19. Verá resultados coincidentes y no coincidentes del UUT (Unit Under Test) en la consola, los cuales se insertaron deliberadamente para demostrar la funcionalidad TEXTIO del banco de pruebas VHDL.
20. El panel “objects” a la izquierda de la ventana del simulador nos permite visualizar la jerarquía de nuestro diseño y navegar a través de ella. De esta forma

es posible seleccionar y visualizar señales correspondientes a niveles inferiores de la jerarquía.

21. En la ventana de formas de onda también nos aparece un marcador que podemos mover arrastrándolo con el ratón y que permite conocer el instante de simulación sobre el que se encuentra. La escala de tiempo comienza en 0.
  22. Dele al “play” en la barra de herramientas principal y simule durante otros 500ns o el tiempo que considere.

### 3.4 SÍNTESIS

Una vez simulado el diseño y comprobado su funcionamiento, podemos pasar a la síntesis.

23. Sintetizamos el programa pinchando en “Síntesis->Run Síntesis” en Flow Navigator. Si todo va bien obtenemos la siguiente confirmación:

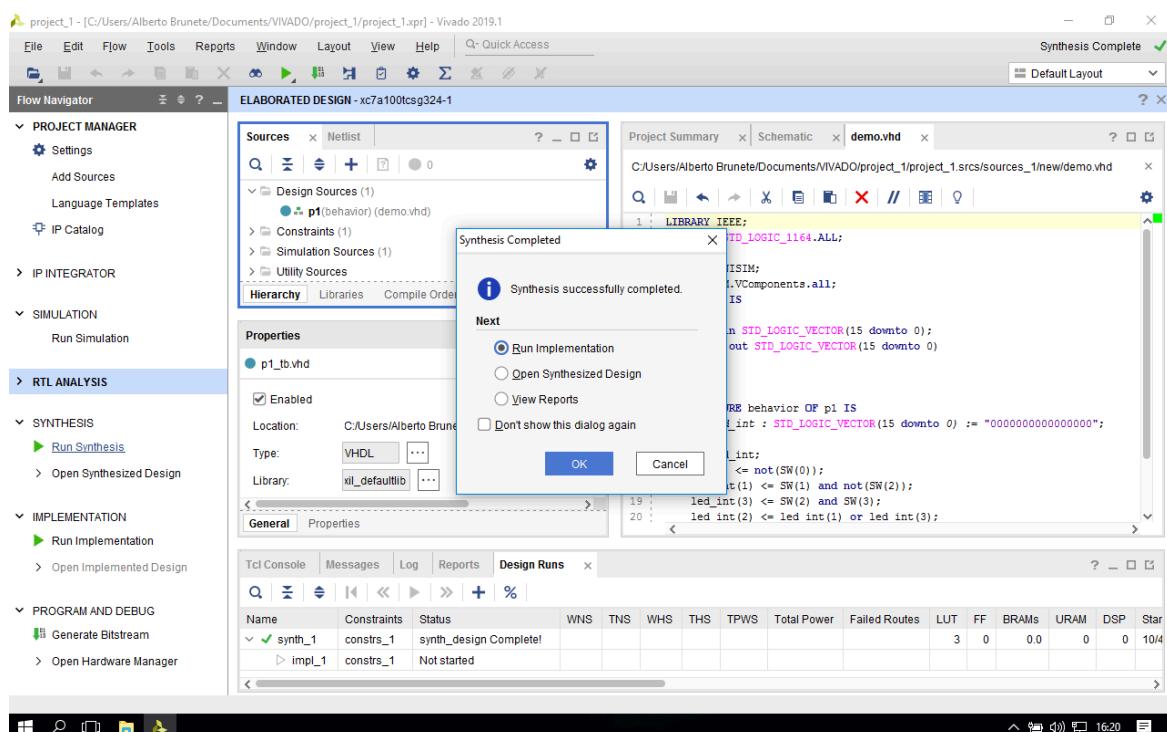


Figura 17. Síntesis completada

24. Podamos seleccionar “Abrir diseño sintetizado” en Flow Navigator y obtenemos el diseño sintetizado (se puede hacer zoom y ver el detalle de bajo nivel):

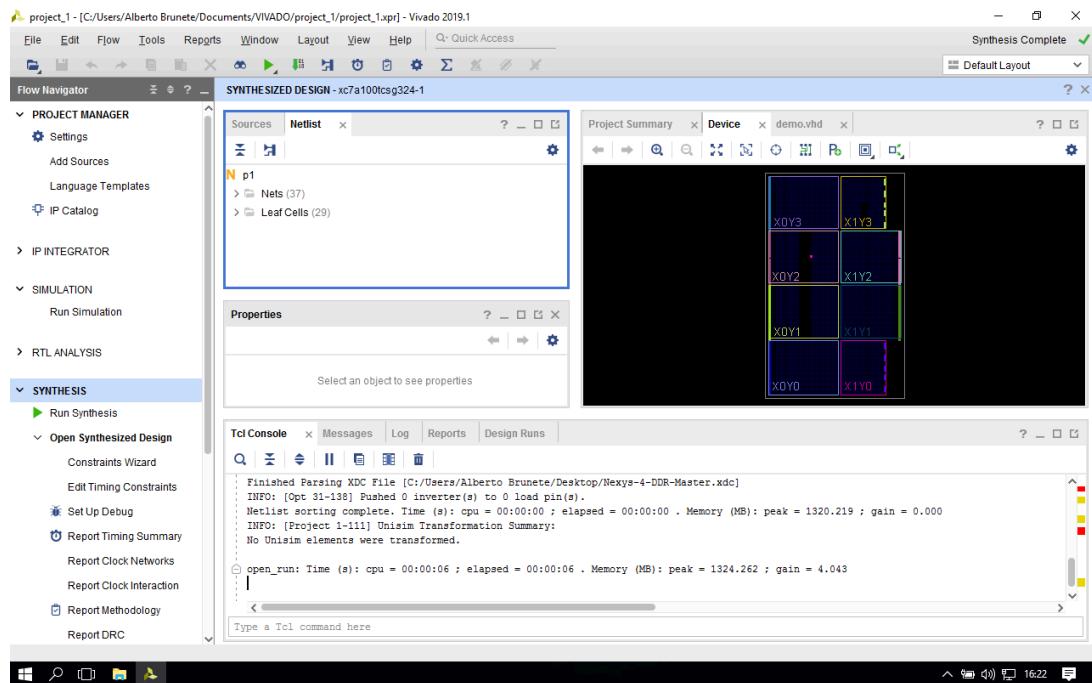


Figura 18. Diseño sintetizado

25. Seleccione la pestaña **Resumen del proyecto** y comprenda lo que muestran las ventanas, en especial es relevante los recursos utilizados (*utilization*).

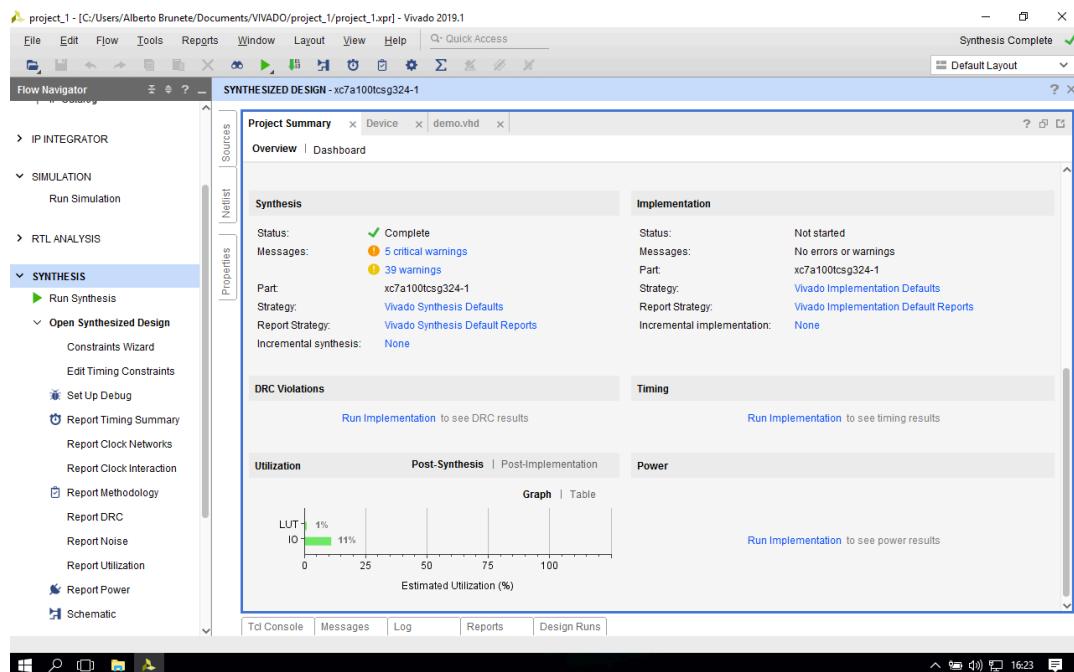


Figura 19. Resumen del proyecto

26. Haga clic en **Esquemático** debajo de las tareas de **Diseño sintetizado** de las tareas de **Síntesis** del panel de **Flow Navigator** para ver el diseño sintetizado en vista esquemática. Será algo parecido a la figura siguiente. Compruebe las diferencias con el esquemático anterior antes de sintetizar.

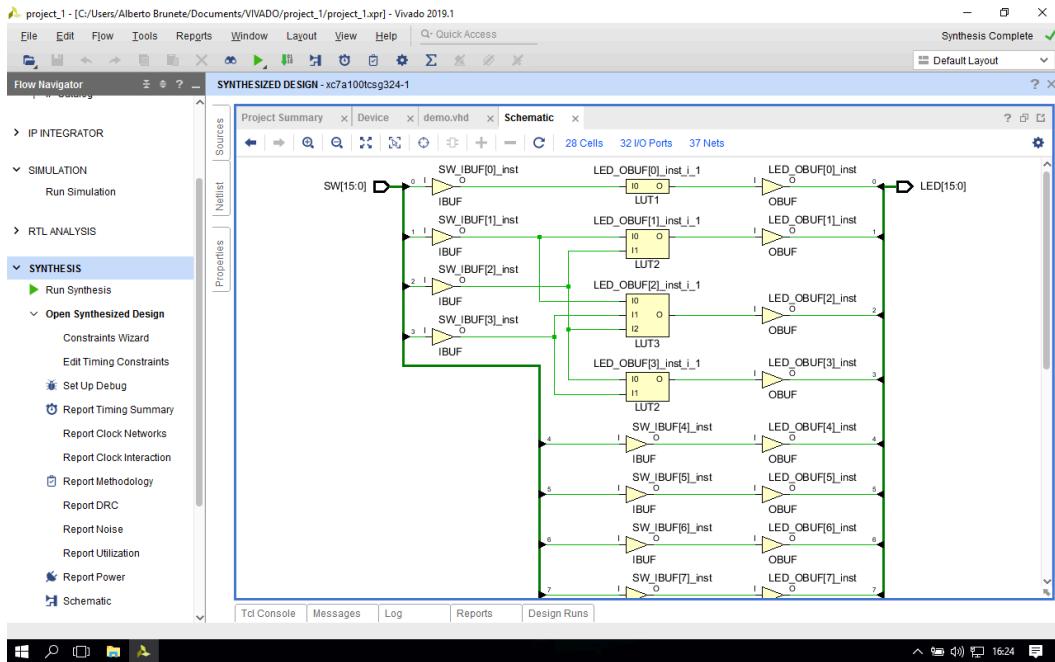


Figura 20. Esquemático del diseño sintetizado

Verá que las puertas lógicas se implementan en LUT. Cuatro puertas en la salida del análisis de RTL se mapean en cuatro LUT en la salida sintetizada. ¿Por qué la LUT 3 tiene 3 entradas?

Ya dispondríamos de un diseño listo para implementar en nuestra FPGA en la siguiente práctica.

--- Fin de la práctica ---

### 3.5 Resolución de errores

[1] Si obtenemos errores/warning parecidos a los de la siguiente figura se debe a que hay líneas que están sin comentar en el archivo .xdc



Figura 21. Posibles errores

```
1  ##Switches
2
3  set_property -dict { PACKAGE_PIN J15  IOSTANDARD LVCMOS33 } [get_ports { SW[0] }];
4  #IO_L24N_T3_RSO_15 Sch=sv[0]
5  set_property -dict { PACKAGE_PIN L16  IOSTANDARD LVCMOS33 } [get_ports { SW[1] }];
6  #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sv[1]
7  set_property -dict { PACKAGE_PIN M13  IOSTANDARD LVCMOS33 } [get_ports { SW[2] }];
8  #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
9  set_property -dict { PACKAGE_PIN R15  IOSTANDARD LVCMOS33 } [get_ports { SW[3] }];
10 #IO_L13N_T2_MRCC_14 Sch=sv[3]
11 set_property -dict { PACKAGE_PIN R17  IOSTANDARD LVCMOS33 } [get_ports { SW[4] }];
12 #IO_L12N_T1_MRCC_14 Sch=sv[4]
13 set_property -dict { PACKAGE_PIN T18  IOSTANDARD LVCMOS33 } [get_ports { SW[5] }];
14 #IO_L7N_T1_D10_14 Sch=sv[5]
```

Figura 22. Comentarios en el fichero .xdc

[2] VIVADO 2021.1. Si tiene problemas al abrir el editor de texto, y se queda atascado en “initializing language server”, una posible solución es: cambiar la opción "Tools -> Settings -> Tool Settings -> Text Editor -> Syntax Checking -> Syntax checking" de Sigasi a Vivado. Y reiniciar. Fuente:

[https://support.xilinx.com/s/question/0D52E00006hprvBSAQ/vivado-20211-stuck-initializing-language-server?language=en\\_US](https://support.xilinx.com/s/question/0D52E00006hprvBSAQ/vivado-20211-stuck-initializing-language-server?language=en_US)