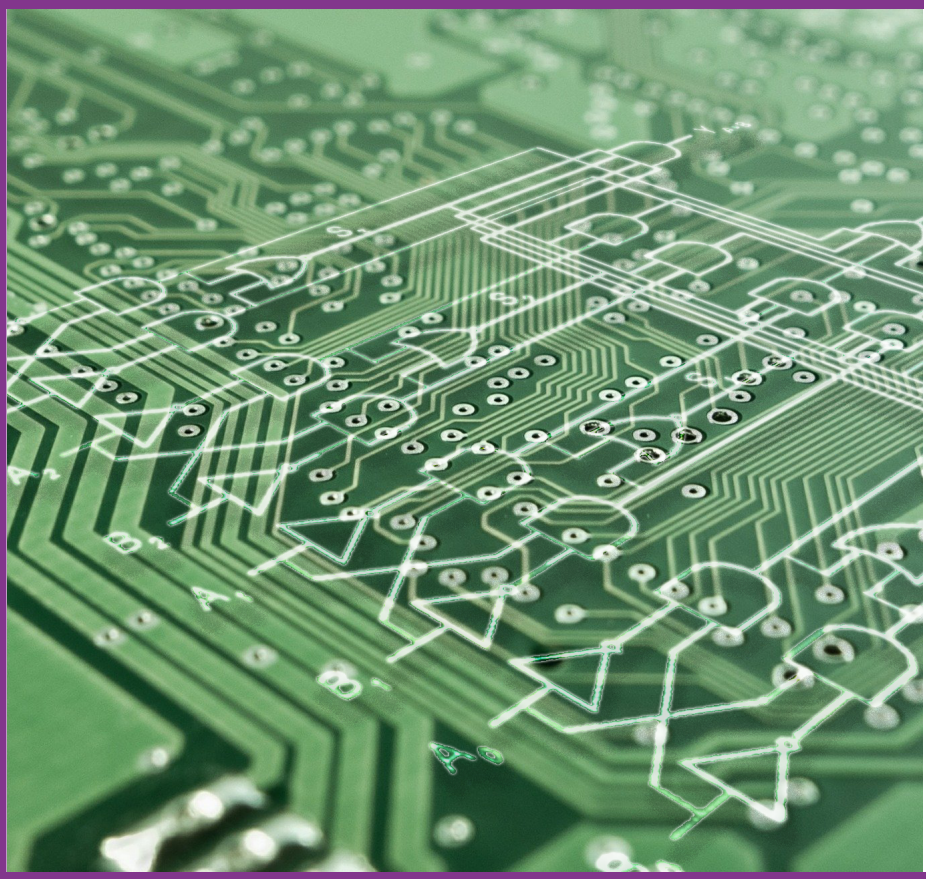
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Lerma

Dirección
Ciencias Básicas
e Ingeniería

Manual de prácticas de laboratorio para el curso Fundamentos de Diseño Lógico



Gerardo Abel Laguna Sánchez
Daniela Aguirre Guerrero
Karen Samara Miranda Campos

Departamento
de Recursos
de la Tierra

Departamento
de Procesos
Productivos

Departamento de
Sistemas de Información
y Comunicaciones

Universidad Autónoma Metropolitana
Unidad Lerma
Av. de las Garzas No. 10
Col. El Panteón, Lerma de Villada,
Estado de México. C.P. 52005.

www.ler.uam.mx

Primera Edición Digital:
Manual de prácticas de laboratorio para el curso
Fundamentos de Diseño Lógico

ISBN: 978-607-28-2517-8

Este trabajo ha sido revisado por pares, de conformidad con los lineamientos del Consejo Editorial de la División de Ciencias Básica de Ingeniería, de la Universidad Autónoma Metropolitana, Unidad Lerma.

Diseño editorial:
LDG. José Uriel Hernández Pérez

UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo

Dr. José Antonio De los Reyes Heredia
Rector General de la UAM

Dr. José Mariano García Garibay
Rector de la UAM Unidad Lerma

División de Ciencias Básicas e Ingeniería
UAM Lerma

Dr. Edgar López Galván
Director de División

Dr. Carlos Eduardo Díaz Gutiérrez
Secretario Académico

Dr. Daniel Librado Martínez Vázquez
Encargado del Departamento de Procesos Productivos

Dr. Yuri Reyes Mercado
Jefe del Departamento de Recursos de la Tierra

Dr. Guillermo López Maldonado
Jefe del Departamento de Sistemas de Información y Comunicaciones

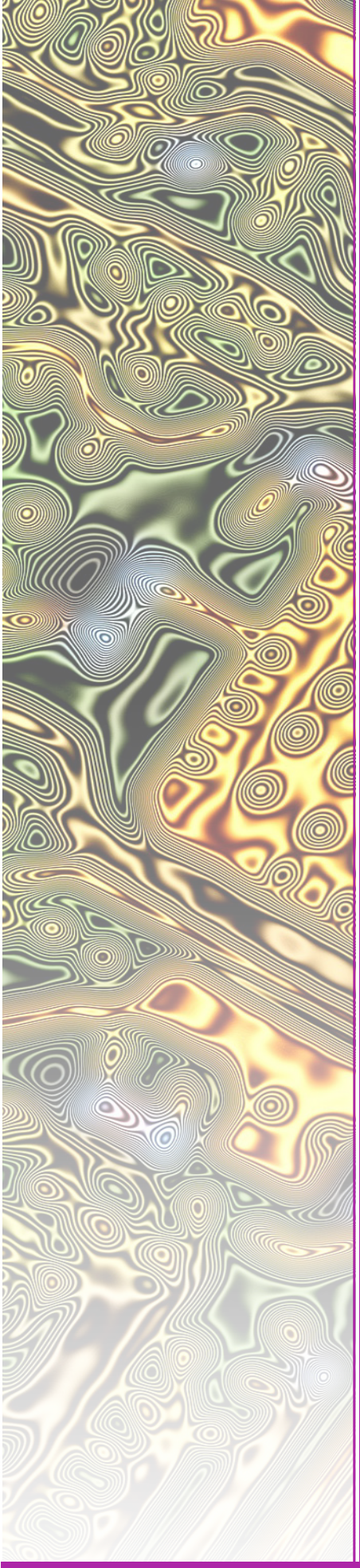
Dirección
Ciencias Básicas
e Ingeniería

Manual de prácticas de laboratorio para el curso Fundamentos de Diseño Lógico

Gerardo Abel Laguna Sánchez,
Daniela Aguirre Guerrero y
Karen Samara Miranda Campos

Marzo 2021

CONTENIDO



I. PRESENTACIÓN

Los circuitos lógicos son el principal elemento o "componente" de las computadoras, pero su utilidad no se limita a ellas. En la actualidad, vivimos en una sociedad donde los sistemas digitales son cada vez más ubicuos en una variedad cada vez más amplia de facetas. Desde los relojes digitales, las telecomunicaciones —pasando por los teléfonos celulares—, las consolas de videojuegos y las televisiones inteligentes, robots —capaces de realizar cirugías a distancia— hasta llegar a ser la piedra angular de las nuevas tecnologías como Internet de las Cosas. Por lo anterior, la motivación para crear este Manual de Prácticas es coadyuvar al apropiado entendimiento de los circuitos lógicos, lo que es particularmente importante tanto para los Ingenieros en Computación y Telecomunicaciones como para los Ingenieros en Sistemas Mecatrónicos Industriales.

El material que se presenta en este documento introduce al alumno en la aplicación práctica del Contenido Sintético de la Unidad de Enseñanza Aprendizaje (UEA) Fundamentos de Diseño Lógico, cuyo objetivo principal es que el alumno sea capaz de comprender el funcionamiento de los componentes digitales, así como conocer las bases para el diseño, descripción y programación de dispositivos lógicos. Lo cual se plasma en el Programa de Estudios de la UEA como:

Al final de la UEA el alumno será capaz de: ***analizar, diseñar y aplicar sistemas digitales tanto combinatorios como secuenciales.***

Contenido Sintético

1. Introducción a los sistemas digitales; elementos de diseño digital
2. Diseño de circuitos combinatorios
3. Diseño, programación y simulación de dispositivos de lógica programable mediante lenguajes de descripción de hardware
4. Registros y memorias
5. Fundamentos de máquinas secuenciales
6. Diseño de circuitos secuenciales
7. Diseño de máquinas de estado finitas

Para esta UEA, el desarrollo de habilidades prácticas de los alumnos es indispensable, ya que ello les permite aplicar y comprobar la teoría vista en el curso y, de esta manera, madurar los conceptos. Se trata de una estrategia didáctica que se puede sintetizarse en el aforismo del sabio chino Xunzi: "*Dímelo y lo olvidaré, enséñame y recordaré, involúcrame y aprenderé*". Así, es evidente que las prácticas de laboratorio son fundamentales para el proceso de enseñanza-aprendizaje del alumno.

Este Manual de Prácticas está diseñado para una modalidad presencial, donde los alumnos pueden asistir al laboratorio y disponer de los elementos necesarios para llevar a cabo cada una de las prácticas propuestas. En particular, utilizamos el enfoque del Diseño Asistido por Computadora (CAD, por sus siglas en inglés) para ilustrar la manera de diseñar y describir a los circuitos lógicos. Específicamente, nos apoyamos en el lenguaje de descripción de hardware definido por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) denominado como VHDL (o también VHSIC-HDL, por las siglas de *Very High*

Speed Integrated Circuit Hardware Description Language). Por otro lado, también recurrimos a materiales como tarjetas de prototipado, circuitos integrados de la familia 74LS y un circuito FPGA (por las siglas de *Field Programmable Gate Array*), mediante las facilidades que otorga una tarjeta de desarrollo denominada comercialmente como Basys 3, entre otros recursos de apoyo.

Este manual comprende diez prácticas de laboratorio, cada una presentada junto con su objetivo, antecedentes, materiales, equipo disponible en el laboratorio, procedimiento y actividades a realizar. Asimismo, también se incluyen once anexos, con relación a las sugerencias para la entrega de los reportes, los tutoriales paso a paso y los códigos fuente VHDL que apoyan y complementan las prácticas. La ruta de trabajo planteada en este manual aspira a que el alumno vaya comprendiendo y aplicando los conceptos desarrollados en la parte teórica del curso de Fundamentos de Diseño Lógico de forma incremental, es decir, paso a paso.

Aunque la descripción de las prácticas presentadas es específica para el equipo y los recursos disponibles en los laboratorios de la Unidad Lerma de la UAM, es relativamente sencillo que cualquiera de estas prácticas pueda llevarse a cabo con equipos o recursos similares, ya sea para adaptarse a las condiciones particulares de otros laboratorios, o bien, para ampliar la oferta de sesiones prácticas en otras UEA semejantes dentro de la misma Universidad Autónoma Metropolitana o, en general, de cualquier otra institución de educación superior.

Por último, se sugiere ampliamente a los profesores que impartan la UEA Fundamentos de Diseño Lógico, que consideren apoyarse en el enfoque del Eje Integrador del Modelo Educativo de la Unidad

Lerma a fin de incluir actividades colaborativas entre los alumnos como investigar, comprender lecturas o responder cuestionarios.

Generalidades sobre las bases tecnológicas empleadas

Circuitos integrados de la serie 74LS

Es una familia de circuitos lógicos integrados que incluyen compuertas lógicas básicas tales como AND, OR, NAND, NOR y NOT. La familia 74LS emplea diodos Schottky (*Lower Power Schottky*) y Lógica de Transistor a Transistor (TTL) que, aunque es robusta, requiere mayor consumo de energía con respecto a otras familias más recientes como, por ejemplo, la familia 74HC. Los circuitos que se consideran en este trabajo corresponden a los siguientes tipos de compuertas:

74LS00 - 4 compuertas NAND de dos entradas.
74LS04 - 6 compuertas inversoras (NOT).

Tarjeta de desarrollo modelo Basys 3 con FPGA Artix-7 XC7A35T

Un FPGA, del inglés *Field-Programmable Gate Array*, es un dispositivo de lógica programable con base en tecnología de semiconductores para disponer de matrices con bloques de compuertas lógicas configurables. Su interconexión puede configurarse o "programarse", según se requiera, conforme a las características de un proyecto en particular. La configuración de un circuito digital se realiza a través de un lenguaje de descripción de hardware (HDL, por sus siglas en inglés) como, por ejemplo, VHDL o Verilog. Gracias a su gran flexibilidad para

programarse y reprogramarse, sus aplicaciones se han incrementado hasta cubrir campos tan diversos como lo son el aeroespacial, el automotriz, los centros de datos y el cómputo de alto rendimiento.

Los dispositivos FPGA básicos se componen principalmente, en su variante sustentada por SRAM, de arreglos de memoria, conexiones programables, celdas lógicas, flip-flops, y puertos de entrada y salida. Las versiones más modernas de los FPGA ya son, en realidad, sistemas del tipo SoC (por las siglas en inglés de *System on Chip*), es decir que ya incluyen periféricos y unidades centrales de procesamiento, como lo es el caso de la familia Artix-7. En particular, la tarjeta de desarrollo modelo Basys 3 es una plataforma de experimentación, de bajo costo, lista para usarse y que, además, incluye interfaces USB y VGA. La tarjeta de desarrollo Basys 3 puede soportar desde diseños para circuitos combinacionales básicos hasta circuitos secuenciales complejos.

Para mayor información, sobre la arquitectura y funciones de la familia Artix-7, el lector puede remitirse a la documentación completa que se encuentra disponible, en línea, en la página del fabricante:
https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html

Para mayor información sobre la tarjeta Basys 3 y su documentación completa, el lector puede remitirse a la página del fabricante:
https://reference.digilentinc.com/reference/programmable-logic/basys-3/start

Simbología básica

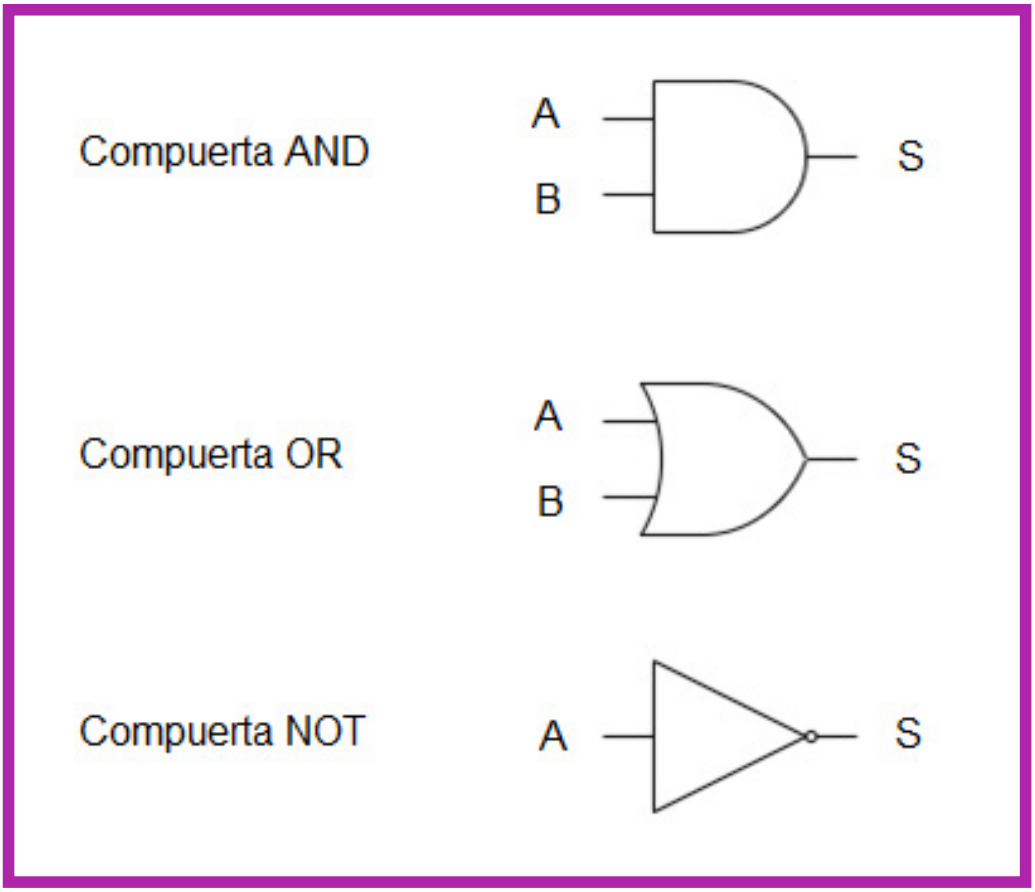


Figura I. Simbología básica para el diseño lógico.

Antes de comenzar

Se sugiere que los alumnos tomen la Unidad de Enseñanza Aprendizaje Fundamentos de Diseño Lógico durante el Trimestre II del Plan de Estudios de su licenciatura. Esta UEA no requiere haber llevado ninguna otra UEA previamente, lo que en la UAM se conoce como seriación, por lo que Fundamentos de Diseño Lógico es el primer acercamiento de los alumnos a los sistemas digitales y no es necesaria ninguna experiencia en diseño digital. No obstante, dado que los circuitos lógicos son la base de los sistemas computacionales modernos, se recomienda que los alumnos se empapen sobre algunos temas para aprovechar de mejor manera las prácticas descritas en este manual.

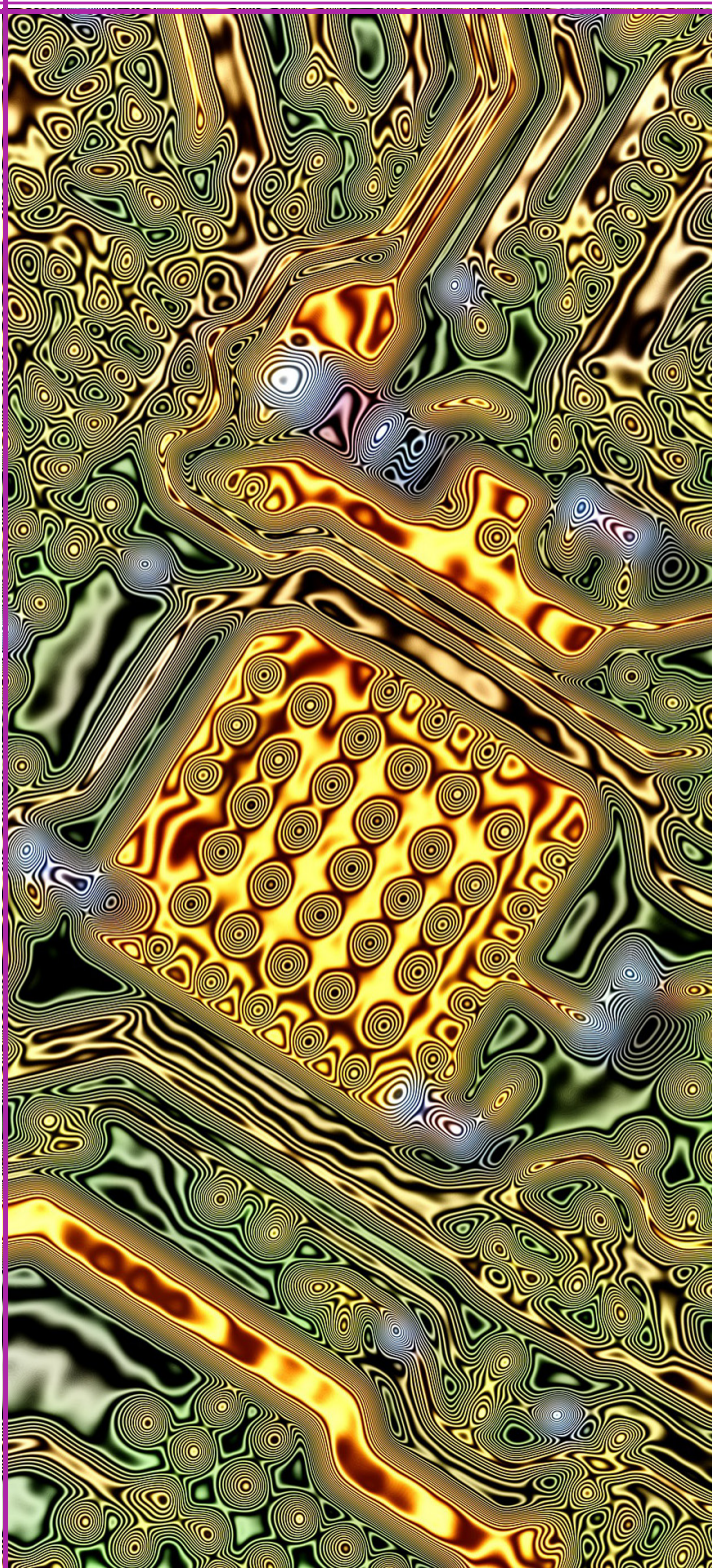
En particular recomendamos revisar:

- Series y progresiones numéricas
- Sistemas numéricos (binario, octal y hexadecimal)
- Igualdades y desigualdades
- Carga, corriente, voltaje, potencia y energía
- Ley de Ohm
- Resistencias y conductancias

Evidentemente, durante las sesiones teóricas que implica esta UEA, se abordarán los temas propios necesarios para realizar cada una de las prácticas, ya que ambos van de la mano.

PRÁCTICA 1

Implementación de circuitos lógicos combinatorios con componentes discretos



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVOS

Construir y probar un circuito lógico combinatorio a partir de componentes lógicos, electrónicos y eléctricos discretos.

ANTECEDENTES

Los circuitos lógicos combinatorios son componentes de hardware que procesan información binaria, de modo que sus salidas dependen de la combinación de las entradas que reciben. Por ejemplo, para un circuito combinatorio de dos entradas y una salida, se tienen cuatro posibles combinaciones de entradas y para cada una de ellas hay un valor de salida definido. De este modo, el comportamiento de un circuito combinatorio puede ser descrito por medio de una tabla de verdad que indica el valor salida para cada combinación de entradas.

Los circuitos combinatorios se construyen a partir de la interconexión de un conjunto de circuitos básicos llamados compuertas lógicas, las cuales ejecutan las tres operaciones lógicas básicas: AND, OR y NOT. Es importante mencionar que, a nivel físico, las operaciones lógicas se implementan a través de la interconexión de dispositivos de conmutación, por ejemplo, transistores o diodos, pero también pueden construirse usando relés (electromagnéticos, neumáticos, etc.) o incluso con elementos mecánicos. Sin embargo, los circuitos combinatorios se diseñan abstrayendo los detalles de estos dispositivos de conmutación y se presentan como un conjunto compuertas lógicas. En la actualidad, es posible embeber desde miles hasta millones de compuertas lógicas en un pequeño circuito integrado para implementar circuitos combinatorios más complejos. En esta práctica se construirá y probará un circuito lógico combinatorio a partir de componentes discretos.

MATERIAL

(el alumno lo debe llevar)
- Tarjeta de prototipado (protoboard).
- 1 circuito 74LS00 o equivalente, con su hoja técnica.
- 1 circuito 74LS04 o equivalente, con su hoja técnica.
- 1 m. de cable UTP CAT5e y 1 pinza de corte, o bien juego de cables tipo Dupont, plug a plug, calibre 28 AWG.
- 1 LED, con su hoja técnica.
- 1 resistencia de 180 ó 470 Ohms de 1/4W.
- 2 resistencias de 10 kOhms, 1/4 W.
- 2 micro switches (push buttons) con 4 terminales.

Equipo
(disponible en el laboratorio)

- Fuente de poder.
- Multímetro de mano.

PROCEDIMIENTO

Considere el siguiente diagrama

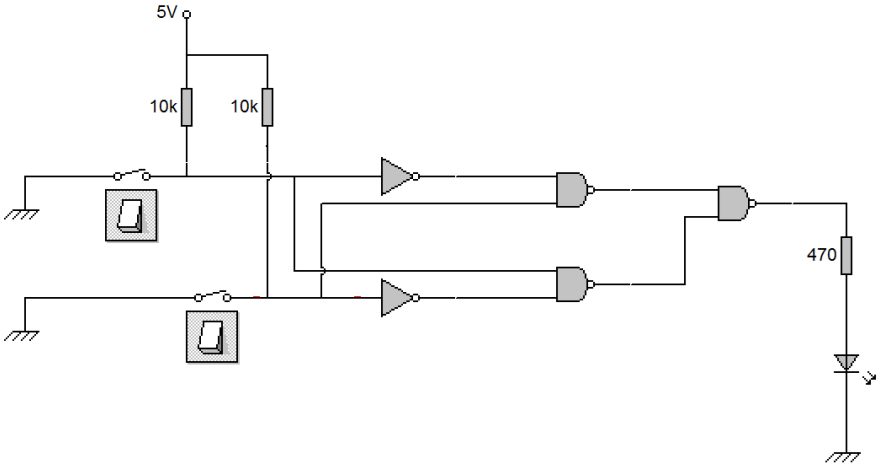


Figura 1.1. Circuito lógico de la Práctica 1.

Construya el circuito de Figura 1.1, sobre una tarjeta de prototipado, usando compuertas 74LS00 y 74LS04, push buttons y LED, con sus respectivas resistencias de pull-up (10 kOhms) y limitadoras (180 Ohms ó 470 Ohms). Obtenga la tabla de verdad del circuito.

ACTIVIDAD

Realice el diseño del circuito, consultando las hojas de especificación de los componentes, para hacer referencia a las terminales de alimentación de potencia (Vcc y GND) y las que se van a conectar entre sí. Los micro switches se pueden usar para proporcionar las entradas y los LED para señalar el estado de las salidas (recuerde que todo LED requiere una resistencia limitadora o corre el riesgo de inutilizar al LED y al circuito que lo alimenta). Después, defina la lógica (positiva/negativa) con que operará el LED y conéctelo de conformidad con las especificaciones técnicas.
Al probar su diseño en la tarjeta de prototipado, no olvide tomar las precauciones necesarias para garantizar la integridad de los equipos y componentes electrónicos empleados. Por equipo y siguiendo las indicaciones del Anexo 1, realice el reporte de esta práctica.

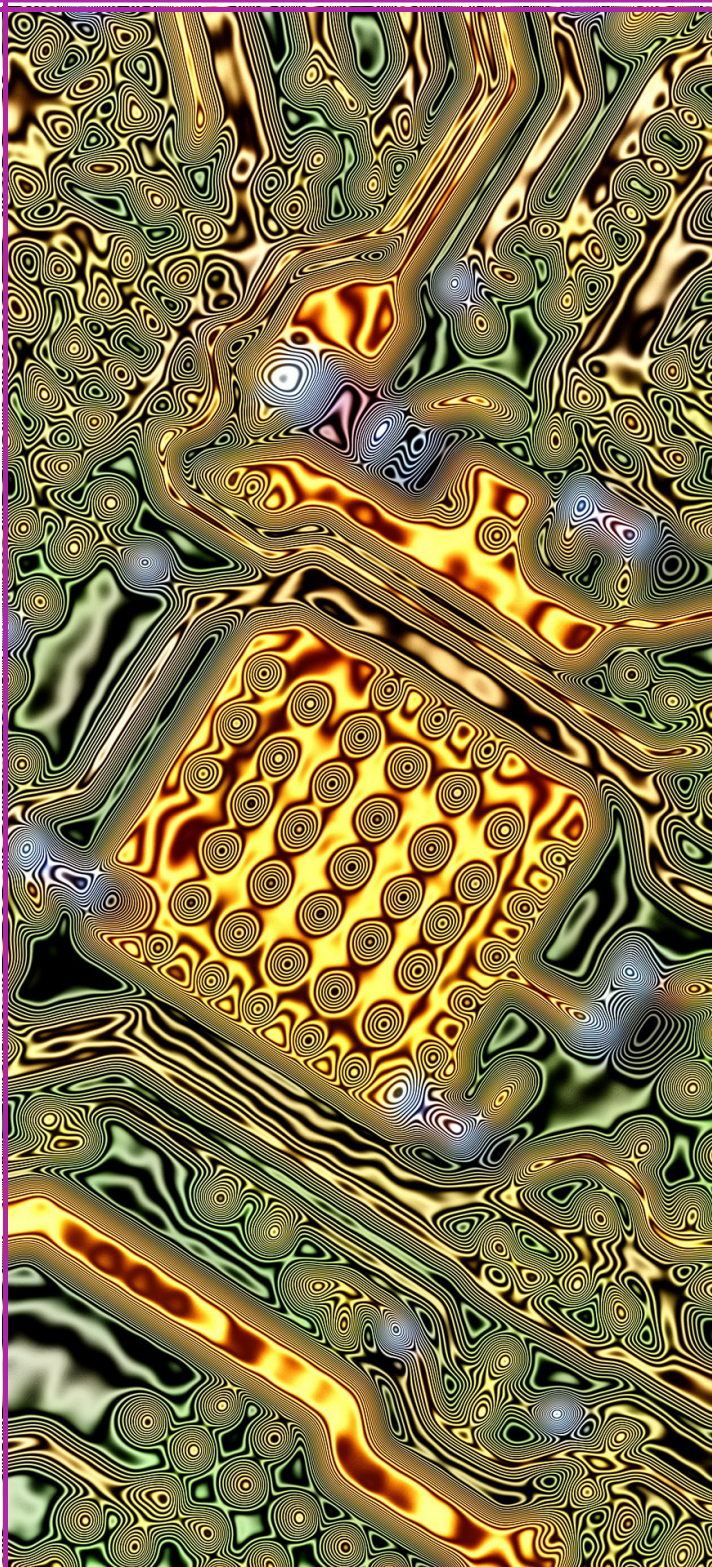
Referencias para consulta

- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.

- Laguna, G. (2020). Video LAB DL 04 (HD). WWW: Youtube.
  Enlace: https://youtu.be/Vi84ffedRbI

PRÁCTICA 2

Desarrollo de proyectos con lenguaje de descripción de hardware



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVOS

Conocer el entorno de desarrollo Vivado para diseño lógico e identificar las etapas básicas en la construcción de un proyecto con Lenguaje de Descripción de Hardware para circuitos integrados de alta velocidad (VHDL, por sus siglas en inglés).

ANTECEDENTES

Vivado es un entorno de desarrollo, proporcionado por el fabricante Xilinx, que funge como software de Automatización de Diseño Electrónico (EDA, por sus siglas en inglés) y permite la realización de un diseño lógico mediante los lenguajes de descripción de hardware VHDL y Verilog. Las prácticas presentadas en este manual se realizan en lenguaje VHDL. En este lenguaje, el diseño del hardware se especifica a través de un conjunto de módulos que constan de dos componentes básicos: 1) la declaración de la entidad, en donde se definen los puertos del hardware; y 2) el bloque de la arquitectura, en donde se realiza la descripción funcional del hardware.

Durante la creación de un proyecto en el entorno de desarrollo Vivado es importante identificar dos tipos de archivos básicos: 1) los archivos fuente que son de tipo VHDL (*.vhd), donde se definen los módulos a través de la declaración de su entidad y su arquitectura; y 2) el archivo de restricciones que tiene extensión xdc, donde se definen las restricciones de hardware, es decir las condiciones específicas en la que van a operar los puertos y las terminales del hardware disponible. En el caso de la tarjeta Basys 3, el archivo genérico de restricciones (es decir, una plantilla o template, en inglés, que contiene un ejemplo con todas las posibles restricciones para los puertos del FPGA Artix-7 XC7A35T, en términos de las conexiones específicas de la tarjeta

Basys 3) puede descargarse desde el sitio web de su fabricante (www.xilinx.com).

Una vez que se ha creado un proyecto VHDL, añadiendo sus archivos fuente y su archivo de restricciones, se procede con la construcción del proyecto que consiste en las etapas de síntesis e implementación del hardware. Así, la herramienta de síntesis de Vivado traduce el código VHDL en una lista de componentes electrónicos interconectados (netlist) y, luego, la herramienta optimiza la netlist para su implementación en un FPGA específico (en nuestro caso, el Artix-7 XC7A35T de la tarjeta Basys 3). Finalmente, se procede a la generación de la cadena de bits de programación (bitstream) que será utilizada para programar el FPGA destino en la tarjeta de desarrollo.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 5, que a continuación se describen:

- **bin_counter_25b.vhd**. Contiene el código del módulo binary_counter25 que define un contador binario de 25 dígitos y salidas de la 0 a la 24.
- **Practica02.vhd.** Contiene el código de alto nivel (top level) que genera una instancia del componente binary_counter25 (definida en el archivo bin_counter_25b.vhd).
- **Practica02.xdc.** Contiene las restricciones y la asignación de terminales definidas por el usuario, específicamente para el hardware usado.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. **Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto a partir de los códigos: Practica02.vhd, bin_counter_25b.vhd, Practica02.xdc. El proyecto se debe llamar "Practica02" (ver Anexo 2).**
2. **Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).**
3. **Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).**

ACTIVIDAD

Identifique las secciones, dentro del entorno Vivado, donde se muestra el progreso de las etapas de síntesis, implementación y generación de cadena de bits para la programación. Al finalizar la secuencia indicada, ¿qué información se reporta y dónde aparece?

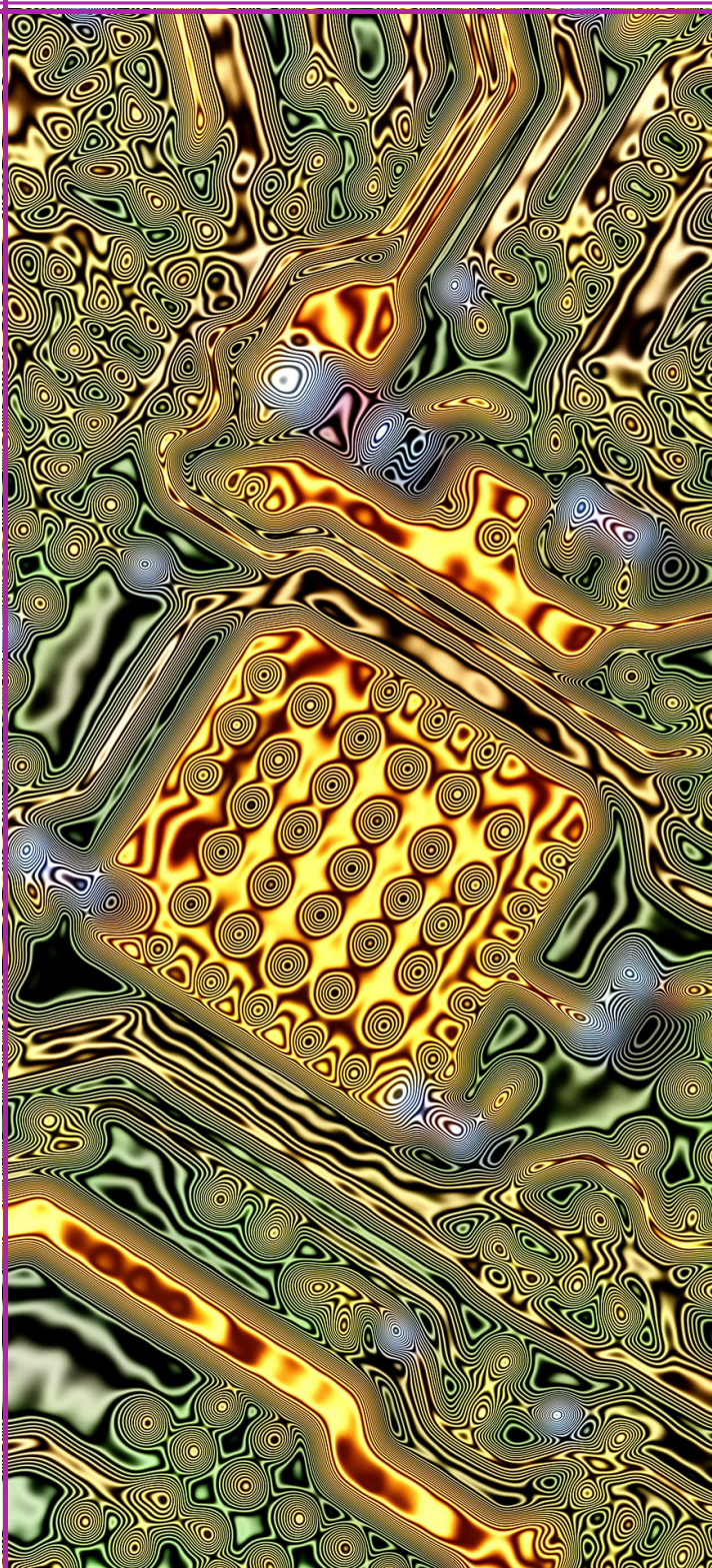
Por equipo y siguiendo las indicaciones del Anexo 1, realice el reporte de esta práctica. Dentro del marco teórico de su reporte, haga énfasis en las etapas del proceso de construcción de un diseño lógico con la herramienta Vivado (síntesis, implementación y generación de cadena de bits para la programación).

Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 3

Descripción de hardware de tipo estructural y funcional



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVOS

Desarrollar un proyecto VHDL básico definiendo y aplicando componentes, así como reconociendo los dos principales estilos de descripción de hardware: el estructural y el funcional.

ANTECEDENTES

Los módulos de hardware en VHDL constan de dos componentes básicos: 1) la declaración de la entidad, en donde se definen los puertos del hardware; y 2) el bloque de la arquitectura, en donde se realiza la descripción del hardware con la funcionalidad deseada. Es posible aplicar dos enfoques para la descripción de hardware: la descripción funcional (o de comportamiento) y la descripción estructural (o de componentes constructivos).

En la descripción estructural, se describen los componentes del hardware y las conexiones (señales) entre ellos, es decir que se describe la estructura del hardware. De este modo, la descripción estructural es equivalente a una transcripción, en código VHDL o Verilog, del circuito esquemático o a bloques del diseño lógico.

En la descripción funcional, se especifica el funcionamiento (o comportamiento) del hardware, ciertamente mediante funciones booleanas, pero, sobre todo, aprovechando al máximo la versatilidad de las sentencias y los constructos del lenguaje de descripción de hardware, a saber: sentencias de selección concurrente y sentencias de control de flujo dentro de procesos secuenciales. Para ello, no es tan importante contar con un diagrama esquemático del hardware sino, más bien, saber con precisión lo que se requiere del hardware, es decir cómo cambian las salidas en función de las entradas.

La Figura 3.1 muestra representaciones de las descripciones funcional y estructural. Note que, en la descripción funcional, las funciones booleanas se comunican mediante interconexiones, mientras que, en la descripción estructural, los componentes se comunican mediante señales.

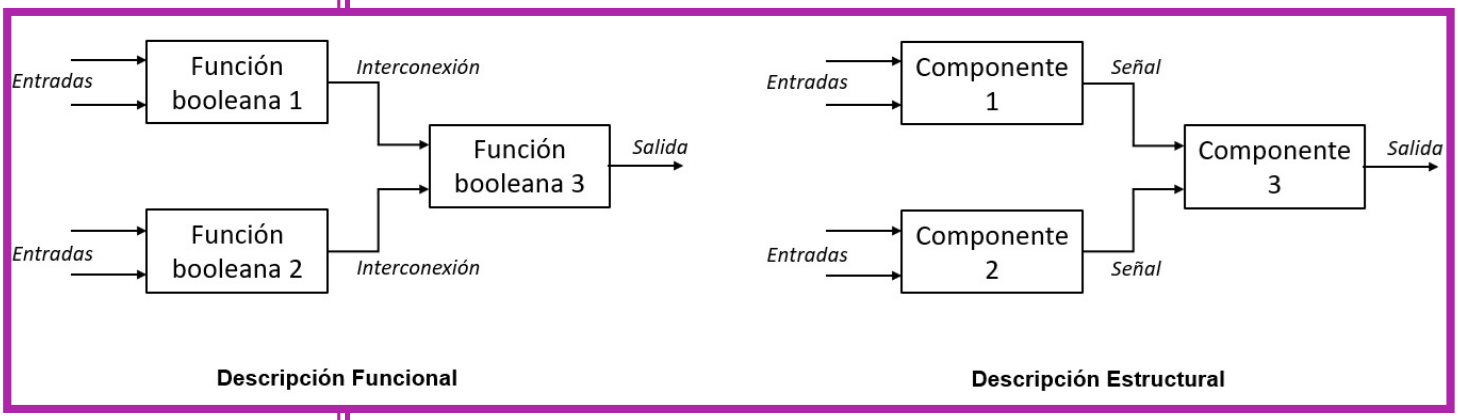


Figura 3.1. Representaciones de las descripciones de hardware funcional y estructural.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 6, que a continuación se describen:

- **miXORa.vhd.** Contiene el código de una compuerta lógica XOR descrita de forma estructural.
- **miXORb.vhd.** Contiene el código de una compuerta lógica XOR descrita de forma funcional.
- **Practica03a.vhd.** Contiene el código de alto nivel (top level) que genera una instancia de la versión estructural de la compuerta XOR.
- **Practica03b.vhd.** Contiene el código de alto nivel (top level) que genera una instancia de la versión funcional de la compuerta XOR.
- **Practica03.xdc.** Contiene las restricciones y la asignación de terminales definidas por el usuario, específicamente para el hardware usado.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. **Iniciar la aplicación Xilinx Vivado y cree un nuevo proyecto a partir de los códigos Practica03a.vhd, miXORa.vhd, Practica03.xcd. El proyecto se debe llamar "Practica03" (ver Anexo 2).**
2. **Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).**
3. **Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).**

ACTIVIDAD

Práctica 3

Estudie cuidadosamente el diagrama esquemático de la tarjeta de desarrollo Basys 3, además lea el archivo de restricciones *Practica03.xdc*, que como se menciona en la sección Procedimiento, genera una instancia de la compuerta XOR descrita en forma funcional. Igualmente, del archivo *Practica03.xdc*, identifique las referencias específicas que se hacen sobre la tarjeta de desarrollo utilizada en esta práctica. A partir del código proporcionado en el Anexo 6, realice los cambios necesarios para usar otros interruptores (sw) de entrada, o bien otro LED de salida.

Finalmente, construya un nuevo proyecto y aplique lo aprendido hasta el momento para desarrollar una segunda implementación de la misma función XOR, pero esta vez usando una descripción de hardware en forma funcional (*behavioral*), en lugar de estructural. Para esto, emplee los códigos *miXORb.vhd y Practica03b.vhd*. Compare ambos proyectos, estructural y funcional, e identifique las principales diferencias entre ambas implementaciones.

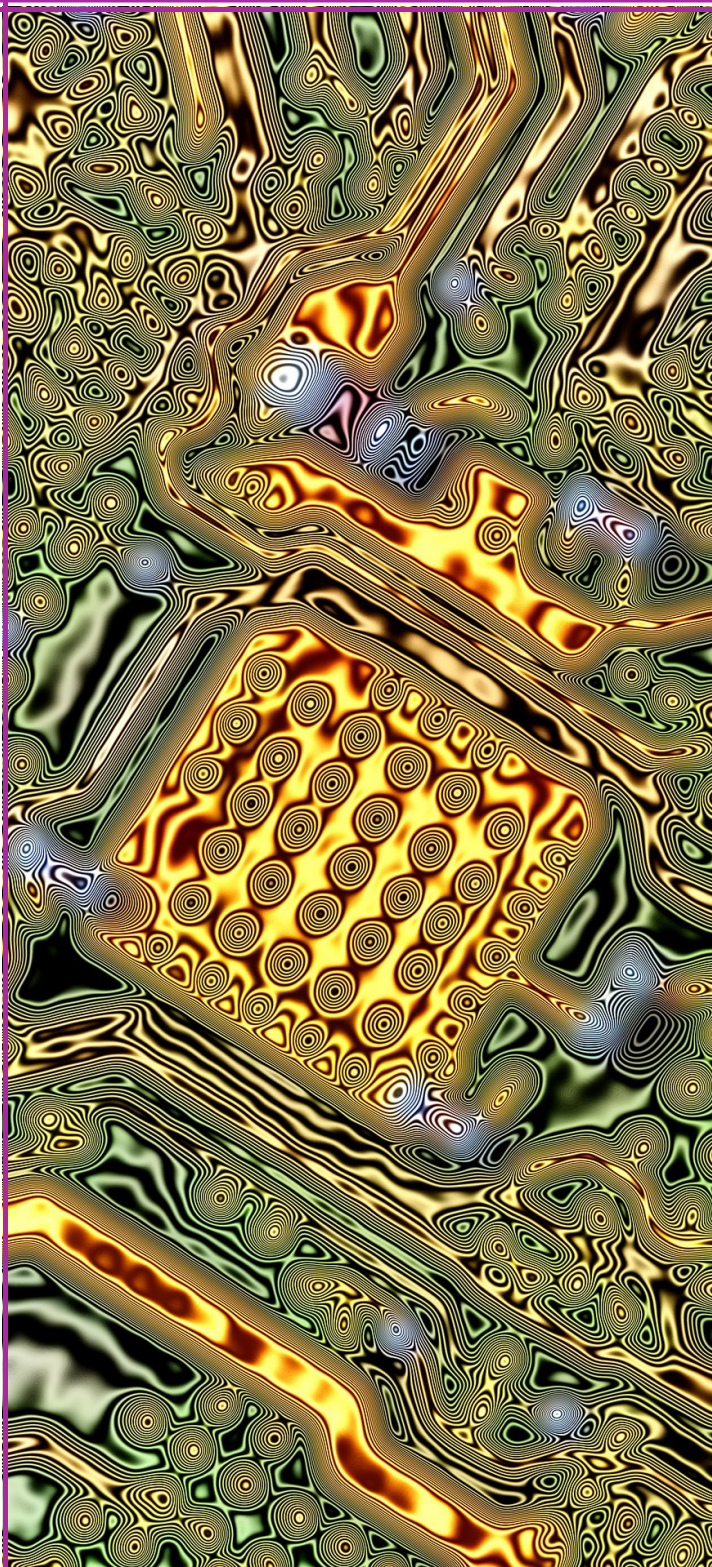
Por equipo, siguiendo las indicaciones del Anexo 1, realice el reporte de esta práctica. En el reporte, exprese con sus palabras cuáles son las diferencias principales entre el estilo de descripción de hardware estructural y el funcional, y cuáles son las principales ventajas y desventajas de implementar la función XOR con cada una de las descripciones de hardware.

Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 4

Definición semiautomática de componentes en el entorno de desarrollo Vivado



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVOS

Desarrollar un proyecto VHDL básico definiendo componentes en forma semiautomática mediante las herramientas del entorno Vivado.

ANTECEDENTES

En el entorno de desarrollo Vivado, los componentes del hardware se definen mediante módulos, en un archivo fuente, en donde se define la entidad y su arquitectura. En la parte de la entidad se indican los puertos de entrada y salida del componente, mientras que en la parte de la arquitectura se realiza la descripción, estructural o funcional, de la operación del componente. Como se muestra en esta práctica, el entorno de desarrollo Vivado también permite la definición semiautomática de los componentes de hardware, mediante un asistente o wizard, partiendo de las entradas y salidas que requieren en el diseño lógico.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

1. **Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto vacío, es decir que no agregue ningún archivo. El proyecto se debe llamar "Practica04" (ver Anexo 2).**

2. **Crear en forma semiautomática un nuevo módulo VHDL con una entidad y su arquitectura.**

En la ventana del margen izquierdo, "F*low Navigator*", ir a la sección "*Project Manager*" y oprimir el icono "*Add Sources*". En la ventana que aparece seleccionar la opción "*<*> Add or create design sources*", luego oprimir el botón *<Next>*. A continuación, en la nueva ventana *"Add Sources"*, oprimir el botón *<Create File>*.

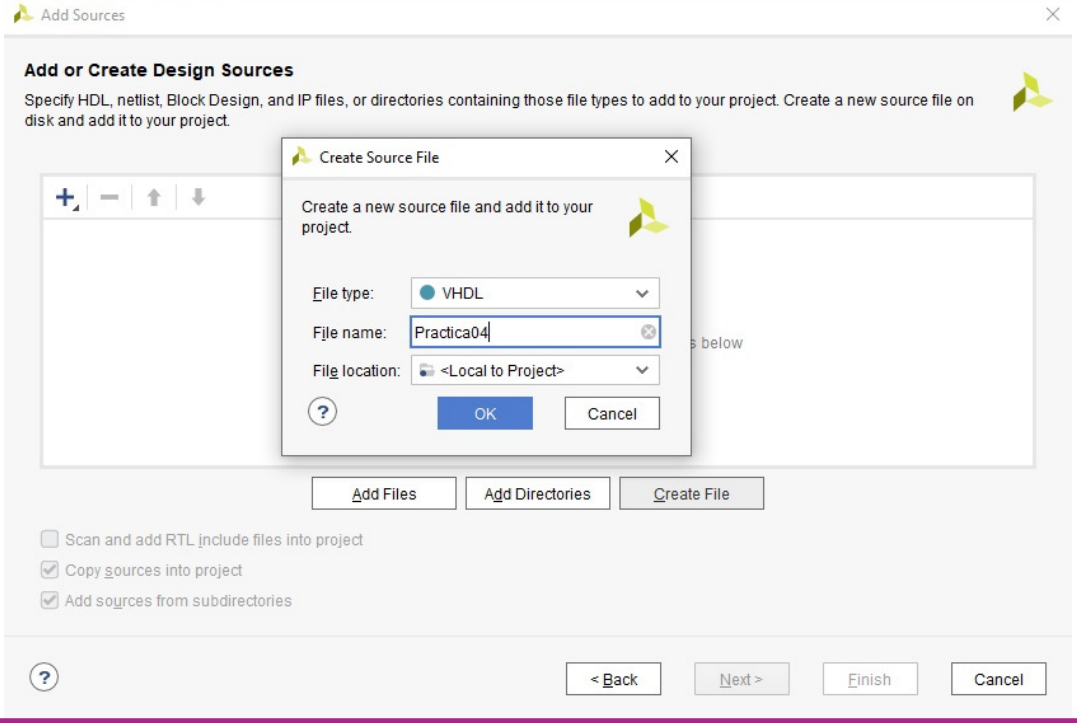


Figura 4.1. Ventana para crear el archivo fuente de un módulo en Vivado.

Comience creando el módulo correspondiente al código de más alta jerarquía, el programa principal. Elija el tipo de archivo *<VHDL>*, nombre al nuevo archivo como *"Practica04"* y deje la ubicación del archivo con el valor por defecto. Oprima el botón *<OK>* para regresar a la ventana *"Add or create design sources"* como se muestra en la Figura 4.1.

Repita el procedimiento para el módulo VHDL con la compuerta XOR, nombrando al nuevo archivo como "*myXOR*". Finalmente, oprima el botón *<Finish>* para pasar a la definición de los módulos (véase Figura 4.2).

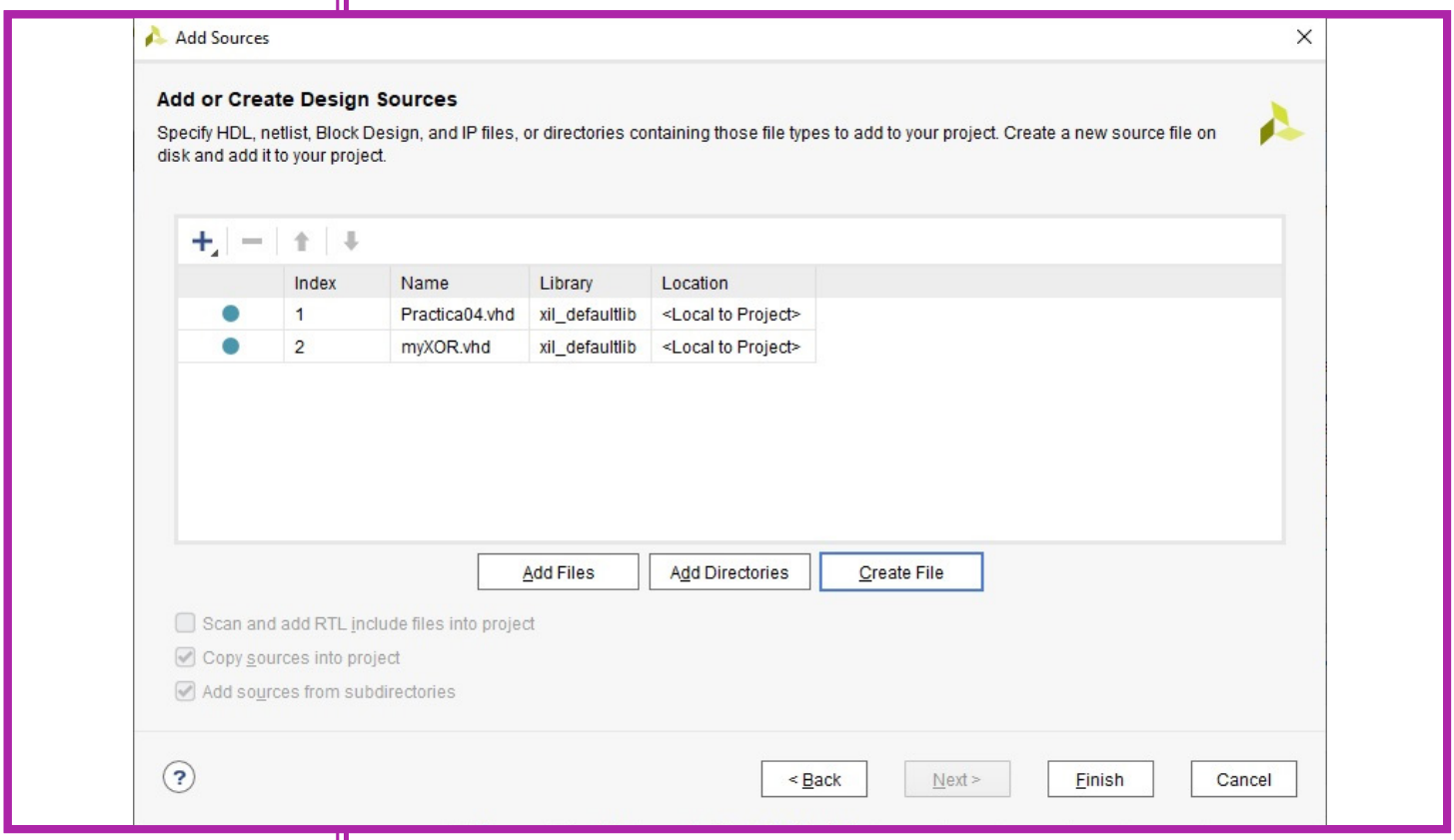


Figura 4.2. Ventana para agregar archivos fuente de un módulo en Vivado. Esta ventana muestra los archivos fuente que se van a generar semiautomáticamente.

En la ventana "*Define Modules*", seleccione, de la lista "*New Source Files*", un archivo para definir su entidad (*Entity*), así como la configuración de sus puertos. Primero, realice la definición del módulo que representa a la tarjeta Basys 3, entonces elija al archivo *Practica04.vhd* (véase Figura 4.3). Nombre a la nueva entidad con el identificador *Basys_system_mod.* Luego, nombre a la arquitectura como *my_system_arch*. En seguida, hay que especificar los puertos para el módulo, en forma individual, indicando si son de entrada o de salida, tal y como se muestra en la siguiente imagen:

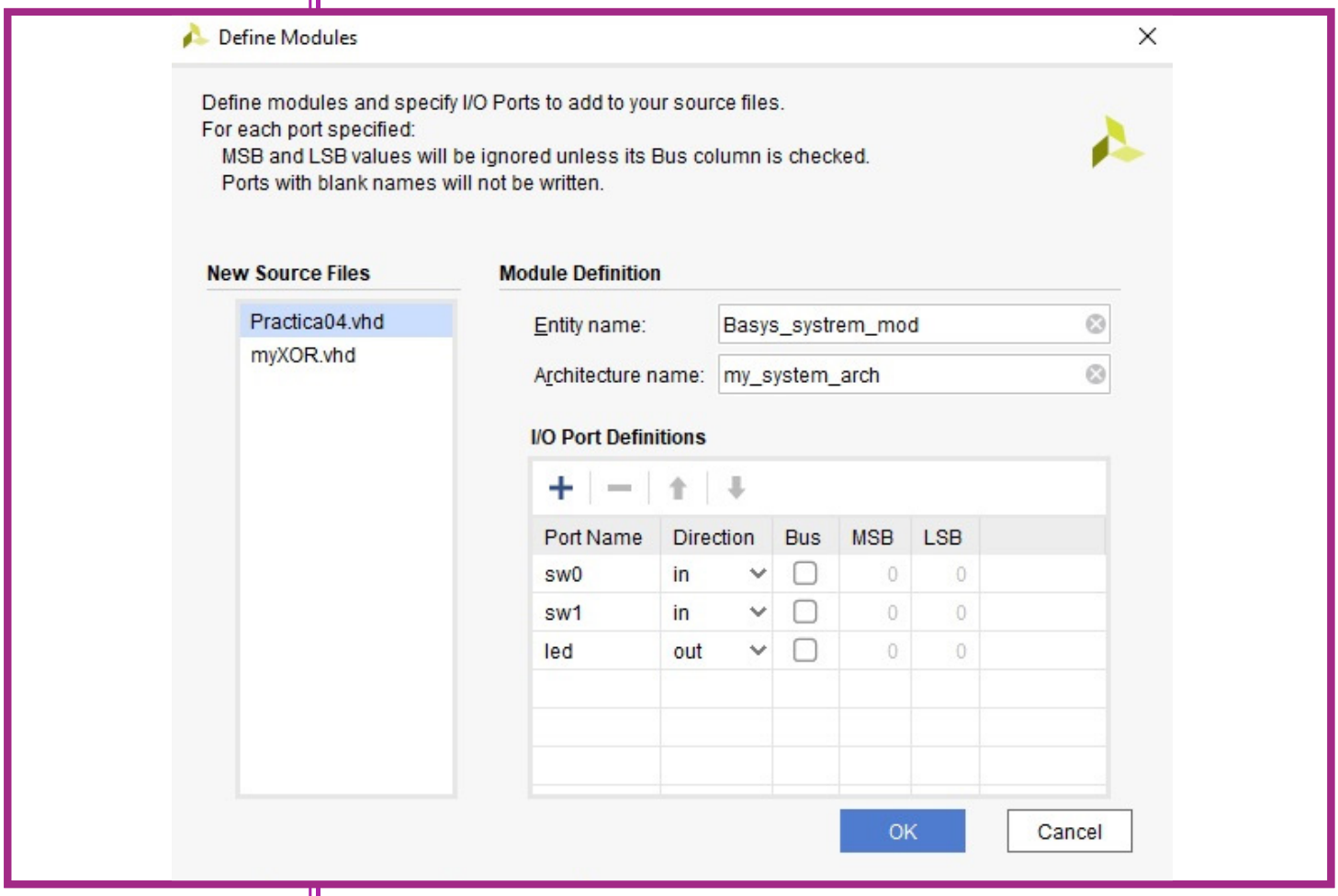


Figura 4.3. Ventana para definir la entidad Basys_system_mod, así como sus puertos y arquitectura.

Ahora, realice la definición del módulo para la función XOR. Entonces, elija al archivo *myXOR.vhd*. Nombre a la nueva entidad con el identificador *my_XOR*. Luego, nombre a la arquitectura como *my_XOR_arch*. En seguida, especifique los puertos para el módulo, en forma individual, indicando si son de entrada o de salida, tal y como se muestra en la siguiente imagen (Figura 4.4):

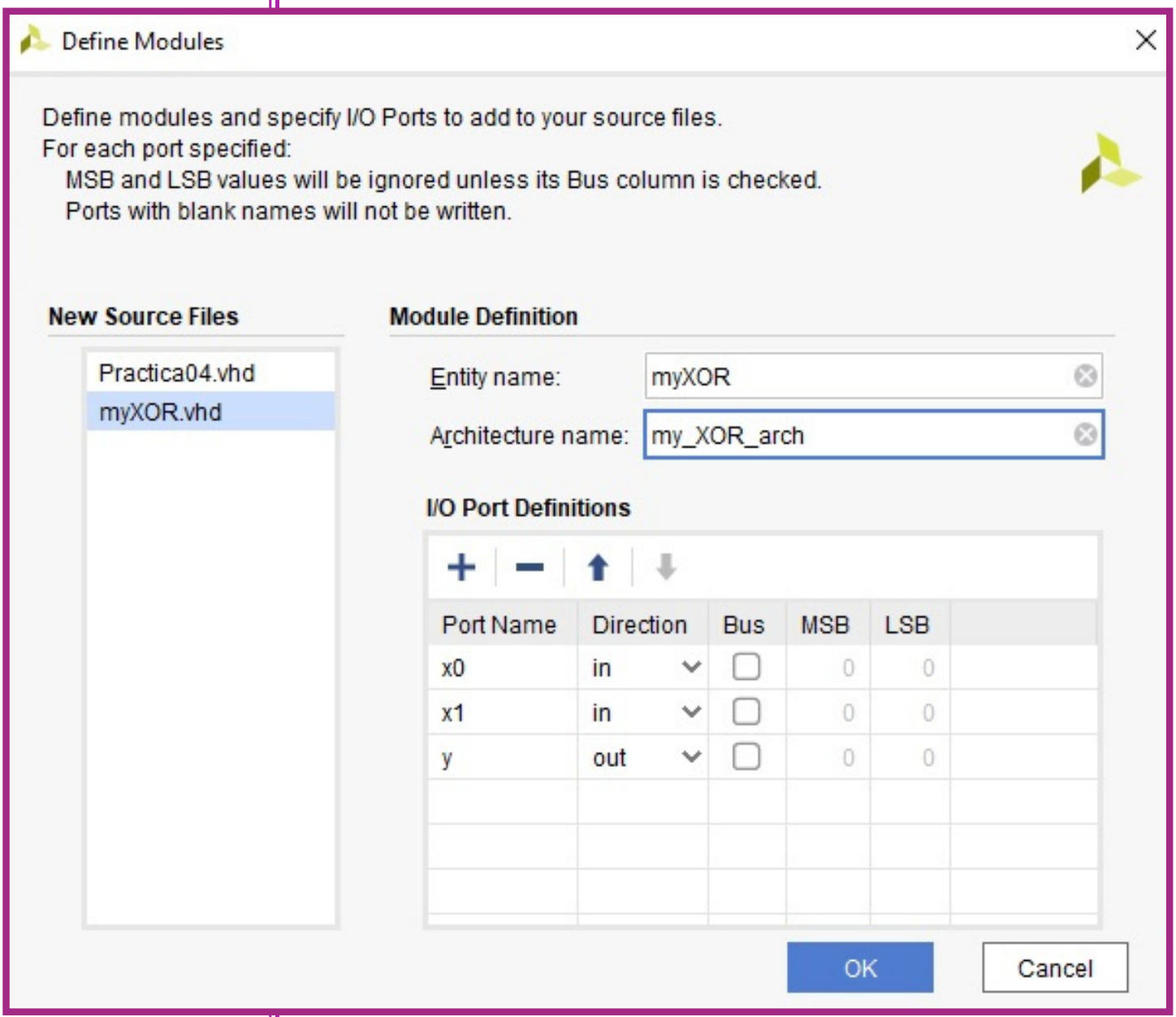


Figura 4.4. Ventana para definir la entidad myXOR, así como sus puertos y arquitectura.

Finalmente, oprima el botón <*OK*> para incluir, en la estructura jerárquica del proyecto, las plantillas de los módulos definidos como en la Figura 4.5:

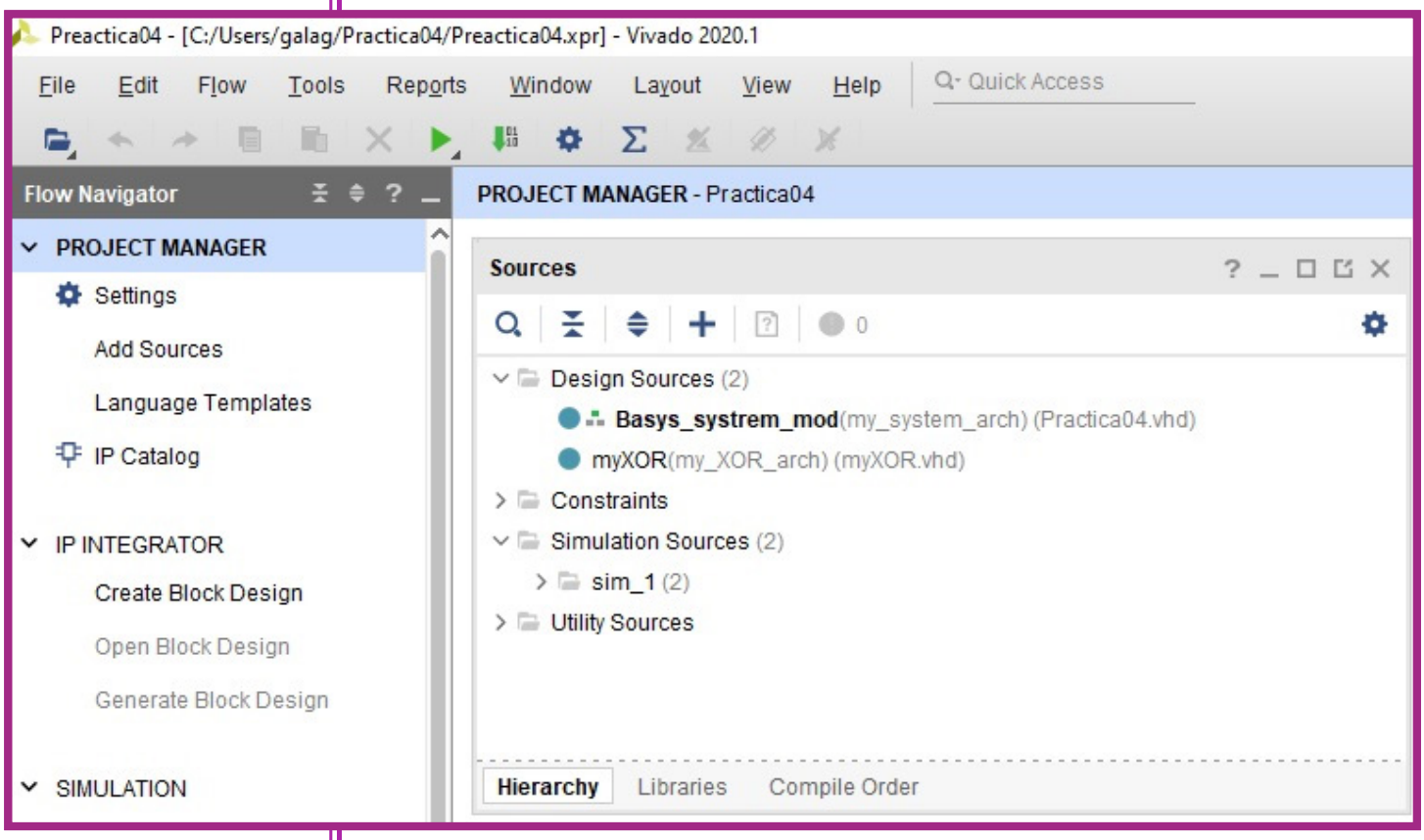


Figura 4.5. Vista de jerarquía de la ventana Sources que muestra los módulos del proyecto, Basys_system_mod y myXOR, en donde el módulo Basys_system_mod es el de más alta jerarquía.

Lo que sigue es completar las arquitecturas colocando el código que convenga. Se comienza con el código del componente XOR. Dé doble clic en el archivo *myXOR.vhd* listado en la vista de jerarquía ("*Hierarchy*") de la ventana "*Sources*".

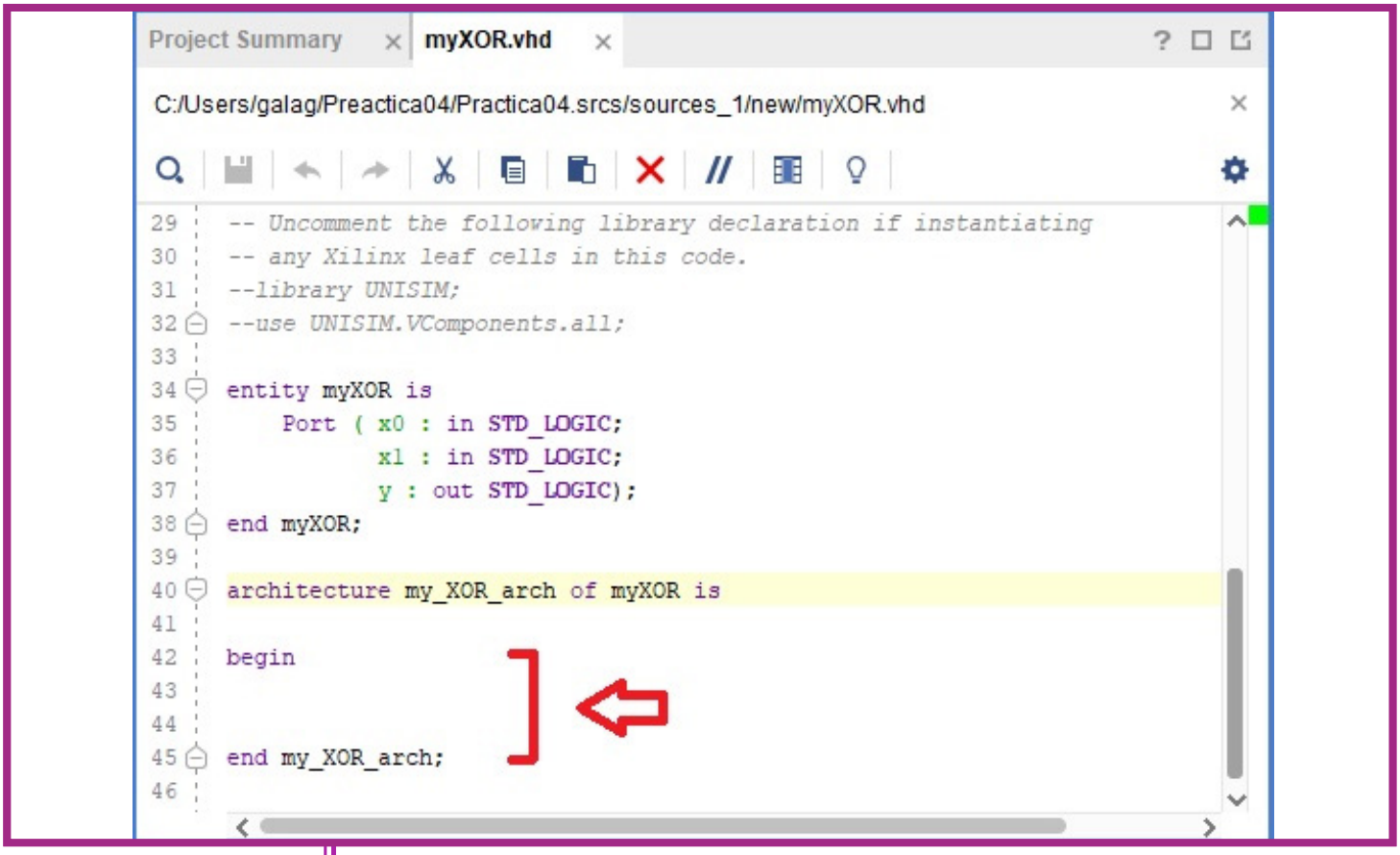


Figura 4.6. Vista del código del módulo myXOR, en donde se señala la sección que define la arquitectura del módulo.

Observe cómo la sección de la arquitectura (*architecture*) aparece declarada pero vacía (véase Figura 4.6). Complétela para que luzca como sigue:

```
architecture my_XOR_arch of myXOR is

   signal p0, p1: std_logic;

begin
   y <= p0 or p1;
   p0 <= (not x0) and x1;
   p1 <= x0 and (not x1);
end my_XOR_arch;
```

Continúe con el código del componente que representa al sistema. Dé doble clic en el archivo *Practica04.vhd*, listado en la vista de jerarquía ("*Hierarchy*") de la ventana "*Sources*".

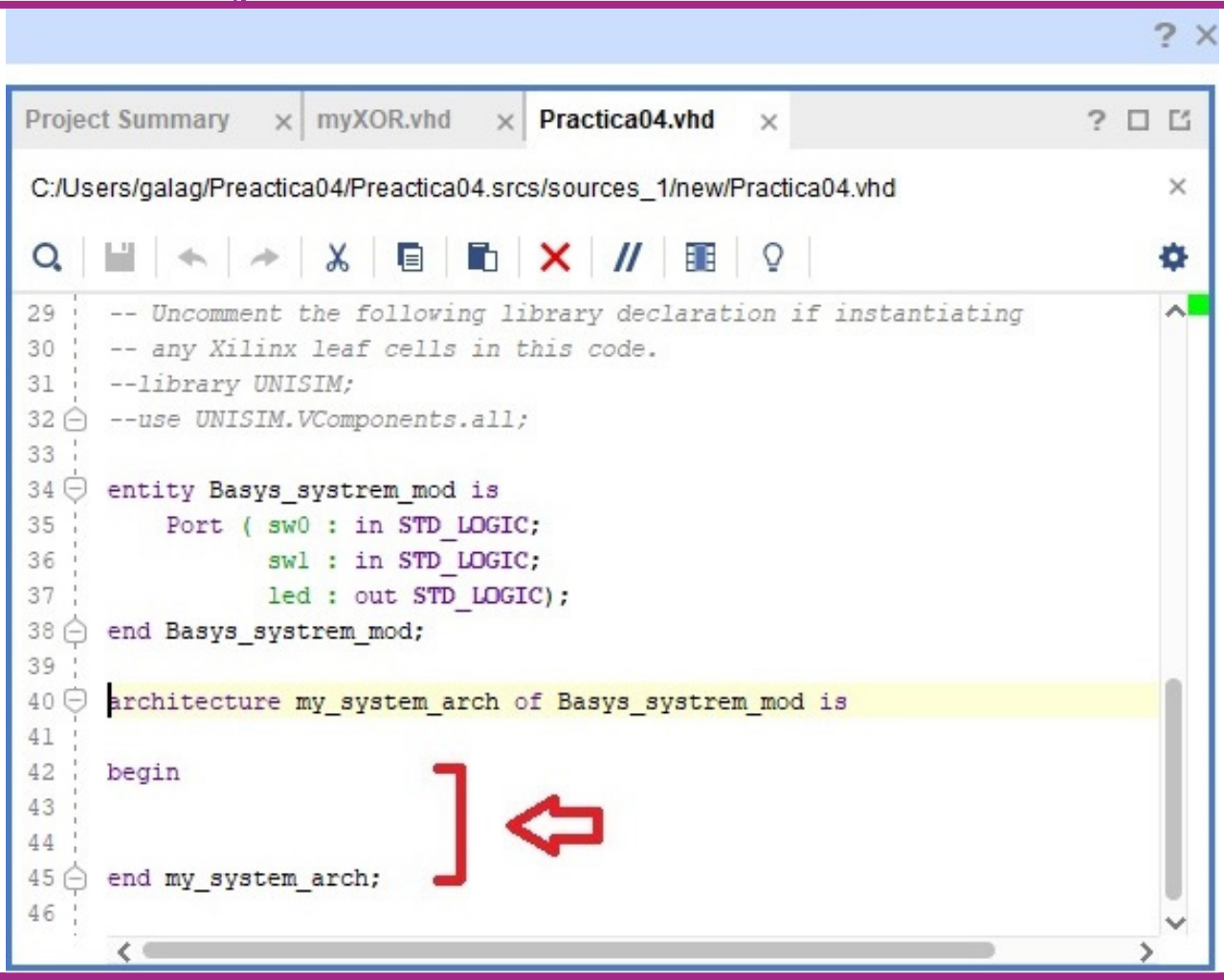


Figura 4.7. Vista del código del módulo Basys_system_mod, en donde se señala la sección que define la arquitectura del módulo.

Observe cómo aquí (Figura 4.7) también la sección de la arquitectura (*architecture*) aparece vacía. Complétela para que luzca como el siguiente código:

```
architecture my_system_arch of Basys_system_mod is

component myXOR
   port(
      x0, x1: in std_logic;
      y: out std_logic
   );
end component;

begin

xor_01 : myXOR
  PORT MAP (
    x0 => sw0,
    x1 => sw1,
    y => led0
  );

end my_system_arch;
```

3. Crear y agregar el archivo de restricciones con la asignación de terminales en el hardware destino.

Ahora cree un código de definición de restricciones de usuario, con extensión *.xdc, que define las restricciones y la asignación de terminales (también denominados como "pines") para la tarjeta de desarrollo Basys 3.

En la ventana del margen izquierdo, "*Flow Navigator*", ir a la sección "*Project Manager*" y oprimir el icono "*Add Sources*". En la ventana que aparece seleccionar la opción "*<*> Add or create constraints*", luego,

oprimir el botón <Next>. A continuación, en la nueva ventana "*Add or create constraints*", oprimir el botón <*Create File*>.

Elija el tipo de archivo <XDC>, nombre al nuevo archivo como "*Practica04*" y deje la ubicación del archivo con el valor por defecto.

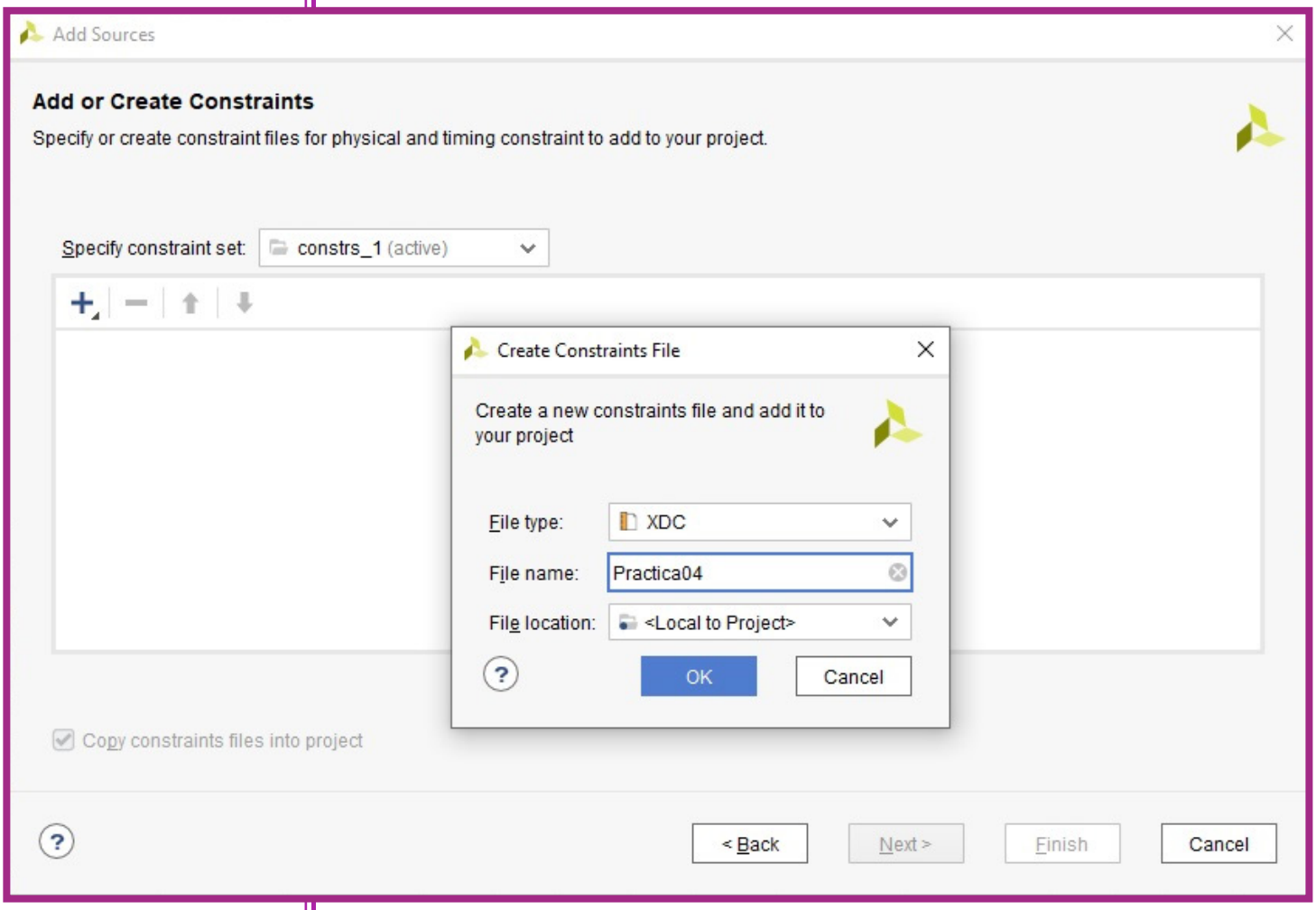


Figura 4.8. Ventana para crear un archivo de restricciones.

Oprima el botón <OK> para regresar a la ventana "*Add or or create constraints*". Oprima el botón <*Finish*> para terminar (Figura 4.8).

Se ha agregado el archivo Practica04.xdc al proyecto, pero está completamente vacío. Dé doble clic en el archivo Practica04.xdc, listado en la vista de jerarquía ("Hierarchy") de la ventana "Sources", y complételo para que luzca como el siguiente código:

```
## Archivo .xdc para tarjeta Basys3 rev B
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw0}]

set_property IOSTANDARD LVCMOS33 [get_ports
{sw0}]
set_property PACKAGE_PIN V16 [get_ports {sw1}]

set_property IOSTANDARD LVCMOS33 [get_ports
{sw1}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led0}]

set_property IOSTANDARD LVCMOS33 [get_ports
{led0}]
```

4. Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).

5. Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).

Práctica 4

ACTIVIDAD

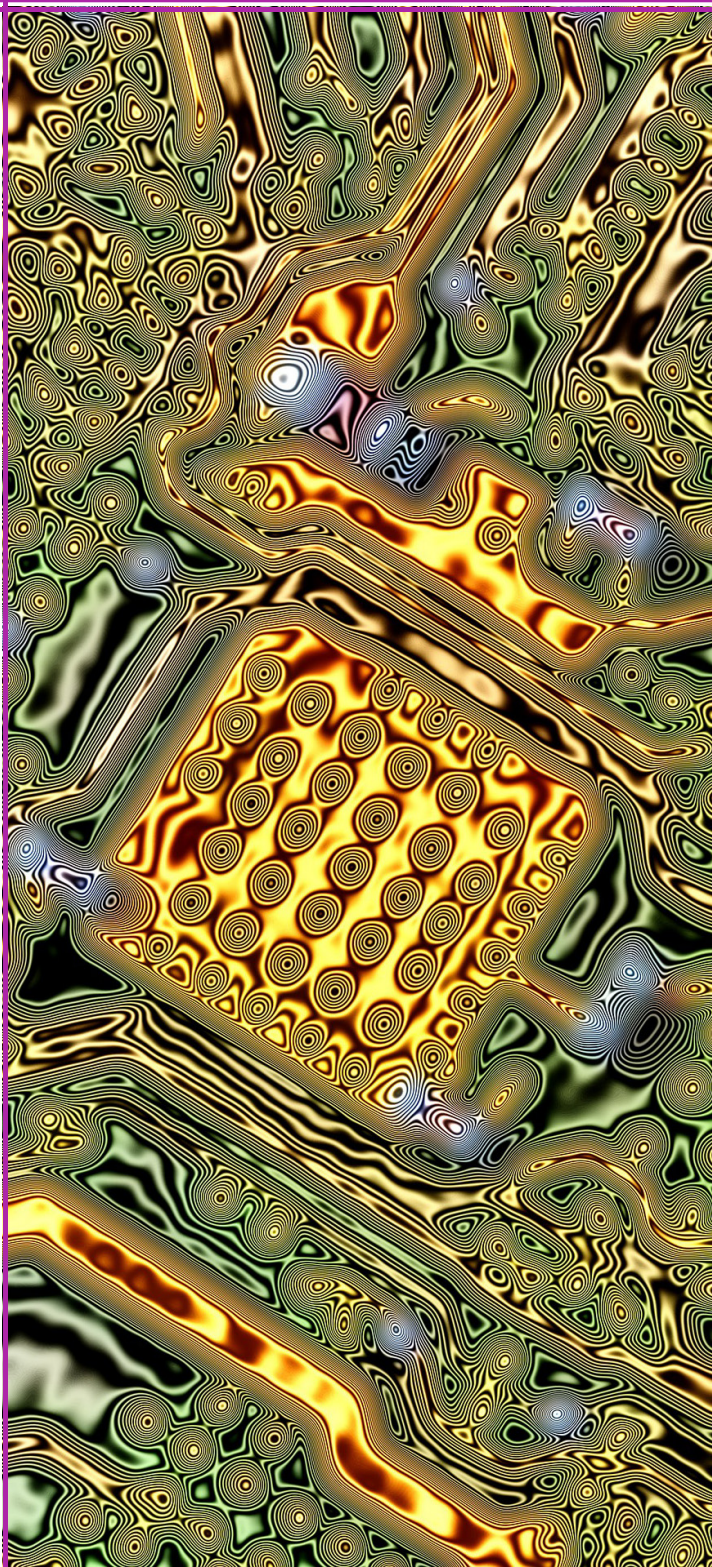
A partir del ejemplo desarrollado en el procedimiento, defina en forma semiautomática los módulos de los componentes para las compuertas lógicas NOT, AND y OR. Sintetice el hardware correspondiente y pruebe sus módulos comprobando sus tablas de verdad. Finalmente, guarde cada entidad (módulo) en un archivo independiente e intégrelos dentro del proyecto para ser usados dentro del código top (*Practica04.vhd*). Compruebe que el desempeño del sistema es el mismo, aunque el código se ha distribuido y modularizado. Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica.

Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 5

Simulación de circuitos lógicos en el entorno Vivado



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVOS

Desarrollar un proyecto VHDL básico haciendo uso de la herramienta de simulación del entorno Vivado.

ANTECEDENTES

Los proyectos de desarrollo de hardware se componen de varios módulos con arquitectura propia. Para asegurar el correcto funcionamiento de un proyecto, se debe probar que cada uno de sus módulos (tanto en forma individual como conjunta), verificando que se cumpla con los requerimientos funcionales establecidos y se comporten como se espera ante distintas señales de entrada. Lo anterior hace posible diseñar proyectos de hardware escalables, al verificar el comportamiento del proyecto conforme se agregan nuevos módulos y comprobar que los nuevos módulos no afectan negativamente el funcionamiento de los módulos que ya han sido probados. Para este propósito, el entorno de desarrollo Vivado provee herramientas de simulación que permiten verificar la respuesta de los módulos de hardware, partiendo de distintos patrones para las señales lógicas de entrada —mediante lo que se conoce como vectores de prueba— y graficando las señales involucradas, tanto entradas como salidas, con relación al eje temporal (diagramas de tiempo).

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 7, que a continuación se describen:

- **miXOR.vhd.** Contiene el código con una versión estructural de la compuerta XOR.
- **test_bench.vhd.** Contiene el código del programa testbench. Note que la entidad de este programa no tiene declarados puertos de entrada/salida, lo que es típico en un programa testbench.
- **Practica05.vhd.** Contiene el código que define a la entidad de más alto nivel (top level), el código principal, que representa a la tarjeta, en forma lógica y a nivel de código, y que invoca a todos los demás códigos.
- **Practica05.xdc**. Contiene las restricciones y la asignación de terminales definidas por el usuario, específicamente para el hardware usado.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto a partir de los códigos: Practica05.vhd, miXOR.vhd, test_bench.vhd y Practica05.xdc. El proyecto se debe llamar "Practica05" (ver Anexo 2).

2. Realizar la construcción del proyecto (ver Anexo 3).

Esta vez no abra el administrador de hardware. Oprima el botón <Cancel> en la ventana "*Bitstream Generation Completed*".

3. Simular el proyecto construido en el entorno Vivado.

En la ventana del margen izquierdo, "*Flow Navigator*", ir a la sección "*Simulation*" y oprimir botón derecho del ratón para poder elegir la opción "*Simulation Settings*". Se abre la ventana de ajustes del proyecto ("*Settings*"), particularmente para la función de simulación. Vaya al campo "*Simulation top module name*" y elija la entidad que corresponde al código de simulación, en este caso *practica5_test_bench* (Figura 5.1).

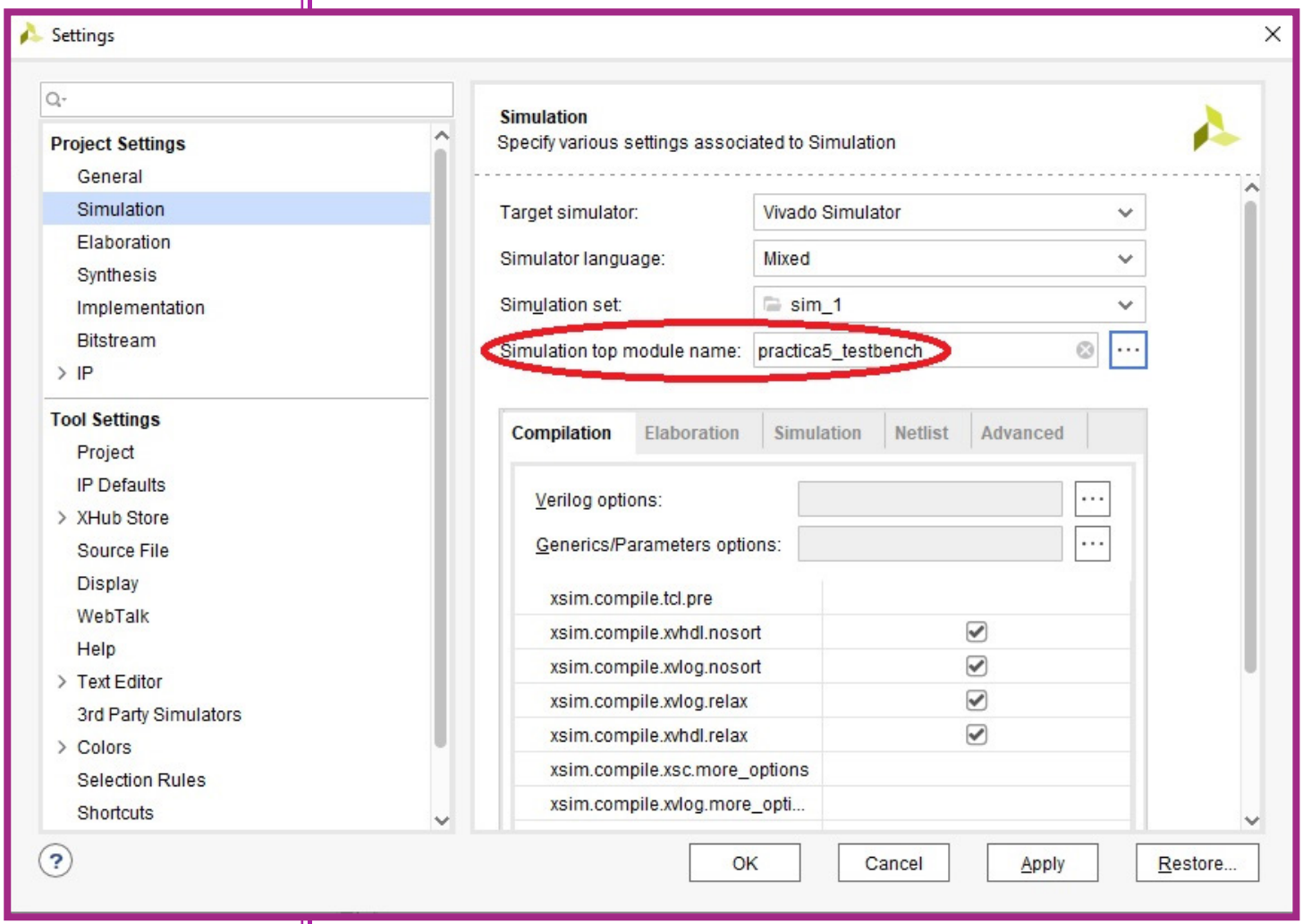


Figura 5.1. Ventana principal para definir la configuración de una simulación.

Deje las demás opciones con los valores por defecto excepto, tal vez, el valor de tiempo de simulación, que se encuentra en la pestaña “simulation”, y que aparece como la variable “xsim.simulate.runtime*”.

Debe especificar un tiempo suficiente para la prueba.

Dado que se generan cuatro vectores de prueba, cada uno con una duración de 200 ns, si ejecuta la simulación por 1000 ns, será suficiente (Figura 5.2).

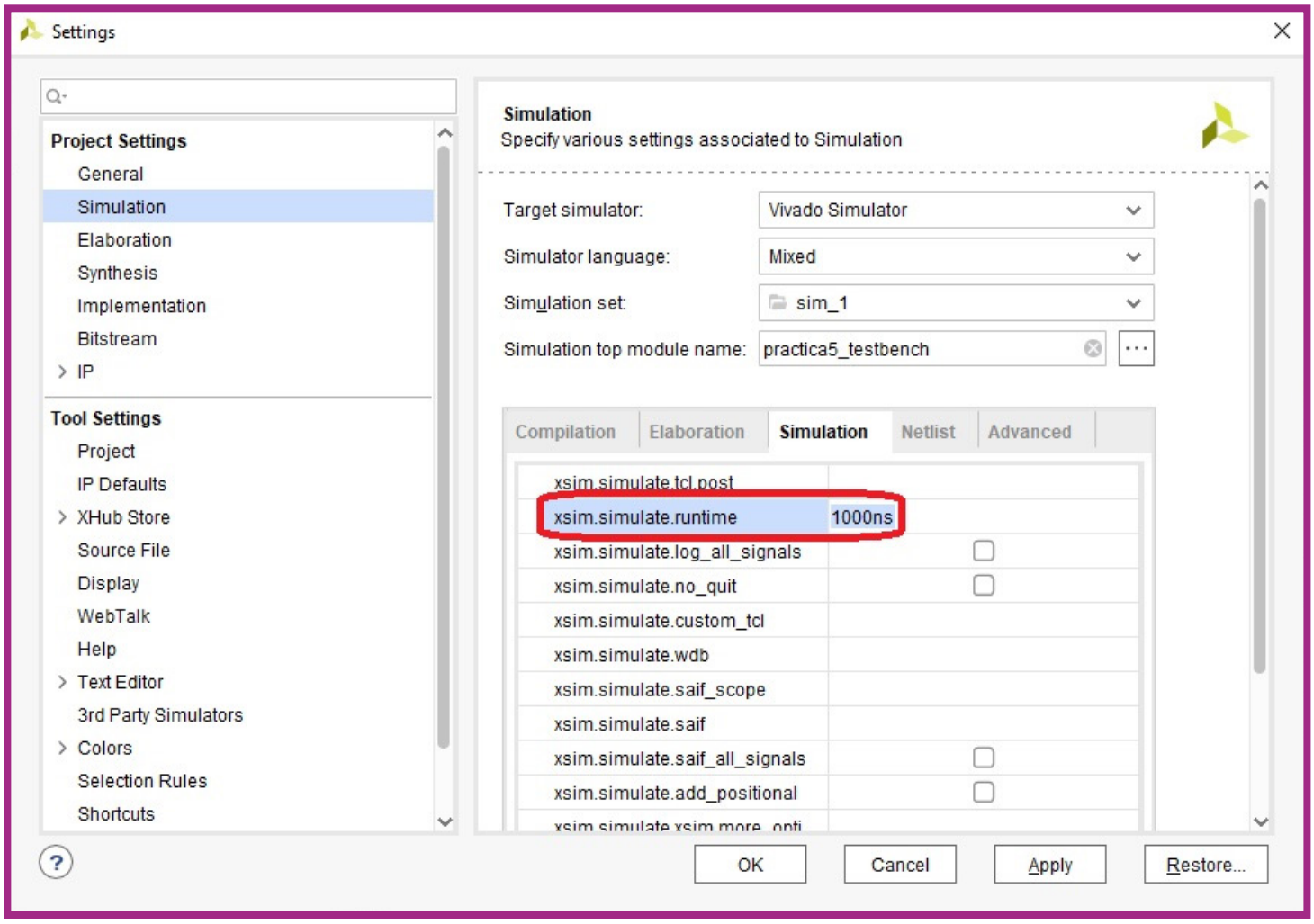


Figura 5.2. Ventana para definir la duración de la simulación.

Para terminar con la configuración de la simulación, oprima el botón <OK>. En la ventana del margen izquierdo, "Flow Navigator", ir a la sección "Simulation" y oprimir la opción "Run Simulation". Elija la opción "Run Behavioral Simulation" y se abre la ventana de simulación (Figura 5.3).

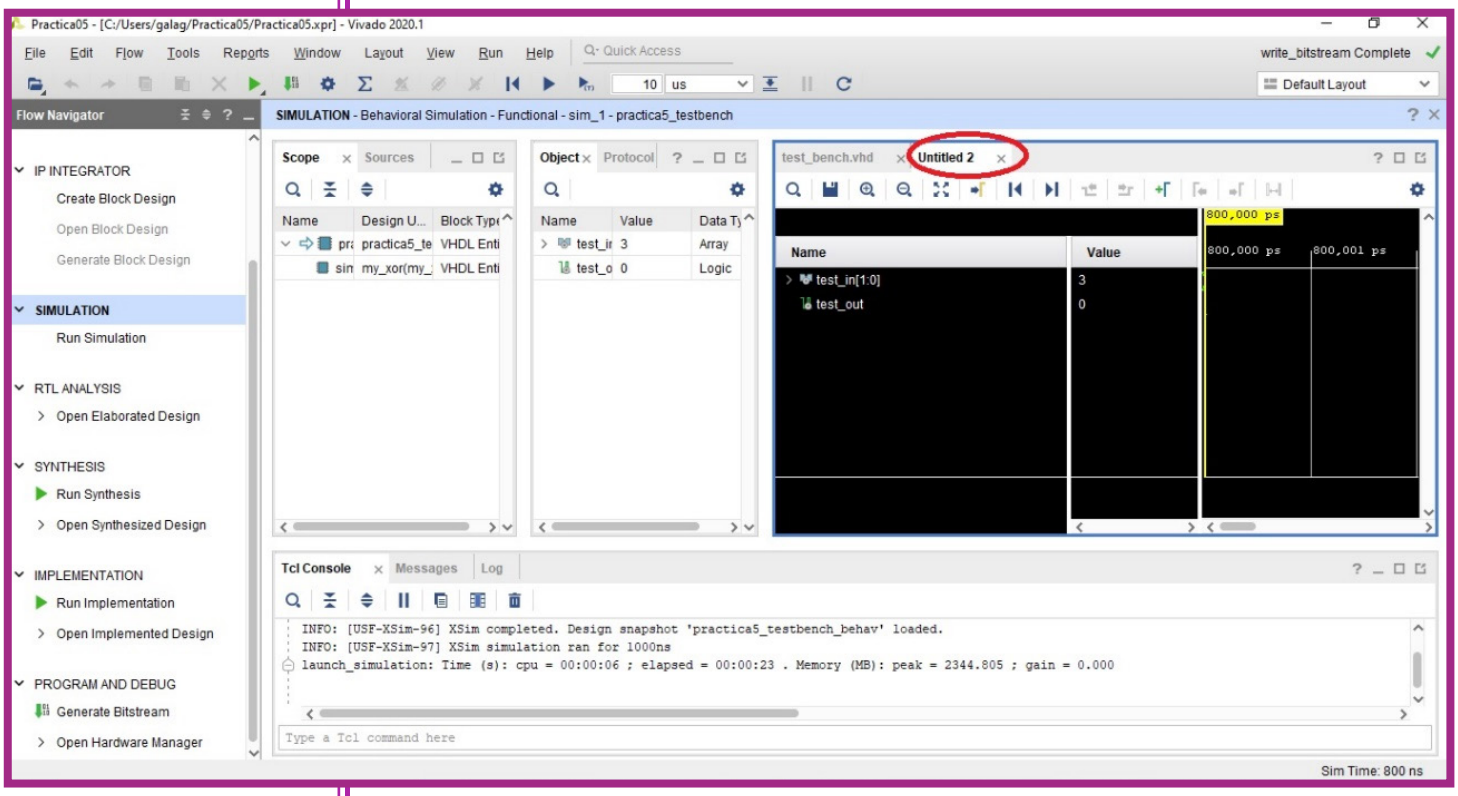


Figura 5.3. Ventana de resultados de simulación.

Maximice la ventana con los diagramas de tiempo y oprima el botón "Zoom Fit", en la barra de herramientas en su costado izquierdo, para que las señales simuladas se ajusten al espacio visible (Figura 5.4).

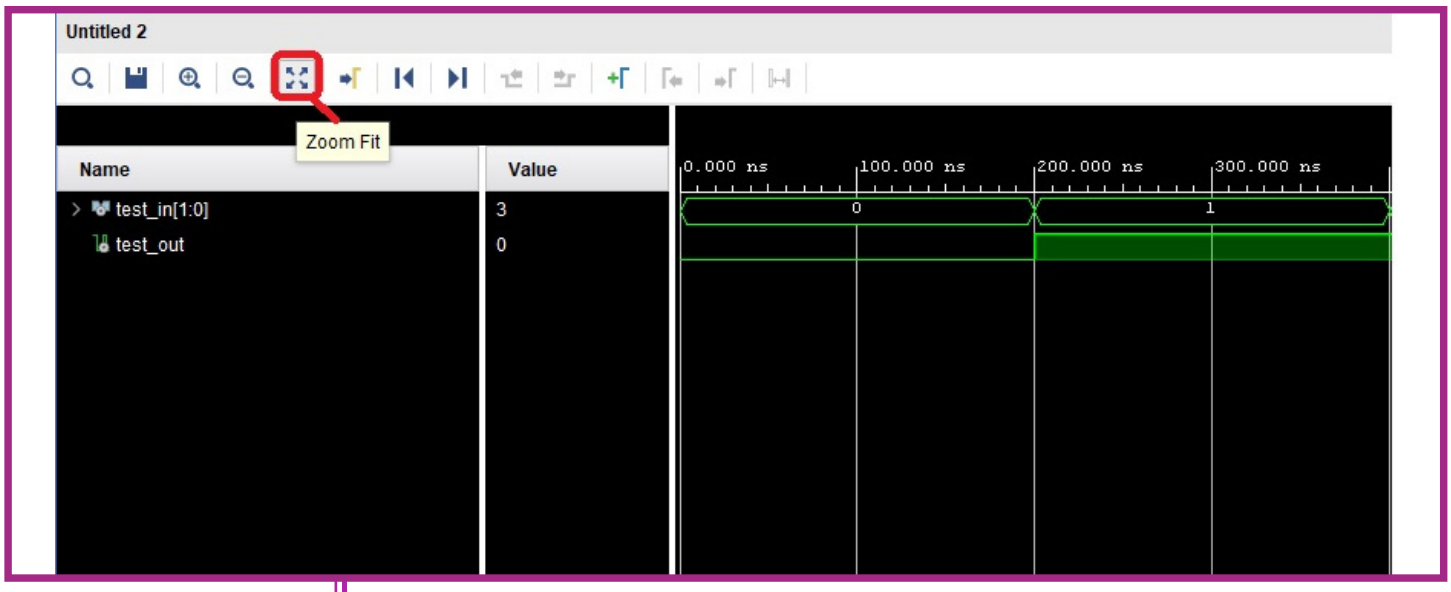


Figura 5.4. Ventana para definir la configuración de una simulación con zoom ajustado para mejor visualización.

Se puede expandir el vector asociado a las señales de entrada (test_in) oprimiendo el icono (+) que le precede en la columna Name (Figura 5.5).

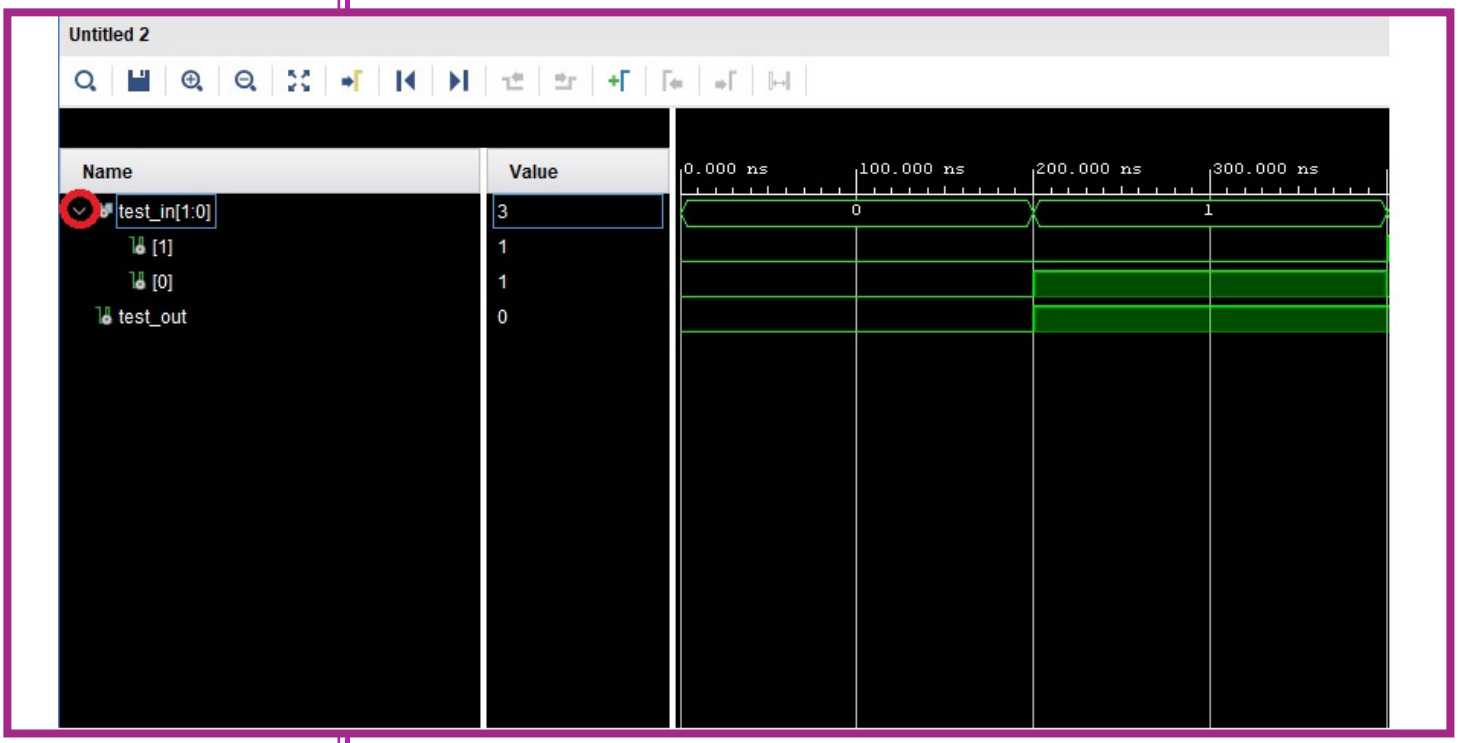


Figura 5.5. Expansión de los valores del vector de señales de entrada (*test_in*).

Nótese cómo las señales muestran pulsos con un ancho de 200 ns, según lo especificado en el banco de prueba test_bench.vhd. Esta última vista es la versión gráfica de la tabla de verdad para la compuerta XOR que se está simulando.

Práctica 5

ACTIVIDAD

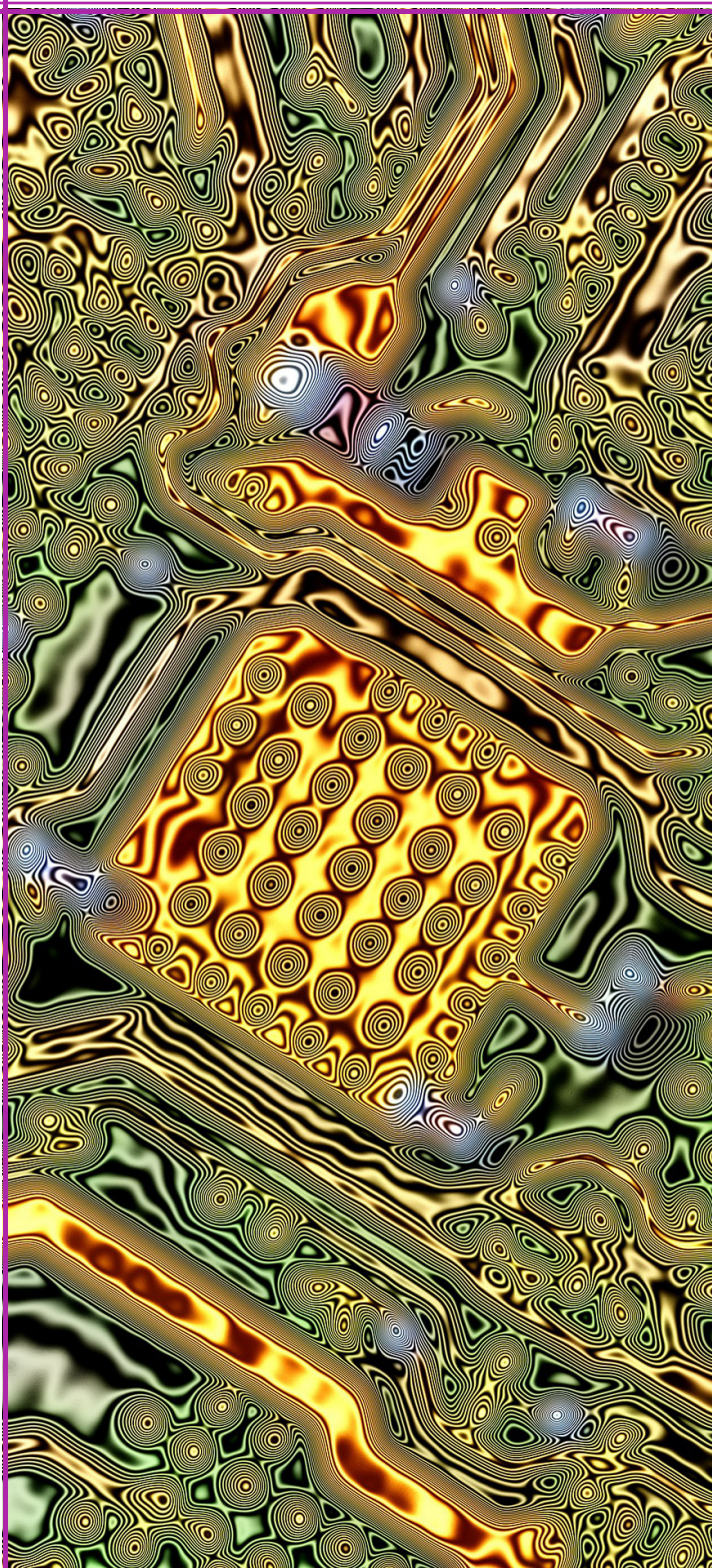
Desarrolle los módulos de prueba (*testbench*) para los componentes que definió en la Práctica 4, es decir, para las compuertas lógicas NOT, AND y OR. Simule el comportamiento de estos componentes y compruebe gráficamente sus tablas de verdad. Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Capture y reporte las ventanas en pantalla de las gráficas, con los diagramas de tiempo, correspondientes a las tablas de verdad de cada compuerta lógica.

Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 6

Convertidor binario a carácter hexadecimal (estructural)



Dirección
Ciencias Básicas
e Ingeniería

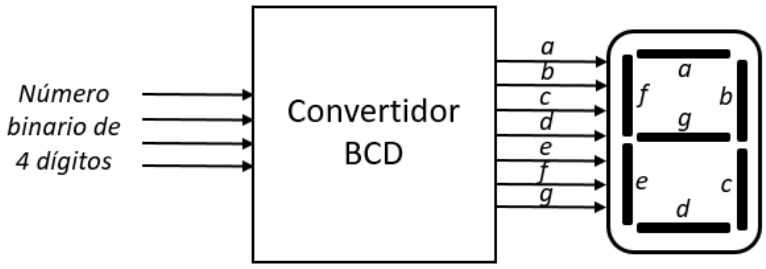
OBJETIVOS

Desarrollar un proyecto VHDL con un componente que convierta un número, codificado en binario de 4 bits, a un carácter hexadecimal (BCH). El componente del convertidor se realizará con base en una arquitectura estructural, usando funciones booleanas con la forma canónica de suma de productos.

ANTECEDENTES

Los circuitos convertidores (ya sean codificadores o decodificadores) convierten cada combinación de bits de entrada en una combinación diferente para los bits de salida. Por ejemplo, en un decodificador de 4 a 16, tenemos 4 bits de entrada y 16 bits de salida, de tal forma que a cada una de las 16 combinaciones de entradas se le asigna una combinación de salida, donde sólo un bit de salida se mantiene en alto, mientras que los demás se mantienen en bajo, indicando, con ello, la combinación que se detectó a la entrada.

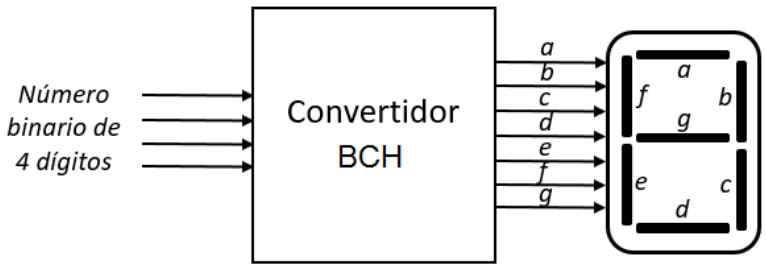
Entre los convertidores digitales más populares se encuentra el convertidor de decimales codificados en binario o BCD (Binary Coded Decimal), el cual asigna un código binario de 4 bits al digito correspondiente en decimal. Una aplicación común de los convertidores BCD es su uso en la activación de displays de 7 segmentos, estos dispositivos se utilizan en varios sistemas digitales, tales como calculadoras digitales, máquinas expendedoras de bebidas, etc. En estas aplicaciones, un convertidor BCD recibe como entrada un número binario de 4 bits y devuelve como salida un código de 7 bits indicando los segmentos del display que se deben activar para mostrar el dígito decimal (ver Figura 6.1).



Entrada binaria	Carácter desplegable
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	–
1011	–
1100	–
1101	–
1110	–
1111	–

Figura 6.1. Diagrama a bloques de un display de 7 segmentos activado por un convertidor BCD.

Un convertidor similar al BCD es el que puede representar no sólo los dígitos del 0 al 9, sino también los caracteres A, B, C, D, E y F, a fin de desplegar un dígito hexadecimal. A este convertidor lo denominaremos BCH, por las siglas en inglés de Binary Coded Hexadecimal (ver Figura 6.2).



Entrada binaria	Carácter desplegable
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Figura 6.2. Diagrama a bloques de un display de 7 segmentos activado por un convertidor BCH.

Práctica 6

En esta práctica se diseñará un convertidor de binario a carácter hexadecimal (mediante una arquitectura estructural) para representar un número binario de 4 bits mediante un carácter hexadecimal, de modo que cada número binario de 4 bits será convertido a una secuencia de 7 bits, la cual será utilizada para mostrar el carácter hexadecimal en un display de 7 segmentos. La arquitectura estructural del convertidor se definirá usando funciones booleanas, mediante el enfoque de la forma canónica de suma de productos, las cuales pueden ser reducidas aplicando las técnicas de Mapas de Karnaugh o álgebra de Boole.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 8, que a continuación se describen:

- ***structural_hex2led.vhd.*** Contiene el código que define a nuestro componente con el convertidor de binario a hexadecimal. Note que la arquitectura está definida con base en funciones booleanas de sumas de productos.
- ***Practica06.vhd.*** Contiene el código que define a la entidad de más alto nivel, el código principal, que representa a la tarjeta, en forma lógica y a nivel de código, y que invoca a todos los demás códigos.
- ***Practica06.xcd.*** Contiene la asignación de terminales y las restricciones definidas por nosotros para el hardware específico que se usará.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto a partir de los códigos: *structural_hex2led.vhd, Practica06.vhd y Practica06.xcd*. El proyecto se debe llamar "Practica06" (ver Anexo 2).
2. Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).
3. Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).

ACTIVIDAD

Práctica 6

A partir del código trabajado en esta práctica, desarrolle una versión simplificada para el código structural_hex2led.vhd, usando la técnica de mapas de Karnaugh o, en su defecto, mediante álgebra de Boole. Construya el proyecto, genere el archivo programable y pruebe su propuesta (ver Anexos 3 y 4). Renombre la arquitectura de la versión simplificada y asegúrese de que genera una instancia de su propia arquitectura y no una de la entidad en el código proporcionado por el profesor. Finalmente, usando la herramienta de simulación del entorno Vivado, simule su propuesta (revise la Práctica 5).

Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Acompañe su reporte con un anexo que incluya los códigos obtenidos.

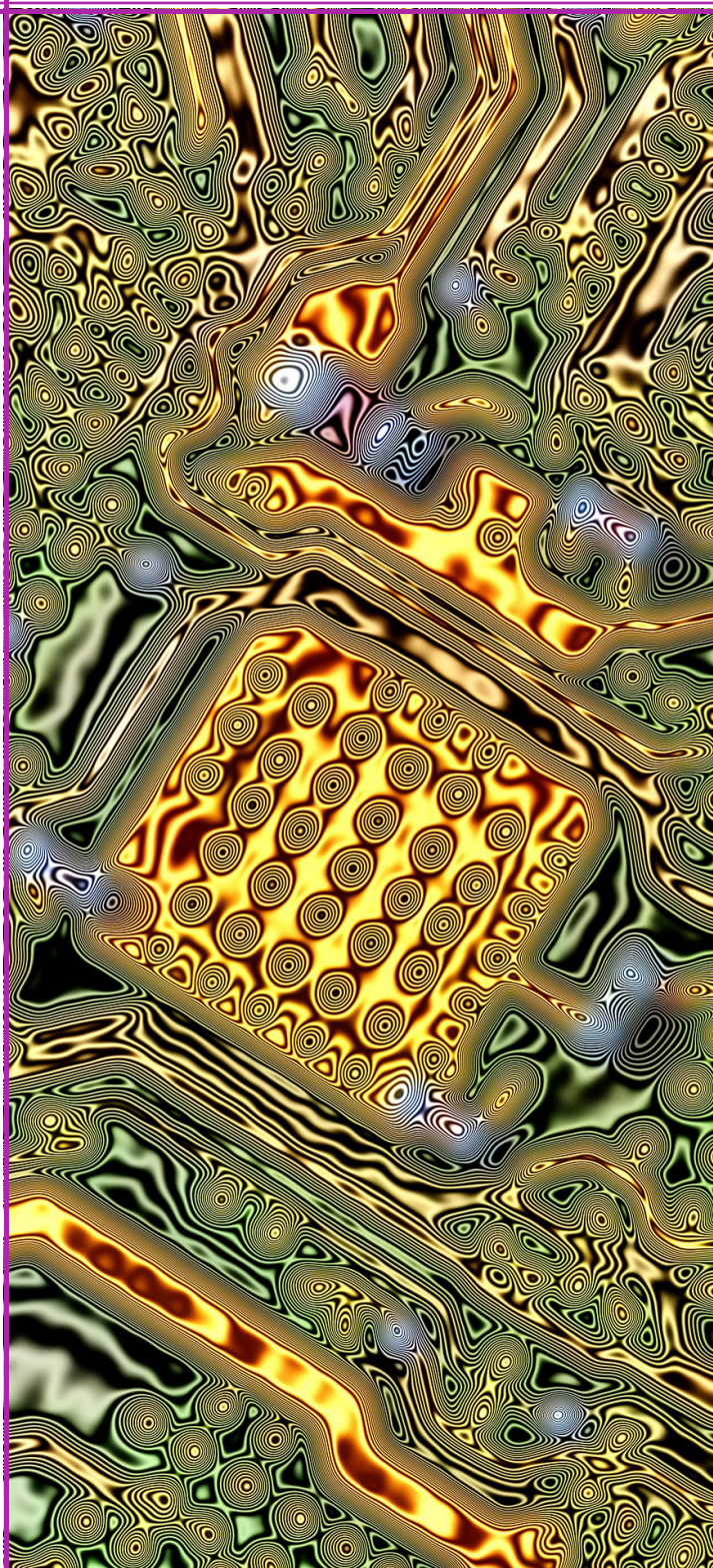
Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video LAB FDL 05. WWW: Youtube. Enlace: https://youtu.be/xk9xWvyIWAA

- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 7

Convertidor binario a carácter hexadecimal (funcional)



Dirección
Ciencias Básicas
e Ingeniería

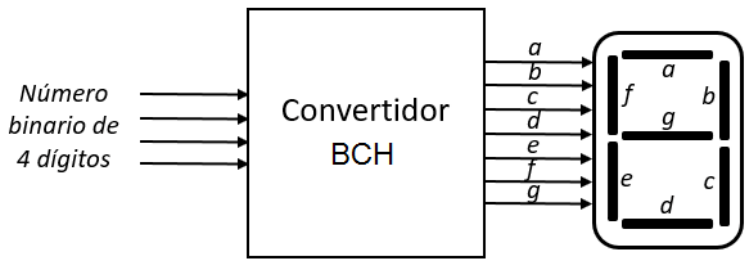
OBJETIVO

Desarrollar un proyecto VHDL con un componente que convierta un número, codificado en binario de 4 bits, a un carácter hexadecimal. El componente del convertidor se realizará con base en una arquitectura funcional, usando estructuras de asignación del lenguaje VHDL.

ANTECEDENTES

Los convertidores son parte fundamental de los circuitos digitales y, entre otras cosas, permiten la comunicación de distintos módulos de hardware, adaptando la salida de un módulo a la entrada de otro. Por ejemplo, como ocurre en las computadoras convencionales, donde un cierto tipo de convertidor adapta la salida de un teclado digital al puerto de entrada de la computadora.

En la Práctica 6 se diseñó un convertidor BCH con arquitectura estructural, en esta práctica, se diseñará un convertidor BCH con arquitectural funcional, en contraparte (ver Figura 7.1). Al igual que en la Práctica 6, el convertidor BCH será conectado a un display de 7 segmentos, de manera que se pueda comparar el comportamiento de ambos convertidores (con arquitectura estructural y funcional).



Entrada binaria	Carácter desplegable
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Figura 7.1. Diagrama a bloques de un display de 7 segmentos activado por un convertidor BCH.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 9, que a continuación se describen:

- ***behavioral_hex2led.vhd.*** Contiene el código que define a nuestro componente con el convertidor de binario a hexadecimal. Note que la arquitectura está definida con un enfoque funcional mediante la estructura de control with-select.
- ***Practica07.vhd.*** Contiene el código que define a la entidad de más alto nivel, el código principal, que representa a la tarjeta, en forma lógica y a nivel de código, y que invoca a todos los demás códigos.
- **Practica07.xcd.** Contiene la asignación de terminales y las restricciones definidas por nosotros para el hardware específico que se usará.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. **Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto a partir de los códigos: behavioral_hex2led.vhd, Practica07.vhd y Practica07.xdc. El proyecto se debe llamar "Practica07" (ver Anexo 2).**
2. **Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).**
3. **Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).**

ACTIVIDAD

Práctica 7

A partir del código trabajado en esta práctica, desarrolle el código para un convertidor decimal codificado en binario (BCD) a 7 segmentos. Construya el proyecto, genere el archivo programable y pruebe su propuesta vía simulación. Asegúrese de renombrar la nueva entidad (por ejemplo, bcd2led), así como su arquitectura, y guardar el código resultante en un nuevo archivo para diferenciarlo del proporcionado en este manual. En el código Practica07.vhd, genere una instancia de su módulo BCD y reemplace el módulo original hex2led por el propio. Modifique el código para activar los 7 segmentos, del display LED, con patrones diferentes a los del código original proporcionado (por ejemplo, el 1 y el 7 se pueden desplegar de forma ligeramente diferente).

Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Acompañe su reporte con un anexo que incluya los códigos obtenidos.

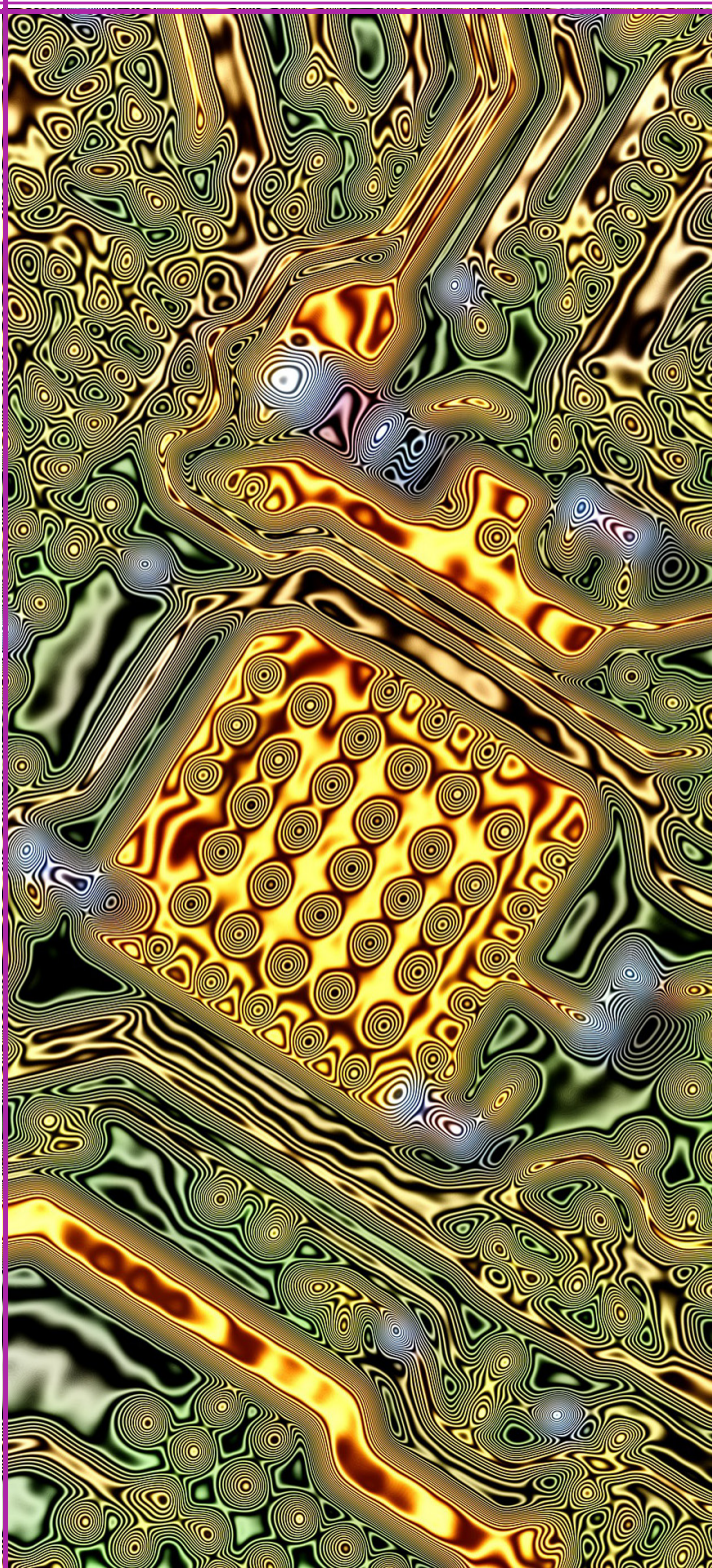
Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.

- Laguna, G. (2020). Video LAB FDL 05. WWW: Youtube. Enlace: https://youtu.be/xk9xWvyIWAA
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 8

Uso de módulos de propiedad intelectual de terceros



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVO

Desarrollar un proyecto VHDL que incluya el uso del reloj del sistema, un contador binario predefinido por un módulo de propiedad intelectual de terceros (*Intellectual Property, IP*) y un registro básico definido con una arquitectura del tipo funcional (*behavioral*).

ANTECEDENTES

Los proyectos VHDL involucran el diseño de distintos módulos de hardware de propósito general que se interconectan y que en conjunto con los módulos IP permiten implementar un componente de hardware personalizado. El entorno Vivado provee una biblioteca de módulos de hardware desarrollados por Xilinx y sus socios comerciales, los cuales ya han sido probados y validados, por lo que su uso es seguro y puede reducir considerablemente el tiempo de desarrollo, permitiendo enfocarse en el diseño del hardware final. Algunos de módulos IP básicos incluyen contadores, registros, acumuladores, entre otros que ampliamente usados y que permiten reducir el tiempo empleado en el diseño. En esta práctica se diseñará un circuito lógico que consiste en un módulo de registro, de diseño propio, y un módulo contador binario de propiedad intelectual de terceros, del cual se deberá deducir su funcionamiento.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán los códigos, disponibles en el Anexo 10, que a continuación se describen:

- ***Register.vhd.*** Contiene el código para construir un registro con arquitectura funcional.
- ***Practica08.vhd.*** Contiene el código que define a la entidad de más alto nivel, es decir, el código principal. En este código se declaran e instancian los componentes: contador *Bin_Counter* y el registro reg8.
- ***Practica08.xcd.*** Contiene la asignación de terminales y las restricciones definidas por nosotros para el hardware específico que se usará.

Una vez que tenga los archivos mencionados en su computadora, proceda con los siguientes pasos:

1. Iniciar la aplicación Xilinx Vivado y crear un nuevo proyecto a partir de los códigos: *Register.vhd, Practica08.vhd y Practica08.xdc.* El proyecto se debe llamar "Practica08" (ver Anexo 2).

Durante la creación del proyecto aparece la ventana para adicionar bloques IP (con propiedad intelectual de terceros) "Add Existing IP (optional)". Aunque usará un contador definido por Xilinx, aún no se ha sintetizado. Por ello, omita esta opción oprimiendo el botón <Next>.

2. Incluir el recurso *Binary Counter* desde el catálogo IP.

En la ventana del margen izquierdo, "*Flow Navigator*", ir a la sección "*Project Manager*" y oprimir el icono "*IP Catalog*". En el catálogo de componentes (*Cores*), ir a la carpeta de elementos básicos (*Basic Elements*) y, luego, a la subcarpeta de contadores (*Counters*). Seleccionar el componente "*Binary Counter*" con un doble clic (Figura 8.1).

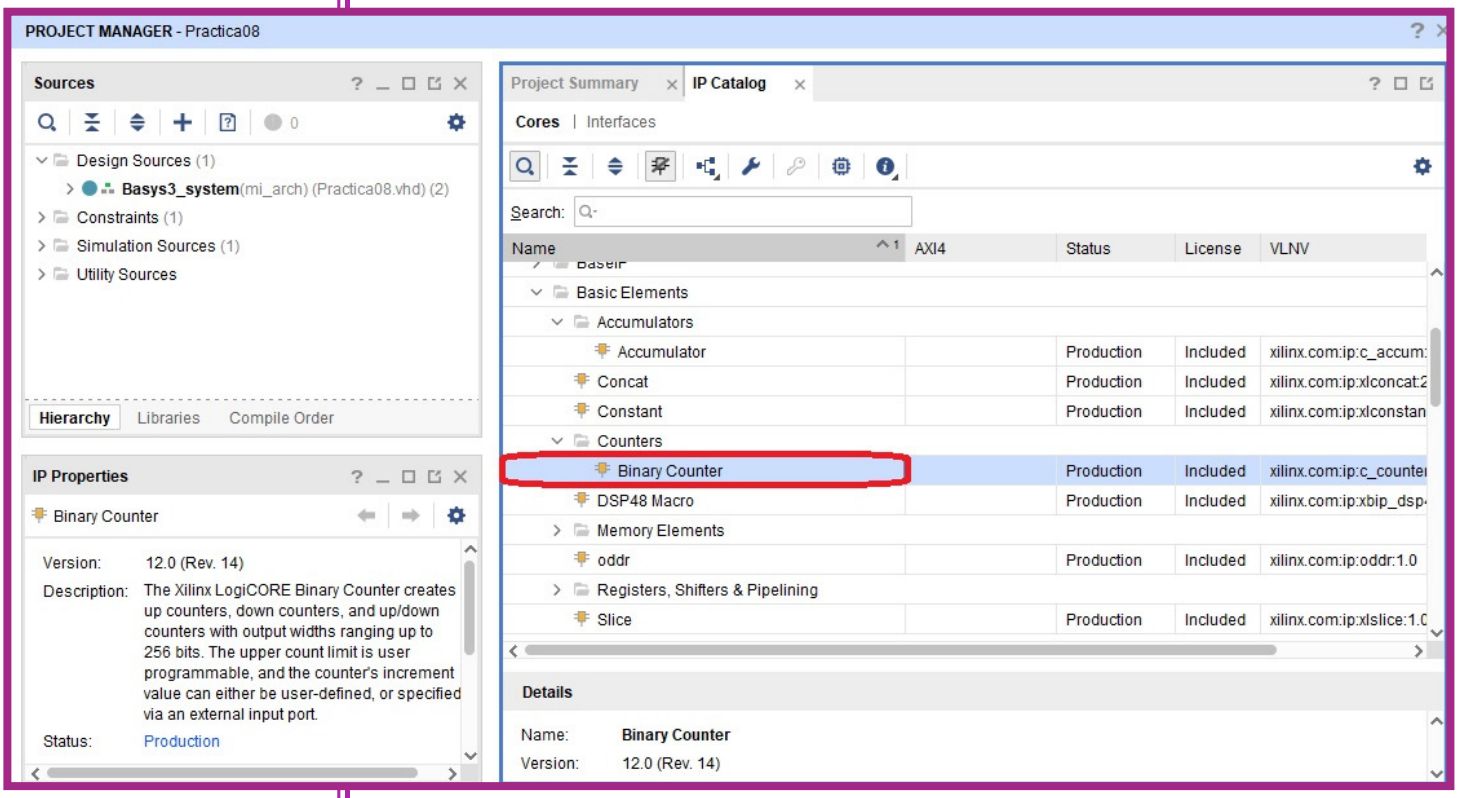


Figura 8.1. Vista del catálogo de componentes IP de Xilinx. En esta práctica se utilizará un componente Binary Counter.

Defina el nombre del componente, en este caso, emplee el nombre *Bin_Counter*. En la pestaña "*Basic*", coloque el valor 24 para el número de salidas (*Output Width*) (Figura 8.2).

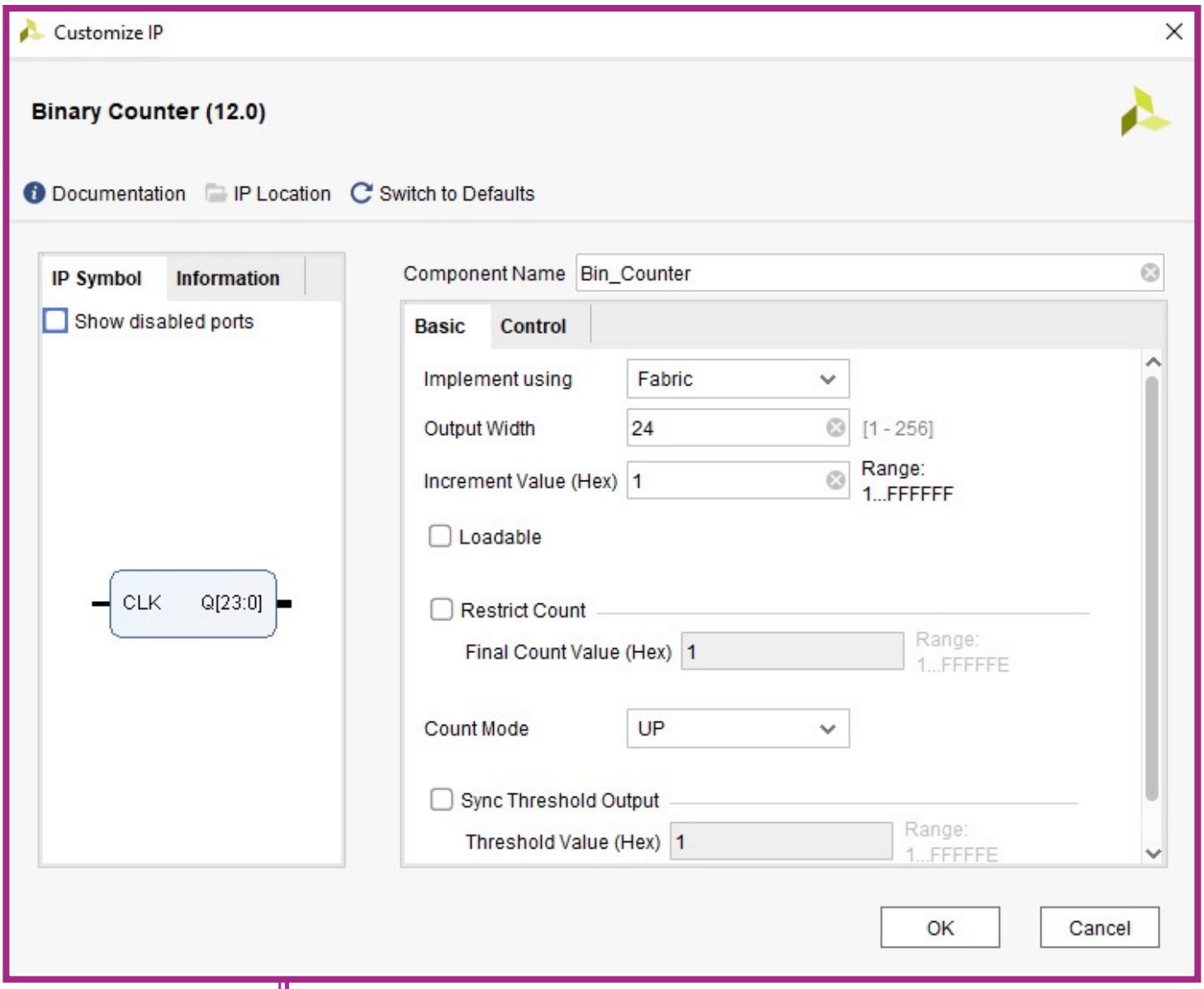


Figura 8.2. Ventana para establecer las características de nuestro componente Bin_Counter.

Deje a las demás opciones con los valores por defecto y oprima el botón <*OK*>. En su caso, también acepte la ruta donde se creará el nuevo componente. Entonces, aparecerá la ventana que nos indica que se creará el bloque Contador Binario (*Bin_Counter.xci*) (Figura 8.3).

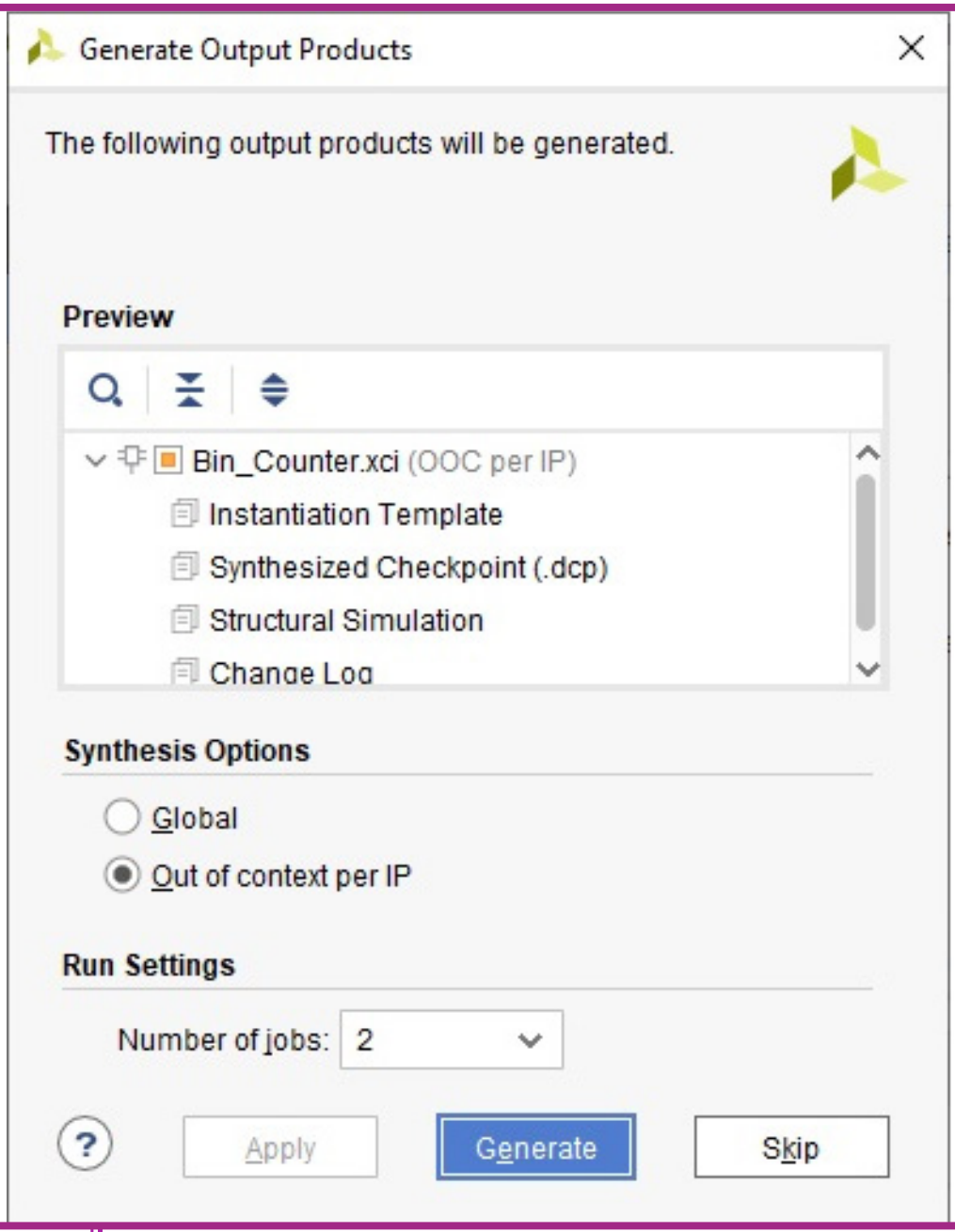


Figura 8.3. Ventana de confirmación del componente Bin_Counter.

Deje los valores por defecto y oprima el botón <*Generate*> para iniciar la síntesis del componente IP. En su caso, también acepte que la generación se realice en un segundo plano (*Out-of-context o* "fuera de contexto").

Después de un par de minutos, aparecerá el elemento IP listado en la pestaña de "*IP Sources*", de la ventana "*Sources*" (Figura 8.4).

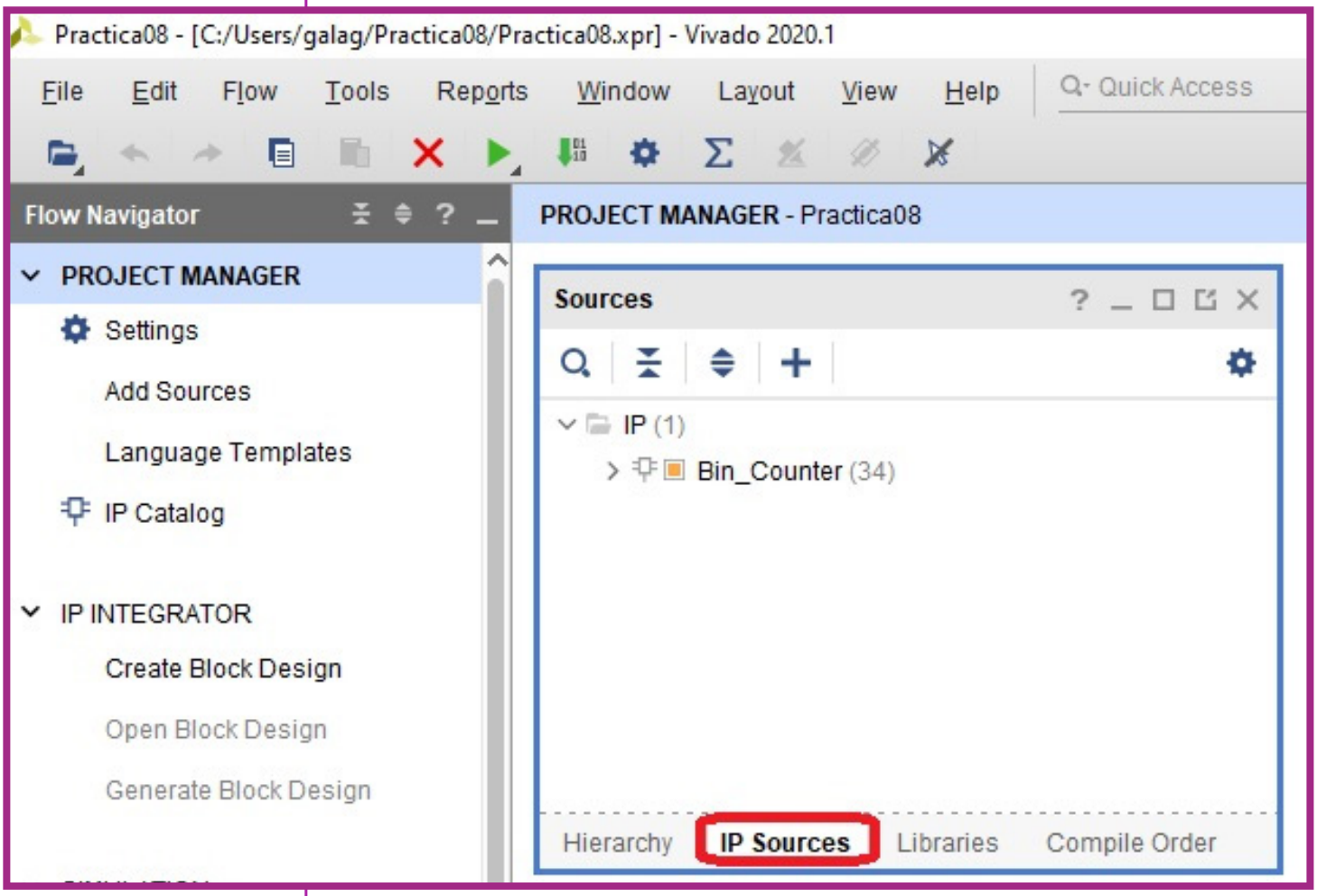


Figura 8.4. Vista del Project Manager después de la creación del componente Bin_Counter. También aparece listado como uno de los módulos invocados en el código principal (Basys3_system en Practica08.vhd).

Expanda el contenido asociado al núcleo del Contador (en nuestro caso, *mi_Contador: Bin_Counter*). Luego, dé doble clic en el archivo *Bin_Counter.vhd* (Figura 8.5).

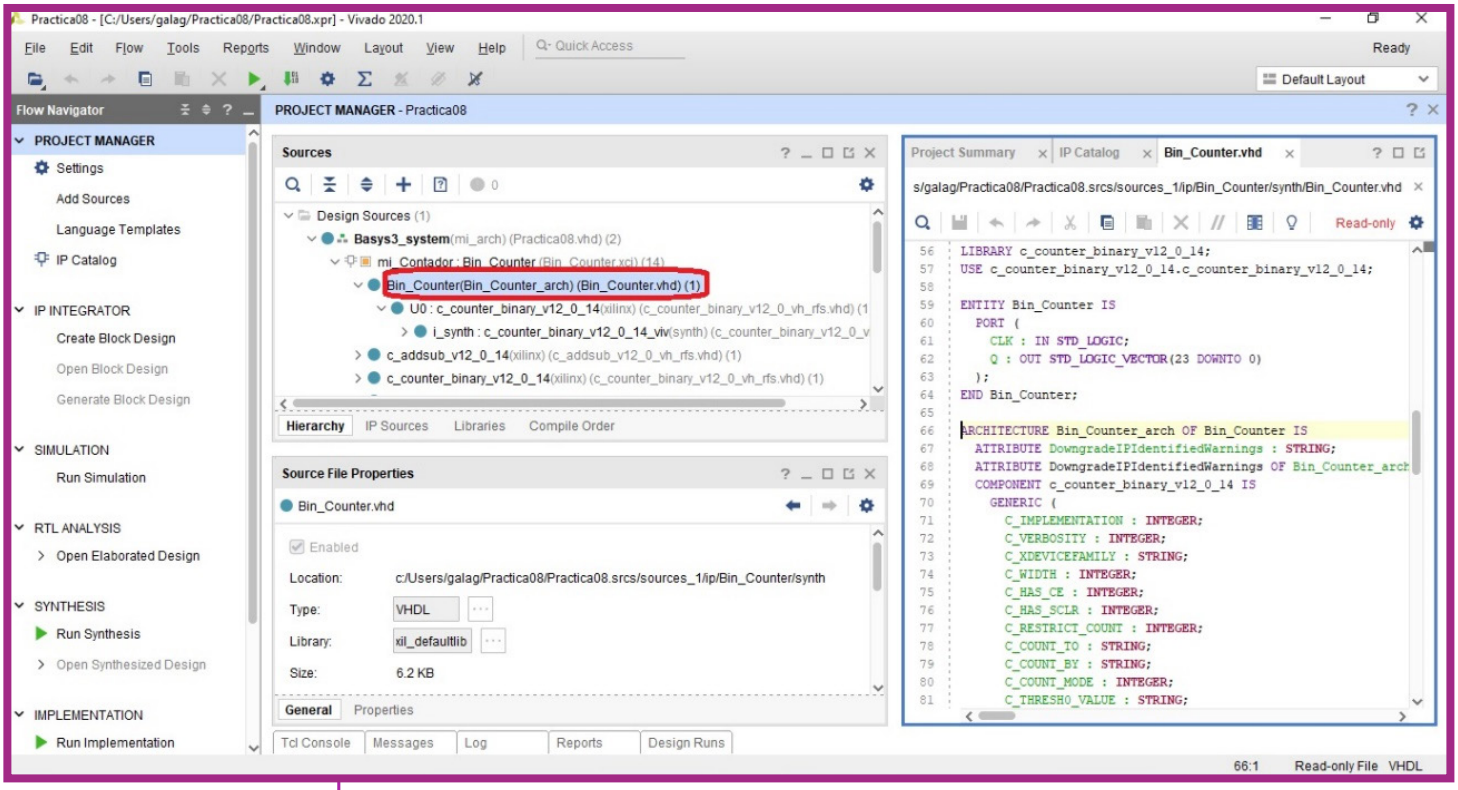


Figura 8.5. Vista del archivo VHDL que define al módulo Bin_Counter.

Use la información de este archivo para declarar al componente Bin_Counter y crear una instancia dentro del código principal (*Basys3_system* en *Practica08.vhd*) de nuestro proyecto.

3. Inclusión de instancias del contador y del registro.

El código *Practica08.vhd* contiene la declaración y la instancia del contador *Bin_Counter*, de acuerdo con lo sugerido en el archivo *Bin_Counter.vhd*, así como una instancia del componente registro (*Register*) (Figura 8.6).

Específicamente, se declaró al contador binario que se usará dentro de la sección reservada para la declaración de componentes:

```
COMPONENT Bin_Counter
  PORT (
    clk : IN STD_LOGIC;
    q : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END COMPONENT;
```

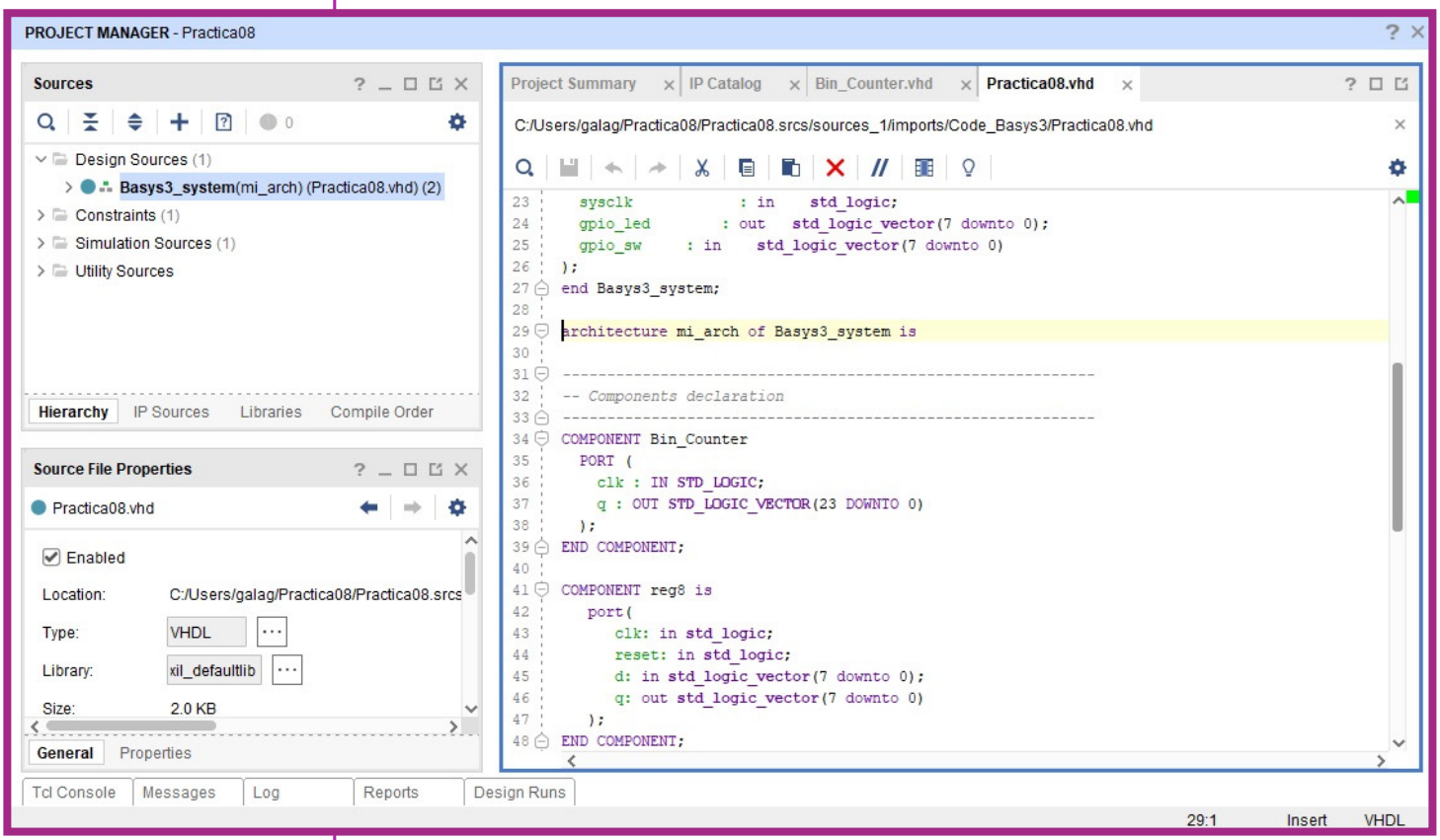


Figura 8.6. Vista del archivo VHDL que define al módulo principal (Basys3_system), en donde se muestra la declaración de sus dos componentes: Bin_Counter y reg8.

También se incluye la declaración del componente reg8:

```
COMPONENT reg8 is
  port(
    clk: in std_logic;
    reset: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
  );
end COMPONENT;
```

Luego, dentro de la arquitectura *mi_arch*, justo después de la palabra clave *begin*, aparece la instancia del contador binario:

```
Mi_Contador : Bin_Counter
  PORT MAP (
    clk => sysclk,
    q => usrclk
  );
```

Así mismo, se encuentra una instancia, con el nombre *mi_Register*, para el componente de registro *reg8*, asignando las señales y terminales correspondientes:

```
mi_Register : reg8
  PORT MAP (
      clk=>usrclk(23),
      reset=>mi_reset,
      d=>gpio_sw,
      q=>gpio_led
);
```

En este código, se emplea el contador como un divisor de frecuencia para el reloj del sistema. Nótese cómo la derivación con la señal de reloj más lenta ($usrclk(23)$) se aplica a la entrada de reloj del registro $mi\_Register$.

4. Realizar la construcción del proyecto y generar el archivo de bits de programación (ver Anexo 3).

5. Programar el proyecto construido en la tarjeta Basys 3 (ver Anexo 4).

Si la operación se concluyó con éxito, el dispositivo se comportará de acuerdo con lo programado: se trata de un registro con carga en paralelo y control por flanco de reloj. Mueva los interruptores y compruebe como cambian los LED asignados a la salida del registro después de un brevísimo instante (el periodo de la señal $usrclk(23)$ que tarda el registro en actualizar su contenido. El programa se mantendrá operando mientras la tarjeta no se apague. Una vez que se apaga la tarjeta, el dispositivo será recargado con el programa especificado por el modo indicado por el puente JP1.

ACTIVIDAD

A partir del código trabajado en esta práctica, haga los ajustes necesarios para crear un módulo (entidad) para un registro de 16 bits, con su propia interfaz de entrada/salida y su arquitectura en un archivo independiente. Luego, incluya el componente de registro en el código top (*Basys3_system* en Practica08.vhd) y sustituya al registro de 8 bits. Conecte todas las entradas y salidas de su nuevo registro y pruebe que opera conforme lo esperado. Infiera el diagrama lógico a bloques del diseño y haga el diagrama esquemático usando una herramienta CAD. ¿Por qué se conecta el reloj del registro a la salida 23 del contador? Respecto del punto de vista del registro, ¿cuál es la función del contador?

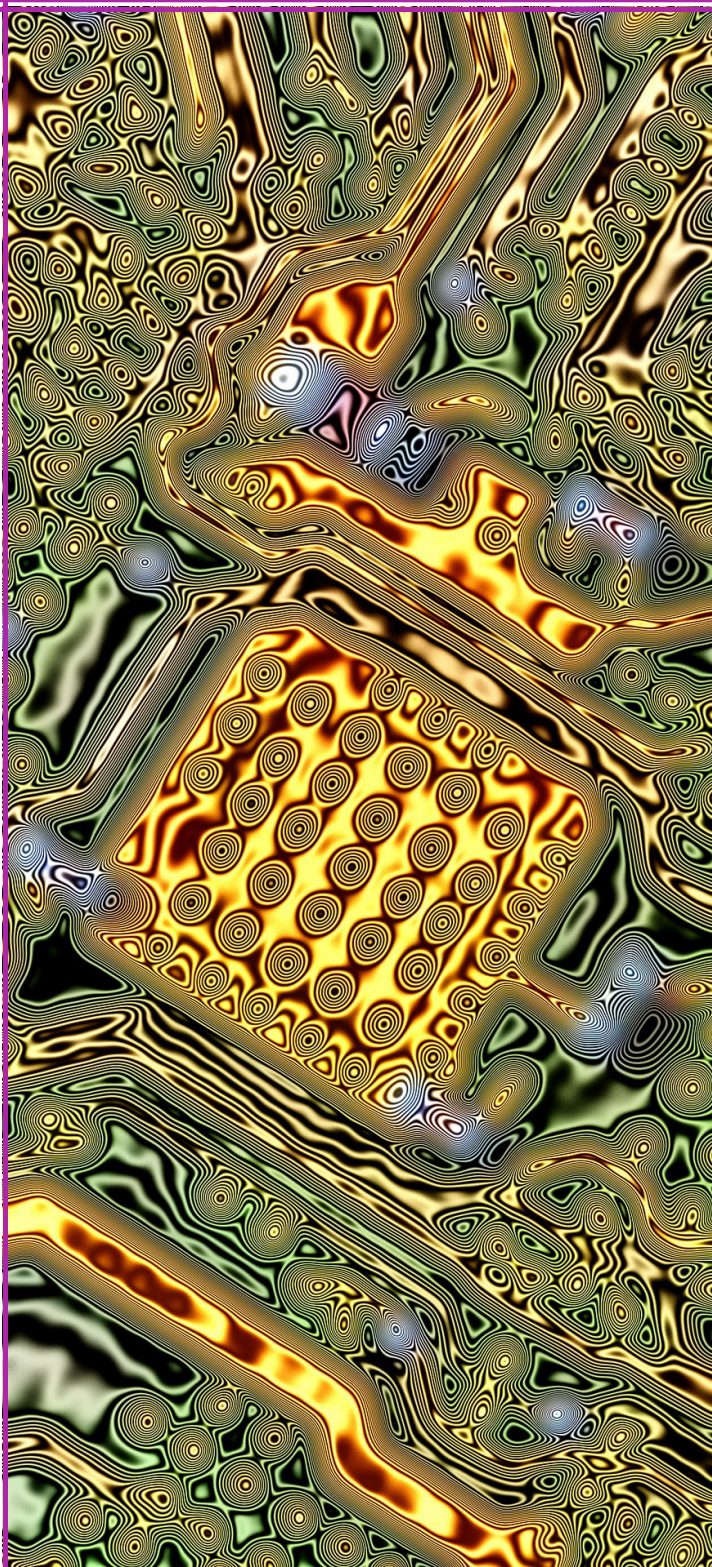
Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Dentro del reporte, en la parte de discusión de resultados, conteste a las preguntas y explique el funcionamiento del circuito. Acompañe su reporte con un anexo que incluya el diagrama a bloques solicitado.

Referencias para consulta

- **Chu, Pong P. (2008).** FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- **Digilent (2020).** Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- **Floyd, T. L. (2014).** Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- **Laguna, G. (2020).** Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- **Xilinx (2020).** 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 9

Proyecto VHDL para un bloque de lógica combinatoria



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVO

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

ANTECEDENTES

A lo largo de las prácticas anteriores se han diseñado circuitos de lógica combinatoria, es decir circuitos en donde cada una de sus salidas están dadas como funciones de la combinación actual de las entradas (ver Figura 9.1).

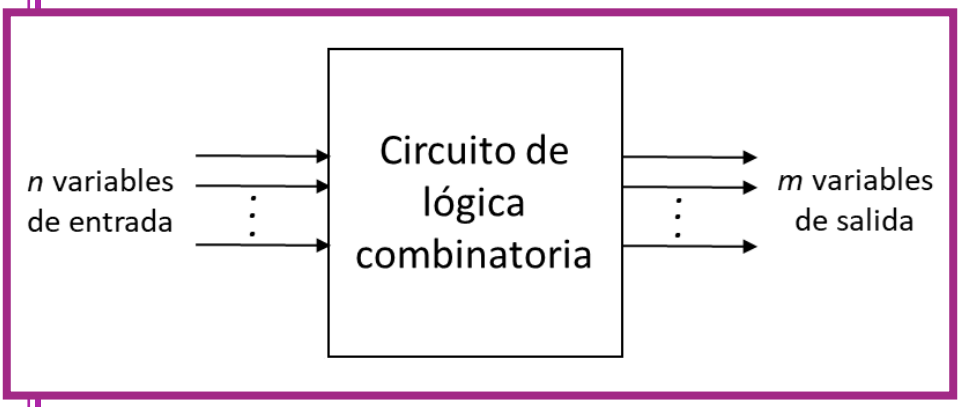


Figura 9.1. Diagrama a bloques de un circuito de lógica combinatoria.

El desarrollo de un proyecto VHDL para un bloque de lógica combinatoria, involucra conocimientos tanto teóricos como prácticos. En lo que respecta a los conocimientos teóricos, es fundamental ser capaces de obtener las formas canónicas de funciones booleanas, aplicando los enfoques de minitérminos o maxitérminos, así como ser capaz de simplificar dichas expresiones, mediante mapas de Karnaugh o álgebra de Boole. En lo que respecta a los conocimientos prácticos, se deben tener conocimientos para la creación de proyectos en el entorno Vivado, así como diseñar e interconectar

los distintos módulos del proyecto. Finalmente, para probar nuestro diseño en la tarjeta destino, es necesario realizar una correcta asignación de terminales y restricciones de hardware, así como hacer un uso correcto de la tarjeta. Todos estos conocimientos están contenidos en las anteriores prácticas del manual y se aplican e integran en esta práctica.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán el código, disponible en el Anexo 11, que continuación se describe:
- Empty_Top_Basys3_template.vhd. Contiene la plantilla (template) del código para la entidad de más alto nivel, es decir, el código principal, que representa a la tarjeta, en forma lógica y a nivel de código, y que invoca a todos los demás códigos.

Una vez que tenga el archivo mencionado en su computadora, proceda con los siguientes pasos:

1. A partir de la siguiente tabla de verdad, obtenga las funciones booleanas en su forma canónica, simplifíquelas con mapas de Karnaugh.

Entradas			Salidas		
e_0	e_1	x	n_0	n_1	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	0

2. Usando la platilla de código, desarrolle un proyecto VHDL para probar que el diseño cumple con la especificación.

ACTIVIDAD

Práctica 9

Use los recursos de la tarjeta Basys 3 para realizar y probar su diseño vía simulación con el entorno Vivado. Los interruptores (*sw*) se pueden usar para proporcionar las entradas y los LED para señalar el estado de las salidas. Revise el diagrama esquemático de la tarjeta para determinar la lógica (positiva/negativa) con que operan ambos. No olvide especificar en el archivo *.xdc los recursos de la tarjeta que van a ser usados, así como verificar la correspondencia de etiquetas.

Al momento de probar su diseño en la tarjeta Basys, no olvide tomar las precauciones necesarias para garantizar la integridad de la tarjeta (ver Anexo 4).

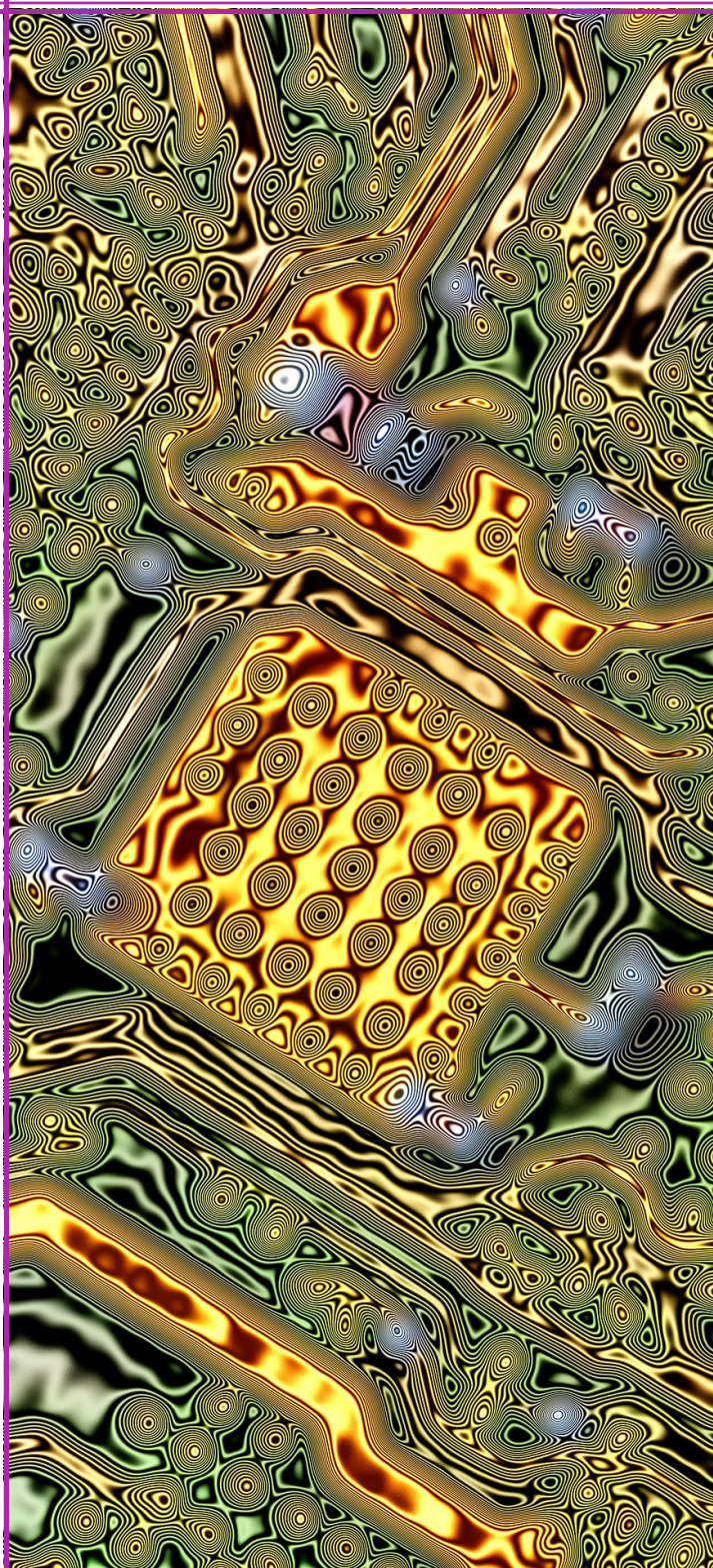
Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Reporte el desarrollo del diagrama a bloques (o esquemático), el código y el circuito obtenido, así como imágenes fotográficas de su experimento.

Referencias para consulta

- Chu, Pong P. (2008). FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- Floyd, T. L. (2014). Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- Xilinx (2020). 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

PRÁCTICA 10

Circuitos Secuenciales



Dirección
Ciencias Básicas
e Ingeniería

OBJETIVO

Desarrollar un proyecto VHDL para un circuito secuencial síncrono y aplicar los conocimientos adquiridos para realizar y probar el diseño obtenido con el FPGA y los recursos de una tarjeta de desarrollo.

ANTECEDENTES

En diseño lógico, se puede distinguir entre dos tipos de circuitos: combinatorios y secuenciales. Los circuitos combinatorios son circuitos en donde cada una de sus salidas están dadas como funciones de la combinación actual de las entradas. Estos circuitos fueron estudiados en las prácticas anteriores. Por otro lado, los circuitos secuenciales se caracterizan por su capacidad de transitar entre distintos estados, de modo que sus salidas y estado futuro están dados en función de su estado presente y la combinación de sus entradas. Por lo tanto, un circuito secuencial, además de contar con circuitos combinatorios, debe contar con elementos de almacenamiento que permitan almacenar el valor de su estado actual. La Figura 10.1 muestra el diagrama a bloques de un circuito secuencial. Note que el circuito secuencial determina el estado futuro a partir de las entradas y el valor del estado presente, el cual se almacena en el circuito.

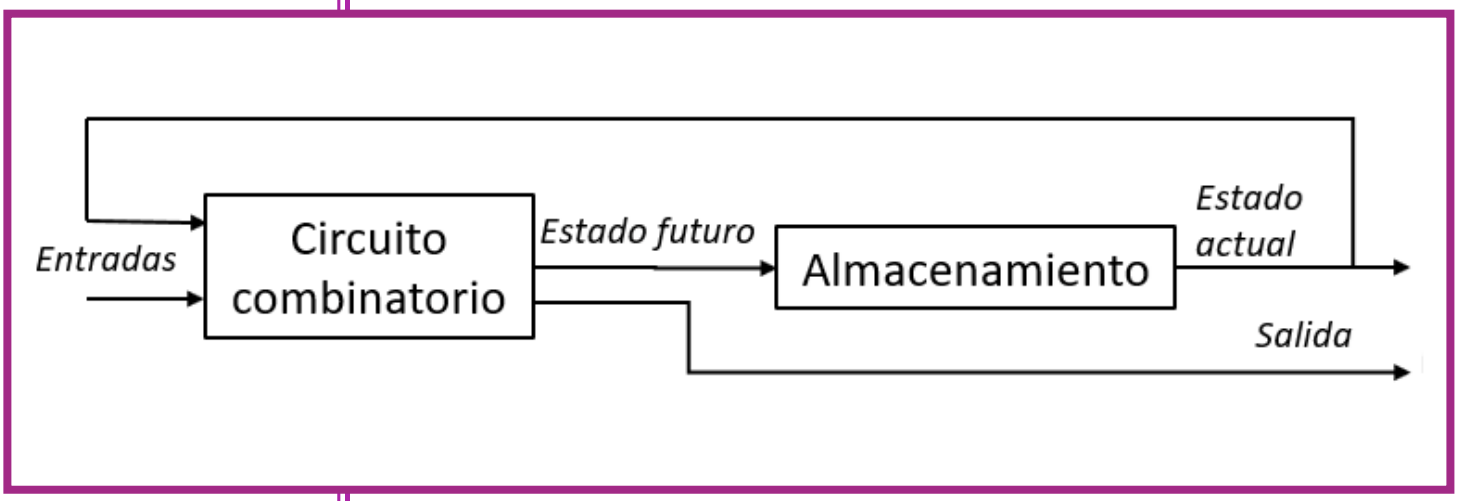


Figura 10.1. Diagrama a bloques de un circuito secuencial.

Otra característica importante de los circuitos secuenciales se refiere a la manera en que actualizan su estado, esta puede ser asíncrona o síncrona. Los circuitos asíncronos se actualizan en el momento que se detectan cambios en las señales de entrada. Los circuitos síncronos se actualizan en instantes discretos de tiempo definidos por un reloj.

Los circuitos secuenciales son fundamentales en el diseño de sistemas digitales ya que, además de ser la base de elementos de almacenamiento, registros, temporizadores, contadores, entre otros componentes primordiales, permiten el diseño de circuitos lógicos que operan de acuerdo con la especificación de una máquina de estado finito (FSM, por sus siglas en inglés). En esta práctica se realizará el diseño de un circuito secuencial síncrono a partir del diagrama de estados de una FSM.

MATERIAL

- Tarjeta con FPGA, modelo Basys 3 de marca Digilent.

- Computadora con software EDA Vivado instalado en su versión 2020.1 o compatible.

PROCEDIMIENTO

Para esta práctica, se usarán el código, disponible en el Anexo 11, que continuación se describe:

- Empty_Top_Basys3_template.vhd. Contiene la plantilla del código para la entidad de más alto nivel, es decir, el código principal, que representa a la tarjeta, en forma lógica y a nivel de código, y que invoca a todos los demás códigos.

Una vez que tenga el archivo mencionado en su computadora, proceda con los siguientes pasos:

1. A partir del diagrama de estados presentado en la Figura 10.2, que representa el funcionamiento de un circuito secuencial, obtenga la tabla de verdad correspondiente, las funciones booleanas en su forma canónica.

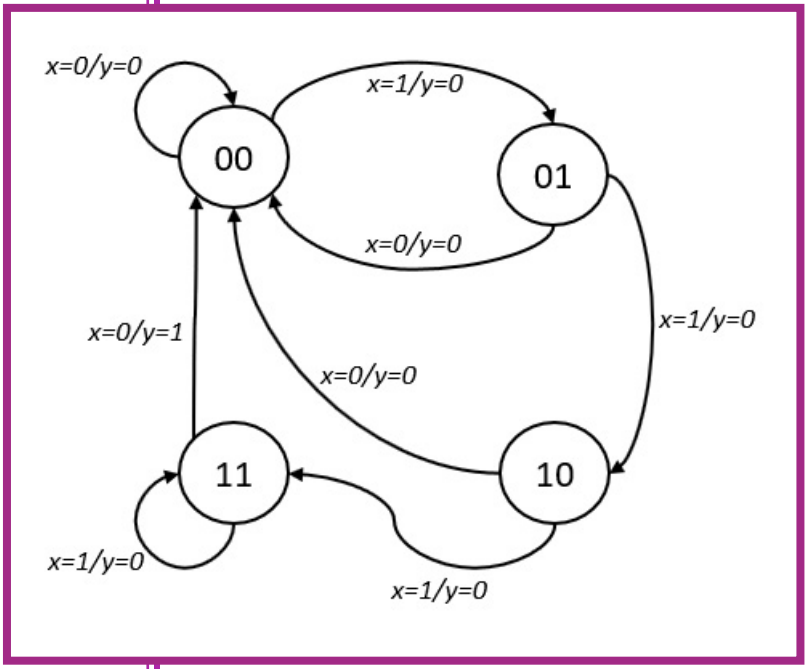


Figura 10.2. Diagrama de estados que representa el comportamiento de un circuito secuencial.

2. Aplique mapas de Karnaugh para simplificar las funciones obtenidas.

3. Usando la platilla de código, desarrolle un proyecto VHDL que integre todos los componentes necesarios (bloque de lógica combinatoria, registros y reloj del sistema) para la realización del circuito secuencial representado.

4. Probar que el diseño cumple con la especificación.

ACTIVIDAD

Práctica 10

Use los recursos de la tarjeta Basys 3 para realizar y probar su diseño. Los interruptores (sw) se pueden usar para proporcionar las entradas, los LED para señalar el estado de las salidas y un display LED de 7 segmentos para mostrar el estado actual del circuito secuencial. Como referencia, use los códigos VHDL de la Práctica No. 8, para la adecuada construcción de un registro y el conveniente uso del reloj del sistema. Lo ideal, para propósitos de prueba y depuración del código, es usar una frecuencia de reloj baja (digamos, de 10 ciclos por segundo o menos). No olvide especificar en el archivo *.xdc los recursos de la tarjeta que van a ser usados, así como verificar la correspondencia de etiquetas.

Al momento de probar su diseño en la tarjeta Basys, procure probar cada módulo (lógica combinatoria, registro, reloj, etc.) por separado, antes de integrarlos a todos ellos en un su proyecto final. Siempre tome las precauciones necesarias para garantizar la integridad de la tarjeta (ver Anexo 4).

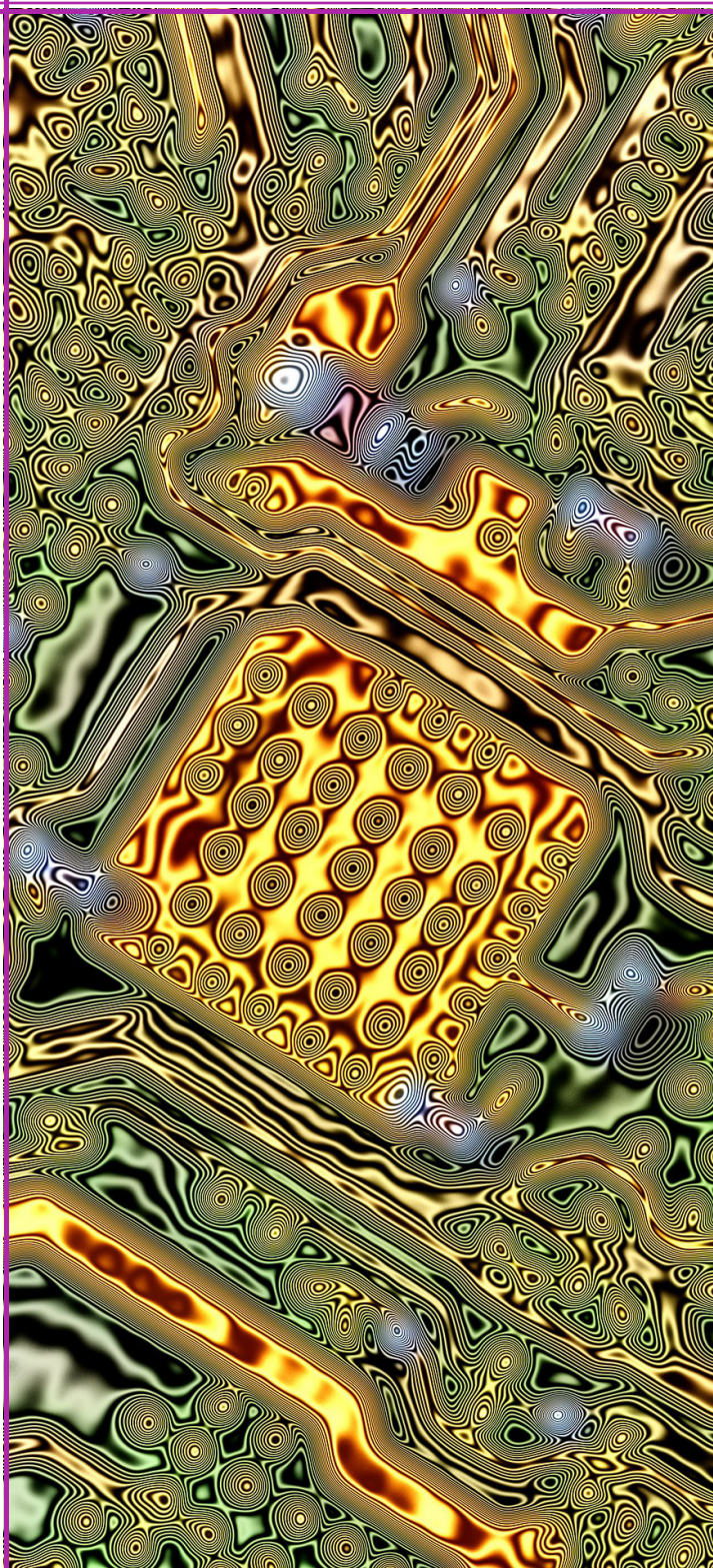
Por equipo y siguiendo las indicaciones del Anexo 3, realice el reporte de esta práctica. Reporte el desarrollo del diagrama a bloques (o esquemático), el código y el circuito obtenido, así como imágenes fotográficas de su experimento.

Referencias para consulta

- **Chu, Pong P. (2008).** FPGA Prototyping by VHDL Examples. EUA: Wiley-Interscience.
- Digilent (2020). Basys 3 Reference Manual. EUA: Digilent. Disponible en línea: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual
- **Floyd, T. L. (2014).** Digital Fundamentals (11a ed.). EUA: Pearson / Prentice Hall.
- Laguna, G. (2020). Video VHDL4Legos 02. WWW: Youtube. Enlace: https://youtu.be/ssaoJU74Ov8
- **Xilinx (2020).** 7 Series FPGAs Data Sheet: Overview (DS180). EUA: Xilinx. Disponible en línea: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

ANEXO 1

Formato para los reportes y metodologìa sugerida



Dirección
Ciencias Básicas
e Ingeniería

Formato para los reportes y metodologìa sugerida

Anexo 1

Se sugiere que, por cada práctica, los alumnos entreguen un reporte por equipo. Los reportes de prácticas permiten registrar cada paso de lo sucedido o lo realizado durante la práctica, lo que ayuda a los estudiantes a entender los resultados, así como recolectar y organizar la información en tablas, gráficas o ilustraciones, ayudan a determinar una explicación de los resultados de la práctica. Finalmente, los reportes ayudan a los profesores a valorar la apropiación que tienen los estudiantes de los conceptos, ya que los estudiantes deberán redactar con sus propias palabras el reporte. Así, estos reportes son la forma principal en que los alumnos comunican su trabajo.

Es importante recordar que un documento bien escrito al menos contiene un lenguaje claro, la sintaxis y la gramática son correctas, no contiene faltas de ortografía y utiliza correctamente las mayúsculas y minúsculas. Debe reflejar una idea clara de lo reportado describiendo con precisión lo realizado en la práctica, por lo que incluir diagramas y fotografías será un recurso valioso.

Se recomienda que el reporte no exceda las cinco cuartillas y que incluya las siguientes secciones:

- Título.
- Objetivo(s).
- Introducción con marco teórico y conceptual. No “copiar y pegar” ningún contenido o recurso digital. Parafrasear los contenidos e indicar la fuente consultada.
- Materiales y procedimiento (o metodología) sintético. No repetir textualmente el procedimiento de la práctica, parafrasear y sólo indicar, grosso modo, los pasos.

- Resultados y discusión de los resultados. Aquí van las imágenes (Incluyan, al menos, una imagen fotográfica de su experimento) y las tablas con las observaciones y mediciones, además de los comentarios al respecto que expliquen los resultados obtenidos. Asimismo, se deberá incluir una breve discusión sobre cómo los resultados permiten, o son relevantes, para llegar al objetivo de la práctica.
- Conclusiones. En esta sección se indica si se cumplió o no el objetivo de la práctica, así como sus impresiones generales sobre la actividad.
- Referencias a las fuentes consultadas. El listado de referencias deberá seguir algún formato de citas y referencias, preferentemente usar el formato de la IEEE.
- Anexo. Debe incluir el diagrama esquemático del circuito que armó, con todos los componentes o, en su caso, el código fuente VHDL modificado que describe al diseño lógico solicitado.

Anexo 1

De igual manera, se recomienda a los alumnos seguir algunos pasos generales para la implementación de las prácticas presentados en la Figura A1.1:

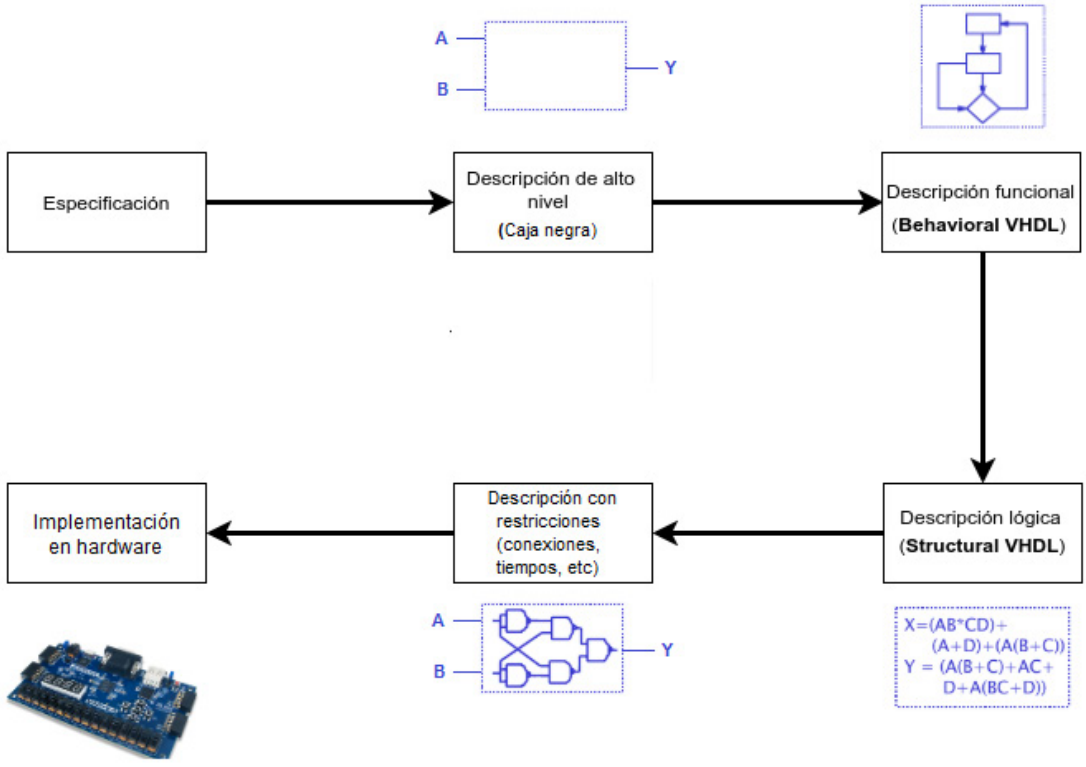
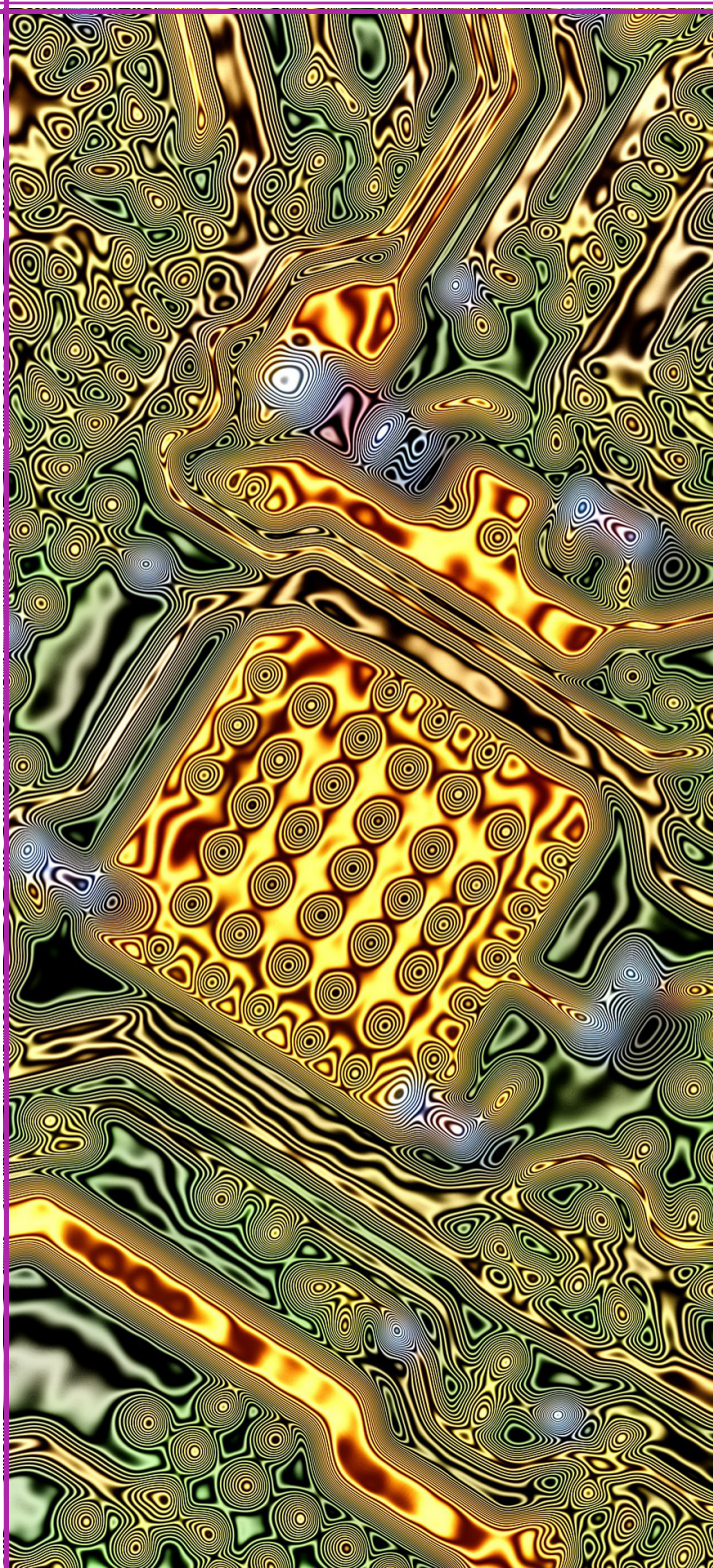


Figura A1.1. Diagrama de estados para la implementación de las prácticas.

ANEXO 2

Creación de proyectos VHDL en el entorno Vivado



Dirección
Ciencias Básicas
e Ingeniería

Formato para los reportes y metodologìa sugerida

Anexo 2

Creación de proyectos VHDL en el entorno Vivado

1. Iniciar la aplicación Xilinx Vivado (Figura A2.1).

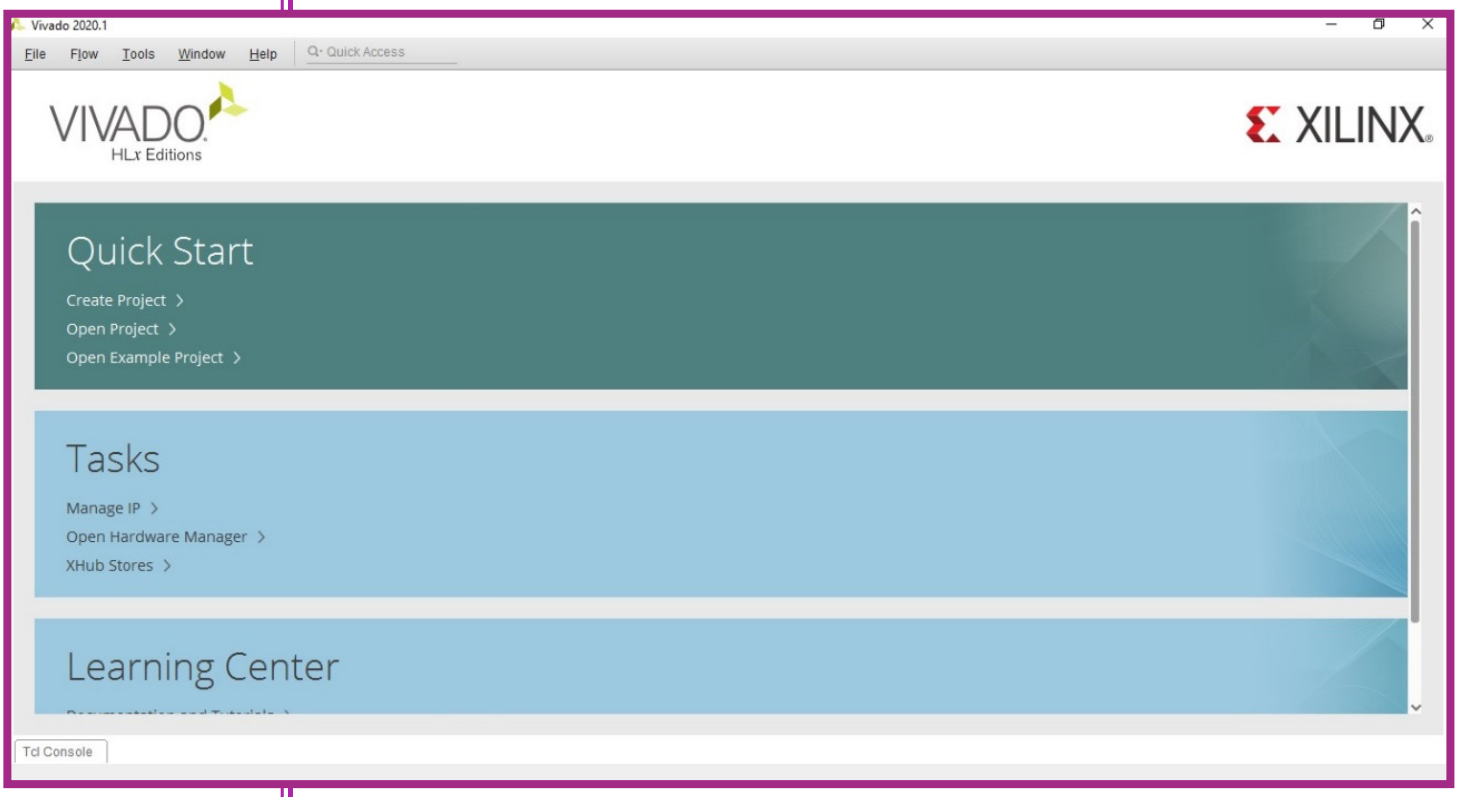
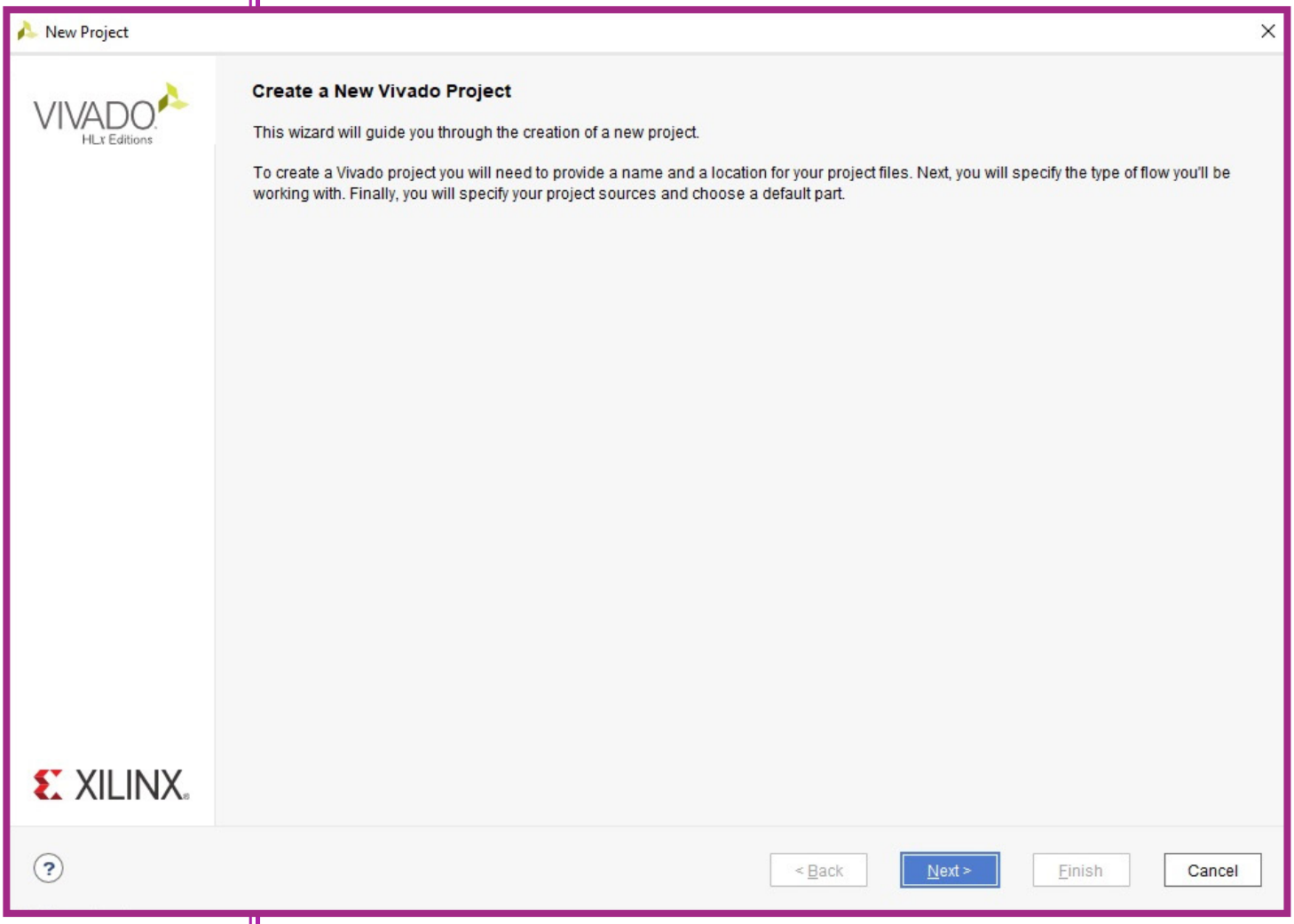


Figura A2.1. Ventana de inicio del entorno de desarrollo Vivado.

Anexo 2

2. Crear un nuevo proyecto.

Vaya a la sección de la ventana denominada Quick Start y oprima el icono “Create Project”.



Se abre la ventana “New Project” (Figura A2.2), entonces oprima el botón <Next>. A continuación, introduzca el nombre del proyecto, por ejemplo “Practica_xx” y defina la ruta del proyecto (Figura A2.3). La ruta no debe contener espacios. Dejar las demás opciones con los valores por defecto y oprima el botón <Next>.

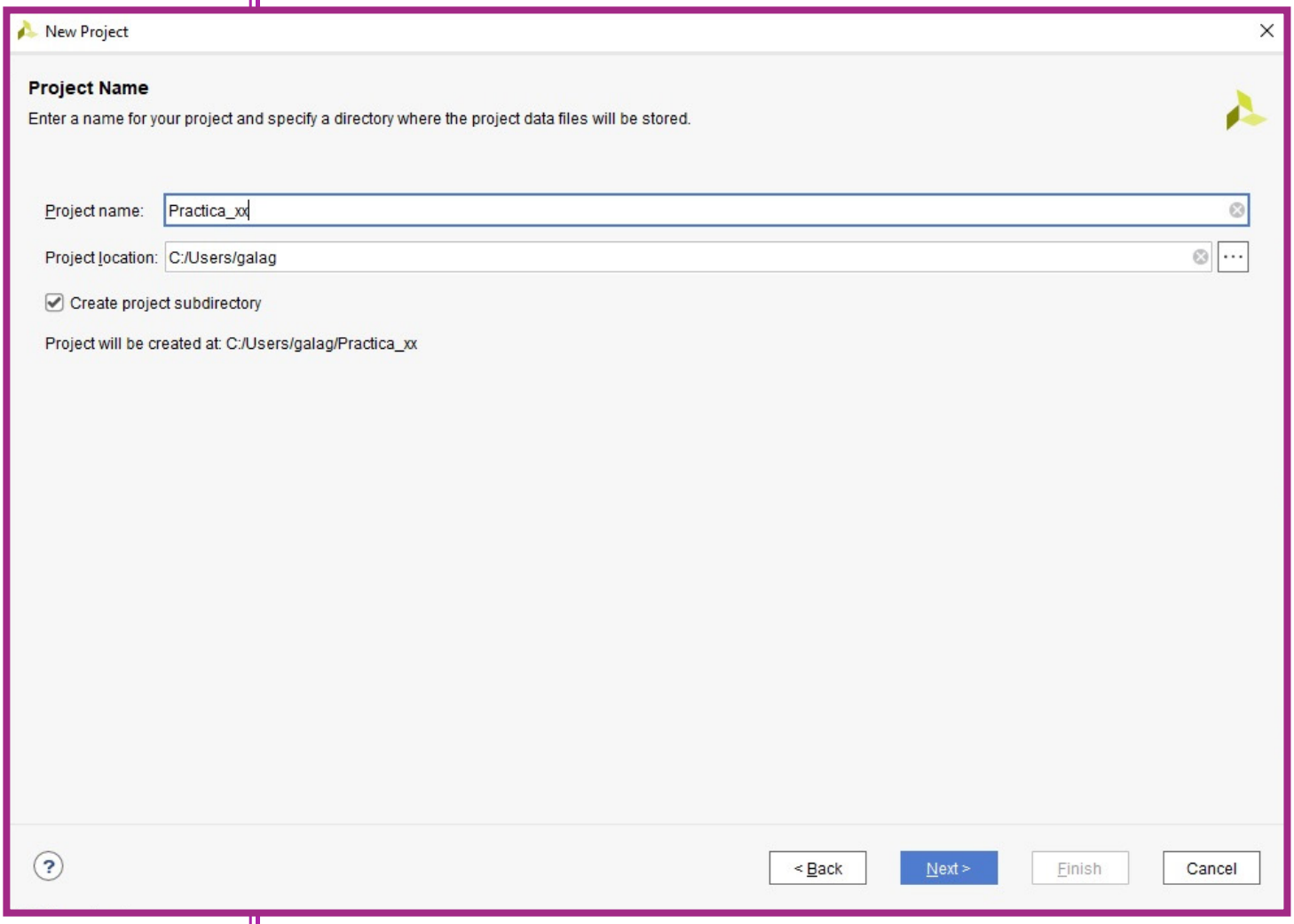


Figura A2.3. Ventana para indicar el nombre y ruta del nuevo proyecto.

En la siguiente ventana (Figura A2.4) aparecen las opciones "Project Type". Elija la opción "<*> RTL Project". Marcar la casilla "Do not specify sources at this time". Oprima el botón <Next>.

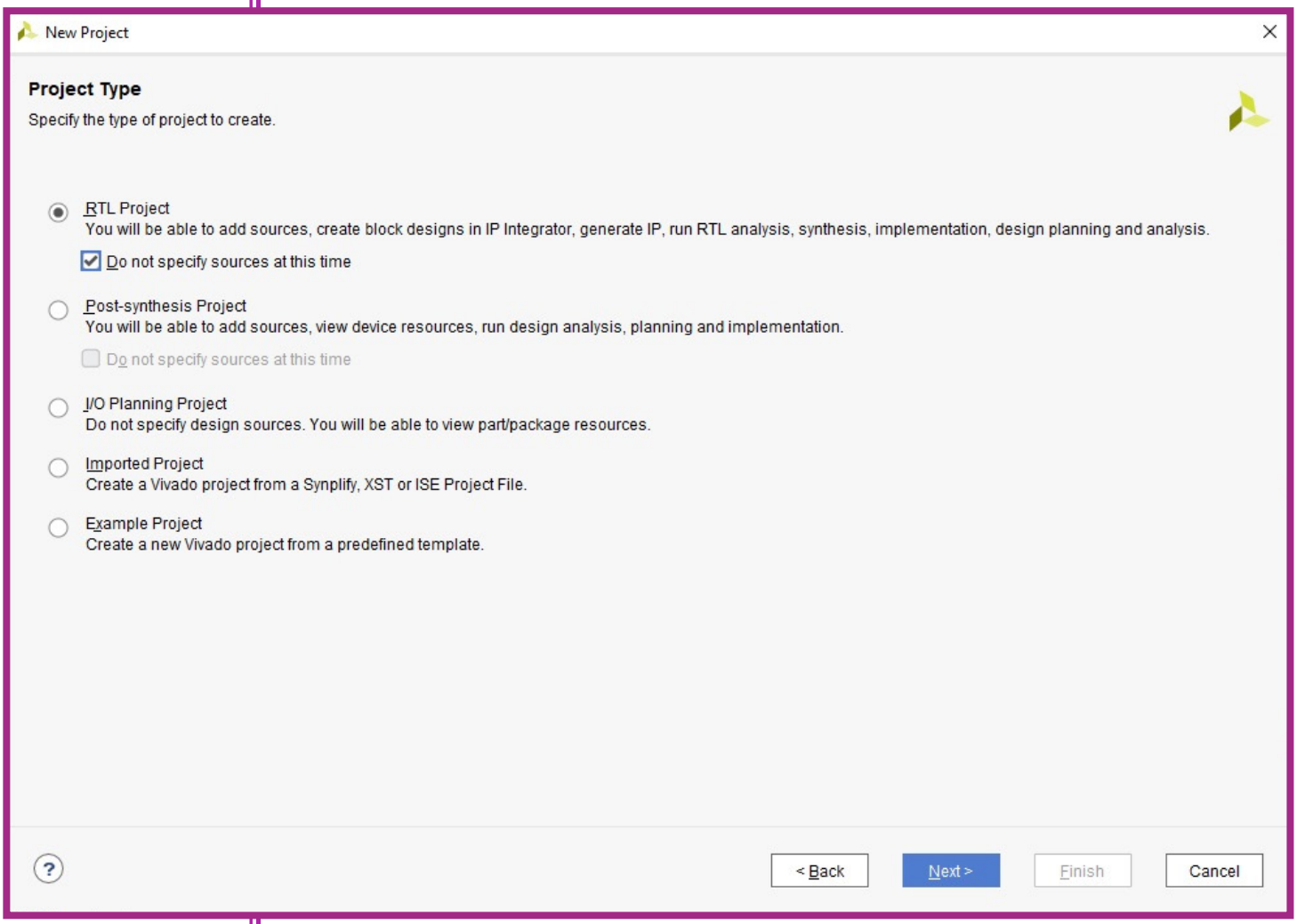


Figura A2.4. Ventana para indicar el tipo de proyecto, en nuestro caso RTL y especificar si es que se agregarán archivos fuente o no.

Ahora, aparecen la ventana con las opciones "Default Part" (Figura A2.5). Hay que especificar las características del dispositivo FPGA presente en la tarjeta de desarrollo. Para la tarjeta Basys 3:

- Family: Artix-7
- Package: CPG236
- Part: xc7a35tcpg236-1

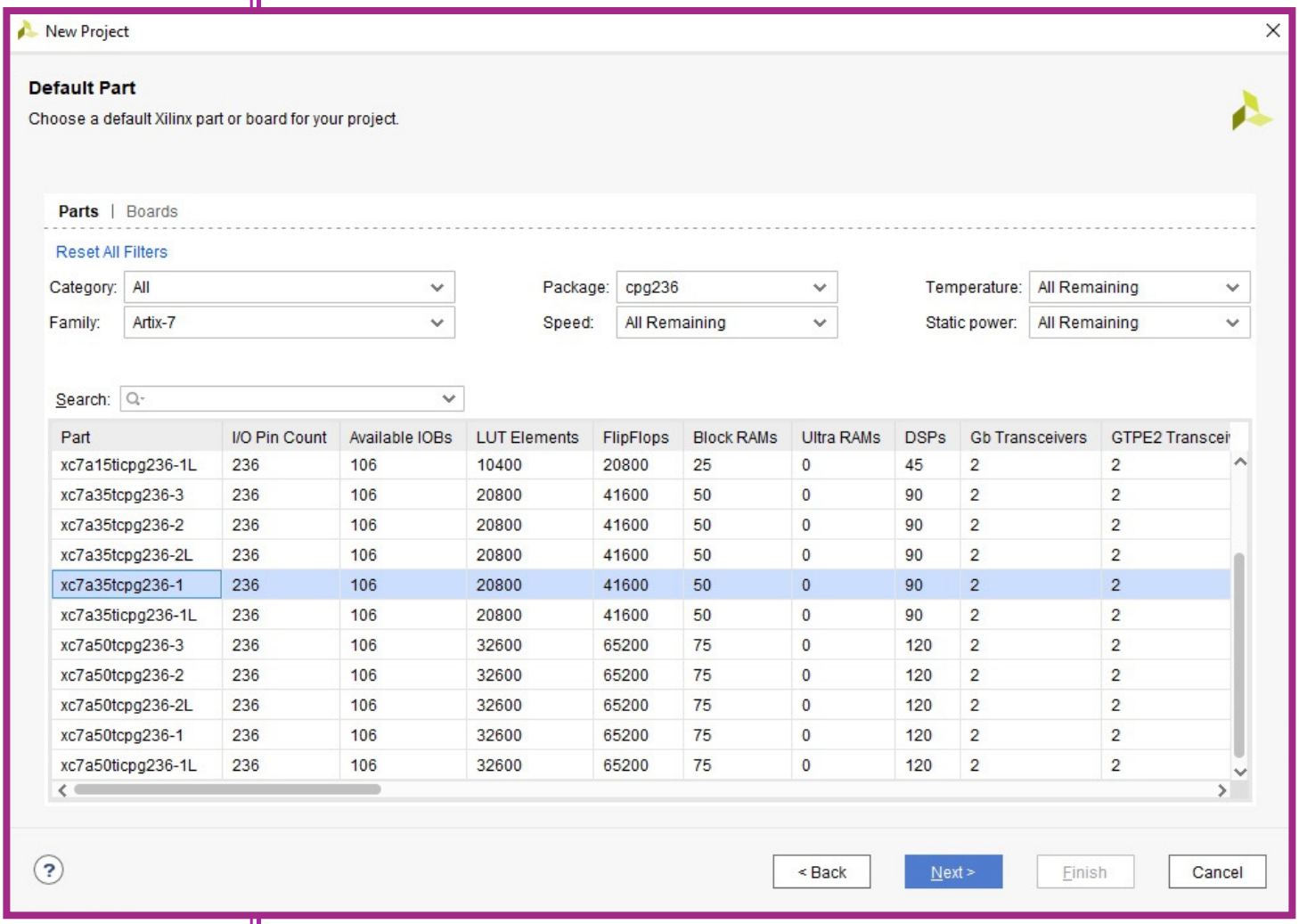


Figura A2.5. Ventana para especificar las características de la tarjeta de desarrollo.

Se oprime el botón <Next>, para ver el resumen, y luego el botón <Finish> para terminar. El proyecto ahora luce vacío, con una vista como la siguiente (Figura A2.6):

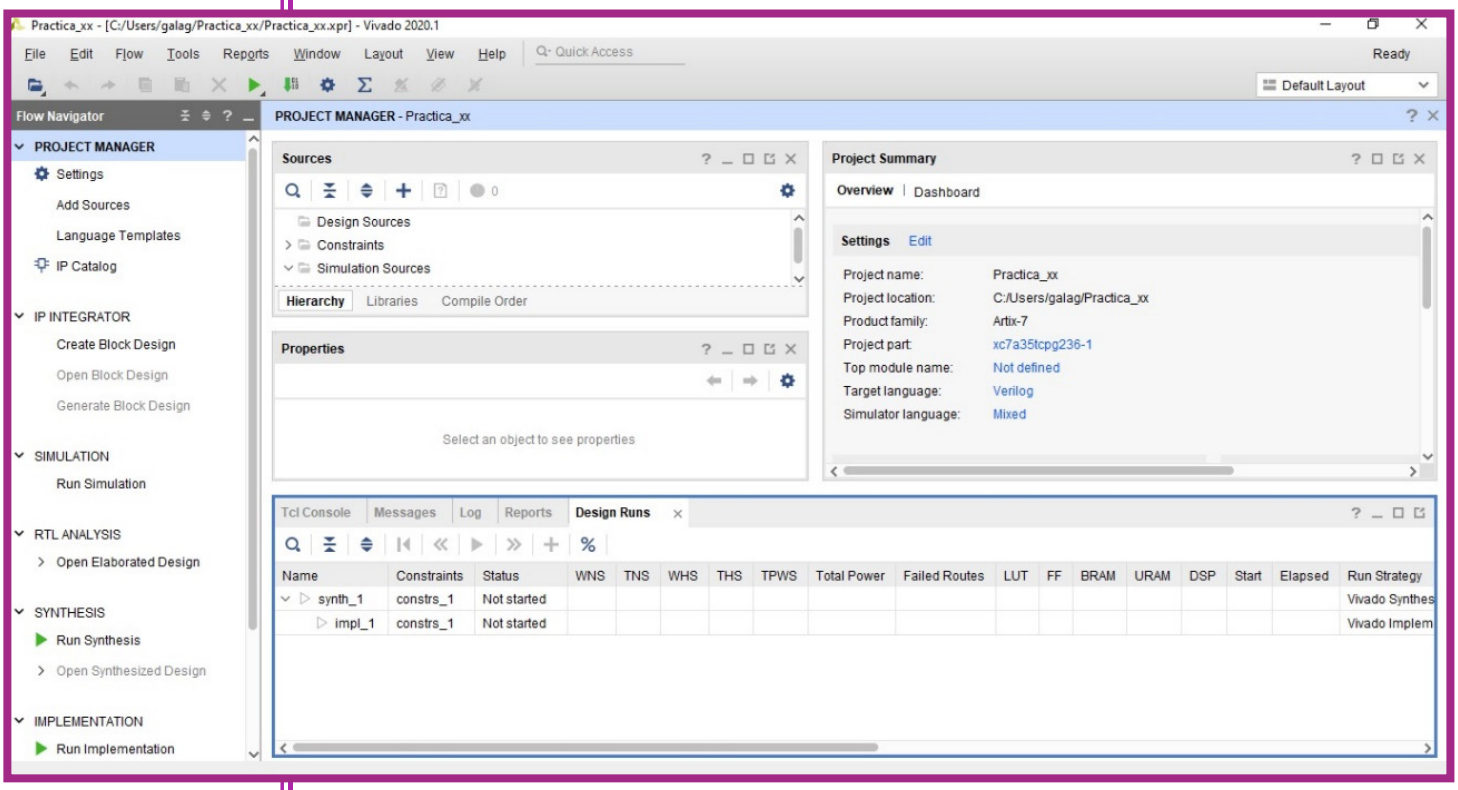


Figura A2.6. Vistas de las secciones "Project Manager" y "Project Summary".

3. Agregar el código fuente con la descripción del hardware.

En la ventana del margen izquierdo, “Flow Navigator”, ir a la sección “Project Manager” y oprimir el icono “Add Sources”. En la ventana que aparece (Figura A2.7) seleccionar la opción “<*> Add or create design sources”, luego oprimir el botón <Next>.

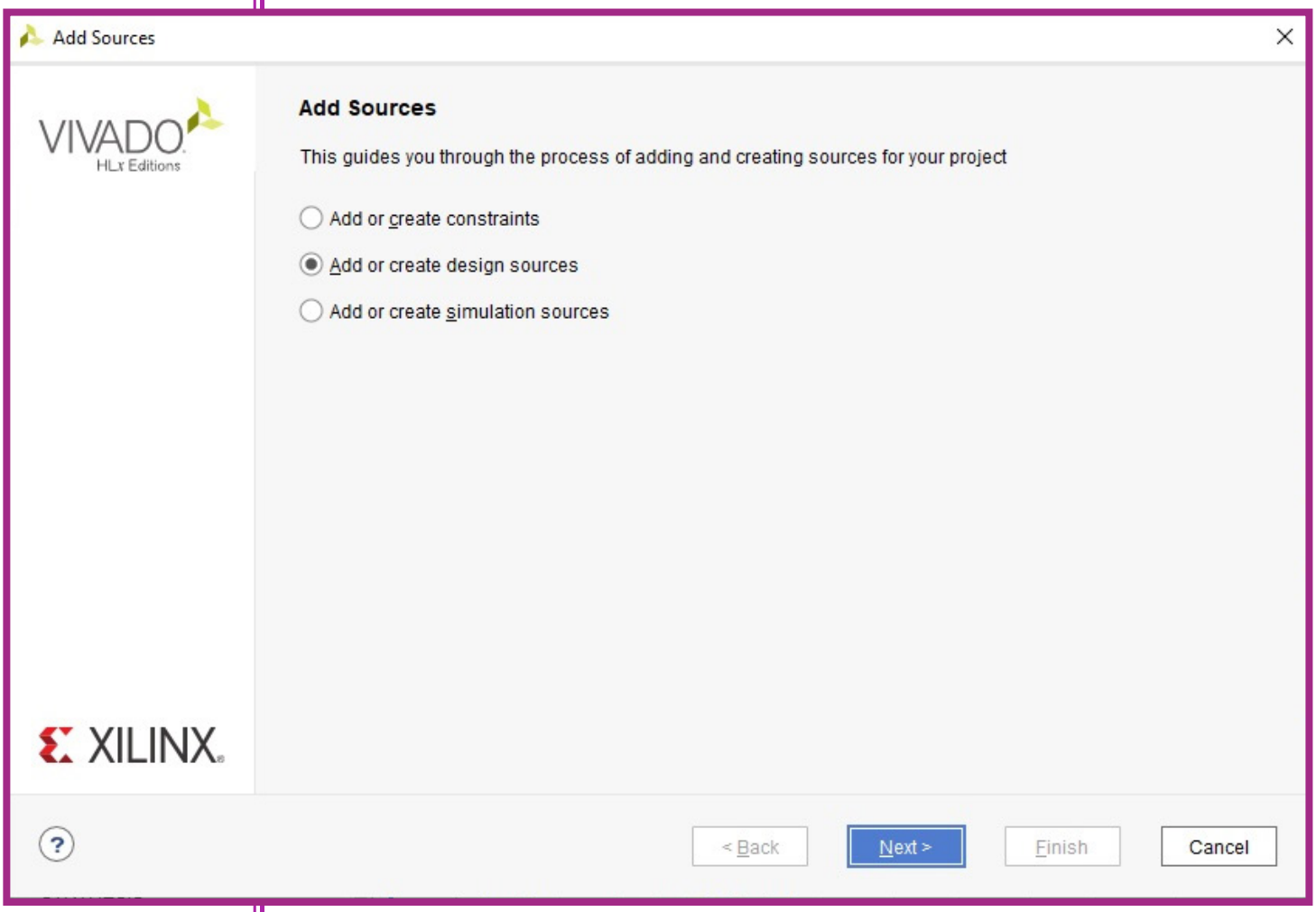


Figura A2.7. Ventana para elegir el tipo de archivos fuente que serán agregados.

A continuación, en la nueva ventana "Add Sources" (Figura A2.8), oprimir el botón <Add Files>.

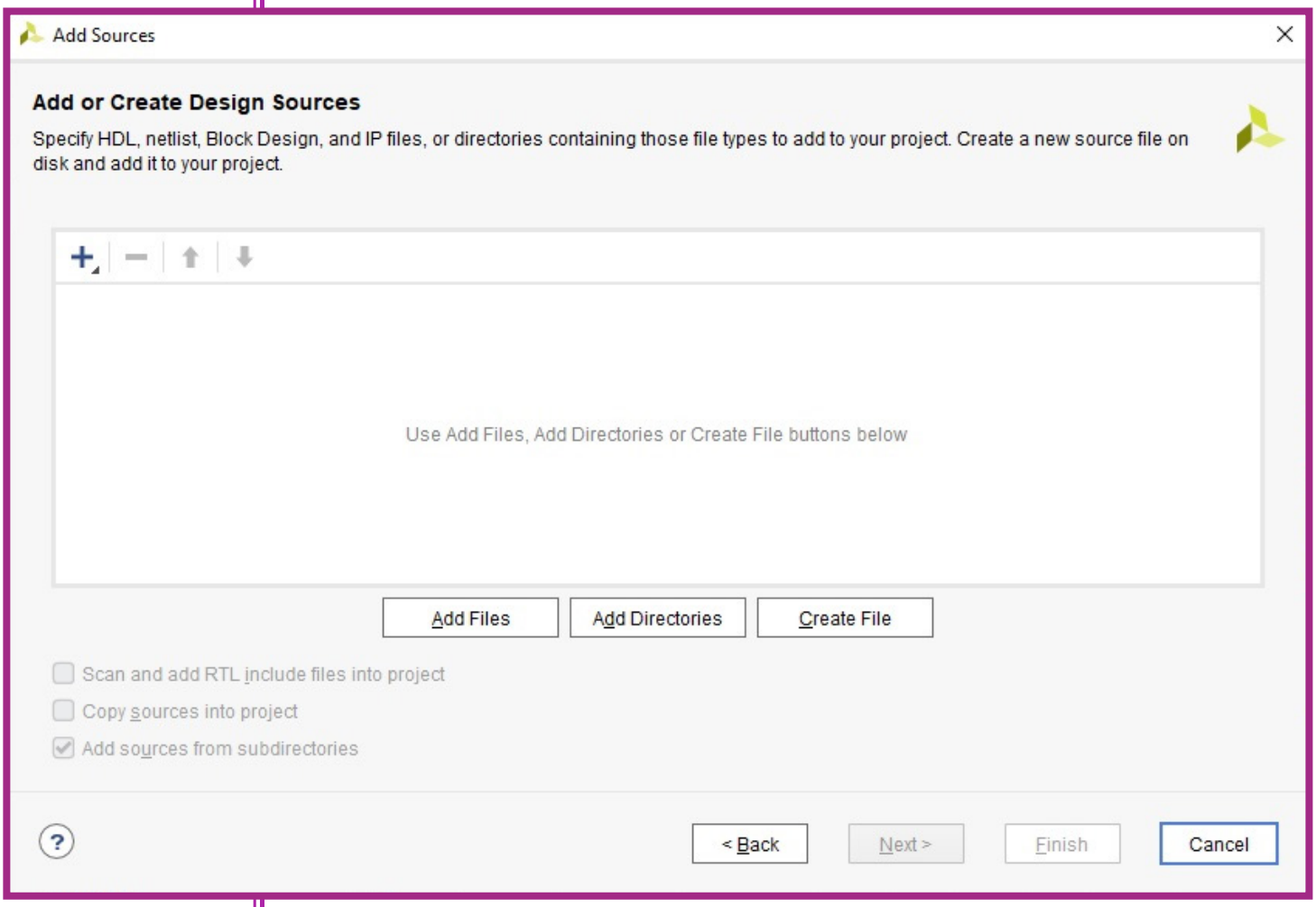


Figura A2.8. Ventana para elegir agregar o crear archivos fuente en un proyecto.

Entonces, proceda a seleccionar y agregar los códigos fuente, es decir, los archivos con código VHDL (Figura A2.9). Una vez seleccionados los códigos fuente, se oprime el botón <OK> y regrese a la ventana "Add Sources" (Figura A2.8). Marque la casilla "<*> Copy Sources into project" y oprima el botón <Finish>.

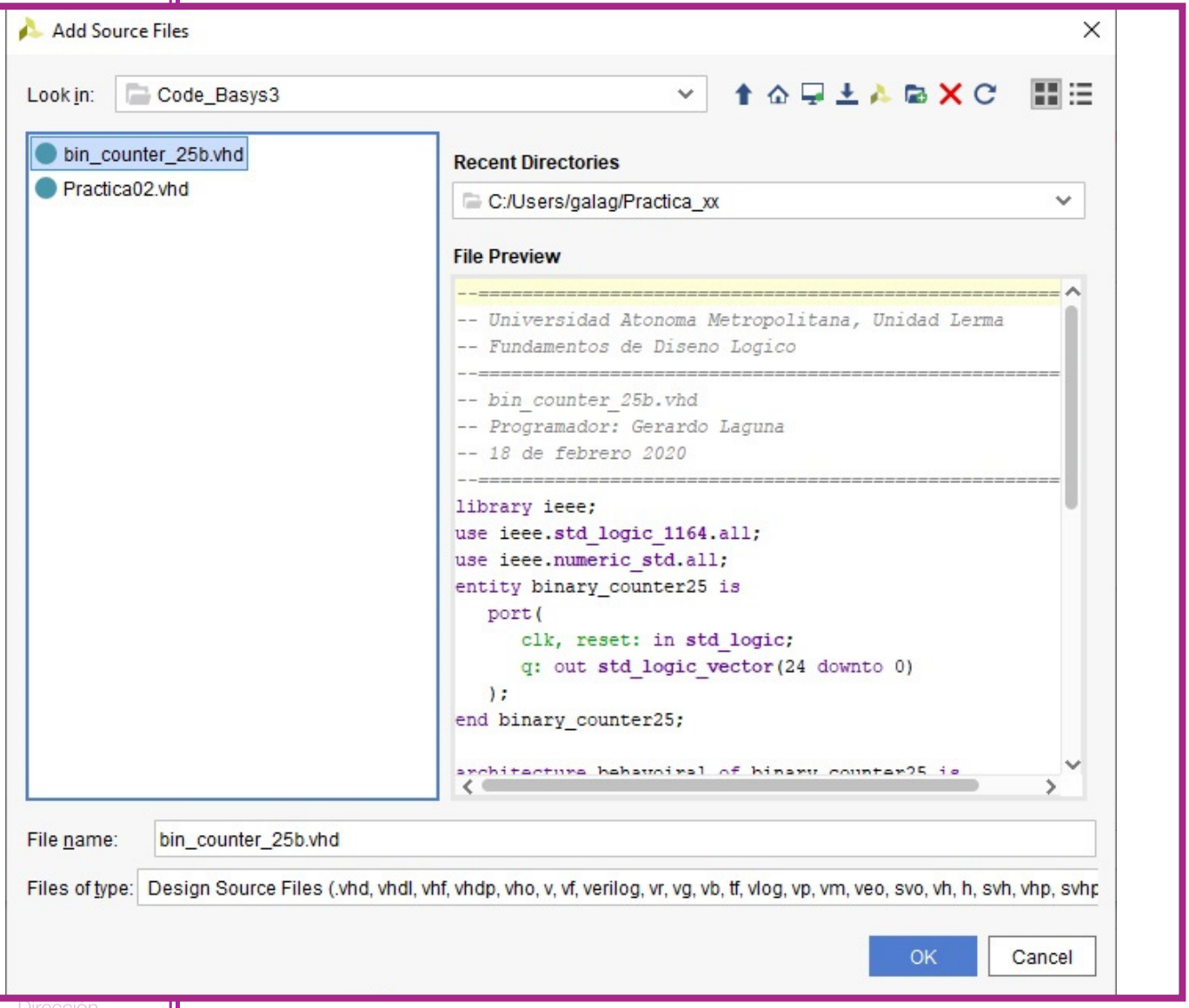


Figura A2.9. Ventana seleccionar los archivos fuente que serán agregados.

4. Agregar el código fuente con las restricciones del usuario.

En la ventana del margen izquierdo, “Flow Navigator”, ir a la sección “Project Manager” y oprimir el icono “Add Sources”. En la ventana que aparece (Figura A2.10) seleccionar la opción “<*> Add or create constraints”, luego oprimir el botón <Next>.

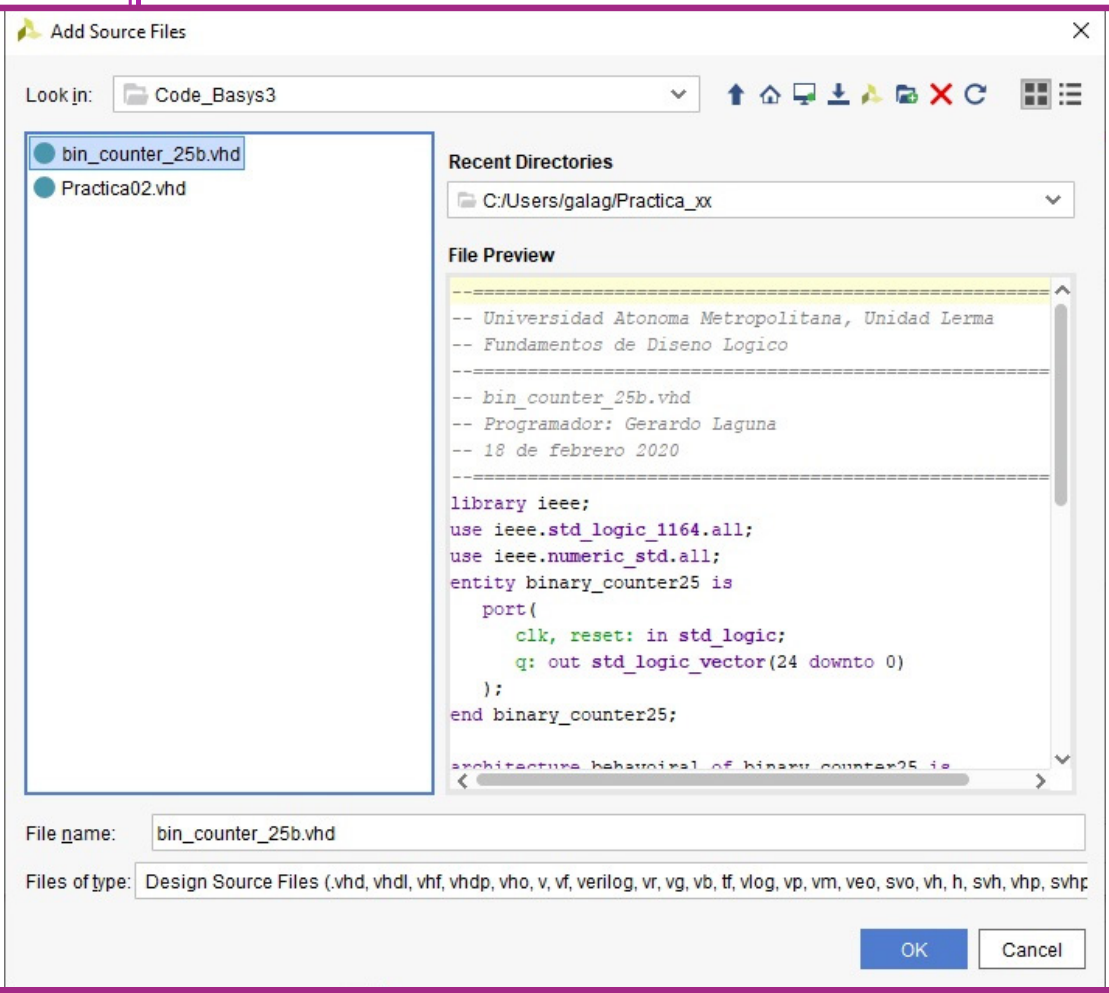


Figura A2.10. Ventana para elegir agregar o crear el archivo de restricciones, de conformidad con el hardware o la tarjeta de desarrollo que contiene al FPGA destino.

A continuación, en la nueva ventana “Add or Create Constraints” (Figura A2.11), oprimir el botón <Add Files>. Entonces, proceder a buscar y agregar el código de restricciones *.xdc (al archivo con las restricciones del usuario).

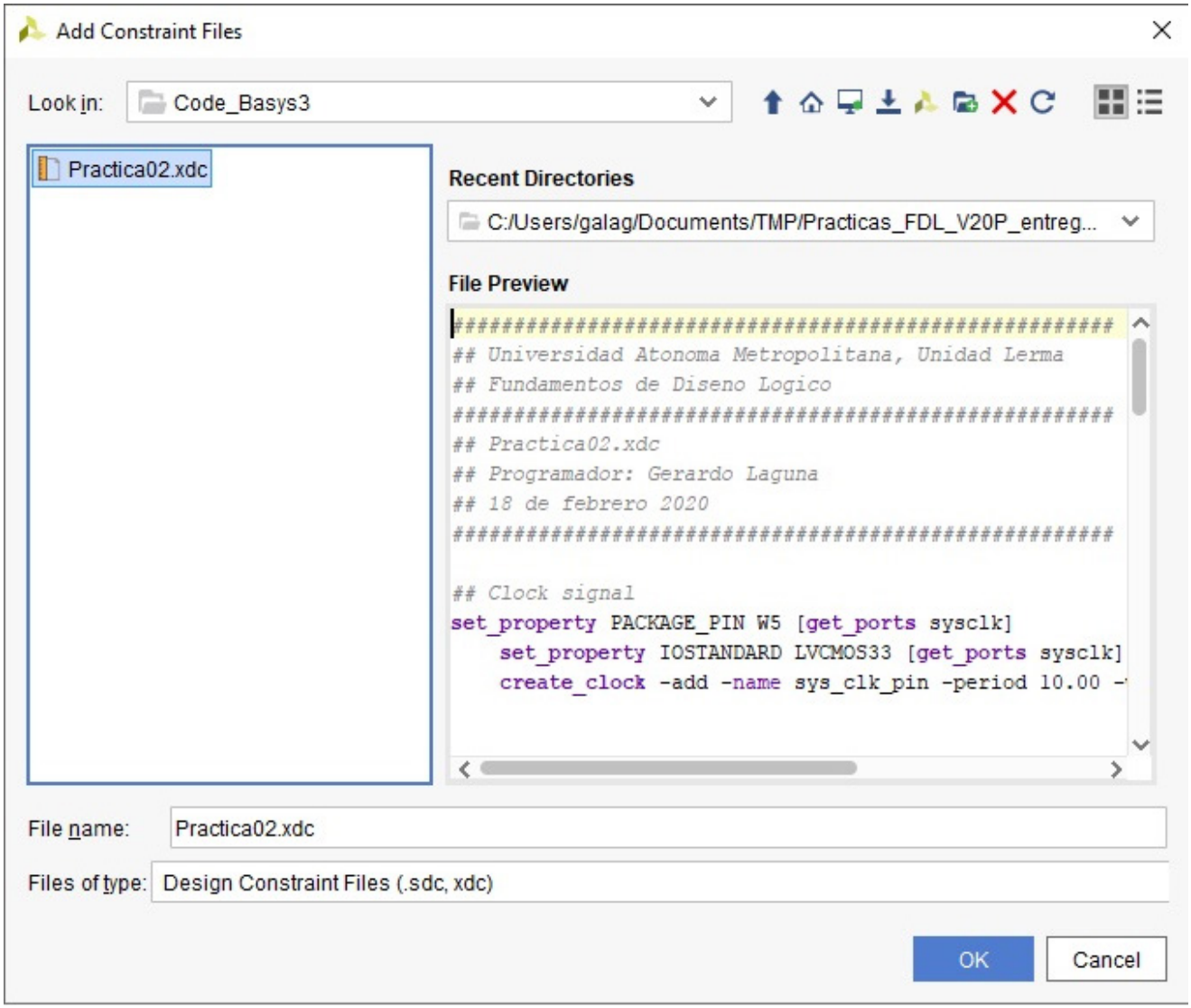


Figura A2.11. Ventana para seleccionar el archivo de restricciones del proyecto.

Una vez seleccionado el código de restricciones (Figura A2.12), se oprime el botón <OK> y regrese a la ventana "Add or Create Constraints" (Figura A2.10). Marque la casilla "<*> Copy Sources into project" y oprima el botón <Finish>.

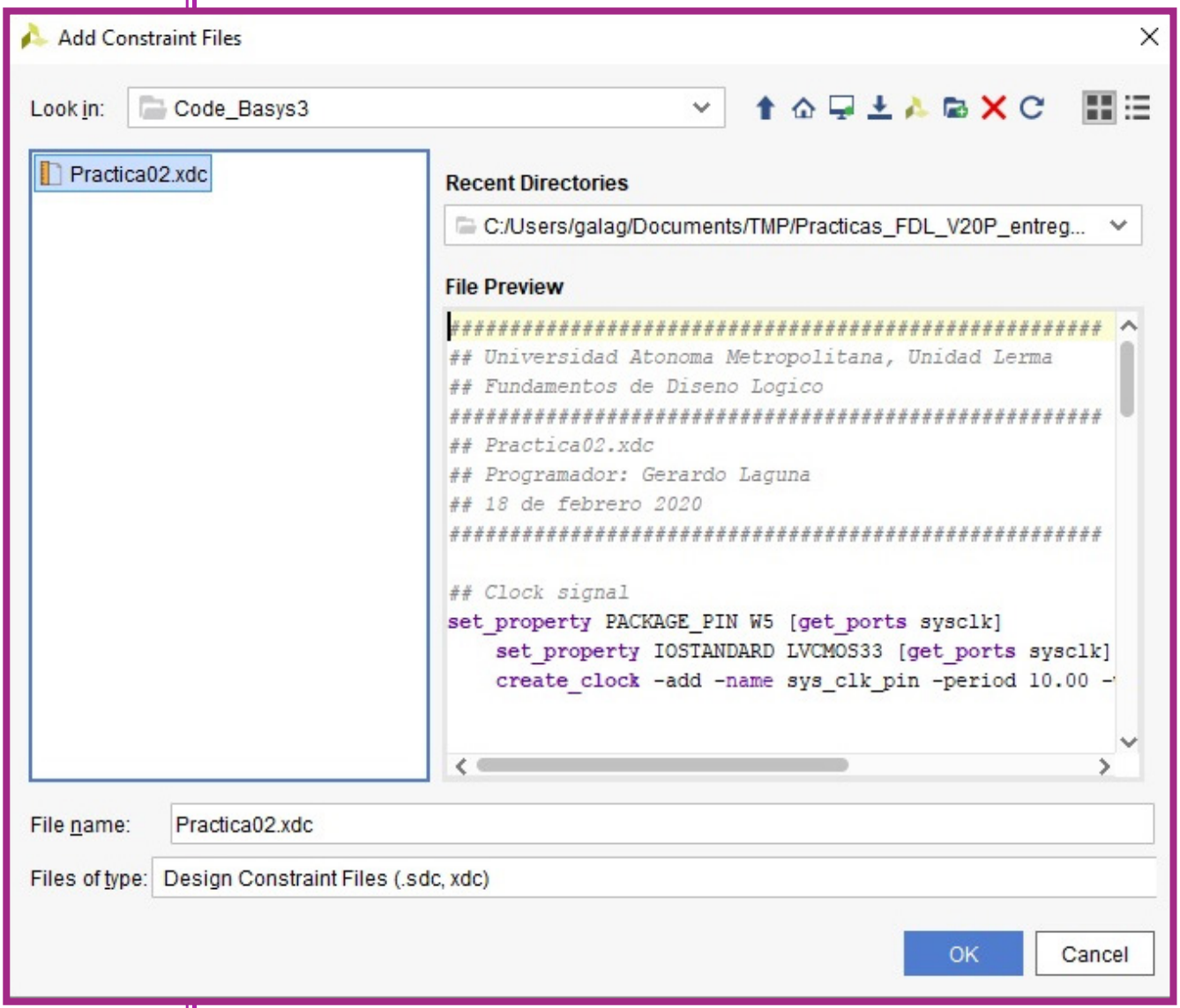


Figura A2.12. Ventana para indicar que el archivo específico de restricciones que ha sisdo seleccionado y deberá ser copiado en la carpeta del proyecto.

Una vez agregados al proyecto, se puede revisar cada código dando doble clic en su nombre listado en la vista de jerarquía ("Hierarchy") de la ventana "Sources" (Figura A2.13).

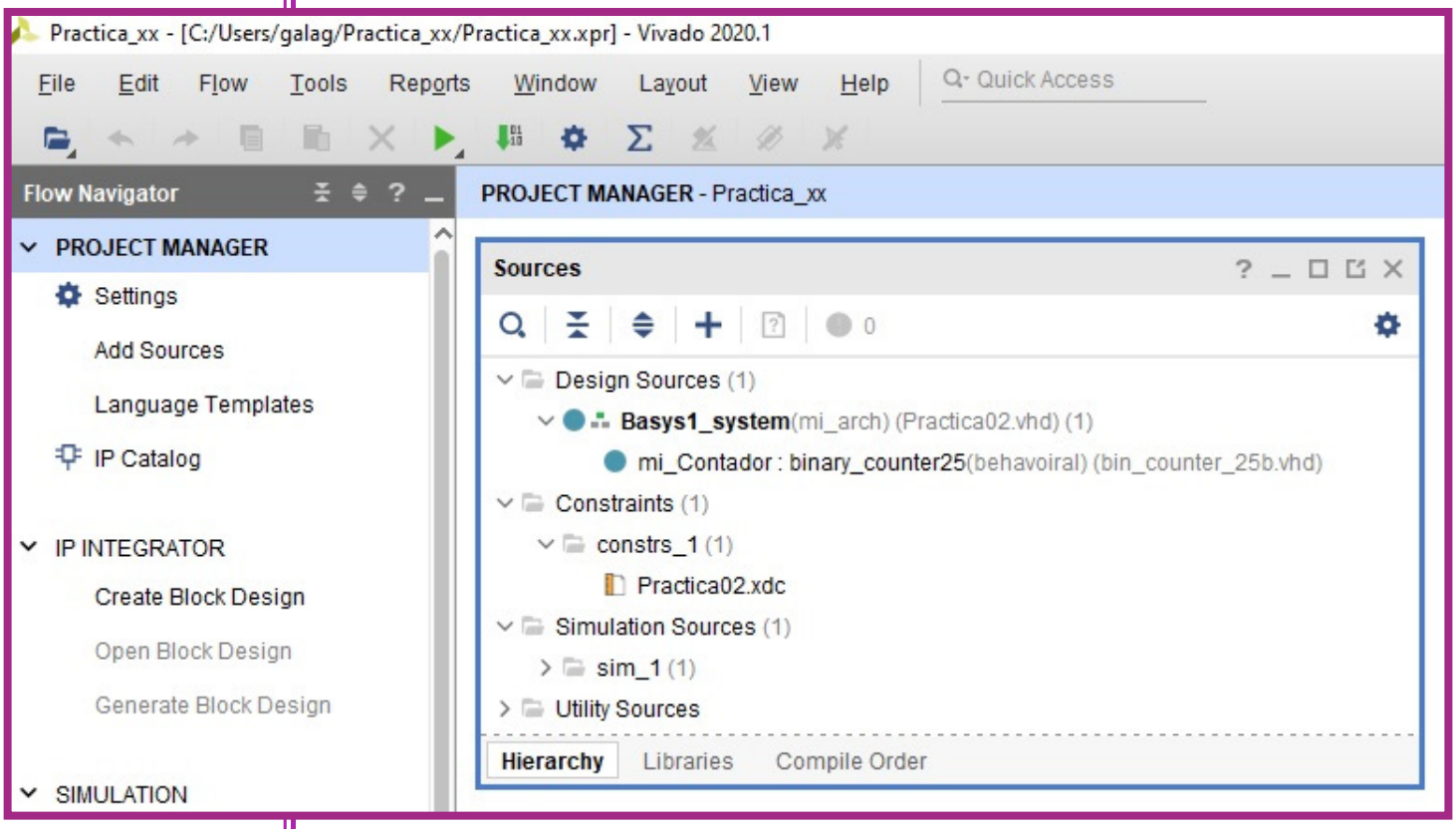
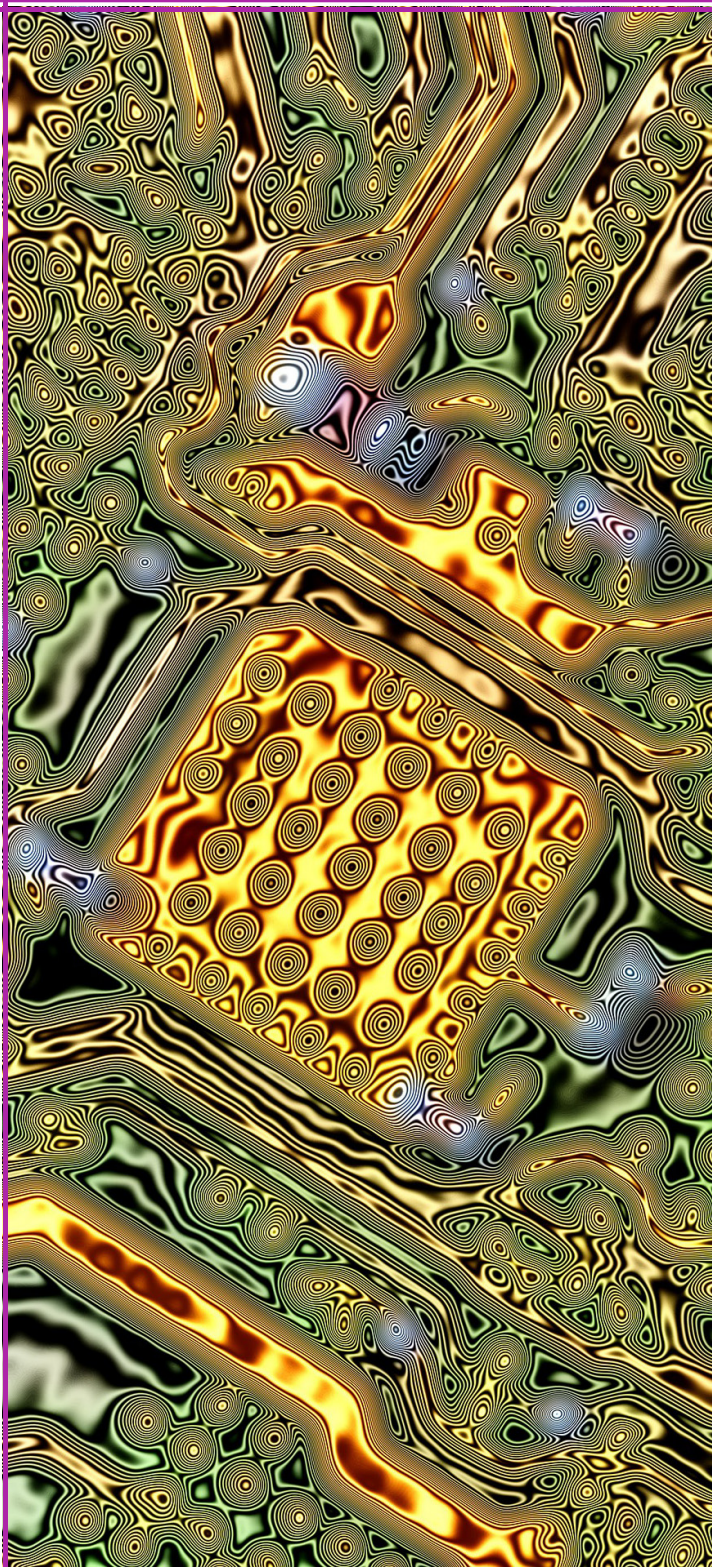


Figura A2.13. Vista de jerarquía del proyecto recién creado. Note que los archivos están organizados en: archivos de diseño (*.vhd), archivos de restricciones (*.xdc) y archivos de simulación (también con extensión vhd).

ANEXO 3

Generación del archivo con los bits de programación



Dirección
Ciencias Básicas
e Ingeniería

Generación del archivo con los bits de programación

Anexo 3

En la ventana del margen izquierdo, **"***Flow Navigator***"**, ir a la sección **"***Program and Debug***"** y oprimir el icono "Generate Bitstream". Si aún no se ha sintetizado ni implementado el proyecto, aparece el siguiente mensaje (Figura A3.1):

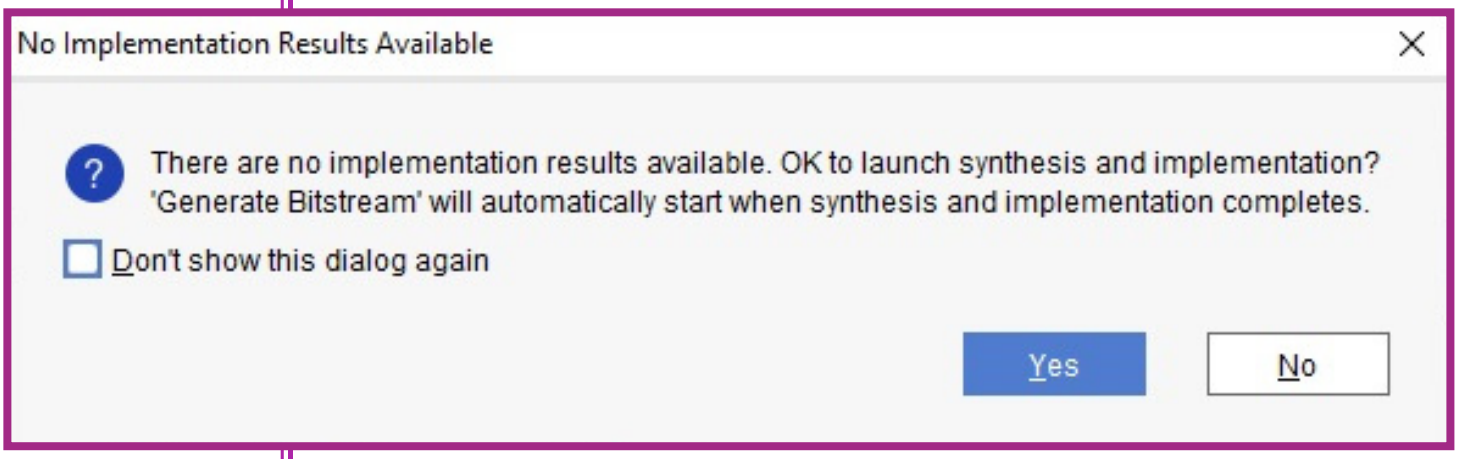


Figura A3.1. Mensaje que indica que él proyecto no se ha sintetizado ni implementado y, por lo tanto, solicita confirmación para dar paso a la síntesis e implementación que concluye con la generación de la cadena de bits de programación.

Oprimir el botón <*Yes*>**.** A continuación, puede aparecer otra ventana intitulada como **"***Launch Runs***", si es el caso, marque la casilla "<*> *Don't show this dialog again*" y** oprima el botón <*OK*> para iniciar la secuencia del flujo de construcción del proyecto hasta llegar a la generación de la secuencia de bits a programar en el dispositivo FPGA (Figura A3.2).

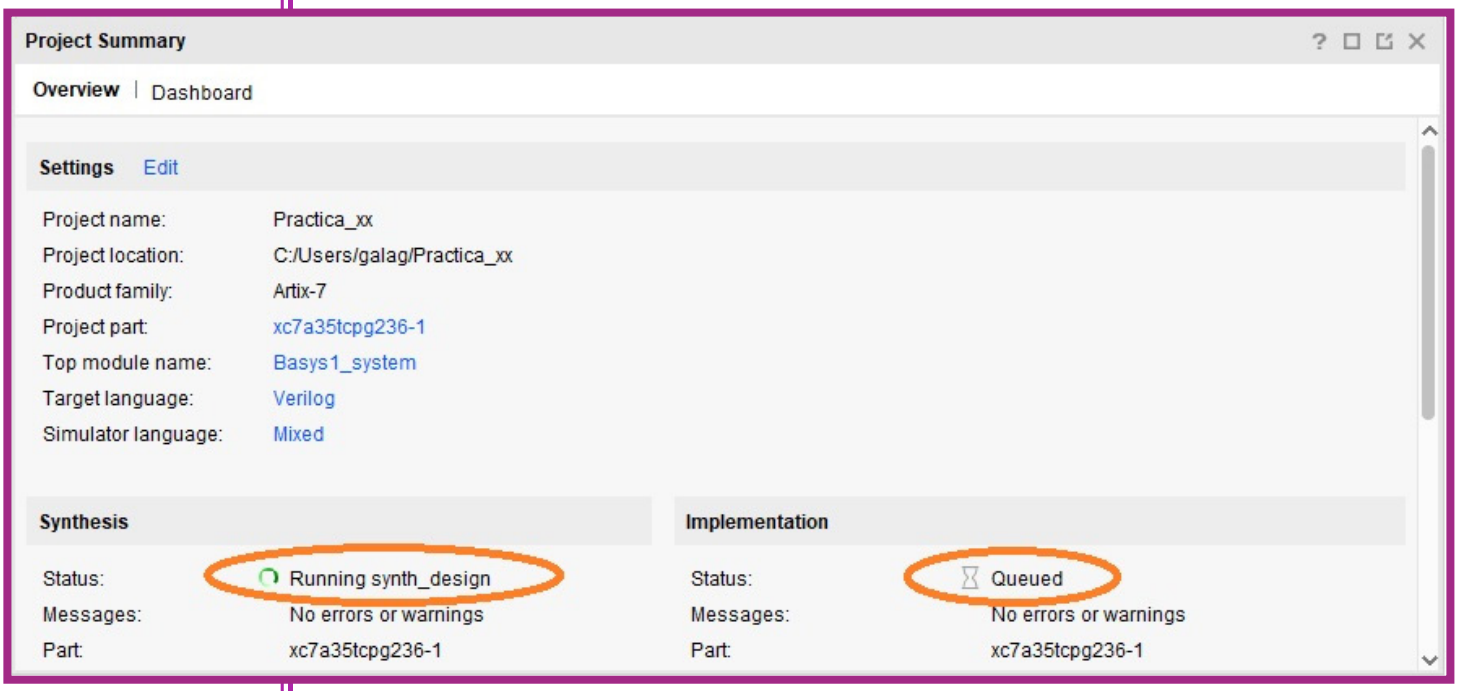


Figura A3.2. Vistas de avance de las fases de Síntesis e Implementación

Si ya no hay errores, al terminar la secuencia aparece el siguiente mensaje (Figura A3.3):

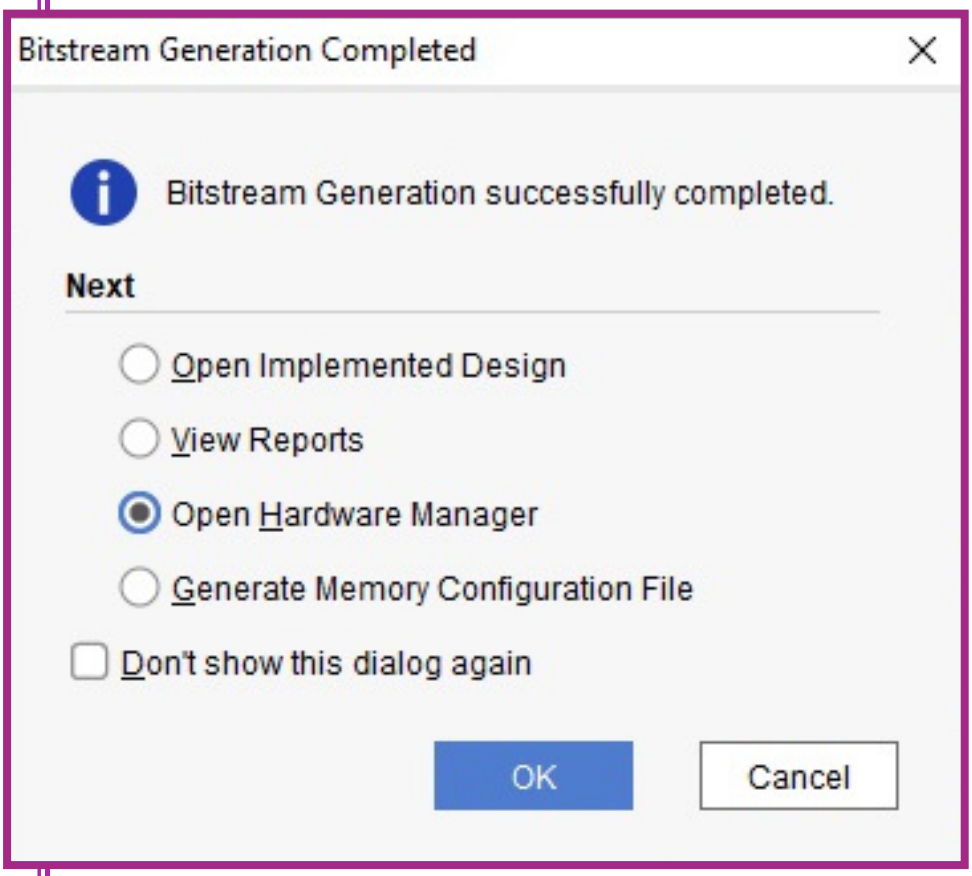
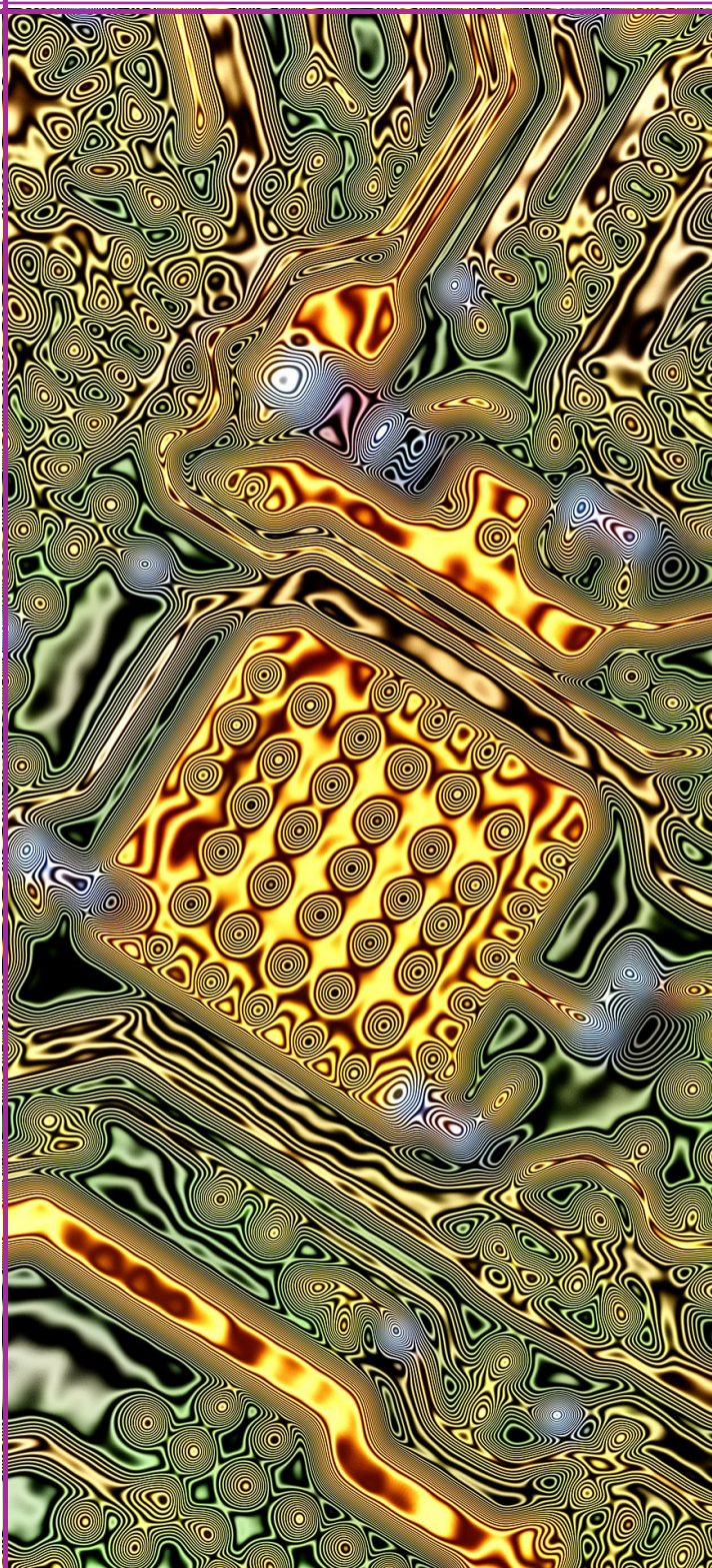


Figura A3.3. Mensaje para indicar que la cadena de bits de programación se ha completado correctamente y elegir la siguiente acción.

Marcar la opción **"<*> *Open Hardware Manager*"**. Oprimir el botón <*OK*> para continuar y proceder a la programación del dispositivo con ayuda del **"*Hardware Manager*"**. Antes de continuar, asegúrese que la tarjeta está conectada y encendida (Ver Anexo 4).

ANEXO 4

Encendido, apagado y programación de la tarjeta Basys 3



Dirección
Ciencias Básicas
e Ingeniería

Encendido, apagado y programación de la tarjeta Basys 3

Anexo 4

Antes de encender la tarjeta Basys 3

Asegúrese de que la tarjeta se encuentra en modo de operación de carga automática de programa desde la memoria Flash a través de SPI (en las terminales JP1 [MODE], debe haber un puente entre los postes 1 y 2). Antes de conectar el cable USB a la computadora, asegúrese de que el interruptor de alimentación (POWER SWITCH) esté pagado (posición OFF). Una vez conectado el cable USB, puede encender la tarjeta (POWER SWITCH en ON) y esperar a que la detecte el sistema.

Inmediatamente después de encender la tarjeta, puede comprobar que funciona verificando que se ejecuta exitosamente la rutina de prueba, cargada desde la memoria Flash, al observar en el display LED la secuencia ascendente de números decimales.

Antes de apagar la tarjeta Basys 3

Antes de desconectar el cable USB de la tarjeta Basys 3, asegúrese de que el interruptor de alimentación (POWER SWITCH) esté pagado (posición OFF). Una vez apagada la tarjeta Basys 3, puede proceder a desconectarla y guardarla.

En la parte superior de la ventana del administrador de hardware aparece una barra con el mensaje “No hardware target is open.” Oprimir el vínculo Open target.

Anexo 4

Programación de la tarjeta Basys 3

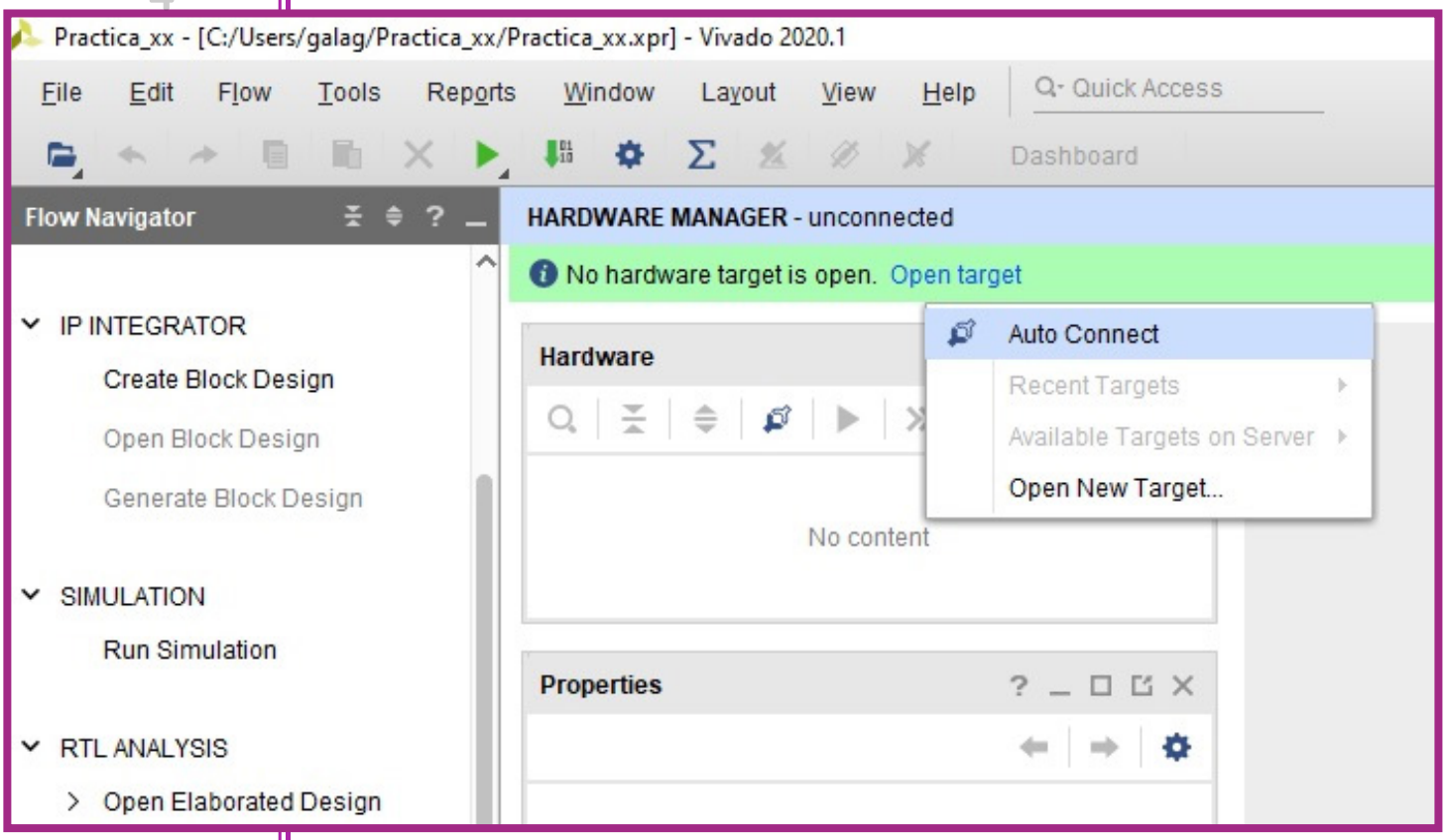


Figura A4.1. Vista del "Hardware Manager". Si la tarjeta está correctamente conectada a la computadora, al seleccionar las opciones "Open target" >> "Auto Connect", la tarjeta debe ser reconocida por Vivado.

En el menú de opciones, elegir "Auto Connect" (Figura A4.1). Ahora la barra superior de la ventana "Hardware Manager" se actualiza con el mensaje "There are no debug cores" (Figura A4.2). Oprimir el vínculo Program Device. Entonces, aparece la ventana para buscar el archivo con la secuencia de bits a programar.

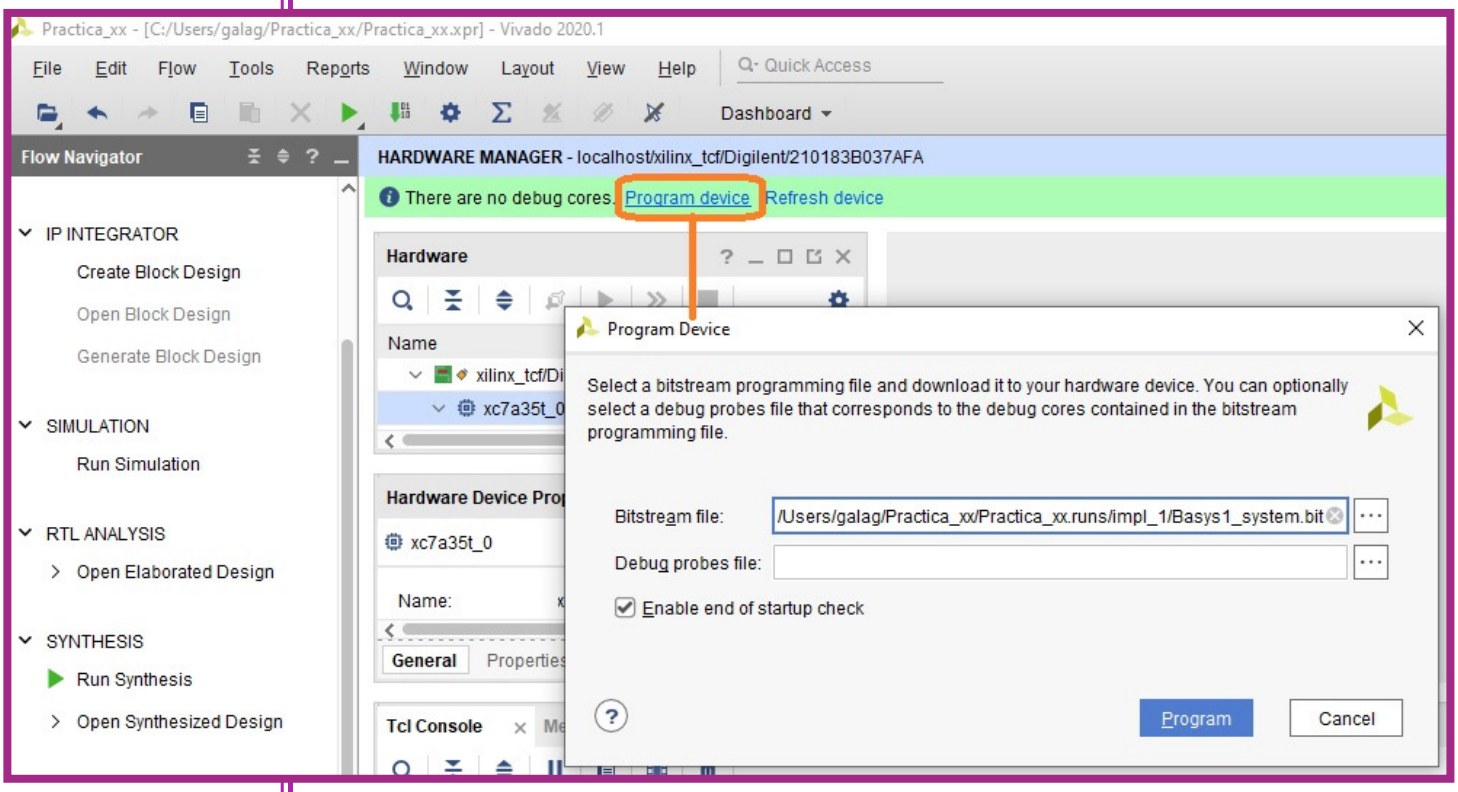


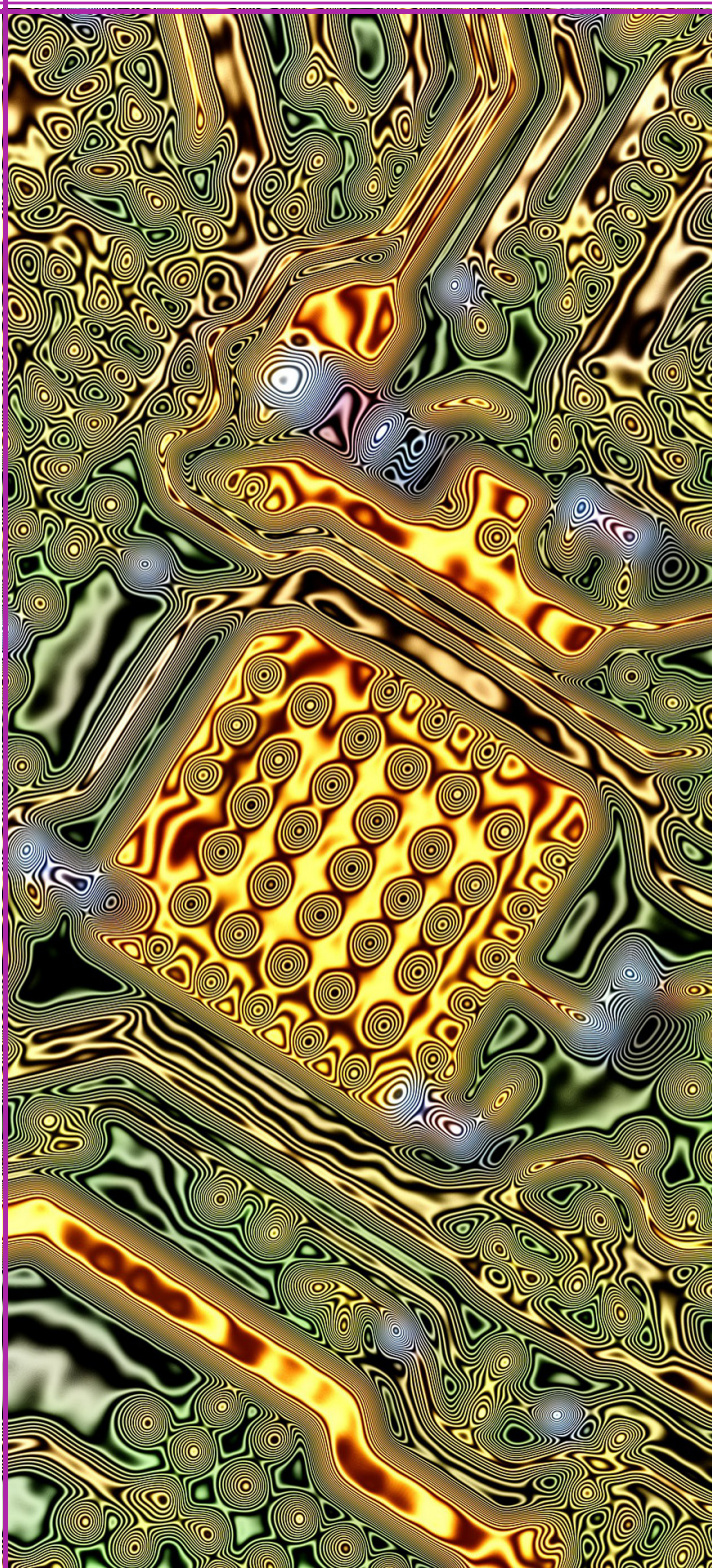
Figura A4.2. Venta para seleccionar el archivo con la secuencia de bits a programar (*.bit).

Por defecto, aparece el archivo de bits generado en el flujo de construcción del proyecto abierto. Dejar las opciones por defecto y oprimir el botón <*Program*>.

Si la operación se concluyó con éxito, el dispositivo se comportará de acuerdo con lo programado. El programa se mantendrá operando mientras la tarjeta no se apague. Una vez que se apaga la tarjeta, el dispositivo será recargado con el programa especificado por el modo indicado por el puente JP1.

ANEXO 5

Código para la Práctica 2 bin_counter.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 5

Código para la Práctica 2
bin_counter.vhd

```
---==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- bin_counter_25b.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--==========================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity binary_counter25 is
  port(
    clk, reset: in std_logic;
    q: out std_logic_vector(24 downto 0)
  );
end binary_counter25;

architecture behavoiral of binary_counter25 is
  signal r_reg: unsigned(24 downto 0);
  signal r_next: unsigned(24 downto 0);
begin
  -- register
  process(clk,reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;
  -- next-state logic
  r_next <= r_reg + 1;
  -- output logic
  q <= std_logic_vector(r_reg);
end behavoiral;
```

Practica02.vhd

```
--==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- Practica02.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--==========================================================
------------------------------------------------------------------------------------------------------------
-- Library declarations
------------------------------------------------------------------------------------------------------------
library ieee;
  use ieee.std_logic_1164.all;

------------------------------------------------------------
-- Entity declaration
------------------------------------------------------------
entity Basys1_system is
port (

  --Basys3 Resources
  mi_reset    : in    std_logic; -- BTNC
  sysclk      : in    std_logic;
  led         : out   std_logic_vector(7 downto 0);
  seg         : out   std_logic_vector(6 downto 0);
  an          : out   std_logic_vector(3 downto 0)
);
end Basys1_system;

architecture mi_arch of Basys1_system is

------------------------------------------------------------------------------------------------------------
-- Components declaration
------------------------------------------------------------------------------------------------------------
component binary_counter25
   port(
      clk, reset:   in std_logic;
      q:            out std_logic_vector(24 downto 0)
   );
end component;
```

```
-----------------------------------------------------------------------------------------------------
-- Signal declaration
-----------------------------------------------------------------------------------------------------
signal usrclk                   : std_logic_vector(24 downto 0);
signal caseCounter              : std_logic_vector(2 downto 0);
signal LedReg                   : std_logic_vector(7 downto 0);
signal SegReg                   : std_logic_vector(6 downto 0);
-----------------------------------------------------------
-- Begin
-----------------------------------------------------------
begin

caseCounter(0)<=usrclk(22);
caseCounter(1)<=usrclk(23);
caseCounter(2)<=usrclk(24);

mi_Contador : binary_counter25
  port map (
    clk => sysclk,
    reset => mi_reset,
    q => usrclk
  );

process(caseCounter)
begin
  case caseCounter is
    when "000" =>
          LedReg <= "11111110";
          SegReg <= "1111110";

    when "001" =>
          LedReg <= "11111101";
          SegReg <= "1111101";

    when "010" =>
          LedReg <= "11111011";
          SegReg <= "1111011";

    when "011" =>
          LedReg <= "11110111";
          SegReg <= "1110111";
```

```
    when "100" =>
        LedReg <= "11101111";
        SegReg <= "1101111";

    when "101" =>
        LedReg <= "11011111";
        SegReg <= "1011111";

    when "110" =>
        LedReg <= "10111111";
        SegReg <= "0111111";

    when "111" =>
        LedReg <= "01111111";
        SegReg <= "1111111";

  end case;
end process;

  led <= LedReg;
  an <= "0000";
  seg <= SegReg;

end mi_arch;
```

Practica02.xdc

Anexo 4

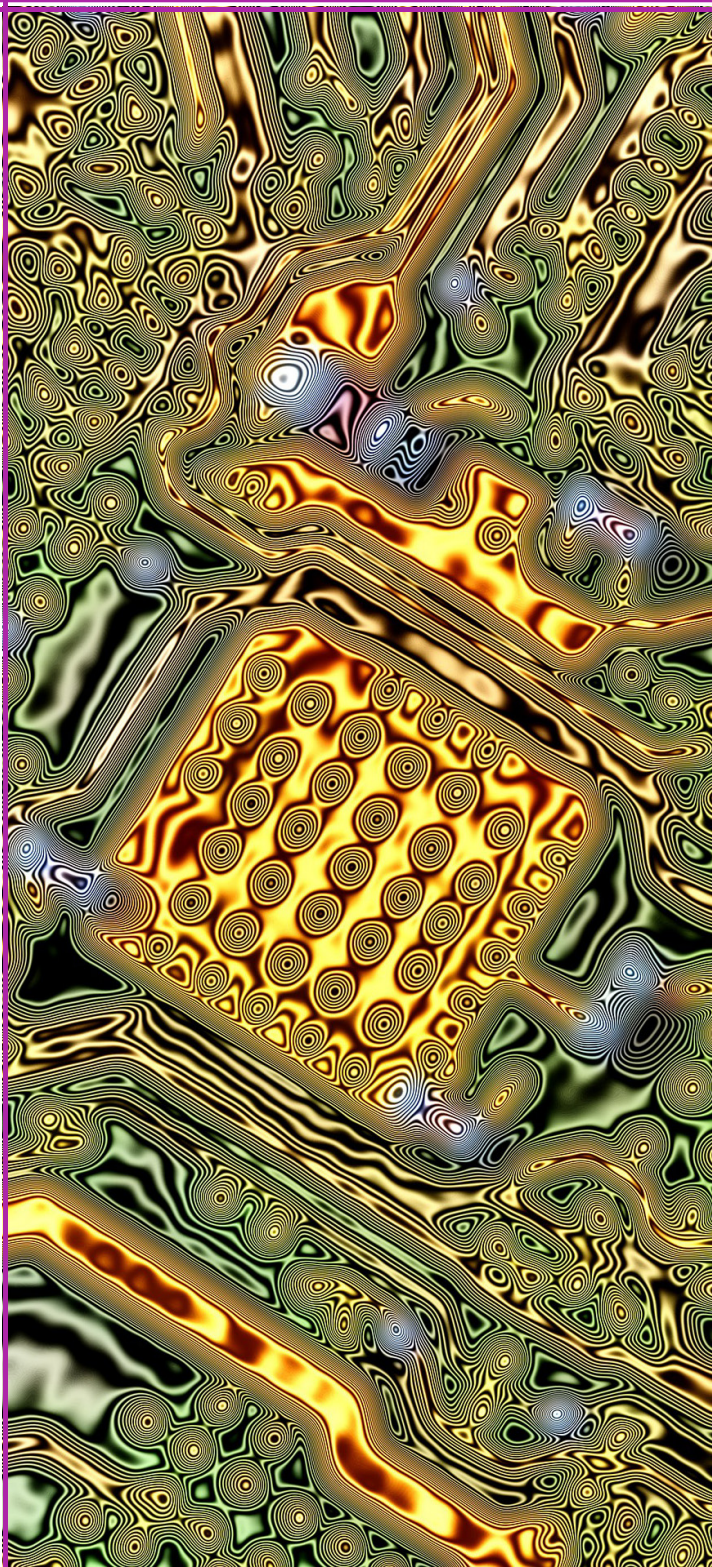
```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica02.xdc
## Programador: Gerardo Laguna
## 18 de febrero 2020
#####################################################

## Clock signal
set_property PACKAGE_PIN W5 [get_ports sysclk]

        set_property IOSTANDARD LVCMOS33 [get_ports sysclk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_
ports sysclk]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
```

Anexo 4

```
##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports mi_reset]

        set_property IOSTANDARD LVCMOS33 [get_ports mi_reset]
```

ANEXO 6

Código para la Práctica 3 miXORa.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 6

Código para la Práctica 3
miXORa.vhd

```
--=========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=========================================================
-- miXORa.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--=========================================================
-----------------------------------------------------------
-- Library declarations
-----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------------------
-- Entity declaration
-----------------------------------------------------------
entity my_xor is
  port(
    x0, x1: in std_logic;
    y: out std_logic
  );
end my_xor;

architecture structural_xor_arch of my_xor is
-----------------------------------------------------------
-- Components declaration
-----------------------------------------------------------
-----------------------------------------------------------
-- Signal declaration
-----------------------------------------------------------
  signal p0, p1: std_logic;
-----------------------------------------------------------
-- Architecture body
-----------------------------------------------------------
begin
  -- suma de productos
  y <= p0 or p1;
  -- productos
  p0 <= (not x0) and x1;
  p1 <= x0 and (not x1);
end structural_xor_arch;
```

Anexo 6

miXORb.vhd

```
--=========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=========================================================
-- miXORb.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--=========================================================
-----------------------------------------------------------
-- Library declarations
-----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------------------
-- Entity declaration
-----------------------------------------------------------
entity my_xorb is
  port(
    x: in  std_logic_vector (1 downto 0);
    y: out std_logic
  );
end my_xorb;

architecture behavioral_xor_arch of my_xorb is
-----------------------------------------------------------
-- Components declaration
-----------------------------------------------------------
-----------------------------------------------------------
-- Signal declaration
-----------------------------------------------------------
-----------------------------------------------------------
-- Architecture body
-----------------------------------------------------------
begin
  with x select
  y<= '0' when "00",   --0
     '1' when "01",   --1
     '1' when "10",   --2
     '0' when "11";   --3

end behavioral_xor_arch;
```

Anexo 6

Practica03a.vhd

```
--=============================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=============================================================
-- Practica03a.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--=============================================================
-------------------------------------------------------------
-- Library declarations
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------
entity Basys_system is
  port(
   --Basys Resources
    sw0, sw1: in std_logic;
    led0: out std_logic
  );
end Basys_system;

architecture my_system_arch of Basys_system is
-------------------------------------------------------------
-- Components declaration
-------------------------------------------------------------
component my_xor
  port(
    x0, x1: in std_logic;
    y: out std_logic
  );
end component;
```

Anexo 6

```
---------------------------------------------------------
-- Signal declaration
---------------------------------------------------------


---------------------------------------------------------
-- Architecture body
---------------------------------------------------------

begin

xor_01 : my_xor
  PORT MAP (
    x0 => sw0,
    x1 => sw1,
    y => led0
  );

end my_system_arch;
```

Anexo 6

Practica03b.vhd

```
--=================================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=================================================================
-- Practica03b.vhd
-- Programador: Gerardo Laguna
-- 18 de febrero 2020
--=================================================================
-----------------------------------------------------------
-- Library declarations
-----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------------------------
-- Entity declaration
-----------------------------------------------------------
entity Basys_system is
  port(
   --Basys Resources
     sw0, sw1: in std_logic;
     led0: out std_logic
  );
end Basys_system;

architecture my_system_arch of Basys_system is
-----------------------------------------------------------
-- Components declaration
-----------------------------------------------------------
component my_xorb
  port(
     x: in  std_logic_vector (1 downto 0);
     y: out std_logic
  );
end component;
```

Anexo 6

```
------------------------------------------------------------
-- Signal declaration
------------------------------------------------------------


------------------------------------------------------------
-- Architecture body
------------------------------------------------------------

begin

xor_01 : my_xorb
  PORT MAP (
    x(0) => sw0,
    x(1) => sw1,
    y => led0
  );

end my_system_arch;
```

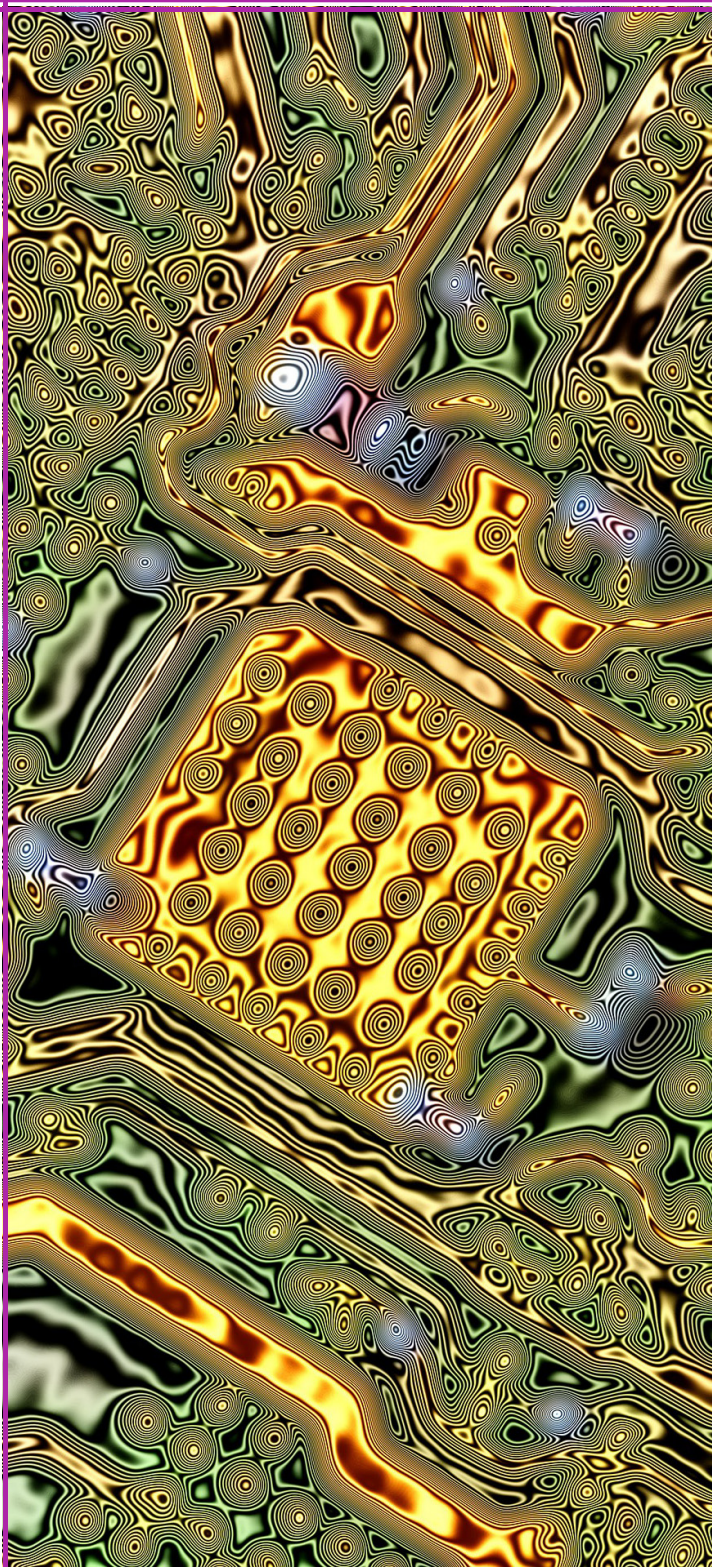
Anexo 6

Practica03a.xdc

```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica03.xdc
## Programador: Gerardo Laguna
## 18 de febrero 2020
#####################################################


## Switches
set_property PACKAGE_PIN V17 [get_ports {sw0}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw0}]
set_property PACKAGE_PIN V16 [get_ports {sw1}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw1}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led0}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led0}]
```

ANEXO 7

Código para la Práctica 5 miXOR.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 7

Código para la Práctica 5
miXOR.vhd

```
--=========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=========================================================
-- miXOR.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--=========================================================
-----------------------------------------------------------
-- Library declarations
-----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------------------
-- Entity declaration
-----------------------------------------------------------
entity my_xor is
  port(
    x0, x1: in std_logic;
    y: out std_logic
  );
end my_xor;

architecture my_xor_arch of my_xor is
-----------------------------------------------------------
-- Components declaration
-----------------------------------------------------------
-----------------------------------------------------------
-- Signal declaration
-----------------------------------------------------------
  signal p0, p1: std_logic;
-----------------------------------------------------------
-- Architecture body
-----------------------------------------------------------
begin
  -- suma de productos
  y <= p0 or p1;
  -- productos
  p0 <= (not x0) and x1;
  p1 <= x0 and (not x1);
end my_xor_arch;
```

Anexo 7

test_bench.vhd

```
--============================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--============================================================
-- test_bench.vhd
-- Programador: Gerardo Laguna
-- 30 de noviembre 2020
--============================================================
-------------------------------------------------------------------------------
-- Library declarations
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------------------------

entity practica5_testbench is
--Notese como en una entidad de prueba (testbench) no define puertos de entrada/
salida
end practica5_testbench;

architecture tb_arch of practica5_testbench is
-------------------------------------------------------------------------------------------------
-- Components declaration
-------------------------------------------------------------------------------------------------
-----
--Notese como en una entidad de prueba (testbench) no define explicitamente el
componente a probar

-------------------------------------------------------------------------------------------------
-- Signal declaration
-------------------------------------------------------------------------------------------------
  signal test_in: std_logic_vector(1 downto 0);
  signal test_out: std_logic;
```

```
-------------------------------------------------------------------------------------------
-- Architecture body
-------------------------------------------------------------------------------------------

begin
  -- Instancia del circuito bajo prueba:
  -- Notese el uso de la referencia al espacio de trabajo actual (library) como “work.” y
  -- la especificacion de la arquitectura del componente (my_xor_arch) como para-
metro

  simulated_xor: entity work.my_xor(my_xor_arch)
    port map (
    x0=>test_in(0),
    x1=>test_in(1),
    y=>test_out
    );

  -- Proceso para generador de vectores de prueba:
  process
  begin
    -- test vector 1
    test_in <= “00”;
    wait for 200 ns;
    -- test vector 2
    test_in <= “01”;
    wait for 200 ns;
    -- test vector 3
    test_in <= “10”;
    wait for 200 ns;
    -- test vector 4
    test_in <= “11”;
    wait for 200 ns;
    -- termina la simulacion
    assert false
      report “Simulacion completa”
     severity failure;
  end process;
end tb_arch;
```

Anexo 7

Practica05.vhd

```
--=============================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=============================================================
-- Practica05.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--=============================================================
-------------------------------------------------------------
-- Library declarations
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------
entity Basys_system is
  port(
   --Basys Resources
     sw0, sw1: in std_logic;
     led0: out std_logic
  );
end Basys_system;

architecture my_system_arch of Basys_system is
-------------------------------------------------------------
-- Components declaration
-------------------------------------------------------------
component my_xor
  port(
     x0, x1: in std_logic;
     y: out std_logic
  );
end component;
```

Anexo 7

```
---------------------------------------------------------
-- Signal declaration
---------------------------------------------------------


---------------------------------------------------------
-- Architecture body
---------------------------------------------------------

begin

xor_01 : my_xor
  PORT MAP (
    x0 => sw0,
    x1 => sw1,
    y => led0
  );

end my_system_arch;
```

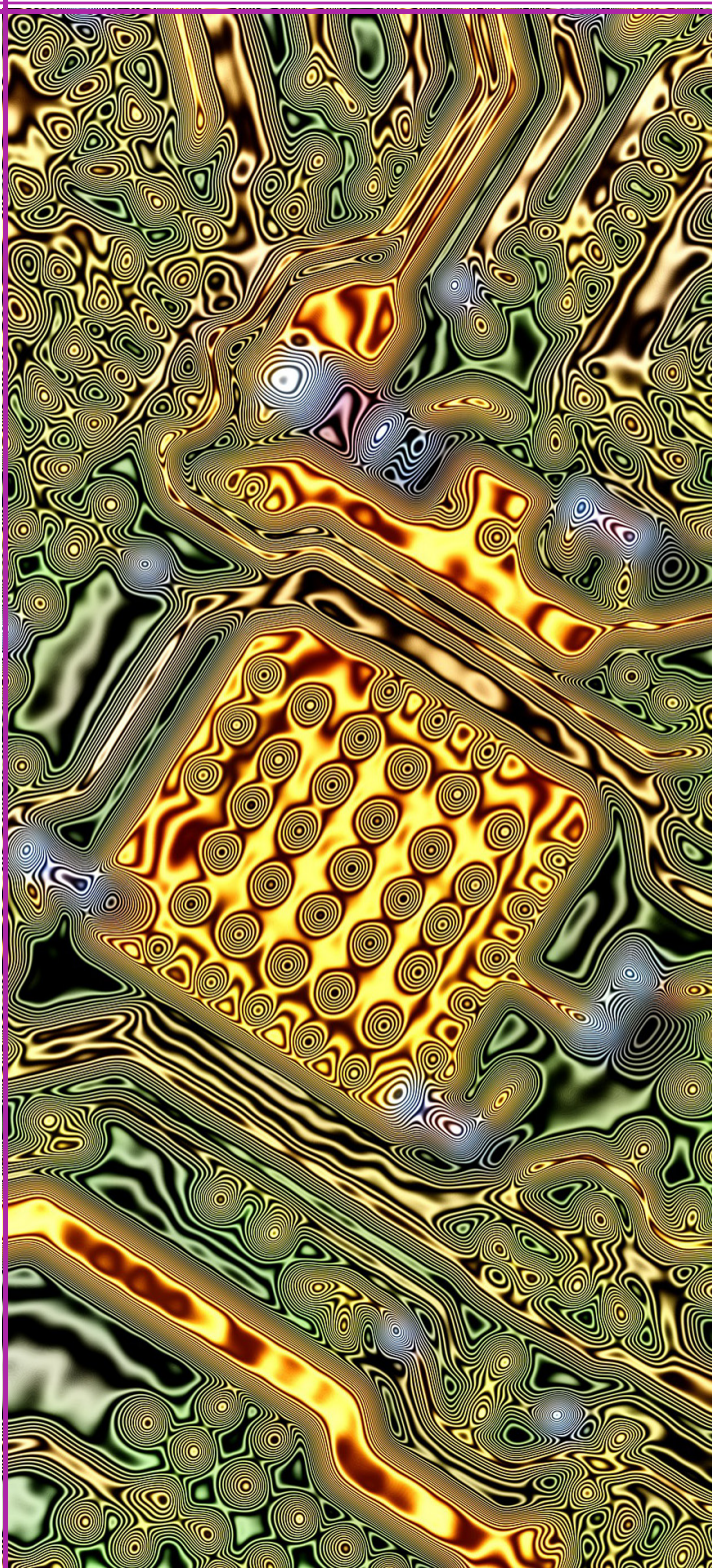
Anexo 7

Practica05.xdc

```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica05.xdc
## Programador: Gerardo Laguna
## 19 de febrero 2020
#####################################################

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw0}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw0}]
set_property PACKAGE_PIN V16 [get_ports {sw1}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw1}]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {led0}]

        set_property IOSTANDARD LVCMOS33 [get_ports {led0}]
```

ANEXO 8

Código para la Práctica 6 structural_hex2led.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 8

Código para la Práctica 6
structural_hex2led.vhd

```
--==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- structural_hex2led.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--==========================================================
------------------------------------------------------------
-- Library declarations
------------------------------------------------------------

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

------------------------------------------------------------
-- Entity declaration
------------------------------------------------------------

entity hex2led is
  Port (
    i_w, i_x, i_y, i_z : in std_logic; --MSB: i_z, LSB: i_w
    o_a, o_b, o_c, o_d, o_e, o_f, o_g : out std_logic
  );
end hex2led;

architecture Structural of hex2led is
------------------------------------------------------------
-- Components declaration
------------------------------------------------------------
```

```
------------------------------------------------------------
-- Signal declaration
------------------------------------------------------------

signal   w, x, y, z : std_logic; -- Li¬neas para senales de entrada
signal   nw, nx, ny, nz : std_logic; -- Li¬neas para senales de entrada negadas
signal   nznynxnw, nznynxw, nznyxnw, nznyxw, nzynxnw, nzynxw, nzyxnw, nzyxw,
         znynxnw, znynxw, znyxnw, znyxw, zynxnw, zynxw, zyxnw, zyxw  : std_logic; --
Li¬neas para senales con “productos”

------------------------------------------------------------
-- Architecture body
------------------------------------------------------------

begin

--
-- segment encoding
--      o_a
--     ---
--o_f |   | o_b
--     ---   <- o_g
--o_e |   | o_c
--     ---
--      o_d

  w <= i_w;
  x <= i_x;
  y <= i_y;
  z <= i_z;
  nw <= not i_w;
  nx <= not i_x;
  ny <= not i_y;
  nz <= not i_z;

  nznynxnw <= nz and ny and nx and nw;
  nznynxw <= nz and ny and nx and w;
  nznyxnw <= nz and ny and x and nw;
  nznyxw <= nz and ny and x and w;
  nzynxnw <= nz and y and nx and nw;
```

```
    nzynxw <= nz and y and nx and w;
    nzyxnw <= nz and y and x and nw;
    nzyxw <= nz and y and x and w;
    znynxnw <= z and ny and nx and nw;
    znynxw <= z and ny and nx and w;
    znyxnw <= z and ny and x and nw;
    znyxw <= z and ny and x and w;
    zynxnw <= z and y and nx and nw;
    zynxw <= z and y and nx and w;
    zyxnw <= z and y and x and nw;
    zyxw <= z and y and x and w;

    o_g <= nznynxnw or nznynxw or nzyxw or zynxnw;
    o_f <= nznynxw or nznyxnw or nznyxw or nzyxw or zynxw;
    o_e <= nznynxw or nznyxw or nzynxnw or nzynxw or nzyxw or znynxw;
    o_d <= nznynxw or nzynxnw or nzyxw or znyxnw or zyxw;
    o_c <= nznyxnw or zynxnw or zyxnw or zyxw;
    o_b <= nzynxw or nzyxnw or znyxw or zynxnw or zyxnw or zyxw;
    o_a <= nznynxw or nzynxnw or znyxw or zynxw;

end Structural;
```

Anexo 8

Practica06.vhd

```
--==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- Practica06.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--==========================================================
------------------------------------------------------------
-- Library declarations
------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

------------------------------------------------------------
-- Entity declaration
------------------------------------------------------------

entity Basys_system is
   Port (
      --Basys Resources
          sw : in  STD_LOGIC_VECTOR (3 downto 0);
      disp_seg : out  STD_LOGIC_VECTOR (7 downto 0);
      disp_sel : out  STD_LOGIC_VECTOR (3 downto 0)
   );
end Basys_system;

architecture my_system_arch of Basys_system is
------------------------------------------------------------
-- Components declaration
------------------------------------------------------------

component hex2led
   Port (
      i_w, i_x, i_y, i_z : in std_logic; --MSB: i_z, LSB: i_w
      o_a, o_b, o_c, o_d, o_e, o_f, o_g : out std_logic
   );
end component;
```

Anexo 8

```
------------------------------------------------------------
-- Signal declaration
------------------------------------------------------------

------------------------------------------------------------
-- Architecture body
------------------------------------------------------------

begin

-- Convierte valor binario en sw a display LED de 7 segmentos

disp_sel <= “0111”;

Displed7seg : hex2led
        port map (
                i_w => sw(0),
                i_x => sw(1),
                i_y => sw(2),
                i_z => sw(3),

                o_a => disp_seg(0),
                o_b => disp_seg(1),
                o_c => disp_seg(2),
                o_d => disp_seg(3),
                o_e => disp_seg(4),
                o_f => disp_seg(5),
                o_g => disp_seg(6)
        );

disp_seg(7) <= ‘1’;

end my_system_arch;
```

Anexo 8

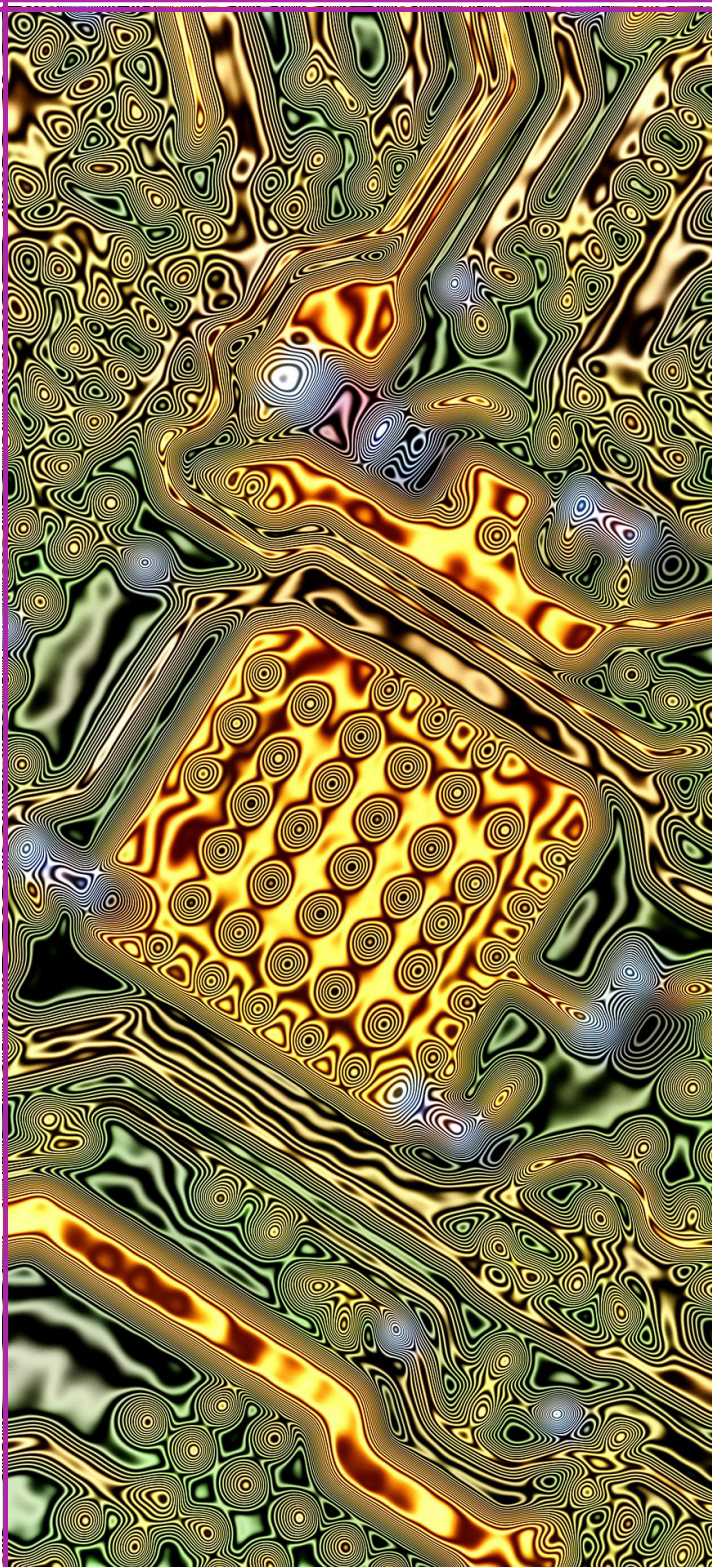
Practica06.xcd

```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica06.xdc
## Programador: Gerardo Laguna
## 19 de febrero 2020
#####################################################

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]



##7 disp_segment display
set_property PACKAGE_PIN W7 [get_ports {disp_seg[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {disp_seg[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {disp_seg[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {disp_seg[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {disp_seg[4]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {disp_seg[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {disp_seg[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[6]}]

set_property PACKAGE_PIN V7 [get_ports {disp_seg[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[7]}]

set_property PACKAGE_PIN U2 [get_ports {disp_sel[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[0]}]
set_property PACKAGE_PIN U4 [get_ports {disp_sel[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[1]}]
set_property PACKAGE_PIN V4 [get_ports {disp_sel[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[2]}]
set_property PACKAGE_PIN W4 [get_ports {disp_sel[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[3]}]
```

ANEXO 9

Código para la Práctica 7 behavioral_hex2led.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 9

Código para la Práctica 7
behavioral_hex2led.vhd

```
--==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- behavioral_hex2led.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--==========================================================
----------------------------------------------------------
-- Library declarations
----------------------------------------------------------

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

----------------------------------------------------------
-- Entity declaration
----------------------------------------------------------

entity hex2led is
   Port (
         HEX : in  STD_LOGIC_VECTOR (3 downto 0);
                        LED : out STD_LOGIC_VECTOR (6 downto 0 )
        );
end hex2led;

architecture Behavioral of hex2led is
----------------------------------------------------------
-- Components declaration
----------------------------------------------------------


----------------------------------------------------------
-- Signal declaration
----------------------------------------------------------
```

Anexo 9

```
----------------------------------------------------------
-- Architecture body
----------------------------------------------------------

begin

--
-- segment encoding
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---
--      3

   with HEX SELect
  LED<= “1111001” when “0001”,   --1
        “0100100” when “0010”,   --2
        “0110000” when “0011”,   --3
        “0011001” when “0100”,   --4
        “0010010” when “0101”,   --5
        “0000010” when “0110”,   --6
        “1111000” when “0111”,   --7
        “0000000” when “1000”,   --8
        “0010000” when “1001”,   --9
        “0001000” when “1010”,   --A
        “0000011” when “1011”,   --b
        “1000110” when “1100”,   --C
        “0100001” when “1101”,   --d
        “0000110” when “1110”,   --E
        “0001110” when “1111”,   --F
        “1000000” when others;   --0

end Behavioral;
```

Anexo 9

Practica07.vhd

```
--==========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--==========================================================
-- Practica07.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--==========================================================
----------------------------------------------------------
-- Library declarations
----------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

----------------------------------------------------------
-- Entity declaration
----------------------------------------------------------

entity Basys_system is
   Port (
         --Basys Resources
                  sw : in  STD_LOGIC_VECTOR (3 downto 0);
         disp_seg : out  STD_LOGIC_VECTOR (7 downto 0);
         disp_sel : out  STD_LOGIC_VECTOR (3 downto 0)
   );
end Basys_system;

architecture my_system_arch of Basys_system is
----------------------------------------------------------
-- Components declaration
----------------------------------------------------------

component hex2led
   Port (
                  HEX : in  STD_LOGIC_VECTOR (3 downto 0);
                        LED :out STD_LOGIC_VECTOR (6 downto 0 )
         );
end component;
```

Anexo 9

```
------------------------------------------------------------
-- Signal declaration
------------------------------------------------------------

signal data_nible : std_logic_vector (3 downto 0); -- Nible de entrada binaria
signal disp_driver : std_logic_vector (6 downto 0); -- Activación para los segmentos LED

------------------------------------------------------------
-- Architecture body
------------------------------------------------------------

begin

-- Convierte valor binario en sw a display LED de 7 segmentos

disp_sel <= “0111”;

data_nible(0) <= sw(0);
data_nible(1) <= sw(1);
data_nible(2) <= sw(2);
data_nible(3) <= sw(3);

Displed7seg : hex2led
   port map (
      hex => data_nible,
      Led => disp_driver
   );

disp_seg(0) <= disp_driver(0);
disp_seg(1) <= disp_driver(1);
disp_seg(2) <= disp_driver(2);
disp_seg(3) <= disp_driver(3);
disp_seg(4) <= disp_driver(4);
disp_seg(5) <= disp_driver(5);
disp_seg(6) <= disp_driver(6);
disp_seg(7) <= ‘1’;

end my_system_arch;
```

Anexo 9

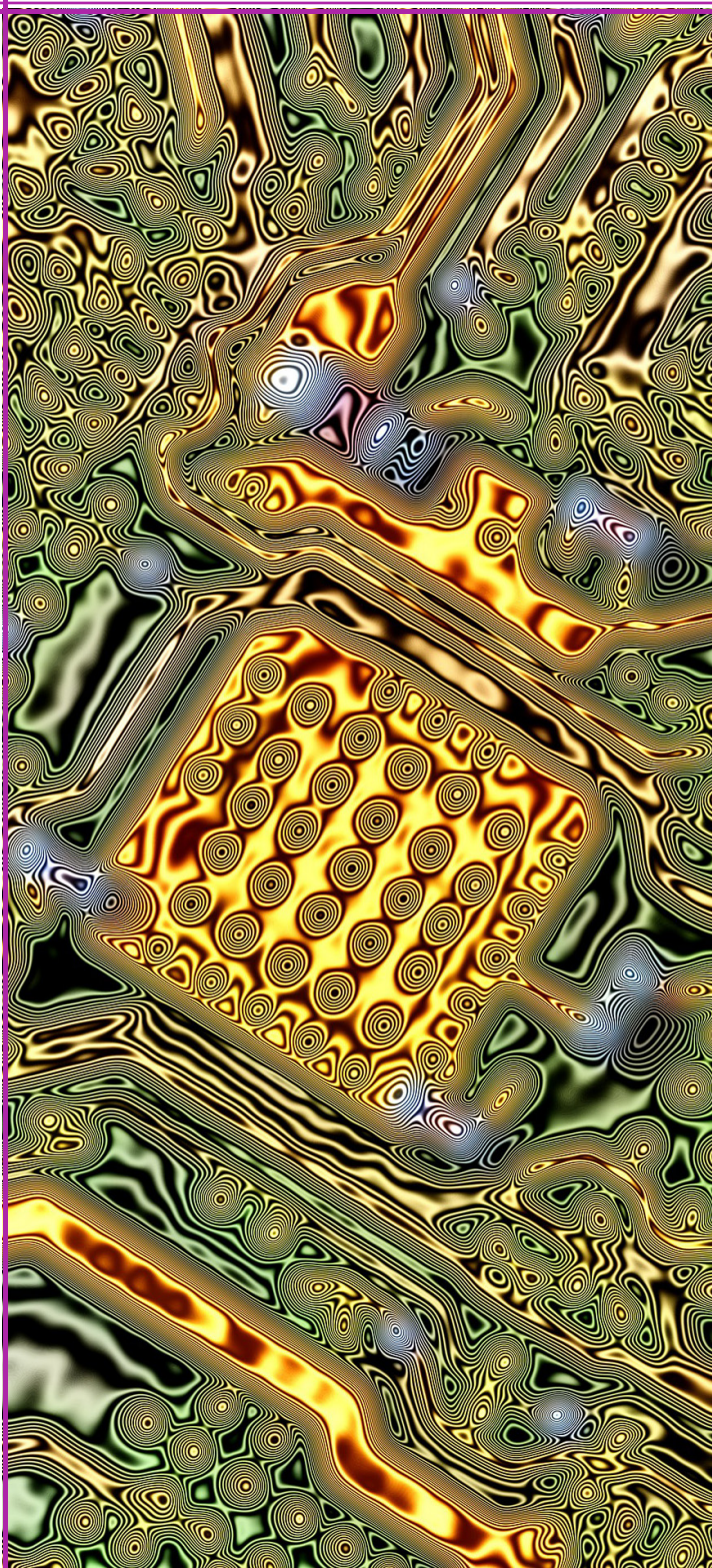
Practica07.xcd

```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica07.xdc
## Programador: Gerardo Laguna
## 19 de febrero 2020
#####################################################

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]


##7 disp_segment display
set_property PACKAGE_PIN W7 [get_ports {disp_seg[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {disp_seg[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {disp_seg[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {disp_seg[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {disp_seg[4]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {disp_seg[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {disp_seg[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[6]}]

set_property PACKAGE_PIN V7 [get_ports {disp_seg[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_seg[7]}]

set_property PACKAGE_PIN U2 [get_ports {disp_sel[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[0]}]
set_property PACKAGE_PIN U4 [get_ports {disp_sel[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[1]}]
set_property PACKAGE_PIN V4 [get_ports {disp_sel[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[2]}]
set_property PACKAGE_PIN W4 [get_ports {disp_sel[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {disp_sel[3]}]
```

ANEXO 10

Código para la Práctica 8 Register.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 10

Código para la Práctica 8
Register.vhd

```
--=========================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=========================================================
-- Register.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--=========================================================

--==============================
-- Registro de 8 bits
--==============================
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
  port(
    clk: in std_logic;
    reset: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
  );
end reg8;

architecture arch of reg8 is
begin
  process(clk,reset)
  begin
    if (reset='1') then
      q <=(others=>'0');
    elsif (clk'event and clk='1') then
      q <= d;
    end if;
  end process;
end arch;
```

Practica08.vhd

```
--=============================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--=============================================================
-- Practica08.vhd
-- Programador: Gerardo Laguna
-- 19 de febrero 2020
--=============================================================
-------------------------------------------------------------
-- Library declarations
-------------------------------------------------------------
library ieee;
  use ieee.std_logic_1164.all;

-------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------
entity Basys3_system is
port (

  --Basys3 Resources
  mi_reset     : in    std_logic; -- BTNC
  sysclk           : in    std_logic;
  gpio_led       : out   std_logic_vector(7 downto 0);
  gpio_sw     : in    std_logic_vector(7 downto 0)
);
end Basys3_system;

architecture mi_arch of Basys3_system is

-------------------------------------------------------------
-- Components declaration
-------------------------------------------------------------
COMPONENT Bin_Counter
  PORT (
    clk : IN STD_LOGIC;
    q : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END COMPONENT;
```

Anexo 10

```
COMPONENT reg8 is
  port(
    clk: in std_logic;
    reset: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
  );
END COMPONENT;

------------------------------------------------------------
-- Signal declaration
------------------------------------------------------------
signal usrclk                : std_logic_vector(23 downto 0);

------------------------------------------------------------
-- Begin
------------------------------------------------------------
begin

mi_Contador : Bin_Counter
 PORT MAP (
  clk => sysclk,
  q => usrclk
 );


mi_Register : reg8
 PORT MAP (
    clk=>usrclk(23),
    reset=>mi_reset,
    d=>gpio_sw,
    q=>gpio_led
);

end mi_arch;
```

Anexo 10

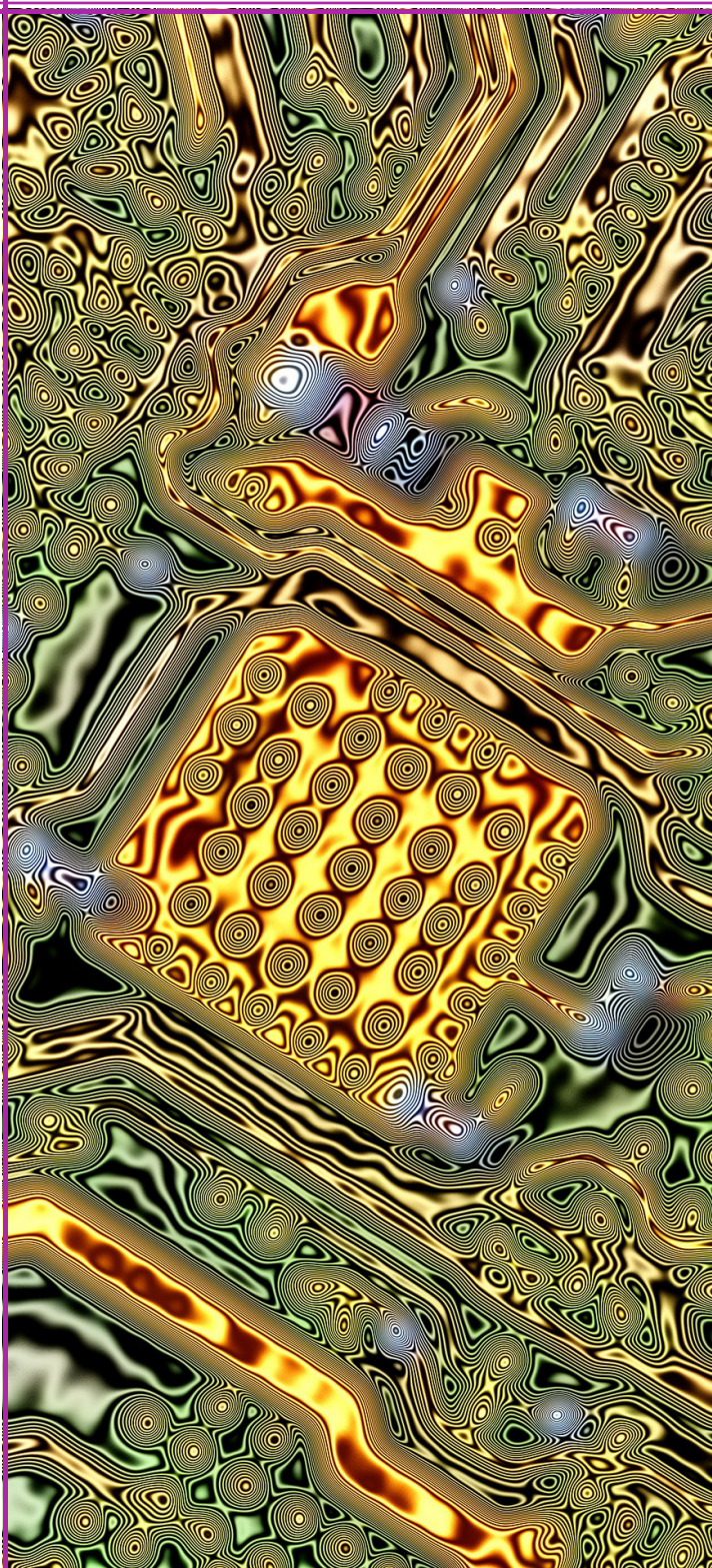
Practica08.xcd

```
#####################################################
## Universidad Autonoma Metropolitana, Unidad Lerma
## Fundamentos de Diseno Logico
#####################################################
## Practica08.xdc
## Programador: Gerardo Laguna
## 19 de febrero 2020
#####################################################

## Clock signal
set_property PACKAGE_PIN W5 [get_ports sysclk]

        set_property IOSTANDARD LVCMOS33 [get_ports sysclk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_
ports sysclk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {gpio_sw[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {gpio_sw[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {gpio_sw[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {gpio_sw[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {gpio_sw[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {gpio_sw[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {gpio_sw[6]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {gpio_sw[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_sw[7]}]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {gpio_led[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[0]}]
set_property PACKAGE_PIN E19 [get_ports {gpio_led[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[1]}]
set_property PACKAGE_PIN U19 [get_ports {gpio_led[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[2]}]
set_property PACKAGE_PIN V19 [get_ports {gpio_led[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[3]}]
set_property PACKAGE_PIN W18 [get_ports {gpio_led[4]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[4]}]
set_property PACKAGE_PIN U15 [get_ports {gpio_led[5]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[5]}]
set_property PACKAGE_PIN U14 [get_ports {gpio_led[6]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[6]}]
set_property PACKAGE_PIN V14 [get_ports {gpio_led[7]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {gpio_led[7]}]



##Buttons
set_property PACKAGE_PIN U18 [get_ports mi_reset]

        set_property IOSTANDARD LVCMOS33 [get_ports mi_reset]
```

ANEXO 11

Código para la Práctica 9 y 10 Empty_Top_Basys3_template.vhd



Dirección
Ciencias Básicas
e Ingeniería

Anexo 11

Código para la Práctica 9 y 10
Empty_Top_Basys3_template.vhd

```
--================================================================
-- Universidad Autonoma Metropolitana, Unidad Lerma
-- Fundamentos de Diseno Logico
--================================================================
-- file.vhd
-- Programador:
-- dd de mm 20aa
--================================================================
----------------------------------------------------------------
-- Library declarations
----------------------------------------------------------------
library ieee;
  use ieee.std_logic_1164.all;
----------------------------------------------------------------
-- Entity declaration
----------------------------------------------------------------
entity Basys3_system is
port (

  --Basys3 Resources
    mi_reset    : in    std_logic; -- BTNC
    sysclk      : in    std_logic;
    gpio_led    : out   std_logic_vector(7 downto 0);
    gpio_sw     : in    std_logic_vector(7 downto 0)
);
end Basys3_system;

architecture mi_arch of Basys3_system is
----------------------------------------------------------------
-- Components declaration
----------------------------------------------------------------
----------------------------------------------------------------
-- Signal declaration
----------------------------------------------------------------


----------------------------------------------------------------
-- Begin
----------------------------------------------------------------
begin

end mi_arch;
```


Casa abierta al tiempo
UNIVERSIDAD
AUTÓNOMA
METROPOLITANA
Unidad Lerma


CBI
Dirección
Ciencias Básicas
e Ingeniería