Programación de sistemas digitales con VHDL

David G. Maxinez

Primera edición ebook

México, 2014

Grupo Editorial Patria

Dedicatoria

Para ti Alexsandra Sarai,
porque siempre estás cuando más te necesito.

Para establecer comunicación con nosotros puede hacerlo por:

correo:
Renacimiento 180, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México, D.F.

fax pedidos:
(01 55) 5354 9109 • 5354 9102

e-mail:
info@editorialpatria.com.mx

home page:
www.editorialpatria.com.mx

Dirección editorial: Javier Enrique Callejas
Coordinadora editorial: Estela Delfín Ramírez
Diseño de interiores: Visión Tipográfica Editores, S.A.
Diseño de portada: Juan Bernardo Rosado Solís/Signx
Supervisor de producción: Gerardo Briones González

Revisión Técnica:
Ing. Guillermo Castillo Tapia
Universidad Autónoma Metropolitana-Azcapotzalco

Programación de sistemas digitales con VHDL

Renacimiento 180, Colonia San Juan Tlihuaca

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro Núm. 43

ISBN ebook: 978-607-438-932-6

Impreso en México
Printed in Mexico

Primera edición ebook: 2014

Contenido

Acerca del autor

David G. Maxinez realizó sus estudios de Ingeniero Mecánico Electricista en la Universidad Nacional Autónoma de México. Posteriormente obtuvo el grado de maestro en ingeniería con especialidad en electrónica dentro de la división de estudios de posgrado de la Facultad de Ingeniería de la UNAM. Dentro del Instituto Tecnológico y de Estudios Superiores de Monterrey cursó el diplomado en habilidades docentes y el diplomado en microelectrónica, realizó sus estudios de doctorado en el área de microelectrónica dentro de la Universidad Autónoma Metropolitana en convenio con la Universidad de TEXAS A&M en Estados Unidos de América. Se ha distinguido como catedrático en diversas universidades: Universidad Nacional Autónoma de México, Instituto Tecnológico y de Estudios Superiores de Monterrey, Universidad del Valle de México, Universidad Americana de Acapulco y Universidad Autónoma Metropolitana.

Dentro de su labor profesional se ha desempeñado como: Jefe de Mantenimiento y Secretario Técnico de la carrera de Ingeniería en Computación en la UNAM FES Aragón, Jefe del Departamento de Electrónica de la Facultad de Ingeniería de la UNAM, Jefe de Laboratorios del grupo Sigma Commodore, Jefe del Departamento de Innovación y Desarrollo Tecnológico de la empresa Robotics 3D. Actualmente es presidente y creador del concepto Exporobótica, Exporobots y Expociencia, eventos de divulgación de ciencia y tecnología educativa celebrados a nivel nacional en varios estados de la República Mexicana.

Es autor de las *Prácticas de Electrónica de Potencia*, editado para la UNAM FES Aragón. *Amplificación de Señales*, para el Tecnológico de Monterrey Campus Estado de México. *Herramientas en Hardware y Software para el diseño electrónico*, en el Tecnológico de Monterrey Campus Estado de México. *VHDL El arte de la programación de sistemas digitales* publicado por Grupo Editorial Patria.

Ha sido distinguido como el asesor y jefe del equipo de robótica del Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Estado de México, en la competencia de robots de manipulación médica, ganador de 1° y 2° lugar en Sao Luis, Brasil. Asesor y jefe del equipo representativo de la UAM Azcapotzalco, ganador de 1° y 3er lugar en la categoría de robot transportador en el evento de Exporobots, que se llevo a cabo en Acapulco, Guerrero. Formó parte del grupo de asesores de la Universidad Autónoma Metropolitana Unidad Azcapotzalco, que representó a México en la competencia Lunabotics Mining Competition, National Aeronautics and Space Administration, NASA 2012.

Distinguido por el Departamento de Ciencia y Tecnología del Distrito Federal en la Ciudad de México como el ganador del segundo lugar en el desarrollo de Materiales y Recursos Digitales Educativos.

Su área de investigación con trabajos arbitrados y publicados se ubica en el campo de la lógica programable, sistemas de lógica difusa y desarrollo de robots educativos.

Prólogo

Hoy en día, en nuestro ambiente familiar o de trabajo nos encontramos rodeados de sistemas electrónicos de alta sofisticación: teléfonos celulares, computadoras personales, televisores de alta definición, equipos de sonido, dispositivos de telecomunicaciones, equipos de medición, etc., son tan solo algunos ejemplos del desarrollo tecnológico que ha cambiado nuestro estilo de vida haciéndolo cada vez más confortable. Todos los sistemas anteriores tienen una misma similitud, su tamaño, de dimensiones tan pequeñas que parece increíble que sean igual o más potentes que los sistemas de mayor volumen que existieron hace algunos años. Estos avances son posibles gracias al desarrollo de la microelectrónica "antes" y ahora con la nanotecnología, es probable que tengamos aplicaciones en tan solo un espacio de 1×10^{-9}.

Con la invención del transistor en 1947 y el posterior desarrollo del circuito integrado "chip" por Jack S.T Clair Kilby en 1958, se abrió un campo ilimitado de aplicaciones que día con día reducen el espacio físico. En la década de 1980 el desarrollo de circuitos integrados de muy larga escala de integración con más de 100 000 transistores integrados en una base de silicio permitía el desarrollo de sistemas electrónicos complejos que impactaría no solo en la reducción del espacio físico sino también en su campo de aplicación: máquinas expendedoras de café, teléfonos, televisores, equipo de diagnóstico y medición, computadoras, juguetes, ignición electrónica de automóviles, cajas registradoras y pequeños misiles "weapons", etcétera.

La microelectrónica en todo su apogeo permitía para el 2002 integrar en un solo chip con tecnología de 0.12 micrómetros (μm), 60 millones de transistores, esta capacidad de integración que maravilló al mundo, permitió desarrollar aplicaciones cada vez más complejas. Hemos sido testigos de la miniaturización y aparición de: teléfonos celulares, internet, TV de alta definición, CD players para auto, MP3, computadoras con cámaras, dispositivos con reconocimiento de voz, cámaras digitales, sistemas de posicionamiento global "GPS", juegos con alto rendimiento gráfico "NINTENDO", televisión satelital, display de pantalla plana, computadoras de alto rendimiento y bajo costo, misiles de corto alcance, etcétera.

La integración de millones de transistores, no solo permitió la reducción del espacio físico, sino también disminuyó el costo de las aplicaciones, un comparativo entre los desarrollos de la década de 1970 y el año 2002 es el siguiente.

- En 1962 un radio de 9 transistores tenía un costo aproximado de 30 dólares.
  - En el 2002 un módulo de memoria con más de 64 millones de transistores es de bajo costo.
- En 1970 una computadora PDP9 tenía un costo muy elevado.
  - En el 2002 una computadora con capacidad de procesamiento de 1.7 GHz con memoria incluida tenía un costo de aproximado de 1 000 dólares.
- En 1970 las computadoras eran utilizadas exclusivamente por técnicos y científicos.
  - En el 2002 las computadoras son utilizadas por niños desde niveles de primaria y secundaria hasta llegar al entorno familiar: procesadores de texto, internet, juegos, etcétera.
- En 1970 el diseño de circuitos integrados era realizado por personas con amplios conocimientos en física de semiconductores, materiales, procesos de fabricación, diseño de circuitos y diseño de sistemas.
  - En el 2002 el diseño VLSI puede realizarse a través de herramientas CAD ("diseño asistido por computadora"), que permiten trasladar las especificaciones del diseño al archivo "layout" que contiene la información y geometría de los transistores utilizados en la aplicación.

Actualmente, con el desarrollo de la nanociencia, "manipulación de la materia a través del reordenamiento de átomos y moléculas, propuesta por el físico Dr. Richard Feynman en 1959, permitieron establecer las bases para la

generación de una nueva revolución tecnológica; recordemos que un nanómetro es la mil millonésima parte de un metro. En 1974, el Dr. Taniguchi, de la Universidad de Ciencia de Tokio, formula el concepto de nanotecnología, como la tecnología capaz de separar, consolidar y deformar átomos y moléculas de materiales. Pero no es sino hasta 1986 que Erik Drexler publica su libro *Engines of Creation* y describe la conceptualización de nanomáquinas con capacidad para construir desde computadoras, hasta maquinaria pesada, ensamblando átomo por átomo, molécula por molécula. Las aplicaciones de la nanociencia provocarán cambios y adelantos tecnológicos no solo en la industria electrónica sino también en áreas como; física, mecánica, medicina, química, bioquímica, biología molecular, computación y por supuesto en el diseño de chips con tecnología de fabricación de 1×10^{-9}, la creación de chips bioinspirados y nanobots dentro del cuerpo humano, dejaron de ser ciencia ficción para convertirse en una realidad con la que el ser humano aprenderá a convivir.

Tradicionalmente la técnica de diseño de los circuitos integrados, full custom design —diseño totalmente a la medida—, consiste en desarrollar circuitos para una aplicación especifica —ASIC—, integrando transistor por transistor y siguiendo los pasos normales de diseño: preparación de la oblea o base, el crecimiento epitaxial, la difusión de impurezas, la implantación de iones, la oxidación, la fotolitografía, la metalización y la limpieza química. Esta técnica ha permitido que los países altamente desarrollados y denominados de primer mundo ejerzan liderazgo sobre los demás, en el desarrollo e integración de aplicaciones. Sin embargo, en la actualidad la técnica presenta algunas grietas: el tiempo de desarrollo, el elevado costo de fabricación —en el proceso inicial—, y su uso en la producción de alto volumen, podría ser un inconveniente, quizá una desventaja, en las tendencias de competitividad actual.

Es aquí donde el desarrollo de aplicaciones ASIC puede realizarse a través de —en sus inicios—, una innovadora propuesta, que sugiere la utilización de celdas programables preestablecidas e insertadas dentro de un circuito integrado, lo que da origen a la familia de dispositivos lógicos programables (los PLD), cuyo nivel de densidad de integración ha evolucionado a través de los años desde los PAL (Arreglos Lógicos Programables), GAL (Arreglo Lógico Genérico), hasta llegar al uso de los CPLD (Dispositivo Lógico Programable Complejo) y los FPGA (Arreglo de Compuertas Programables en Campo), cuya utilización depende de la cantidad de lógica a integrar. Su ventaja directa es que la aplicación a realizar se obtiene de manera inmediata, se recomienda para aplicaciones en bajo volumen con opción a ser fabricadas en alto volumen, vía pedido al fabricante de la tecnología. La utilización de los dispositivos programables, en conjunto con las herramientas actuales de diseño (apéndice A), favorecen la creatividad y la competitividad en el desarrollo de aplicaciones en países en vías de desarrollo.

David G. Maxinez

Organización del libro

En el año 2000 se editó el libro **VHDL el arte de programar sistemas digitales**, con reimpresiones anuales desde 2001 hasta la fecha y que han ayudado a varias generaciones de estudiantes a comprender y aplicar el lenguaje. Un panorama muy distinto al enfrentado en el año 1994, cuando por primera vez tuve conocimiento de los lenguajes de descripción en hardware que venían a sustituir a los compiladores tradicionales —PALASM, OPAL, CUPL, etc.—, y la problemática para hacerme de las herramientas de trabajo y la bibliografía especializada sobre este tema. Actualmente las condiciones son muy diferentes, los circuitos programables son muy populares y las herramientas de trabajo fácilmente conseguibles, aunque aún pueden ser costosas en algunos sistemas de desarrollo.

Durante estos años, de la primera publicación a la fecha, alumnos y colegas me han preguntado por qué no incluir los últimos trabajos de investigación y aplicaciones realizadas con dispositivos programables —específicamente FPGA—, y los resultados obtenidos en campos como: robótica, visión, lógica difusa, sistema neurofuzzy, redes neuronales y en general las aplicaciones en el área de SoftComputing. La razón es simple, este nuevo libro pretende hacer llegar a los instructores y estudiantes de nivel preparatoria y licenciatura los conceptos y campo de aplicación, en temas comúnmente expuestos en una licenciatura —del diseño lógico a la arquitectura de computadoras—; es decir, el presente trabajo fue pensado con la idea de mostrar el uso del lenguaje de forma didáctica y amena sin importar la complejidad del proyecto a realizar —el fondo y no la forma—, buscando mostrar que los principios y conocimientos obtenidos dentro de un plan de estudios son totalmente válidos y utilizados en los cambios tecnológicos actuales y las herramientas futuras.

De igual modo es importante mencionar que los códigos o programas expuestos pueden ser sintetizados en dispositivos programables pequeños o en sistemas de desarrollo de alta densidad de integración, la diferencia es el costo y cantidad de lógica a integrar.

En resumen, el trabajo está formado por 12 capítulos, apéndices A, B, C, D, E, F y un manual de prácticas diseñado para dar soporte a la asignatura de diseño lógico (se incluye en el CD-ROM del libro), aunque el lector o profesor puede posteriormente realizar aplicaciones y desarrollar prácticas para temas específicos. El contenido ha sido pensado para poder usarse en diversas situaciones de diseño considerando el contenido o enfoque de varias asignaturas.

En términos generales, del capítulo 1 al 5 se establece lo que considero las características del lenguaje, los tipos de instrucciones y su aplicación. Estos capítulos son la base formal de conocimiento.

El **capítulo 1** describe las características principales del lenguaje, la descripción de la entidad y arquitectura, como los módulos básicos de programación, se pone especial interés en la programación mediante vectores, el uso de librerías y paquetes y se describe el formato de los diversos estilos convencionales de diseño, los cuales son utilizados extensamente a lo largo del libro. Es un capítulo de lectura esencial y obligada para el lector.

El **capítulo 2** muestra en detalle los tipos de declaraciones concurrentes y secuenciales, el manejo de señales para la interconexión de bloques lógicos y en conjunto con el apéndice E presentan los bloques o elementos lógicos combinacionales más utilizados. Básicamente presenta la base formal del diseño lógico combinacional.

El **capítulo 3**, de lectura obligada para el usuario, presenta el uso de las declaraciones secuenciales y su aplicación en el diseño de flip-flop, registros, contadores y diagramas de estado, se introduce la expresión ***atributo*** utilizado para asignar terminales a un dispositivo. Este capítulo realiza una síntesis de los circuitos secuenciales utilizados en la asignatura de diseño lógico.

El **capítulo 4** detalla el uso del algoritmo de la máquina de estado, **carta ASM**, como la herramienta de control de flujo utilizada para describir no solo el comportamiento de controladores digitales, sino también la herramienta de descripción de circuitos aritméticos. Capítulo indispensable y formativo para el diseño de unidades de control.

El **capítulo 5** presenta algunos de los métodos utilizados para ligar dentro de un solo dispositivo programable diversas entidades lógicas individuales. Capítulo base y muy útil para desarrollar aplicaciones en un solo circuito ***system on chip***.

De los **capítulos 6 al 11** se establece la descripción de conceptos con aplicaciones particulares; es decir, apoyo especifico y práctico a diversas asignaturas, sin tratar de abarcar o involucrar aspectos propios de cada materia, es decir solo se presentan detalles prácticos de realización en VHDL.

El **capítulo 6** muestra el control de movimiento en los robots móviles y su control de velocidad a través de la variación por anchura de pulso. Se utilizan en su realización física dispositivos PLD pequeños y sistemas de desarrollo, específicamente, Nexys™2 Spartan-3E FPGA (el manejo de esta tarjeta se describe en detalle en la práctica 8). Capítulo que permite al lector introducirse en el ámbito de la automatización de sistemas mecatrónicos.

El **capítulo 7** se sustenta en el algoritmo de la carta ASM, describe los sistemas de control más distintivos del diseño de sistemas digitales; control por registros, control por contadores y control microprogramado. La descripción se realiza mediante el estilo flujo de datos "utilizado en diseño lógico" y una descripción funcional, enfoque de la asignatura de microntroladores y arquitectura de computadoras. Este capítulo proporciona la base de conocimiento en el diseño de unidades de control, interesante en el diseño tradicional y su tendencia futura.

El **capítulo 8** presenta de una manera interesante la programación de entidades, visto como un componente y parte fundamental de un diseño más extenso, la incorporación del concepto bits-slice permite al lector desarrollar aplicaciones de manera modular y acorde a los recursos tecnológicos con los que cuenta, desde un punto de vista didáctico el uso de componentes puede ser abordado como una técnica de integración y manejarse de manera independiente a la forma presentada en este capítulo; es decir, puede abordarse después del capítulo 4. Este capítulo presenta el diseño modular con tendencia hacia el procesamiento de aplicaciones en paralelo.

El **capítulo 9** describe una de las técnicas utilizadas para optimizar y desarrollar aplicaciones del tipo *system on chip*, se detalla paso a paso el concepto de diseño modular y programación top level. Obligatorio para las asignaturas de diseño de computadoras secuenciales o paralelas.

El **capítulo 10** describe y resume los conceptos básicos utilizados en la estructura computacional, ***diseño*** de computadoras, capítulos 1, 2, 3 y 4, ***organización*** de computadoras, capítulos 7 y 8 y ***arquitectura*** de computadoras, capítulos 9 y 10. En este capítulo se describe el programa de un microprocesador básico, diseñado de manera modular mediante la interconexión de componentes y con tendencia hacía el concepto de sistemas embebidos, se menciona el uso de una arquitectura pipeline y se deja al lector que profundice sobre este tema en literatura especializada. Capítulo básico de aplicación y conceptualización del manejo de un microprocesador.

El **capítulo 11** complementa el tema de diseño de unidades de control, se incluye con el propósito de establecer que una instrucción, como parte de un set o conjunto de instrucciones debe de cumplir una tarea específica y que para ello es necesario crear el hardware apropiado, capaz de ejecutar la orden. Este capítulo busca que el lector experimente con el diseño de arquitecturas de hardware, para el diseño de unidades aritméticas y lógicas y en el diseño de circuitos microcontroladores con número limitado de instrucciones.

Finalmente el **capítulo 12** y gracias a los sistemas de desarrollo actuales, muestra que es posible sintetizar a nivel de hardware una red neuronal artificial en dispositivos FPGA, se incorpora el desarrollo paso a paso del aprendizaje de la red, simulada con ayuda de la herramienta de Matlab.

Agradecimientos

Esta es para mí una de las secciones más importantes de este trabajo. Dentro de mi labor académica en diversas instituciones: Universidad Nacional Autónoma de México, Campus Ciudad Universitaria y Aragón. Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México. Universidad Americana de Acapulco y Universidad Autónoma Metropolitana Unidad Azcapotzalco, he recibido comentarios, críticas y propuestas de mis amigos, colegas y alumnos, que me fueron expuestas en el salón de clase, en los pasillos universitarios, en la estancia para profesores o, inclusive, aquellas que llegaron a través de internet. Es el momento de agradecerles sus sugerencias y observaciones, que siempre tuve presentes al momento de escribir este trabajo.

En forma particular a mis alumnos: Alfredo Torres Noriega del ITESM-CEM, Hernán Bautista Lezama de la UNAM, Pedro Nicanor Luque y Betzayda D. Velázquez Méndez de la UAM Azcapotzalco por su colaboración en la simulación y desarrollo físico de los programas expuestos en este trabajo. A mis colegas, Dr. Jorge Ramírez Landa, M. en C. Juan Carlos Olguín R., Dr. Alejandro Cruz Sandoval por su valiosa participación en el capítulo de redes neuronales artificiales. Al Ing. Martín Hernández Hernández, M. en C. Rafael Antonio Márquez Ramírez, M. en I. Francisco Javier Sánchez Rangel e Ing. Ricardo Godínez Bravo por su valiosa colaboración en el desarrollo de los prototipos y prácticas. A mis amigos Ing. Guillermo Castillo Tapia por haber aceptado la revisión de este material, al Dr. Isaac Schanadower Barán cuyas propuestas y estructuras de programación vienen a fortalecer este trabajo, al Dr. Andrés Ferreira Ramírez Jefe del Departamento de Electrónica de la UAM Azcapotzalco, por invitarme en calidad de profesor visitante a trabajar al lado de grandes investigadores que ayudaron a mejorar este proyecto. Finalmente a mi editora, Estela Delfín R. y a todo su equipo, cuya calidad y trabajo colaborativo es contagioso y excelente.

David G. Maxinez

1

VHDL Estructura y organización

Introducción

En la actualidad, el lenguaje de programación VHDL (*Hardware Description Language*) constituye una de las herramientas de programación con mayor uso en el ambiente industrial y en el ámbito universitario, debido a la versatilidad con la cual se pueden describir y sintetizar circuitos y sistemas digitales en la búsqueda de soluciones de aplicación inmediata.

El uso correcto del lenguaje hace obsoleto el diseño tradicional, que organiza bloques lógicos de baja y mediana escala de integración, "compuertas, contadores, registros, decodificadores, etcétera". En otras palabras, para qué organizar diversas estructuras lógicas en una determinada solución, si esta se puede crear en una sola entidad, la cual no solo proporciona una reducción considerable de espacio físico, sino que además también produce un resultado más directo y menos susceptible a los errores derivados de la conexión entre varios componentes.

El resultado —la solución— es susceptible de ser encapsulada en dispositivos lógicos programables, llamados PLD (véase figura 1.1).

Al emplear dispositivos programables de muy bajo costo, conocidos por sus siglas en inglés como GAL (Arreglos Lógicos Genéricos), los cuales se utilizan con mucha frecuencia en proyectos donde lo que se requiere de manera primordial es la interconexión de algunos dispositivos convencionales como multiplexores, comparadores, sumadores, etcétera; estos chips pueden adquirirse de manera individual y ser programados en grabadores convencionales.

En el caso de que la solución requiera de un mayor número de componentes lógicos o "pins" de salida, pueden utilizarse circuitos CPLD (por sus siglas en inglés; Dispositivos Lógicos Programables Complejos), los cuales también se consiguen de forma individual o como parte de un sistema de desarrollo. Por otro lado, si el proyecto lo amerita, es posible utilizar los FPGA (por sus siglas en inglés; Arreglo de Compuertas Programables en Campo), recomendados para ser empleados y programados dentro de un sistema de desarrollo y en aplicaciones con tendencia hacia el desarrollo de sistemas completos dentro de un solo circuito integrado SOC (*System On Chip*).

Las opciones que brinda el hardware para la integración de la aplicación permiten introducir y utilizar el lenguaje en diversas materias, cuyo nivel de profundidad y diseño depende de los bloques lógicos por emplear; de esta

manera, el diseño lógico y el de sistemas digitales, la robótica, la arquitectura de computadoras y la algorítmica, entre otras, son algunas de las áreas donde VHDL presenta una alternativa de diseño. Por ejemplo, en un curso de diseño lógico, quizá lo más recomendable sea utilizar el lenguaje y programar en dispositivos GAL; sin embargo, esta no es una condición necesaria e indispensable, pues depende en gran medida de la economía personal o institucional.

Como se verá adelante, los cambios en la programación son irrelevantes respecto de la utilización de diversas tecnologías de fabricación de los chips. Por tanto, quizás los aspectos que deban cuidarse con más atención son la conceptualización y el entendimiento de la arquitectura interna del circuito; no es lo mismo programar en Arreglos Lógicos Genéricos que en FPGA, cuyas arquitecturas internas son muy diferentes.

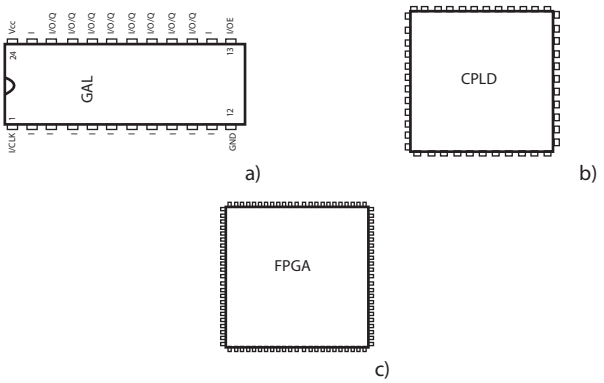


Figura 1.1 Tecnologías de soporte. a) GAL. b) CPLD. c) FPGA.

1.1 VHDL su estructura

El lenguaje de descripción en hardware VHDL se estructura en módulos o unidades funcionales, identificados mediante una palabra reservada y particular de este lenguaje (véase figura 1.2). En tanto, a su vez, cada módulo tiene una secuencia de instrucciones o sentencias, las cuales, en conjunto con las declaraciones de las unidades involucradas en el programa, permiten la descripción, la comprensión, la evaluación y la solución de un sistema digital.

Al interior de la estructura de un programa, las unidades Entidad (**`Entity`**) y Arquitectura (**`Architecture`**) —en conjunto— forman la columna vertebral de este lenguaje. Por su parte, los módulos restantes, no necesariamente utilizados en la búsqueda de una solución, sirven entre otras cosas para optimizar y generalizar la aplicación en futuros desarrollos, como se verá cuando la ocasión se presente. Sin embargo, en este momento nuestra atención se centra en describir la función de la entidad y la arquitectura.

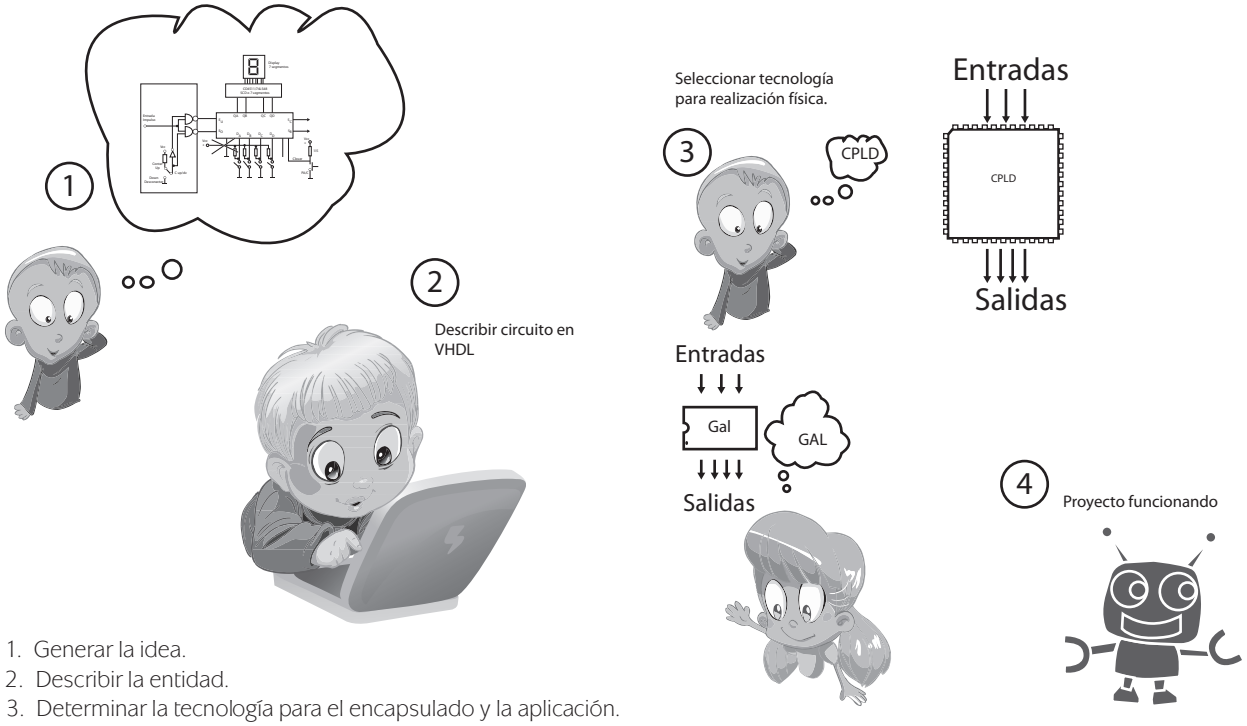
- Library = Bibliotecas
- **Entity = Entidad**
- **Architecture = Arquitectura**
- Package = Paquete
- Component = Componente

Figura 1.2 Módulos de diseño en VHDL.

Entidad (entity)

Una **entidad** básicamente representa la caracterización del dispositivo físico; es decir, exhibe las entradas y las salidas del circuito (llamados pins) que el diseñador ha considerado pertinentes para integrar su idea o aplicación; en la figura 1.3, se puede observar con detalle la secuencia de desarrollo.

Con base en esta idea, una **entity** —por la palabra reservada del programa— constituye un bloque de diseño que puede ser analizado y programado como un elemento individual, ya sea como una compuerta, un sumador o un decodificador, entre otros, incluso ser considerado como un sistema a través de su relación entre entradas y salidas, las cuales representan los puntos de observación o de conexión a elementos periféricos propios de la aplicación.



1. Generar la idea.
2. Describir la entidad.
3. Determinar la tecnología para el encapsulado y la aplicación.

Figura 1.3 Representación del proceso de aplicación y desarrollo de la entidad.

En la figura 1.4 a), la entidad proporciona los pins de salida "d3, d2, d1, d0" para el control de un motor a pasos, mientras que en la figura 1.4 b), la entidad se conecta a un visualizador de 7 segmentos.

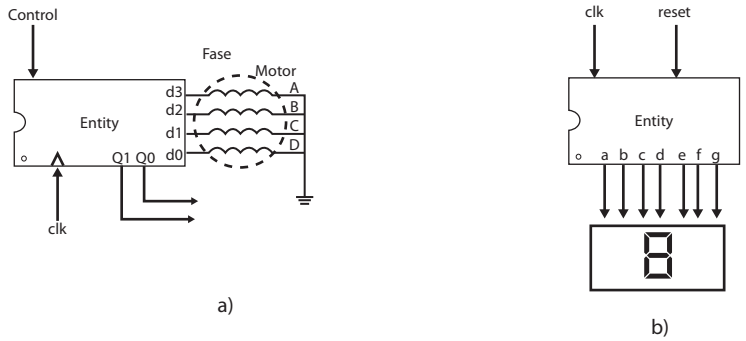


Figura 1.4 Entidades el concepto.

Puertos de entrada-salida

Cada una de las señales de entrada y salida en una entidad es referida como un **puerto**, el cual es equivalente a una terminal (pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un **nombre**, un **modo** y un **tipo de dato**.

El **nombre** es utilizado como una forma de llamar al puerto; el **modo** permite definir la dirección que tomará la información, mientras que el **tipo** precisa qué clase de información se transmitirá a través del puerto. Por ejemplo, en el caso de los puertos de la entidad representada en la figura 1.5, aquellos que son de entrada están indicados por las variables **a** y **b**; mientras que el puerto de salida se representa por la variable **c.** Por otra parte, el **tipo** de dato será tratado más adelante.

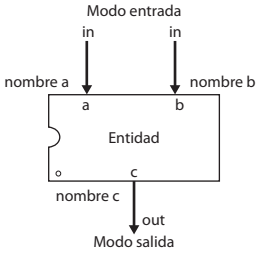


Figura 1.5 Puertos de entrada–salida.

Modos

Como se mencionó antes, un modo permite definir la dirección hacia donde el dato es transferido. Un modo puede tener uno de cuatro valores: **in** (entrada), **out** (salida), **inout** (entrada/salida) y **buffer** (véase figura 1.6).

- **Modo in.** Se refiere a las señales de entrada a la entidad. El modo in es solo unidireccional y únicamente permite el flujo de datos hacia dentro de la entidad.
- **Modo out**. Indica las señales de salida de la entidad.
- **Modo inout.** Permite declarar a un puerto de forma bidireccional, es decir como de entrada/salida, además hace posible la retroalimentación de señales dentro o fuera de la entidad.
- **Modo buffer.** Permite realizar retroalimentaciones dentro de la entidad; pero, a diferencia del modo inout, el puerto declarado se comporta como una terminal exclusiva de salida.

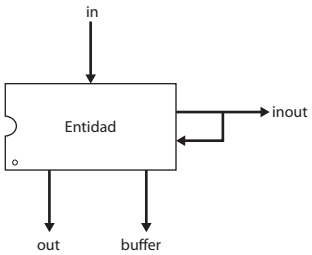


Figura 1.6 Tipos de modos.

Tipos de datos

Los tipos son los valores (datos) que el diseñador establece para los puertos de entrada y salida dentro de una entidad, y que son asignados de acuerdo con las características de un diseño en particular. Algunos de los tipos más utilizados son el **bit**, el cual tiene valores de 0 y 1 lógico; el tipo **boolean** (booleano) define valores de verdadero o falso en una expresión; el **bit_vector** (vectores de bits), el cual representa un conjunto de bits para cada variable de entrada y/o salida, y el tipo **integer** (entero), que representa a un número entero.

Los anteriores son solo algunos de los tipos que maneja VHDL, aunque no son los únicos.[1] Los tipos de datos y su uso se introducirán conforme se vayan requiriendo y empleando a lo largo del texto.

Declaración de entidades

La declaración de una entidad consiste en describir las entradas y las salidas de un circuito identificado como **Entity** (entidad); en otras palabras, la declaración señala las terminales o los pines de entrada y de salida con los que cuenta el circuito. Por ejemplo, considérese la tabla de verdad que se muestra en la figura 1.7; como se puede observar, esta tiene tres entradas, A, B y C, y dos salidas, F0 y F1. En este caso, a la entidad se le ha identificado con el nombre de TABLA, tal y como se muestra.

A B C	F0 F1
0 0 0	0 1
0 0 1	1 0
0 1 0	1 1
0 1 1	1 0
1 0 0	0 0
1 0 1	1 1
1 1 0	1 0
1 1 1	1 1

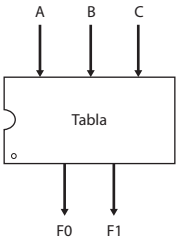


Figura 1.7 Declaración de la entidad Tabla.

La descripción de este programa se muestra en el listado 1.1, donde los números de las líneas (1, 2, 3, 4, 5) no son parte del código y serán utilizadas como referencia para explicar alguna sección en particular. En tanto, las palabras en **negritas** están reservadas para el lenguaje de programación, es decir, tienen un significado especial para el programa.

[1] En el apéndice A se listan los tipos de datos existentes en VHDL.

```
1 --Declaración de la entidad de una tabla de verdad
2 entity Tabla is
3 port (A, B, C:   in bit;
4      F0, F1:  out bit):
5 end Tabla ;
```

Listado 1.1 Declaración de la entidad tabla de la figura 1.7.

Ahora comencemos con el análisis del código línea por línea. Así, la línea 1 inicia con dos guiones (**- -**), los cuales indican que el texto que está a la derecha es un comentario; estos se usan únicamente con el fin de documentar el programa, ya que todos los comentarios son ignorados por el compilador. En la línea 2 se empieza la declaración de la entidad utilizando la palabra reservada **entity**, seguida del **identificador** o nombre de la entidad "Tabla" (para este ejemplo) y la palabra reservada **is**.

Los puertos de entrada y salida (**port**) son declarados en las líneas 3 y 4, respectivamente; en este caso, los pines de entrada son A, B y C, mientras que los puertos de salida están representados por F0 y F1. Si se observa con atención la tabla de verdad de la figura 1.7, es posible apreciar que en su descripción solo se utilizan valores lógicos ('0' y '1'); por tanto, es de suponer que el tipo de dato empleado en la declaración es bit. Por último, en la línea 5 termina la declaración de entidad con la palabra reservada **end,** seguida del nombre de la entidad Tabla.

Aspectos importantes a considerar

Como en cualquier lenguaje de programación, es importante hacer notar que VHDL también sigue una sintaxis y una semántica en el código de programación, las cuales es necesario respetar lo siguiente.

- **Punto y coma (;):** se utiliza para cerrar y finalizar declaraciones.
- **Dos puntos (:):** en este caso se usan como separador entre el nombre de los puertos y los modos de entrada.
- **El paréntesis:** después del modo de salida **out** bit cierra la declaración de los puertos (port).
- El uso de las mayúsculas o las minúsculas en la declaración es irrelevante para el compilador.

Ejemplo 1.1

Declarar la entidad del circuito lógico que se muestra en la figura 1.8.

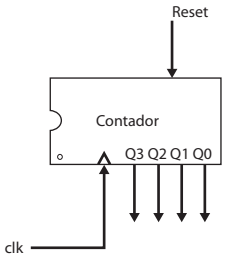


Figura 1.8

Solución

Como se puede advertir en la figura, las entradas y salidas del circuito se encuentran claramente identificadas.

- **Modo in:** clk, Reset
- **Modo out:** Q3,Q2,Q1 y Q0
- **Entidad:** Contador

Entonces, la declaración de la entidad sería de la siguiente forma:

```
1 -- Declaración de la entidad contador
2 entity Contador is
3   port( clk,Reset: in bit;
4      Q3,Q2,Q1,Q0: out bit);
5 end Contador ;
```

Listado 1.2

Ejemplo 1.2

Declarar la entidad del circuito lógico mostrado en la figura 1.9.

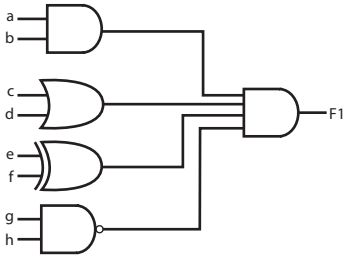


Figura 1.9

Solución

Como se puede observar, las entradas y salidas están claramente identificadas.

- **Modo in:** a, b, c, d, e, f, g, h
- **Modo out:** F1
- **Entidad:** Circuito

Aquí, el diseñador deberá identificar los pines de entrada y de salida para establecer la entidad. Recuérdese que la escritura del código mediante letras mayúsculas o minúsculas es irrelevante en la edición del programa.

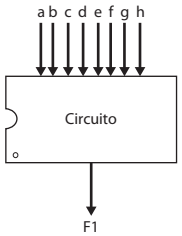


Figura 1.10

```
1 -- Declaración de la entidad
2    entity Circuito is
3    port( a,b,c,d,f,g,h,: in bit;
4                       F1: out bit);
5    end Circuito;
```

Listado 1.3

Ejemplo 1.3

Declarar la entidad del circuito lógico mostrado en la figura 1.11.

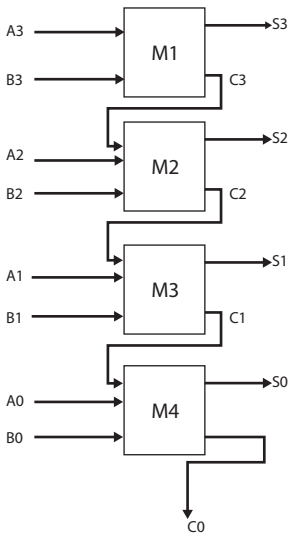


Figura 1.11

Solución

En este caso se puede notar que existen varios módulos individuales, los cuales en conjunto forman una entidad general. Además, las salidas C3,C2,C1,C0 deben considerase como inout, es decir señales de entrada-salida, dado que en la estructura interna se observa que la salida de cada una de estas retroalimenta, a la entrada, al siguiente módulo:

- **Modo in:** A3, A2, A1, A0, B3, B2, B1, B0
- **Modo out:** S3, S2, S1, S0
- **Modo inout:** C3, C2, C1, C0
- **Entidad:** Proyecto

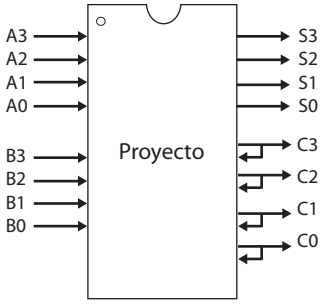


Figura 1.12

```
1 --  Integración de módulos
2 entity Proyecto is
3 port( A3,A2,A1,A0,B3,B2,B1,B0: in bit;
4          C3,C2,C1,C0: inout bit;
5          S3,S2,S1,S0: out bit);
6 end Proyecto;
```

Listado 1.4

1.2 Identificadores

Los identificadores son simplemente los nombres o las etiquetas que se usan para hacer referencia a variables, constantes, señales, procesos, etc. Estos identificadores pueden ser números, letras del alfabeto y/o guiones bajos que separan caracteres. Es importante resaltar que no existe una restricción en cuanto a su longitud. Todos los identificadores deben seguir ciertas especificaciones o reglas para que puedan ser compilados sin errores (véase tabla 1.1).

Tabla 1.1

Regla	Incorrecto	Correcto
El primer carácter siempre debe ser una letra mayúscula o una minúscula.	4suma	Suma4
El segundo carácter no puede ser un guión bajo.	S_4bits	S4_bits
No se permite el uso de dos guiones juntos.	Resta__4	Resta_4_
Un identificador no puede utilizar símbolos.	Clear#8	Clear_8

Es importante destacar que VHDL cuenta con una lista de palabras reservadas, las cuales no pueden utilizarse como identificadores (véase apéndice E).

Declaración de entidades utilizando vectores

La entidad Proyecto se muestra en la figura 1.13 a), y por comodidad nuevamente se representa en la figura 1.13 b) mediante sus bits individuales: A3, A2…, etcétera. La entidad puede esquematizarse mediante la señalización, que se muestra con claridad en la figura 1.3 b), donde se indican, mediante un bus —grupo de cables de conexión—, los cuatro bits de entrada y/o salida de cada una de las variables involucradas.

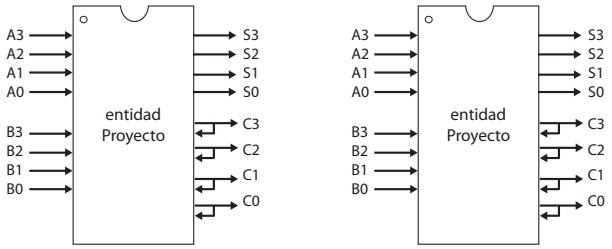


Figura 1.13 Señalización de pins por medio de un bus.

El acomodo de los bits —vector— que componen el bus puede ser ordenado de forma ascendente o descendente, por ejemplo para las entradas A y B se escribiría de la siguiente manera:

Arreglo descendente:

```
A(A3,A2,A1,A0)
B(B3,B2,B1,B0)
```

Arreglo ascendente:

```
A(A0,A1,A2,A3)
B(B0,B1,B2,B3)
```

En VHDL, el conjunto bits ordenados pueden considerarse como **vectores de bits**, los cuales deben tomarse en cuenta en grupo y no como bits individuales. Asimismo, en VHDL la manera de describir una configuración que use vectores radica en la utilización de la sentencia **bit_vector**; así, la forma en la que se ordenan se indica como ascendente "**to**" o como descendente "**downto**". Por su parte, el rango o el número de bits que utiliza el vector se representa en la notación siguiente:

```
0 to 3 (0 hasta 3)
3 downto 0 (3 hacia 0)
```

Así, por ejemplo, la sentencia **3 downto 0** involucra un vector de 4 bits **(3:0)**. Para su ejemplificación, consideremos la entidad proyecto y el código de programación mostrado en la figura 1.14.

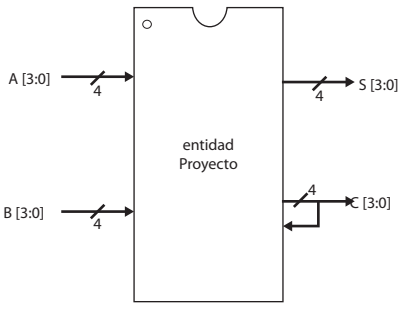


Figura 1.14 Declaración de la **entidad Proyecto** mediante vectores.

```
1 --  Declaración de vectores
2 entity Proyecto is
3 port( A,B,: in bit_vector(3 downto 0);
4       C: inout bit_vector(3 downto 0);
5          S: out bit_vector(3 downto 0));
6 end Proyecto;
```

Listado 1.5

La descripción de este ejemplo es la siguiente.

- En la línea 3, las entradas A y B se declaran como vectores de entrada de 4 bits ordenados de forma descendente (3:0).
- En la línea 4, la variable C se declara como un vector de entrada salida de 4 bits ordenado de forma descendente (3:0).
- En la línea 5, la variable de salida S es un vector de 4 bits ordenados de forma descendente; en esta línea considérese el doble paréntesis al final de la instrucción. El primero es parte del rango del vector y el segundo marca el cierre de la declaración de los puertos (port).

Ejemplo 1.4

Describir en VHDL la entidad del circuito que se muestra en la figura 1.15; considerar el uso de vectores para la solución.

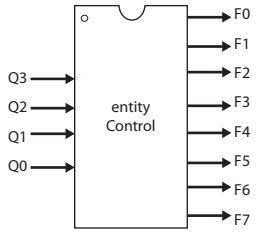


Figura 1.15

Solución

En este caso, nótese que existen cuatro señales de entrada: Q3, Q2, Q1, Q0, las cuales pueden agruparse como un vector al que llamaremos (Q); en tanto, podemos integrar las señales de salida de F0 hasta F7 en un vector denominado (F). Para la solución, el vector Q se ordenará de forma descendente y el vector F de manera ascendente, es decir:

- **Modo in:** Q (3 downto 0). El vector considera 4 bits de entrada.
- **Modo out:** F(0 to 7). El vector considera 8 bits de salida.
- **Entidad:** Control

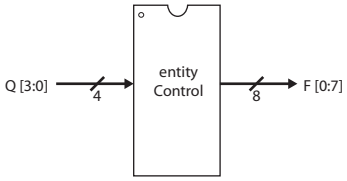


Figura 1.16

```
1 -- Declaración de vectores
2 entity Control is
3 port (Q:  in bit_vector (3 downto 0);
4       F:  out bit_vector (0 to 7));
5       end Control;
```

Listado 1.6

Declaración de entidades utilizando librerías y paquetes

Un aspecto importante en la programación con VHDL radica en el uso de **librerías** y **paquetes** que permiten declarar y almacenar estructuras lógicas, seccionadas o completas, y que facilitan el diseño. Una **librería** (**biblioteca**) es un lugar al cual se accede para emplear las unidades de diseño predeterminadas por el fabricante de la

herramienta (**paquete**), y que se aplican para agilizar el diseño, además de que también permiten almacenar el resultado obtenido de la compilación de un diseño, con el fin de que este pueda ser utilizado dentro de uno o varios programas.

En VHDL se encuentran definidas diversas librerías; su cantidad depende de las herramientas que el fabricante del software ha instalado. Por lo general, siempre hay dos librerías: **ieee** y **work "Cypress Semiconductor Ver 5.2"** (véase figura 1.17). Al interior de la librería **ieee** se ubica el paquete std_logic_1164, mientras que en la librería **work** se localizan los paquetes numeric_std, std_arith y gates.

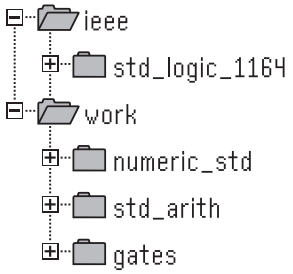


Figura 1.17 Contenido de las librerías **ieee** y **work**.

Un paquete es una unidad de diseño que permite desarrollar un programa en VHDL de manera ágil, debido a que en este se encuentran algoritmos preestablecidos (sumadores, restadores, contadores, etc.) que ya tienen optimizado un determinado comportamiento. Por esta razón, el diseñador no necesita caracterizar paso a paso una nueva unidad de diseño si esta ya se ubica almacenada en algún paquete; en este caso, solo basta llamarlo y especificarlo dentro del programa.

Por tanto, un paquete no es más que una unidad de diseño formada por declaraciones, programas, componentes y subprogramas que incluyen los diversos tipos de datos (bit, booleano, std_logic) empleados en la programación en VHDL, y que por lo general forman parte de las herramientas en software. Cuando en el diseño se utiliza algún paquete, es necesario llamar a la librería que lo contiene, lo cual se lleva a cabo a través de la siguiente declaración:

```
library ieee;
```

Esta declaración permite el empleo de todos los componentes incluidos en la librería **ieee**. En el caso de la librería de trabajo (**work**), su uso no requiere de la declaración **library**, dado que la carpeta work siempre está presente al desarrollar un diseño.

Paquetes

El paquete **std_logic_1164** (estándar lógico_1164), que se encuentra dentro de la librería **ieee**, contiene todos los tipos de datos comúnmente utilizados en VHDL, como: **std_logic_vector**, **std_logic**, std_signed y std_unsigned, entre otros.

La forma en que se accede a la información contenida dentro de un paquete es mediante la sentencia **use** seguida del nombre de la librería y del paquete, respectivamente; esto es:

```
use nombre_librería.nombre_paquete.all;
```

Por ejemplo:

```
use ieee.std_logic_1164.all;
```

En este caso, **ieee** es la librería, **std_logic_1164** es el paquete y **all** es la palabra reservada, la cual indica que todos los componentes almacenados dentro del paquete pueden ser utilizados.

A continuación se listan algunos de los paquetes más importantes al interior de las librerías de VDHL, así como sus características principales.

- El paquete **numeric_std** define funciones para la realización de operaciones entre diferentes tipos de datos; en este, los tipos pueden representarse con y sin signo (véase apéndice C).

- El paquete **numeric_bit** determina tipos de datos binarios con y sin signo.
- El paquete **std_arith** define funciones y operadores aritméticos como: más (+), menos (-), división (/), multiplicación (*), igual (=), mayor que (>), menor que (<), entre otros (véase apéndice F).

A reserva de ir introduciendo los diversos paquetes contenidos en VHDL, conforme la programación lo requiera. En lo sucesivo, se efectuará un uso extensivo de las librerías y de los paquetes dentro de los programas desarrollados en el texto.

Ejemplo 1.5

Realizar la declaración de la entidad Bloque mostrada en la figura 1.18 mediante el uso de librerías y paquetes.

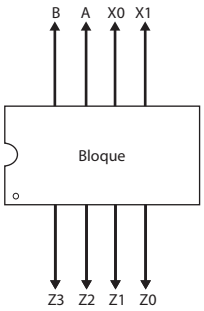


Figura 1.18

Solución

En esta entidad se utiliza la biblioteca **ieee** y el paquete std_logic_1164. Las entradas y salidas se manejan de forma individual y en tipo vector. Es decir:

- **Modo in:** X vector de 2 bits.
- **Modo in:** A, B entradas individuales de un bit.
- **Modo out:** Z salida vector de 4 bits.
- **Entidad:** Bloque

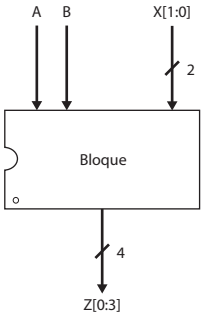


Figura 1.19

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity Bloque is
4 port (X: in std_logic_vector(1 downto 0);
5       A,B: in std_logic);
8       Z: out std_logic_vector(0 to 3));
7 end Bloque;
```

Listado 1.7

Ejemplo 1.6

En la figura 1.20 se representa una entidad con capacidad para realizar una suma de 2 bits, A y B, que da como resultado la operación SUMA= A+B. Realizar la declaración de la entidad correspondiente mediante el uso de paquetes y librerías.

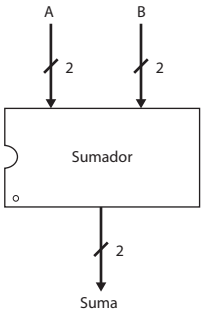


Figura 1.20 Entidad Sumador.

Solución

Como se puede observar, en la línea 3 la librería work utiliza el paquete std_arith, requerido para que la operación SUMA= A+B se pueda efectuar; este paquete permite el uso del operador (+), mismo que hace posible esta operación. En el caso de que la librería no esté dentro del programa al momento de su ejecución, el compilador enviará un mensaje de error.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.std_arith.all;
4 entity Sumador is
5 port (A, B : in std_logic_vector(1 downto 0);
6        SUMA: out std_logic_vector(1 downto 0));
7 end Sumador;
```

Listado 1.8

La librería de trabajo **work** es el lugar preestablecido por el software para almacenar los programas que el usuario va realizando; por tanto, esta siempre se encuentra presente en la compilación de un diseño. Así, mientras no sea especificada otra librería, los diseños invariablemente se guardarán dentro de ella; por dicha razón no requiere el uso de la declaración **library.**

Operadores aritméticos. Como su nombre lo indica, estos permiten realizar operaciones del tipo aritmético, tales como suma, resta, multiplicación, división, cambios de signo, valor absoluto y concatenación. Dichos operadores generalmente son utilizados en el diseño lógico para la descripción de sumadores y restadores o para las operaciones de incremento y decremento de datos. Los operadores aritméticos predefinidos en VHDL se muestran a continuación.

Tabla 1.2 Operadores Aritméticos utilizados en VHDL

Operador	Descripción
+	Suma
-	Resta
/	División
*	Multiplicación
**	Potencia

1.3 Arquitectura (architecture)

Una **arquitectura** define el algoritmo o la estructura de solución de una entidad, en esta se describen las instrucciones o los procedimientos "programa" que deben llevarse a cabo para obtener la solución deseada por el diseñador.

La gran ventaja que presenta VHDL con respecto a los compiladores tradicionales de diseño *PALASM*, *OPAL*, *PLP*, *ABEL* y *CUPL* (véase apéndice A), entre otros disponibles para la programación de Dispositivos Lógicos Programables, es la extensa variedad de formatos con los que cuenta y que utiliza en la descripción de una entidad. Es decir, en VHDL es posible describir en diferentes *niveles de abstracción,* que van desde el uso de ecuaciones y estructuras lógicas, la descripción mediante la simbología de la transferencia de registros RTL, hasta la simplicidad que puede representar una caja negra —sistema—, cuya interpretación "función de transferencia", la relación entre entradas y salidas, describe el funcionamiento de la entidad.

No obstante, en VHDL no existe de manera formal un procedimiento de diseño. Como ya se mencionó, la versatilidad en la abstracción y la descripción de una entidad permiten al usuario la tarea de planear la estrategia por utilizar para una solución en particular. Sin duda, la experiencia en programación puede ser un factor importante de diseño, sin embargo, VHDL posee una estructura de programación tan amigable que es muy fácil de entender, aun por personas que no han tenido relación alguna con lenguajes de programación.

De manera general y con base en cómo se presenta la entidad (**entity**), a primera vista, en VHDL se pueden distinguir los siguientes estilos de programación.

- Estilo por flujo de datos.
- Estilo funcional.
- Estilo estructural.

Descripción por flujo de datos

La descripción por flujo de datos muestra con detalle la transferencia de información entre las entradas y las salidas de una entidad. Este estilo, totalmente comprensible para el usuario, se recomienda para quienes recién se inician en la programación en VHDL, en materias como diseño lógico, donde las tablas de verdad y las ecuaciones lógicas son parte fundamental en la descripción de un circuito lógico.

Para una mejor comprensión de la descripción por flujo, considérese la tabla de verdad de la figura 1.21; como se puede observar la tabla tiene dos entradas, A y B, y dos salidas, F0 y F1. Para fines explicativos, también se ha agregado el minitérmino correspondiente (m) a cada combinación de entrada. Por ejemplo, el minitérmino m0 hace referencia a la combinación cero de entrada A=0 y B=0.

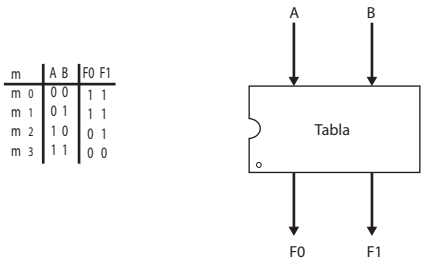


Figura 1.21 Declaración en flujo de datos.

A continuación se realiza la descripción del código línea por línea. En las líneas 2 y 3 se escriben la librería y el paquete respectivamente. En tanto, de las líneas 4 a la 7 se escribe la entidad. Como se puede ver, las entradas y las salidas se utilizaron de manera individual; nótese que en estas se mezcló el uso de letras mayúsculas y minúsculas, lo cual resulta "irrelevante para el compilador".

```
 1 -- Declaración flujo de datos
 2  library ieee;
 3 use ieee.std_logic_1164.all;
 4 entity tabla is
 5 port (A,B:   in std_logic ;
 6     F0,F1 : out std_logic) ;
 7 end tabla ;
 8 architecture flujo of tabla is
 9 begin - - comenzar
10     F0 <= '1' when (A='0' and B='0') else
11           '1' when (A='0' and B='1') else
12           '0' ;
13     F1 <= '0' when (A='1' and B='1') else
14           '1' ;
15 end flujo;
```

Listado 1.9

En la línea 7 se escribe la arquitectura identificada como flujo; la sentencia se lee:

—La arquitectura flujo asignada a la entidad tabla es—.

En la línea 8 se inicia el programa con la declaración "begin". De modo que la línea 9 se asigna a la variable F0 el valor lógico de uno ('1'), correspondiente a la primera combinación de las entradas A=0 y B=0 de la siguiente tabla de verdad:

A B	F0
0 0	1

Figura 1.22

Es decir, asigna a la variable F0 el valor de uno (**<= '1'**), cuando (**when**) la variable A es igual a uno ('**0**'), y (**and**) cuando la variable B es igual a uno ('**0**'), sino (**else**). La sentencia es la siguiente:

```
9     F0 <= '1' when (A='0' and B='0') else
```

El término "else-sino" propicia el análisis de la siguiente condición en la tabla de verdad: minitérmino, m1.

En la línea 10 se asigna el valor de '1' cuando las combinaciones de entrada para A y B son "A='0' and B='1'. La sentencia es la siguiente:

```
10          '1' when  (A='0' and B='1'  else
```

De nueva cuenta, la instrucción sino **else** analiza el siguiente renglón; en este caso, la salida F0 para la combinación m2, m3 son cero (0), lo que origina que en la línea 11 termine la asignación a la variable F0 con el punto y coma final (;).

m	A B	F0
m 0	0 0	1
m 1	0 1	1
m 2	1 0	0
m 3	1 1	0

```
11 ——→ F0=`0´;
```

Figura 1.23

En el caso de la salida F1, es posible observar sin ningún esfuerzo que es más conveniente declarar la combinación m3, es decir F1='0' when A='1' y B='1' sino '1'; líneas 12 y 13, respectivamente.

m	A B	F0 F1
m 0	0 0	1
m 1	0 1	1
m 2	1 0	1
m 3	1 1	0

```
12     F1 <= `0´ when (A=´1´ and B=´1´) else
13           `1´;
```

Figura 1.24

Como se puede observar, en este estilo de declaración, "flujo de datos", es muy fácil dirigir y describir el camino que los datos siguen dentro de la entidad. Otra opción de esta descripción es apoyarse en los operadores lógicos (and, not, or, xor, xnor, etc.) para obtener una solución.

Para ejemplificar considérese otra vez la tabla de verdad de la entidad anterior, que se presenta nuevamente en la figura 1.25.

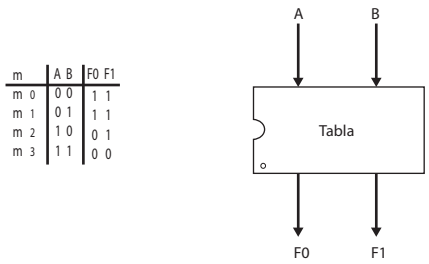


Figura 1.25 Tabla de verdad de la entidad tabla.

Las ecuaciones lógicas para las salidas F0 y F1 son:

$$F0 = \overline{A} * \overline{B} + \overline{A} * B \qquad F1 = \overline{A} * \overline{B} + \overline{A} * B + A * \overline{B}$$

Su declaración utilizando los operadores lógicos incluidos en el lenguaje son:

```
F0 <= ((NOT A AND NOT B) OR (NOT A AND B));
F1 <= ((NOT A AND NOT B) OR (NOT A AND B) OR (A AND NOT B));
```

En consecuencia, el programa queda como se muestra a continuación:

```
  -- Declaración flujo de datos
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3 entity tabla is
 4 port (A,B:   in std_logic ;
 5   F0,F1 : out std_logic) ;
 6 end tabla ;
 7 architecture flujo of tabla is
 8 begin - - comenzar
 9   F0 <= (not A and not B) or (not A and B);
10   F1 <= (not A and not B) or (not A and B) or (A and not B);
11 end flujo;
```

Listado 1.10

Como se puede ver, el diseño que resulta de la utilización de los operadores lógicos del álgebra booleana o de la declaración mediante la estructura when-else se utiliza con mucha frecuencia en entidades o proyectos donde el diseñador puede acceder con facilidad al formato de tabla de verdad.

Ejemplo 1.7

Describir, mediante la declaración **when-else** y con base en la tabla de verdad**,** el funcionamiento de la siguiente compuerta AND.

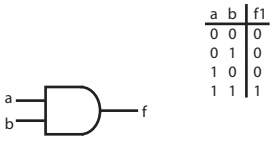


Figura 1.26

Solución

Si consideramos la salida igual a 1 para f1, el programa quedaría como sigue:

```
-- Descripción flujo de datos
library ieee;
use ieee.std_logic_1164.all;
entity com_and is
port( a,b: in std_logic;
        f1: out std_logic);
end com_and;
architecture compuerta of com_and is
begin
f1 <= '1' when (a = '1' and b = '1' ) else
     '0';
end compuerta;
```

Listado 1.11

Ejemplo 1.8

Describir mediante ecuaciones booleanas el circuito mostrado a continuación:

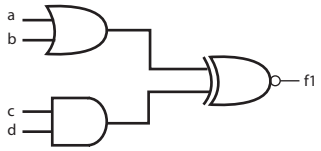


Figura 1.27

Solución

Primero, obtenemos las ecuaciones de salida de la compuerta or y and, y después utilizamos el operador xnor.

```
-- Declaración utilizando ecuaciones boolenas
library ieee;
use ieee.std_logic_1164.all;
entity ejemplo is
port ( a,b,c,d: in std_logic;
             f: out std_logic);
end ejemplo;
architecture compuertas of ejemplo is
begin
     f <= ((a or b) xnor (c and b));
end compuertas;
```

Listado 1.12

Operadores lógicos

Los operadores lógicos que se utilizan en la descripción con ecuaciones booleanas y que están definidos dentro de los diferentes tipos de datos —bit, boolean o std_logic— son los operadores: and, or, nand, xor, xnor y not. Las operaciones que se efectúen entre estos (excepto not) deben realizarse con datos que tengan la misma longitud o palabra de bits.

Los operadores lógicos presentan el siguiente orden y prioridad al momento de ser compilados:

1. Expresiones entre paréntesis.
2. Complementos.

3. Función and.
4. Función or.

Las operaciones **xor** y **xnor** son transparentes al compilador, el cual las interpreta mediante la suma de productos correspondiente a su función.

Aspectos importantes por considerar

- Se utiliza '-' una comilla para expresar un valor individual,
- Se utiliza "- " doble comilla para describir un grupo de bits "vector".

Como ejemplo del uso de operadores lógicos en VHDL, obsérvese la tabla 1.3.

Tabla 1.3

Ecuación	En VHDL
q = a + x · y	q = a *or* (x *and* y)
y = $\overline{a + b \cdot /c + d}$	y = *not* (a *or* (b *and not* c) *or* d)

Descripción funcional

En una descripción funcional lo más importante es el conocimiento global del sistema, razón por la cual las entidades diseñadas bajo este estilo son programadas como una caja negra; es decir, no importa la organización o la estructura interna de la entidad, solo se requiere que el programador conozca lo que espera obtener en la salida y la forma en que operan las pins de entrada. Por ejemplo, considérese una entidad en la cual existe una salida C que adopta el valor C=1 cuando las señales de entrada, A y B, son iguales; en caso contrario, la salida C toma el valor de cero C=0. Entonces, la síntesis del problema sería la siguiente:

```
Si A=B entonces C=1, sino C=0
if A=B   then   C=1, else C=0
```

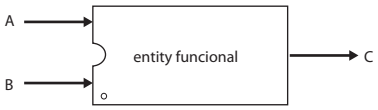


Figura 1.28

A continuación se muestra el código que representa la descripción del circuito.

```
1  -- Ejemplo de una descripción funcional
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity funcional is
5  port (A,B: in std_logic;
6      c: out std_logic);
7  end funcional ;
8  architecture caja of funcional is
9  begin
10 process (A,B)
11 begin
12   if A = B then        -- Si A=B entonces
13      C <='1';
14   else                 -- sino
15      C <='0';
16 end if;
17 end process;
18 end caja;
```

Listado 1.13 Arquitectura funcional de un comparador de igualdad de 2 bits.

Nótese que la declaración de la entidad (**entity**) se encuentra descrita entre las líneas 4 y 7; en tanto que de las líneas 8 a la 18 se desarrolla el algoritmo (**architecture**) que describe el funcionamiento de la entidad. Para iniciar la declaración de la arquitectura (línea 8), es necesario definir un nombre arbitrario con el que esta declaración de la arquitectura pueda ser referenciada. En este caso, el nombre asignado es caja, además de incluir a la entidad con la cual está relacionada (funcional).

Por su parte, en la línea 9 se observa el inicio (**begin**) de la sección donde se comienzan a declarar los procesos que rigen el comportamiento del sistema.

Una declaración funcional utiliza una nueva sentencia denominada **process**, la cual se aprecia en la línea 10. La declaración del *proceso* (**process**) está acompañada de una lista sensitiva entre paréntesis (A,B), que hace referencia a las señales que determinan el funcionamiento del proceso —el cual debe entenderse como el hecho de que el algoritmo es sensible al cambio de valor de estas variables—. En la línea 11 inicia le ejecución del proceso mediante la palabra reservada **begin**.

Siguiendo con el análisis, se puede advertir que de la línea 12 a la 16 el proceso se ejecuta mediante la declaración del tipo **if-then-else** (si-entonces-sino). Esto se interpreta como sigue (línea 12): **si** el valor de la señal A es igual al valor de la señal B, **entonces** el valor de '1' se asigna a la variable C, **sino** se asigna a C el valor de cero '0'.

En la línea 16 se cierra el **if** de la línea 12. Entonces, el proceso se termina con la palabra reservada **end process** y la arquitectura se cierra de la forma acostumbrada **end caja**.

Como se puede observar, la *descripción funcional* se basa de manera principal en el uso de procesos y de declaraciones secuenciales, las cuales permiten de forma rápida el modelado de la función.

Con la finalidad de reforzar este tipo de programación, considérese la figura 1.30, en la cual se ha detallado la compuerta **OR**, cuya función de transferencia es bien conocida, además de que puede distinguirse en la tabla de verdad que se muestra en la figura 1.29. De esta manera, una descripción funcional sería de la forma siguiente:

a	b	f1
0	0	0
0	1	1
1	0	1
1	1	1

Figura 1.29

Figura 1.30 Función conocida de una compuerta OR.

```
1  -- Declaración funcional
2  library ieee;
3  use ieee.std_logic_1164.all;
4  entity com_or is
5  port (a,b: in std_logic;
6      f1: out std_logic);
7  end com_or;
8  architecture funcional of com_or is
9  begin
10 process (a,b) begin
11    if (a = '0' and b = '0') then
12        f1 <= '0';
13    else
14       f1 <= '1';
15 end if;
16 end process;
17 end funcional;
```

Listado 1.14

A partir del texto anterior se distingue cómo la compuerta **or** solo produce el valor cero f1 = 0 cuando a y b son iguales; en caso contrario, la salida es 1, f = 1 línea 14. Nótese que la línea 11 tiene una descripción muy similar al estilo flujo de datos.

Descripción estructural

Como su nombre lo indica, una **descripción estructural** basa su comportamiento en modelos lógicos ya establecidos (compuertas, sumadores, contadores, proyectos especiales, etc.). Es importante destacar que estas estructuras pueden ser diseñadas por el usuario y guardadas para su posterior utilización o extraídas de los paquetes contenidos en las librerías de diseño del software que se esté utilizando, como se verá más adelante.

En la figura 1.31 se muestra una representación esquemática de una entidad que cuenta con dos variables de entrada de 2 bits (a0, a1 y b0, b1). En su construcción se han empleado dos compuertas **nor** exclusivas y una compuerta **and**.

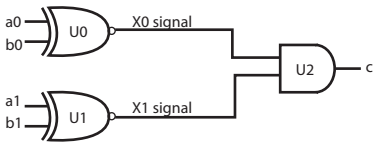


Figura 1.31 Estilo estructural.

Para ejemplificar, considérese que cada una de estas compuertas (modelo lógico) se encuentra dentro del paquete **gates**, del cual son extraídas para estructurar el diseño. Este paquete diseñado con anterioridad (mismo que se describirá con detalle en los próximos capítulos).

A fin de iniciar la programación de una entidad de manera estructural, es necesario la descomposición lógica del diseño en pequeños submódulos (jerarquizar), los cuales permiten analizar de manera práctica el circuito, ya que la función de entrada/salida es conocida (véase figura 1.32). Así, la indicación **xnor2** considera una compuerta **xnor** de 2 entradas.

Es importante resaltar que una **jerarquía** en VHDL se refiere al hecho de dividir en bloques y no a que un bloque tenga mayor peso que otro. Esta manera de fragmentar el problema hace de la descripción estructural una forma sencilla de programar, la cual —dentro del contexto del diseño lógico— es posible distinguir cuando se analiza por separado alguna sección de un sistema integral.

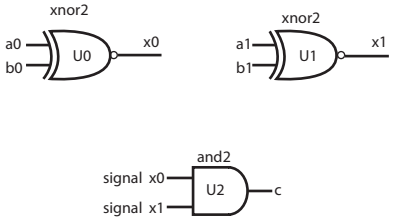


Figura 1.32 Descomposición en módulos conocidos.

De acuerdo con nuestro ejemplo, se conoce la función de salida de las dos compuertas **xnor**, por lo que al unirlas a la compuerta **and**, la salida **c** es solo el resultado de la operación **and** efectuada internamente a través de las señales "signal' **x0** y **x1**.

El siguiente listado muestra el código del programa de esta entidad, donde desde la línea 3 hasta la 6 se detalla la entidad denominada estructura, la cual considera dos variables de entrada de dos bits cada una, a (0:1) y b (0:1), como vector y la salida c individualmente.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity estructura is
4 port( a,b: in bit_vector (0 to 1);
5         c: out bit);
6 end estructura;

7 use work.gates.all;
8 architecture estructural of estructura is
```

```
 9 signal x: bit_vector (0 to 1);
10 begin

11   U0:      xnor2  port map (a(0), b(0), x(0));
12   U1:      xnor2  port map (a(1), b(1), x(1));
13   U2:      and2   port map (x(0), x(1), c);

14 end estructural;
```

Listado 1.15 Descripción estructural de un comparador de igualdad de 2 bits.

De modo que los componentes **xnor** y **and** no se declaran, debido a que se encuentran incluidos en el paquete **gatespkg**, el cual a su vez está contenido dentro de la *librería de trabajo* (**work**), en la línea 7.

En la línea 8 se inicia con la declaración de la arquitectura nombrada *estructural*.

En la línea 9 se declara **signal x**, que involucra las señales (x0 y x1), que se declaran dentro de la arquitectura y no en la entidad, debido a que no representan a una terminal (pin) y solo se usan para conectar bloques de manera interna a la entidad.

La conectividad de los bloques o submódulos se describe de la línea 11 a la 13. Cada compuerta se maneja como un bloque lógico independiente (componente) del diseño original, al cual se le asigna una variable temporal (U0, U1 y U2); en tanto, la salida de los bloques U0, U1 se maneja como una señal.

De forma que la compuerta **and2** recibe las dos señales provenientes de x (x0 y x1), ejecuta la operación y asigna el resultado a la salida c del circuito (línea 13).

```
13   U2:    and2     port  map    (x(0),   x(1),   c);
```

Ejemplo 1.9

Realizar el programa correspondiente para programar la entidad que se muestra en la figura 1.33, mediante el uso de una declaración estructural.

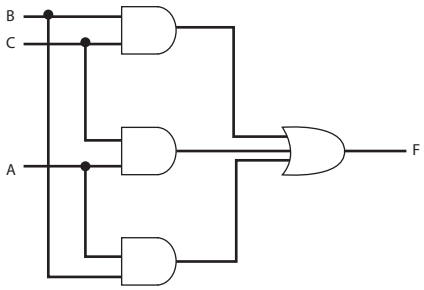


Figura 1.33

Solución

Como se observa, el circuito considera cuatro submódulos denominados:

- U0: and2 con entradas y salidas: B, C, signal x0
- U1: and2 con entradas y salidas: C, A, signal x1
- U2: and2 con entradas y salidas: A, B, signal x2
- U3: or3 con entradas y salidas: x0, x1, x2, F

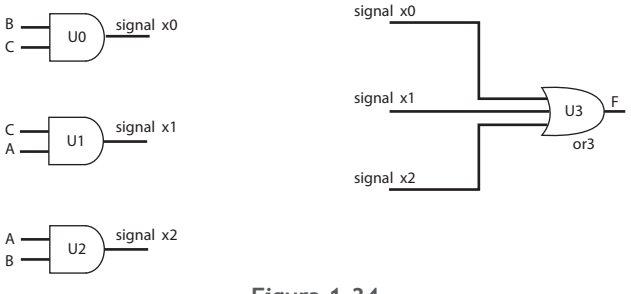


Figura 1.34

```
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3 entity comp is
 4 port( A,B,C : in std_logic;
 5           F: out std_logic);
 6 end comp;
 7 use work.gates.all;
 8 architecture estructura of comp is
 9 signal x: bit_vector (0 to 2);
10 begin
11 U0: and2 port map (B, C, x(0));
12 U1: and2 port map (C, A, x(1));
13 U2: and2 port map (A, B, x(2));
14 U3: or3 port map (x(0), x(1), x(2), F);
15 end estructura;
```

Listado 1.16

Es importante resaltar que en la línea 9 se ha declarado la señal x como un vector ascendente de tres bits: (x(0), x(1) y x(2)).

1.4 Comparación entre los estilos de diseño

El estilo de diseño utilizado en la programación depende de manera exclusiva del diseñador y de la complejidad del proyecto. Por ejemplo, un diseño puede describirse mediante ecuaciones booleanas, pero si este es muy extenso quizá sea más apropiado hacerlo a través de estructuras jerárquicas para dividirlo. Ahora bien, si se requiere diseñar un sistema cuyo funcionamiento dependa solo de sus entradas y sus salidas, es conveniente emplear la descripción funcional, la cual tiene la ventaja de requerir un número menor de instrucciones, además de que el diseñador no necesita un conocimiento previo de la función de cada componente que forma el circuito.

Ejercicios

1.1 ¿Qué es un PLD?

1.2 Determine el significado de los siguientes términos: GAL, CPLD y FPGA.

1.3 ¿La estructura de programación en VHDL requiere del uso de los módulos entidad y arquitectura? Describa la función de cada uno de ellos.

1.4 ¿Cuál es la función de utilizar punto y coma (;) en la estructura de un programa?

1.5 Mencione los tipos de modos válidos en VHDL.