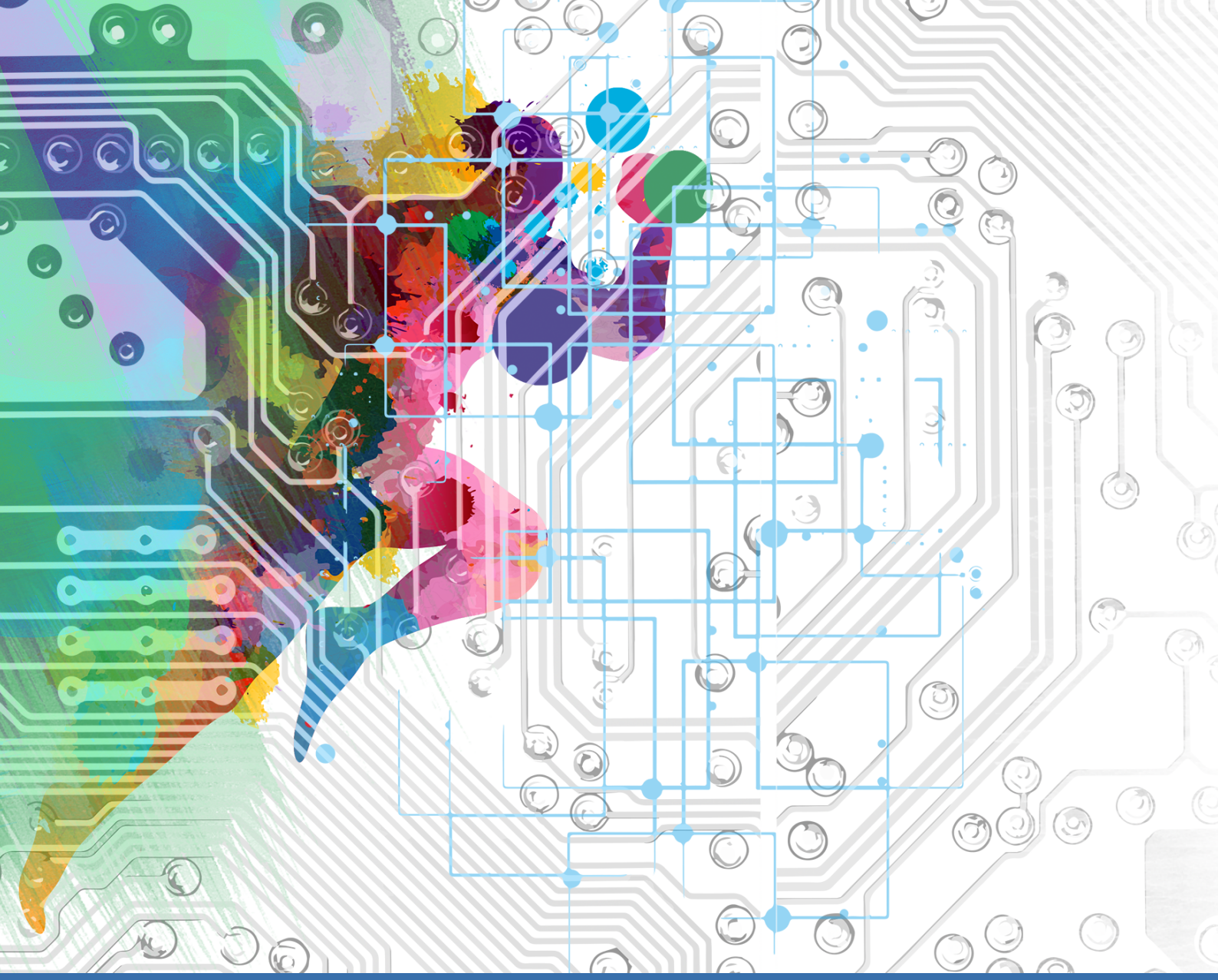


Jonathan Álvarez Ariza
Sergio González Gil

Lenguaje de descripción de *hardware* (VHDL)

UNIMINUTO
Corporación Universitaria Minuto de Dios
Educación de calidad al alcance de todos

UNIMINUTO
Corporación Universitaria Minuto de Dios
Educación de calidad al alcance de todos

Presidente del Consejo de Fundadores
P. Diego Jaramillo Cuartas, cjm

Rector General Corporación Universitaria Minuto de Dios - UNIMINUTO
P. Harold Castilla Devoz, cjm

Vicerrectora General Académica
Marelen Castillo Torres

Rector Sede Principal
Jefferson Enrique Arias Gómez

Vicerrectora Académica Sede Principal
Luz Alba Beltrán Agudelo

Directora General de Docencia y Desarrollo Curricular
Luz Nelly Romero Agudelo

Directora General de Publicaciones
Rocío del Pilar Montoya Chacón

Director de Investigación Sede Principal
P. Carlos Germán Juliao Vargas, cjm

Decano Facultad de Ingeniería
John Camilo Cifuentes Taborda

Director Tecnología en Electrónica
Carlos Arturo Giraldo Gutiérrez

Coordinadora de Publicaciones Sede Principal
Paula Liliana Santos Vargas

Álvarez Ariza, Jonathan
Lenguaje de descripción de hardware (VHDL) / Jonathan Álvarez Ariza, Sergio González Gil. Bogotá: Corporación Universitaria Minuto de Dios - UNIMINUTO. Facultad de Ingeniería UNIMINUTO, 2018.
ISBN digital: 978-958-763-287-3
133 p. il.
1.VHDL (lenguaje de computadores). – 2. Diseño lógico – Procesamiento de datos 3. Circuitos lógicos – Diseño y construcción 4. Circuitos lógicos –Métodos de simulación i.González Gil, Sergio

CDD: 621.392 A59l BRGH Registro Catálogo UNIMINUTO No. 91271

Archivo descargable en MARC a través del link: https://tinyurl.com/bib91271

LENGUAJE DE DESCRIPCIÓN DE HARDWARE (VHDL)

Autores
Jonathan Álvarez Ariza
Sergio González Gil

Corrección de estilo
Carlos Manuel Varon Castañeda

Diseño y diagramación
María Paula Berón Ramírez

Primera edición: digital 2018
ISBN digital 978-958-763-287-3

Calle 81 B # 72 B – 70, piso 8
(+57 1) 2916520, ext. 6012
Bogotá D.C. - Colombia
2018

Índice

Índice de figuras

Índice de códigos fuente

Índice de tablas

Introducción

En general, los lenguajes de alto nivel contienen un conjunto de instrucciones y librerías que hacen más fácil al diseñador estructurar un programa con una finalidad determinada; de igual manera —y a diferencia de los lenguajes de bajo nivel como Assembler—, contienen un intérprete o compilador que transforma las instrucciones en lenguaje de máquina (booleano), con el fin de generar una determinada salida o proceso.

El presente libro, que pretende ser una guía introductoria para estudiantes que deseen aprender sobre lenguaje de descripción de *hardware* (VHDL, por sus iniciales en inglés), consta de cuatro capítulos, dispuestos de la siguiente manera: en el primero, se introducen conceptos iniciales de VHDL, junto con la clasificación de los dispositivos lógicos programables, sus características y forma de construcción, es decir, se presenta una introducción de los dispositivos lógicos programables con sus diversas particularidades, abarcando desde los PLD hasta las FPGA.

En el segundo capítulo, se exponen los diversos fundamentos de VHDL (por ejemplo, la creación de variables y entidades, y el uso de sentencias); con este conjunto de temas, el lector puede aprender varios conceptos y generar entidades a través de VHDL, además de profundizar en los conceptos establecidos en el capítulo I respecto a temáticas como entidad, librerías, paquetes y arquitectura, todos ellos elementos importantes en el proceso de descripción de *hardware*.

En un tercer capítulo, a su turno, se muestran al lector las potencialidades de VHDL a través de un conjunto de aplicaciones prácticas, con el objetivo de evidenciar las diversas aplicaciones que puede tener VHDL en campos como las máquinas de estado y el manejo de dispositivos gráficos (tales como LCD), secuencias neumáticas o señales análogas; se ofrecen también varios ejemplos de creación de entidades para la generación de sistemas de lógica combinacional y secuencial.

El cuarto y último capítulo, presenta aplicaciones a través de VHDL, tales como manejo de LCD, secuencias neumáticas, termómetro y comunicación serial. Es preciso decir que, para la construcción de cada una de las entidades mostradas en el presente libro, se recomienda tener la tarjeta Basys 2; empero, esto no es un requisito en tanto es posible realizar los ejercicios y simularlos observando su comportamiento, y analizando el esquema general de construcción de *hardware* mediante lenguaje VHDL.

Adicionalmente, los códigos usados se han cargado en el repositorio GitHub, por lo cual se encuentran disponibles en https://github.com /VHDLUniminuto/VHDL. Si lo desea, el lector puede ir a este sitio para observar, descargar y modificar los códigos usados en el libro; asimismo, puede acceder a los vínculos suministrados de videos dispuestos en YouTube, en donde encontrará una explicación de los códigos realizados en el capítulo IV y su funcionamiento.

Por último, esperamos que esta obra sea de gran ayuda para quienes desean incorporarse en el mundo de VHDL y sus aplicaciones. Sabemos, de igual manera, que el camino de este aprendizaje es secuencial; por ello, animamos al lector a realizar, comprobar o modificar los ejercicios, a fin de conocer las particularidades de este lenguaje y lograr que este aprendizaje resulte significativo para su vida como estudiante, profesional o practicante en el área de la electrónica. Cabe anotar que todos los códigos mencionados y establecidos han sido comprobados y añadidos al repositorio mencionado.

Reiteramos, entonces, nuestro deseo de que este libro constituya un material de apoyo valioso, e impulse a sus lectores al aprendizaje de la misma manera en que su concepción nos ha motivado a aprender como autores.

Fundamentos de VHDL

Este capítulo presenta los conceptos principales que se emplean en el diseño digital a través del lenguaje VHDL. Primero, se abordarán las características más importantes de los dispositivos lógicos programables (PLD), en especial de los *Field Programmable Gate Array* (FPGA) y *Complex PLD* (CPLD); y luego se introducirán algunas nociones iniciales del lenguaje VHDL.

El concepto de dispositivo lógico programable

Una de las consideraciones iniciales en la contextualización del diseño de circuitos a través de VHDL corresponde a la definición de lo que se entiende como "circuito lógico programable". Estos dispositivos fueron desarrollados en la década de 1970 por compañías como IBM, en busca de sistemas digitales más robustos y eficientes en cuanto a su programación e implementación. La función de estos dispositivos radica en la implementación de una o más expresiones lógicas o booleanas, de manera que el diseñador puede modificar dicha función cuando lo desee. La estructura básica de un dispositivo lógico programable se establece en la figura 1.1.

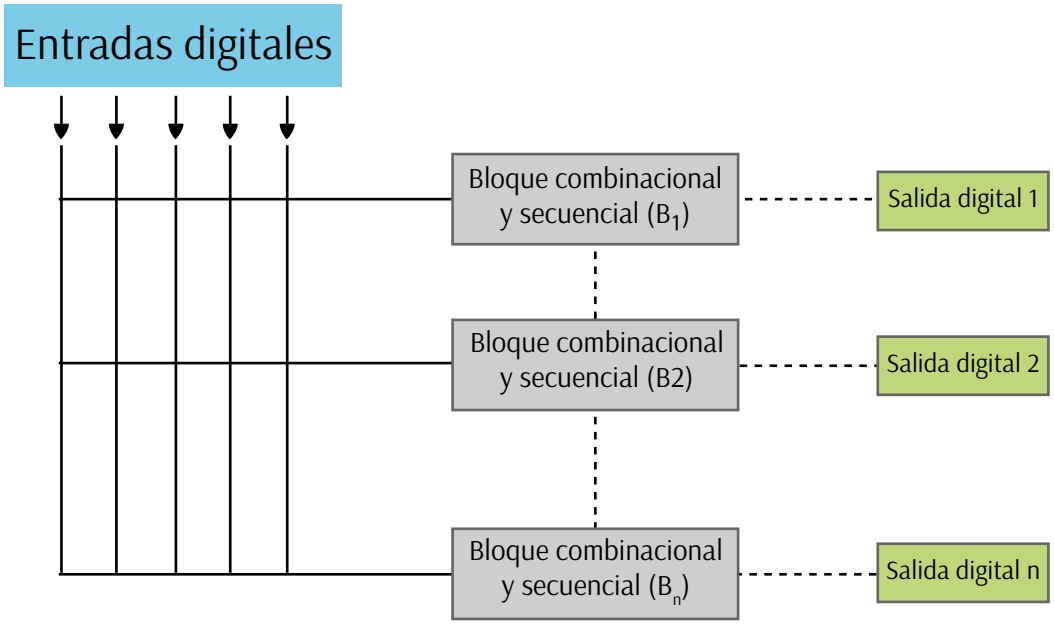


Figura 1.1. Esquema general de dispositivo lógico programable.
Fuente: elaboración propia.

Como se aprecia, existen tres componentes básicos que conforman un PLD:

- **Bloque combinacional y secuencial**: consta de un conjunto de elementos conocidos como *lookup tables* (LUT), las cuales pueden verse como pequeñas memorias de 2^n filas por 1 columna y bloques formados típicamente por compuertas lógicas (AND, OR, XOR) de varias entradas, junto con bloques secuenciales como *flip flops* tipo D. Este bloque tiene la función de implementar una o varias expresiones lógicas especificadas por el diseñador. La estructura de esta unidad se muestra en la figura 1.2.

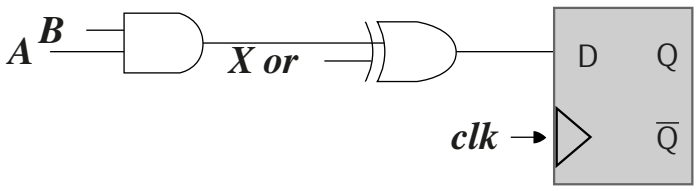


Figura 1.2. Esquema de bloque combinacional y secuencial de PLD.
Fuente: elaboración propia.

Cada bloque puede implementar una o más funciones lógicas en términos de suma de productos (SOP). Para este caso, las entradas A, B están interconectadas con los bloques de entrada y salida, y el *flip flop* tipo D permite transferir la salida de la función lógica hacia algún bloque de salida.

- **Bloques de entrada y salida**: estos elementos interconectan los pines de entrada y de salida con los bloques combinacionales y secuenciales mencionados anteriormente. Su estructura básica está conformada por un conjunto de multiplexores[1] y sumadores, con lo cual se permite la interconexión de pines físicos con las unidades lógicas. Las señales de los pines de entrada y salida, así como los pines de alimentación del PLD, están comprendidos en el rango de 0 a 3.3V para tecnologías CMOS, o en los de 0-1.8V, 0-2.5V y 0-3.3V para LVCMOS[2].

- **Bloque de interconexiones (matriz de interconexiones)**: estos elementos permiten conectar las entradas y salidas del PLD con las unidades secuenciales y combinacionales, con el fin de establecer la función indicada en el diseño. Esta matriz puede visualizarse como un conjunto de conmutadores que realizan las conexiones entre los bloques indicados anteriormente. El resultado de este proceso, conocido como *ruteo*, es el eje central de la implementación de *hardware*.

1 Un multiplexor es un dispositivo que permite seleccionar una determinada señal, transfiriéndola hacia su salida. Un ejemplo de circuito integrado con esta funcionalidad es el integrado 74151.

2 Las tecnologías *Complementary Metal Oxid Semiconductor* (CMOS) y *Low Voltage CMOS* (LVCMOS) permiten la utilización de un menor voltaje de funcionamiento, mayor frecuencia de trabajo del dispositivo y una mayor escala de integración; es decir, el número de transistores dentro de un PLD aumenta. Estas tecnologías reemplazaron a la tradicional TTL, realizada con transistores BJT y un voltaje de operación de 5V. Debe resaltarse que estas tecnologías son construidas a partir de transistores FET. Para más información, véase Baker, J. (2010). *CMOS: Circuit Design, Layout, and Simulation* (3 ed.). Hoboken, Nueva Jersey: John Wiley and Sons, Institute of Electrical and Electronics Engineers – IEEE.

En función del avance de estas tecnologías, el número de bloques combinacionales o secuenciales aumenta de forma gradual. Debe resaltarse que, en la actualidad, estos dispositivos son reprogramables. Una perspectiva de los principales dispositivos programables en la actualidad se mostrará en la siguiente sección.

Principales dispositivos lógicos programables (CPLD y FPGA)

En esta sección se explicitan las características principales de los dispositivos PLD que se usan actualmente, conocidos como CPLD y FPGA, cuyas particularidades los hacen idóneos para la implementación de *hardware* a pequeña y mediana escalas. Así, se abordarán, generalidades y nociones básicas respecto de estos dispositivos; y después se expondrán algunos métodos de construcción de los mismos de forma general, junto con su evolución.

CPLD (Complex PLD)

Un CPLD es un dispositivo basado en una matriz general de interconexión, en conjunto con bloques combinacionales y secuenciales, los cuales aumentan en número frente al PLD convencional (Toronto, 2000). Algunos ejemplos de esta familia son los dispositivos referencia XC2C256 de la empresa Xilinx , o bien la familia MAX II de la empresa Altera[3]. La matriz de interruptores es basada en transistores FET —que pueden ser vistos como interruptores—, los cuales permiten interconectar las diversas unidades lógicas con los bloques de entrada y salida: también se los denomina conmutadores de conexión EEPROM. Las

3 Para más información sobre estas empresas y los productos CPLD y FPGA disponibles, véanse las direcciones web http://http://www.xilinx.com, http://www.altera.com y http://www.latticesemi.com

figuras 1.3 y 1.4 presentan un esquema general de la estructura de un CPLD y el esquema de conexión de los conmutadores EEPROM, respectivamente.

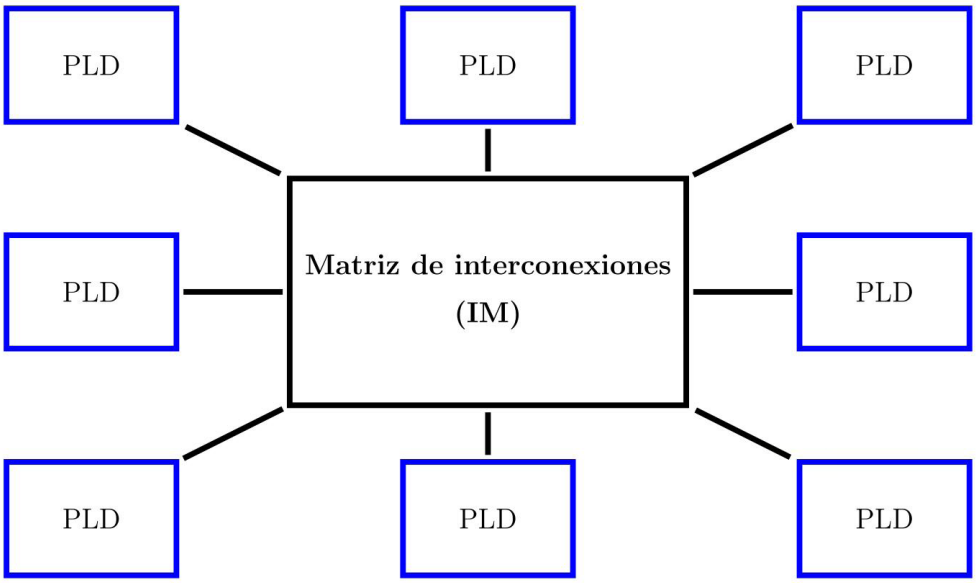


Figura 1.3. Estructura de configuración de PLD.
Fuente: elaboración propia.

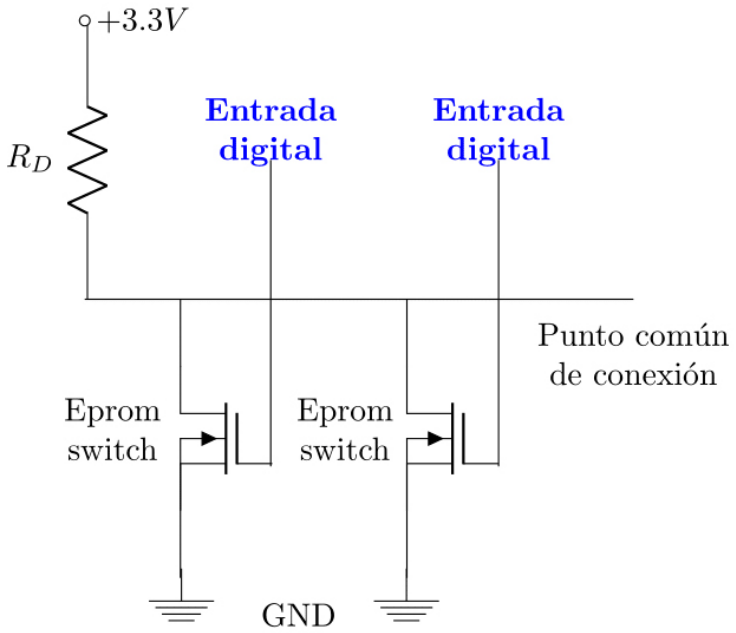


Figura 1.4. Esquema de interconexión de conmutadores EPROM.
Fuente: elaboración propia.

Como ejemplo de los conceptos mencionados, considérese el esquema de conexión de la familia de CPLD Xilinx XC9500 CoolRunner II (Xilinx inc, 2002) que se muestra en la figura 1.5.

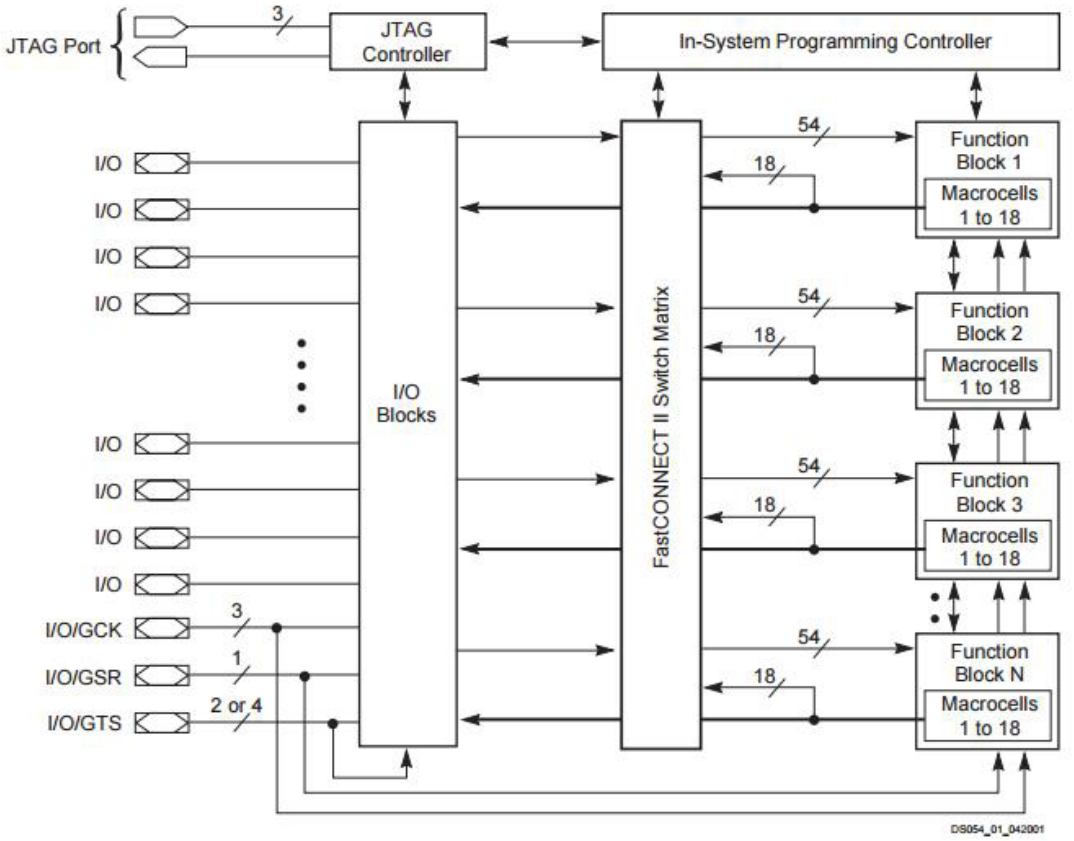


Figura 1.5. Configuración de familia de CPLD XC9500 (Xilinx).
Fuente: imagen tomada de http://www1.cs.columbia.edu/~sedwards/classes/2006/4840/ds054.pdf.

Nótese en el caso de la figura 1.5 que el CPLD tiene el conjunto de elementos que se han mencionado anteriormente: en este caso, la *matriz de interconexiones* es denominada *FastCONNECT II switch matrix*; y los bloques de entrada y salida (*I/O blocks*) se interconectan con los bloques funcionales PLA (*function blocks*), como se indicó en la figura 1.3. La matriz de interconexiones crea los puentes entre diferentes unidades lógicas (PLA), creando un *retraso de propagación* fijo entre estas unidades debido a la resistencia y capacitancia establecida en la conexión de las mismas. Las conexiones entre las

entradas y salidas son realizadas a través de conmutadores como los establecidos en la figura 1.4. Las conexiones pueden crearse entre las entradas, salidas y unidades lógicas, en función de las especificaciones indicadas en el diseño de un sistema lógico combinacional o secuencial. Una consideración de los dispositivos lógicos programables, y en especial, de las unidades CPLD y FPGA, es que la función realizada por el diseñador puede asignarse a cualquier pin de entrada o de salida, lo cual permite usar la totalidad del dispositivo para la implementación de *hardware*. Para la programación, el dispositivo CPLD contiene un puerto especial que funciona mediante protocolo JTAG (*IEEE std 1149.1-IEEE std 1149.1a*)[4], del cual se hablará en secciones posteriores; por ahora, cabe indicar que este protocolo permite asignar al dispositivo CPLD la entidad generada mediante lenguaje VHDL, a fin de crear las interconexiones necesarias que permitan la implementación del *hardware* diseñado: este proceso se denomina *place and route*.

FPGA *(Field Programmable Gate Array)*

Este dispositivo programable difiere en su construcción con respecto a los CPLD: en este caso, la matriz fija cambia a un número mayor de conexiones entre las unidades lógicas, reemplazando la matriz de interconexiones que tienen los CPLD. El proceso de ruteo o unión de las unidades lógicas puede cambiar. Esto es, en un proceso de este tipo se pueden unir los elementos a través de un camino A; y en otro, a través de un camino B: ello genera un retraso de propagación variable. De igual manera, el número de compuertas lógicas usadas aumenta en comparación con los CPLD.

4 Para más información técnica sobre este protocolo consulte la página web: http://grouper.ieee.org/groups/1149/1/

Una FPGA está formada por tres unidades básicas: bloque de lógica combinacional, bloque secuencial (conformado por *flip flops*[5] que forman registros) y el bloque de interconexión de entradas y salidas. En la figura 1.6 se establece el esquema de construcción de una FPGA de acuerdo al esquema suministrado por (Farooq U., 2012), mientras que la figura 1.7 presenta la estructura de un bloque lógico conocido como *Configurable Logic Block* (CLB) (Haskell, 2009) o *Adaptative Logic Module* (ALM) (Altera, Adaptive Logic Module (ALM), 2016).

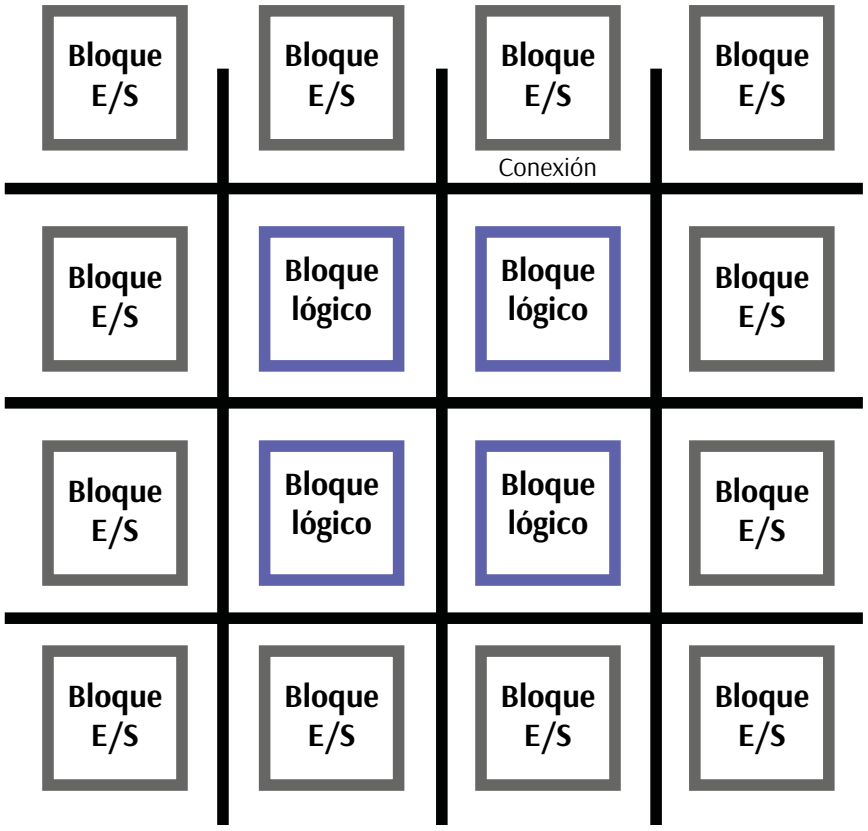


Figura 1.6. Esquema de construcción de FPGA.
Fuente: elaboración propia.

5 Un *flip flop* es la unidad básica de memoria RAM, conformada principalmente por compuertas lógicas AND o NOR. El estado de esta unidad cambia en función de la señal de reloj aplicada y de las entradas que tenga el *flip flop* en un momento dado. Para más información sobre esta terminología, véase Wakerly, J. (2001). *Diseño digital: principios y prácticas*. México: Pearson Educación.

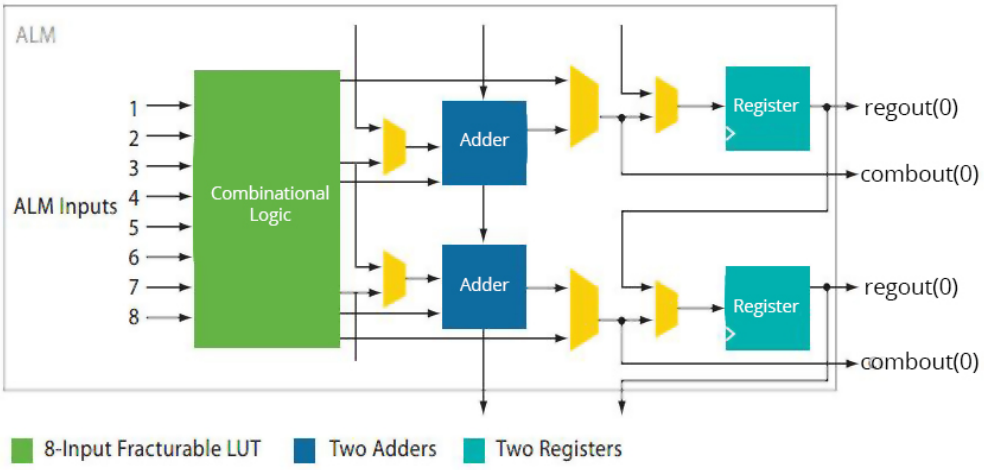


Figura 1.7. Esquema de construcción de FPGA Altera.
Fuente: imagen tomada de https://www.altera.com/en_US/pdfs/literature/wp/wp-01003.pdf

Nótese que, en el caso de la figura anterior, la matriz de interconexiones fija desaparece y, en su lugar, se tiene una conexión entre bloques de Entrada y de Salida (E/S) y bloques lógicos.

Como se ha mencionado, el ruteo o conexión entre unidades lógicas puede hacerse tomando dos caminos, ya que existen dos opciones para llegar al mismo punto: de ahí que el retraso de propagación (*T*)[6] en los dispositivos FPGA sea variable. Debe observarse que los bloques lógicos están constituidos a partir de elementos de lógica combinacional, tales como compuertas lógicas AND, OR, XOR, NOT y elementos de lógica secuencial (*flip flops)*, por ejemplo.

Una diferencia general entre los dispositivos FPGA y CPLD radica en que, en los primeros, las conexiones no se realizan a través de conmutadores EEPROM como los establecidos en la figura 1.3, sino a través de unidades conocidas como *static RAM* (SRAM). Considérese como ejemplo de lo mostrado la figura 1.8, en donde

6 El retraso de propagación es un tiempo en el cual una señal digital "viaja" a través de las conexiones establecidas entre unidades lógicas y los bloques de entrada y salida. Este tiempo depende de la capacitancia y resistencia que hay en estas conexiones o caminos, en el caso de los dispositivos FPGA este tiempo no es fijo debido a que el proceso de ruteo (*route*) es aleatorio, de esta forma el camino de conexión puede tener una *extensión variable*.

se muestra la estructura de un bloque lógico y su interconexión para la familia de FPGA *VIRTEX IV* de Xilinx Inc. (Xilinx inc, 2008); este bloque se denomina *slice*.

Debe tomarse en cuenta que, en función de la empresa fabricante de estos dispositivos, los nombres de las unidades lógicas cambian: por ejemplo, en el caso de la empresa Altera, las unidades o bloques lógicos son conocidos como *Adaptative Logic Module* (ALM) (Altera, Adaptive Logic Module (ALM), 2016); mientras que para la empresa Xilinx son *slices,* como se dijo. A este respecto, y como sugerencia al lector, es importante entender el concepto de construcción de los dispositivos FPGA para que el cambio de nombres entre empresas fabricantes no represente un problema al comprender el funcionamiento de los mismos.

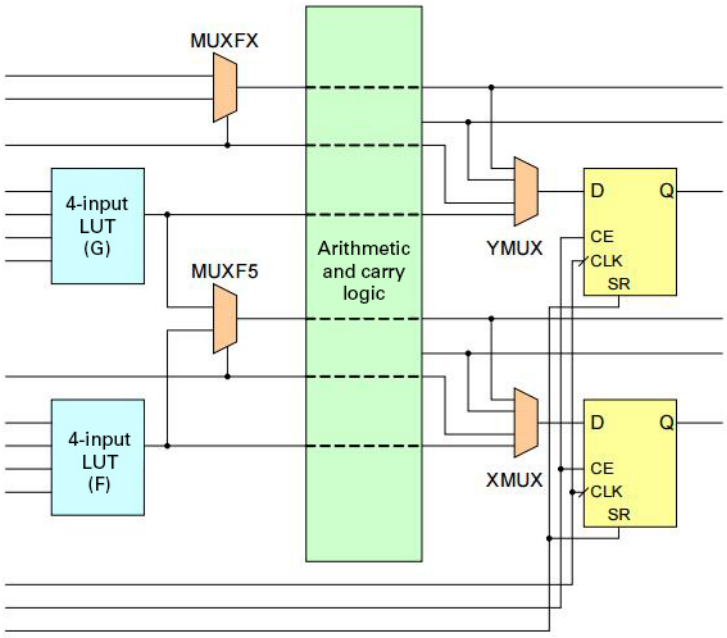


Figura 1.8. Esquema de construcción de bloque lógico para FPGA Virtex-4 de Xilinx, denominado Slice.
Fuente: imagen tomada de http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

Para este caso, las unidades lógicas están distribuidas en los bloques *Lookup table* (LUT), junto con unidades combinacionales, secuenciales y multiplexores; estos últimos permiten seleccionar la señal que

se genera en una determinada conexión o camino para ser transferida hacia los bloques en entrada y salida. Las conexiones punteadas se realizan a través de bloques RAM y registros de desplazamiento[7].

Esquema físico de construcción de CPLD y FPGA

En la actualidad, existen dos tipos de tecnologías en la construcción de CPLD y FPGA: *Dual Layer Metal* (DLM) y *Triple Layer Metal* (TLM) (Altera, CPLDs vs FPGAs Comparing High-Capacity Programmable Logic, 1998). Estas basan su acción en la forma más eficiente de crear las conexiones realizadas a través de terminaciones metálicas, las cuales pueden ser unidas a través del uso de pocos transistores (tales como los que se muestran en la figura 1.3). Las figuras 1.9 y 1.10 presentan esquemas de construcción TLM.

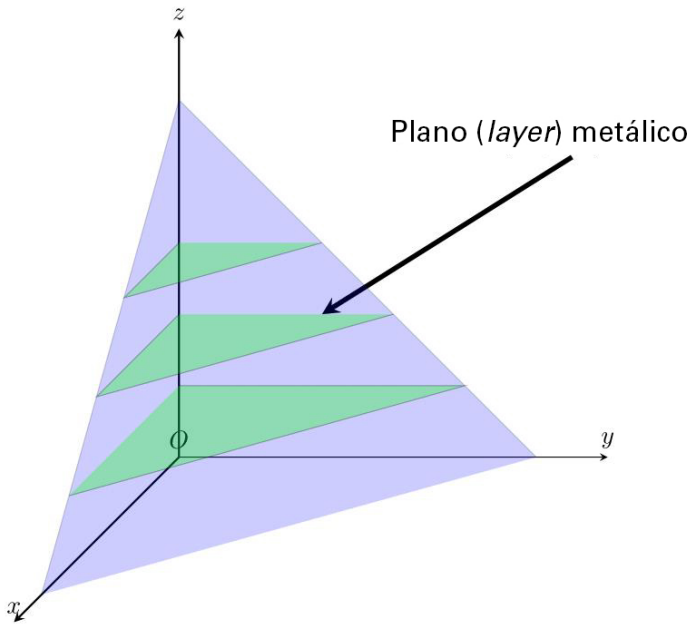


Figura 1.9. Esquemas de construcción TLM para FPGA.
Fuente: elaboración propia.

7 Debido a que las señales digitales deben ser "almacenadas" durante un tiempo para procesarse de acuerdo a las funciones lógicas *sintetizadas*, los resultados de estas últimas deben transferirse hacia los bloques de salida. Para este fin, las FPGA disponen de bloques de memoria RAM y registros de desplazamiento.

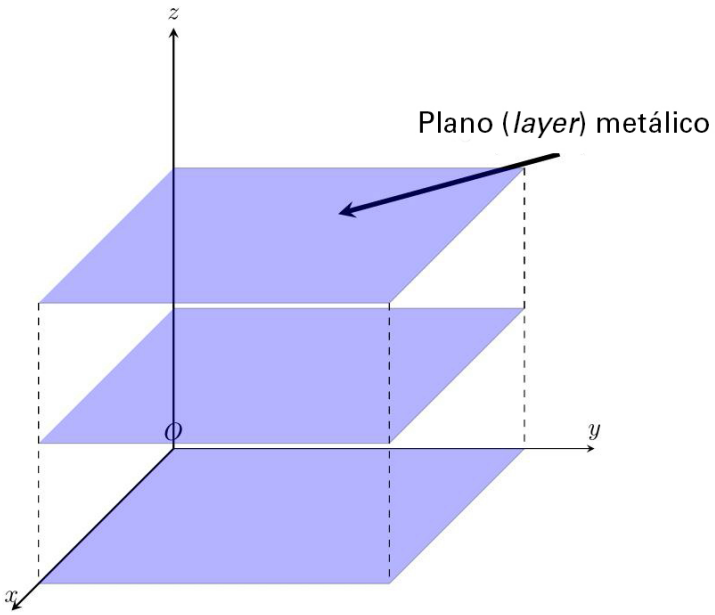


Figura 1.10. Esquema de construcción TLM para CPLD (derecha).
Fuente: elaboración propia.

Nótese que en los esquemas anteriores se muestran dos tipos de construcción: uno para FPGA (piramidal) con un espesor de capas metálicas cercanas a las 0,6 micras; y una estructura rectangular para CPLD. Las tecnologías actuales de estos dispositivos permiten una escala de trabajo cercana a los 28 nm. Estas escalas permiten una mayor eficiencia, debido a que se pueden colocar dispositivos (transistores) en proporción a estas distancias en nm. Adicionalmente, este esquema de diseño optimiza la utilización de las capas de metal (*layers*) de una mejor forma, con lo cual se reducen los costos implicados en la construcción de los dispositivos mencionados. Cabe mencionar que, en la actualidad, las FPGA solo se construyen en estructura piramidal: esta es una restricción propia de estos dispositivos y de la forma de unión entre unidades lógicas. Dicha tecnología es un avance frente a la DLM, la cual basa su funcionamiento en dos capas metálicas (figura 1.11).

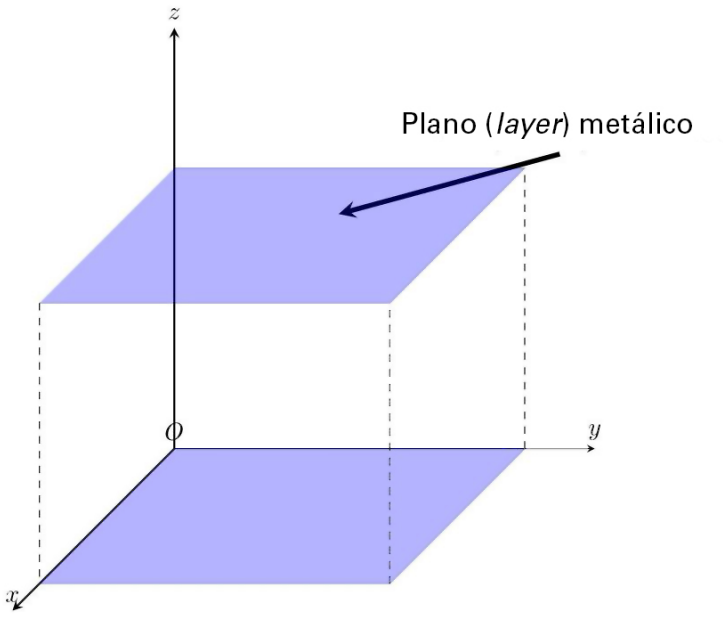


Figura 1.11. Esquema de construcción DLM: Dual Layer Metal.
Fuente: elaboración propia.

Como comparación práctica entre ambas tecnologías, tómense los esquemas presentados en las figuras 1.9, 1.10 y 1.11 y sus áreas (planos metálicos) en la construcción de dos edificios: si bien la altura de las edificaciones sería la misma, con TLM podrían construirse tres pisos, mientras que con DLM solo sería posible construir dos; por lo tanto, el uso de TLM optimizaría recursos en el primer diseño, con lo cual se lograría una mayor asignación de espacio para la misma tarea. El proceso mencionado reduce costos de fabricación, debido a que se puede utilizar una mayor proporción de estos planos metálicos. Así, este tipo de tecnologías, este tipo de tecnologías lidera el mercado de la clase de dispositivos estudiada. En este sentido, se ha establecido que en años futuros se presentará una reducción en la escala de construcción, comprendida entre 20 y 16 nm. En la próxima sección se hará una presentación sintética respecto de los fabricantes que lideran el mercado de estos dispositivos.

Principales fabricantes de dispositivos CPLD y FPGA

En la actualidad, tres empresas lideran el mercado de CPLD y FPGA: Xilinx Inc., Altera Corporation y Lattice Semiconductor Corporation; y las dos primeras son pioneras en la creación de dispositivos lógicos programables (por ejemplo, Xilinx es la creadora de los dispositivos FPGA). Estas empresas disponen de un conjunto de productos, cada uno con su entorno de desarrollo. Este último concepto, por su parte, se remite al conjunto de herramientas de *software* que permiten sintetizar la *entidad*[8] o estructura de *hardware* en los dispositivos mencionados: para el caso particular del presente libro, el entorno de desarrollo será el *software* WebPACK ISE v 14.7 de Xilinx, cuya operación se discutirá en apartados posteriores. El listado de entornos de desarrollo de acuerdo al fabricante se enuncia a continuación:

- Xilinx: entorno WebPACK ISE.
- Altera: entorno Quartus[9].
- Lattice Semiconductor: IspLever Classic[10].

A continuación, se relacionan las principales familias de CPLD y FPGA establecidas por las empresas mencionadas.

8 El concepto de entidad será discutido más adelante.

9 La configuración de este software puede encontrarse en el siguiente recurso web: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/tt/tt_my_first_fpga.pdf

10 La configuración de este software puede encontrarse en el siguiente recurso web: http://www.latticesemi.com/~/media/LatticeSemi/Documents/Tutorials/LZ/LSEforClassicTutorial20.pdf?document_id=51068

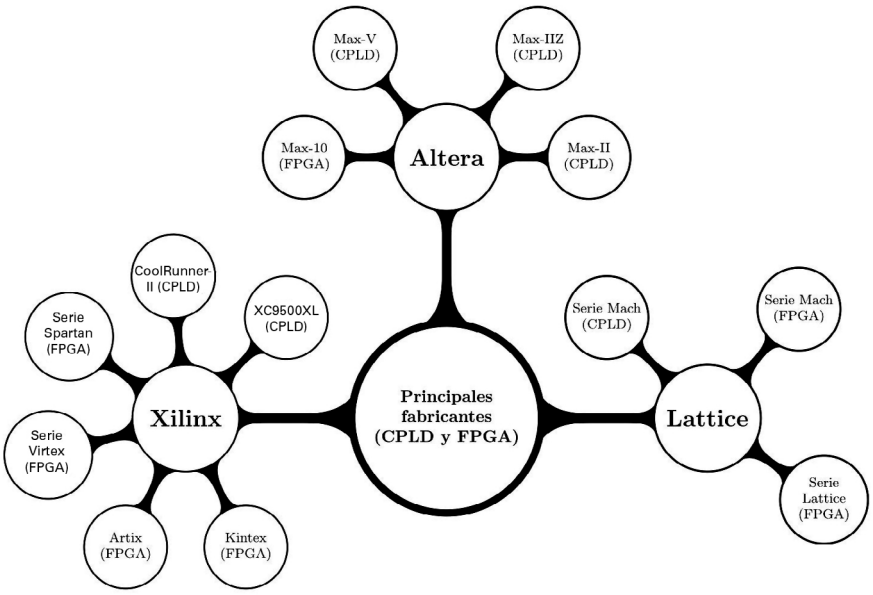


Figura 1.12. Principales fabricantes de dispositivos CPLD y FPGA.
Fuente: elaboración propia.

Nociones iniciales sobre lenguaje VHDL

VHDL (VHSIC *Hardware Description Language*) es un lenguaje orientado a la descripción del comportamiento de *hardware* de un sistema electrónico, es decir, está orientado al *diseño y la implementación de hardware.* Este lenguaje, estructurado a partir de la década de 1980 por el Departamento de Defensa de los Estados Unidos, se ha conocido con nombres como VHDL-87 o VHDL-93, por ejemplo, y luego fue normalizado por el Institute of Electrical and Electronics Engineers (IEEE)[11] mediante los estándares IEEE 1164 y 1076[12]. Debido a que

11 IEEE es una asociación mundial de ingenieros sin ánimo de lucro dedicados a la investigación, estandarización de productos y realización normas técnicas, en las áreas de electrónica y eléctrica.

12 El estándar IEEE 1076 establece los lineamientos de referencia del lenguaje VHDL. Para consultar este manual, véase la página web http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=893288&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%-2F7180%2F19335%2F00893288

VHDL está orientado hacia el *hardware* (Bhasker, 1991), tiene algunas restricciones particulares y no puede ser entendido solo como un lenguaje de computador, cuyos recursos son mayores que los de una FPGA o CPLD. Como cualquier lenguaje, VHDL tiene un conjunto de normas de sintaxis, operadores, declaración de variables, bucles etc., los cuales deben seguirse para implementar una función lógica determinada en los dispositivos programables mencionados.

Uno de los primeros conceptos que debe entenderse en el desarrollo de *hardware* con VHDL es el denominado como ***entidad*** (Pedroni, 2004): esta puede verse como la estructura del *hardware*, por lo cual describe sus entradas y salidas. La declaración de la entidad es el primer paso que debe hacerse en la elaboración de cualquier *hardware* que se desee implementar o simular mediante lenguaje VHDL. El esquema general de la entidad se presenta en la figura 1.13.

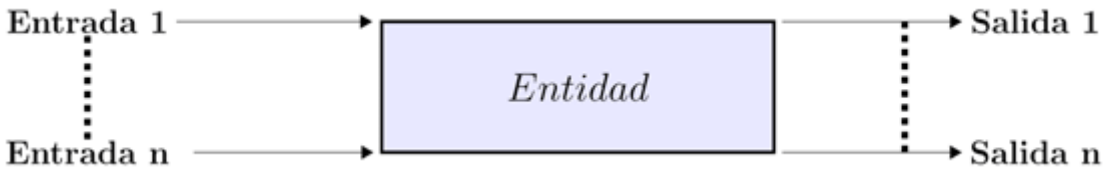
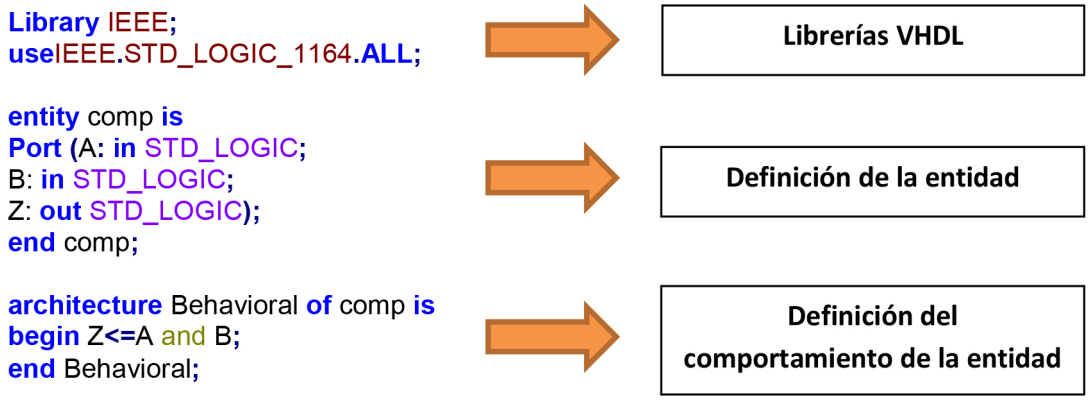


Figura 1.13. Esquema general de una entidad o bloque de hardware.
Fuente: elaboración propia.

Una vez que el concepto de entidad ha sido establecido, corresponde revisar la noción de arquitectura: en esta se determina el *comportamiento* de la entidad, definida previamente. Este comportamiento, a su turno, se establece a través de lenguaje VHDL; si bien en un primer momento esto se logra mediante la definición de funciones lógicas, no es esa la única forma. La estructura de un programa en VHDL tendrá la forma general que se explicita en el código fuente 1; en este ejemplo se evidencia que un programa en VHDL tiene declaraciones de librerías, entidad y arquitectura.

Código fuente 1. Definición de entidad y su comportamiento a través de VHDL.



Fuente: elaboración propia.

El lenguaje VHDL dispone de un conjunto de operaciones lógicas con las cuales es posible definir el comportamiento de una entidad —*not, and, or, xor, xnor, nand, nor*—, las cuales son operaciones generales que se encuentran dentro de una función booleana.

Las expresiones booleanas que típicamente se implementan en los PLD son de tipo suma de productos (SOP). Bajo tal estándar, una función lógica puede tener la siguiente apariencia:

$$f(x,y,z,)=x'yz + xy'z + xyz.$$

Nótese, además, que en la primera parte del código fuente 1 se especifica un conjunto de librerías: estas hacen posible que el lenguaje funcione, además de permitir operaciones como conversiones numéricas y lógicas. En el diseño a realizar, una o varias librerías deben ser incluidas para que la entidad tenga un correcto funcionamiento. Por defecto, la librería general que establece el funcionamiento de VDHL con su sintaxis, variables y operadores, es *IEEE_STD_LOGIC_1164*; esta se incluye por defecto dentro de la plantilla inicial creada en el entorno de desarrollo del PLD (CPLD o FPGA).

Cada entorno de desarrollo (IDE) permite la definición de cualquier entidad y su arquitectura bajo lenguaje VHDL, y está diseñado

para utilizarse únicamente con las familias o referencias de un fabricante específico. Es decir, aunque el lenguaje VHDL sea universal, el proceso de *síntesis y ruteo* de un PLD difiere entre fabricantes.

En la creación e implementación de cualquier entidad en VHDL se establecen una serie de pasos que el diseñador debe tomar en cuenta, los cuales se mencionan a continuación:

1. Codificación (*VHDL coding*).
2. Simulación (*simulation*).
3. Síntesis (*synthesising*).
4. Localización y ruteo (*place and route*).

En el proceso de codificación se describe el comportamiento de la *entidad* a través de lenguaje VHDL, como se ha mencionado. Una vez completado lo anterior, se comprueba si la descripción tiene algún tipo de error de sintaxis. Posteriormente, el diseñador procede con la simulación de la entidad realizada: esta se realiza a través un programa denominado *test bench,* generado también a través de lenguaje VHDL. Si el proceso de simulación está acorde a las especificaciones de diseño de *hardware* establecidas, se avanza al proceso de *síntesis*, en el cual el programa establecido en VHDL se transforma en términos del dispositivo físico (CPLD o FPGA). Este proceso valida si la entidad se ajusta a las restricciones (*constraints*) de macroceldas o bloques lógicos que puede tener una familia de dispositivos *lógicos programables.* Si el diseño cumple con estas restricciones, se continúa con el proceso de localización y ruteo en el cual, por ejemplo, se establecen las conexiones entre unidades lógicas y bloques de entrada y salida, en el caso de las FPGA. Los procesos de síntesis, localización y ruteo difieren entre fabricantes y se ajustan a las técnicas de optimización que estos hayan desarrollado para tal fin; igualmente, estos son los procesos que más tardan tiempo en el desarrollo e implementación de una entidad.

Como se dijo, el programa de simulación recibe el nombre de *test bench*. Este permite realizar la simulación de una entidad como la descrita en el código establecido anteriormente, bajo diferentes combinaciones para las entradas y periodos de tiempo. Este archivo es creado a partir del mismo lenguaje VHDL, definiéndose las combinaciones que se desean para probar la entidad en un momento específico. Una vez creado el programa, el mismo se sintetiza y depura a través del simulador, con lo cual se puede tener un marco global del funcionamiento de dicha entidad ante diversos tipos de entradas. La figura 1.14 muestra un ejemplo de la simulación a través de un *test bench*.

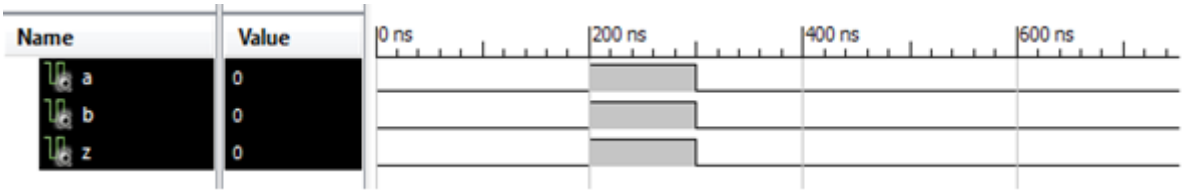


Figura 1.14. Ejemplo de diagrama de tiempos generado a través de test bench.
Fuente: elaboración propia.

La creación de un archivo de simulación *test bench* se ciñe a la siguiente secuencia de pasos generales:

1. Creación del programa mediante código VHDL.
2. Definición de las entradas y salidas, mediante las combinaciones que definen el comportamiento de la entidad en proceso de realización.
3. Síntesis del archivo creado.
4. Simulación del archivo mediante el programa de simulación respectivo. En el caso de los dispositivos FPGA y CPLD de Xilinx Inc., la simulación se realiza mediante el software ISIM.

Como ejemplo de los conceptos establecidos, considérese el siguiente *test bench* para la entidad mencionada en el código fuente 1 y la figura 1.14 (compuerta AND), el cual se describe a continuación:

Código fuente 2. Definición de test bench para entidad establecida en el código fuente 1.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY testb IS
END testb;

ARCHITECTURE behavior OF testb IS


COMPONENT CompuertasB
PORT(
        A :IN std_logic;
        B :IN std_logic;
        Z :OUT std_logic
);
ENDCOMPONENT;


--Inputs
signal A :std_logic:= '0';
signal B :std_logic:= '0';

            --Outputs
signal Z :std_logic;
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name

constant periodo :time:=100 ns;

BEGIN

            -- Instantiate the Unit Under Test (UUT)
uut: CompuertasB PORTMAP(
        A => A,
        B => B,
        Z => Z
);

-- Clock process definitions
   stim_proceso :process
begin
```

Librerías VHDL

Entrada y salida de la entidad. En este caso, A y B son las entradas, y Z es la salidad para la entidad compuerta AND.

Retraso entre operaciones (en este caso ns).

```
        wait for periodo;
        A<='0';
        B<='0';

         wait for periodo;
         A<='0';
         B<='1';
         wait for periodo;
         A<='1';
         B<='0';
         wait for periodo;
         A<='1';
         B<='1';
         wait for periodo;
         --Z<=A and B;
         wait for periodo;

         wait; --Detener el proceso de simulación
end process;

        -- Stimulus process

END;
```

Fuente: elaboración propia.

Nótese que, en este caso, existe una serie de sentencias que *definen* el comportamiento del archivo *test bench*. Al iniciar, se define la entidad para este caso (entradas y salidas). Luego de esto, se define la sentencia **signal**: a este respecto, una señal (**signal)** es un punto de interconexión *interna* del dispositivo lógico programable, es decir, un camino que une varios elementos *dentro* de la FPGA o CPLD.

Pasado lo anterior, se define el periodo de tiempo destinado al cambio de las entradas que tienen lugar dentro de la entidad; esta sentencia se establece mediante la instrucción **constant** periodo: time**: =**100 ns (para este caso, el periodo de actualización de la entidad es de 100 ns). Luego, se establece la *combinación de las entradas* de la entidad: en el caso que nos ocupa, se han definido las combinaciones para las entradas *A, B,* de tal forma que se obtiene una perspectiva general del comportamiento de la entidad

ante dichas entradas a través del *diagrama de tiempos*[13] mostrado en la figura 1.14, producto del *test bench*. Dicho diagrama da una aproximación al comportamiento real de la entidad o *Unit Under Test* (UUT), que para este caso es una compuerta *and*.

Herramientas de desarrollo

Como se expresó en la presentación de esta obra, el desarrollo de los ejemplos del presente libro requiere algunas herramientas a nivel de *hardware* y *software,* tales como la *tarjeta Basys 2* y el aplicativo WebPACK ISE de Xilinx Inc. Se debe mencionar que, para el desarrollo completo de las temáticas en el presente libro, se debe al menos disponer del software de desarrollo WebPACK ISE, en el cual se *codificarán, simularán y sintetizarán* las entidades creadas en VHDL. La tarjeta Basys 2, a su turno, permite implementar las entidades creadas mediante los anteriores pasos, con lo cual se comprueba y depura su funcionamiento, y se optimiza el proceso de aprendizaje de quien desee realizar los ejercicios. Con lo anterior, en esta sección se sintetizan las características principales de las herramientas de *software* y *hardware* mencionadas aquí.

Tarjeta Basys 2

Basys 2 (Digilent, 2016) es una tarjeta de desarrollo que permite implementar y depurar las diversas entidades creadas mediante lenguaje VHDL. Las particularidades de esta tarjeta son mostradas en la figura 1.15.

13 Un diagrama de tiempos es una figura que ilustra las señales de entrada y de salida digitales que conforman, en este caso, una entidad en el proceso de simulación. El diagrama de tiempos contiene las entradas en función de las combinaciones dadas en el *test bench*, así como las salidas producto de las funciones lógicas establecidas en la entidad.

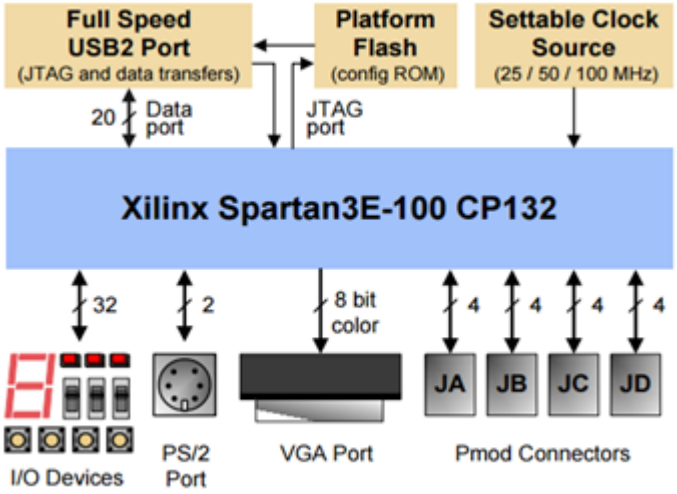


Figura 1.15. Esquema general de construcción tarjeta Basys 2 de Digilent inc.
Fuente: imagen tomada de https://reference.digilentinc.com/_media/reference/programmable-logic/basys-2/basys2_rm.pdf

La tarjeta dispone de una FPGA Xilinx Spartan III (Xilinx inc, 2013) que contiene 100.000 macroceldas[14], suficientes para la implementación de las entidades propuestas en el libro. Además de este núcleo, se encuentra un conjunto de periféricos: VGA, PS/2, conectores PMOD (para conexión con motores, LCD y *drivers*), pulsadores y *display,* entre otros.

La tarjeta se comunica con el Computador Personal (PC) a través de un conector USB con el protocolo *Join Test Action Group* (JTAG) (IEEE std 1149.1-IEEE std 1149.1a); este último transmite la información necesaria para que se establezcan las conexiones necesarias entre los bloques de entrada – salida y los bloques lógicos mencionados (CLB o ALM), con lo cual se implementa la entidad generada de manera eficiente. La figura 1.16 ilustra los componentes de *hardware* usados en la programación de la tarjeta en cuestión.

14 El término macroceldas se refiere al número de *unidades lógicas* (*compuertas lógicas y flip flops*) que contiene una FPGA para este caso la Spartan III.

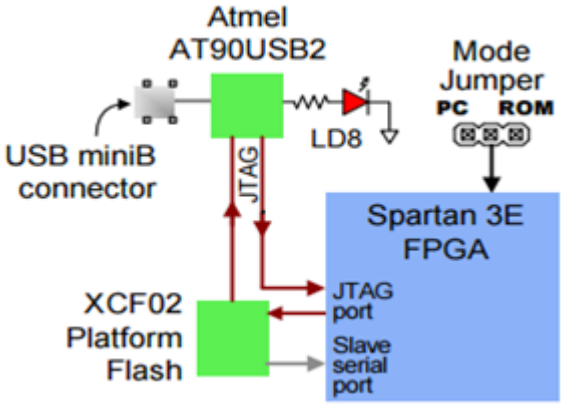


Figura 1.16. Esquema general de la estructura de programación de la tarjeta Basys 2.
Fuente: imagen tomada de https://reference.digilentinc.com/_media/reference/programmable-logic/basys-2/basys2_rm.pdf

Sumado a lo anterior, la tarjeta maneja un nivel de voltaje general de 3.3 V, por lo cual se trata de un sistema *CMOS*[15]; esta consideración debe tomarse en cuenta especialmente en el manejo de señales de entrada y salida mediante los conectores PMOD mencionados. Durante el desarrollo del libro —y como uso general para acoplar señales de diferente valor de voltaje—, se utilizarán intercambiadores de nivel de 3,3 V (CMOS) a 5 V (TTL) como los integrados 74HC243, 74HC244, 74HC245. Con esto se logrará un acondicionamiento de las señales generadas en la tarjeta Basys hacia otro tipo de dispositivos, tales como *Liquid Crystal Display* (LCD) y conversores análogo-digital (ADC) y USB-UART TTL para comunicaciones seriales, entre otros. Adicionalmente, la compañía Digilent ofrece el *software* de programación de esta tarjeta, denominado *Adept*[16]. En la figura 1.17 se muestra la estructura de la misma.

15 *Complementary Oxid Semiconductor* (CMOS) es un tipo de tecnología basada en transistores FET, cuyo voltaje de operación se sitúa entre los 1,6 y 3,3 V. CMOS reemplazó a la tecnología tradicional *Transistor-Transistor-Logic* (TTL), cuyo voltaje de operación es de 5 V.

16 Este software puede ser descargado desde la siguiente dirección web: http://store.digilentinc.com/digilent-adept-2-download-only/

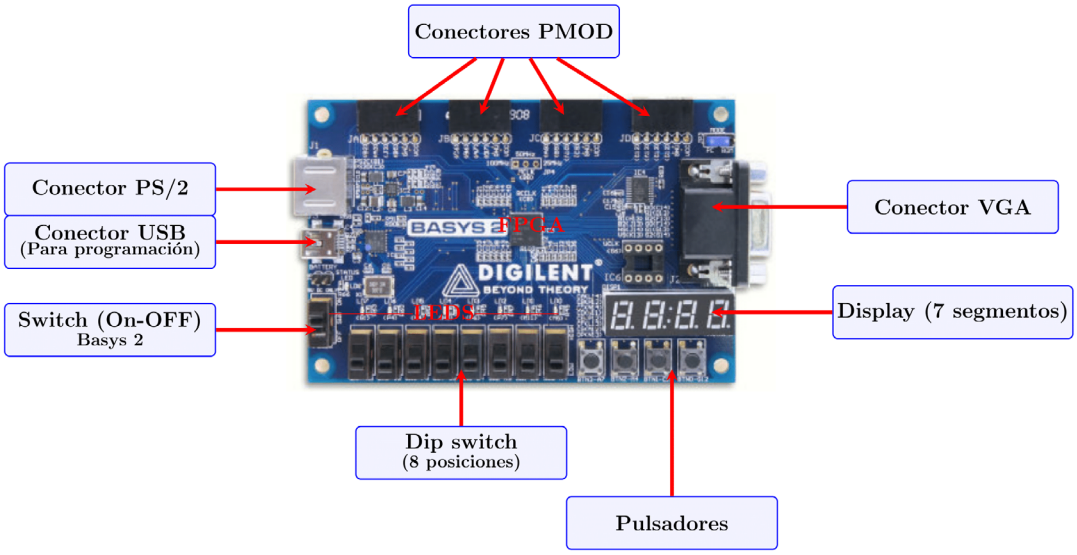


Figura 1.17. Vista general y componentes principales de la tarjeta Basys 2.
Fuente: elaboración propia.

Cada vez que se realiza una entidad con el *software* de desarrollo WebPACK ISE, se genera un archivo ejecutable para FPGA o CPLD con *extensión. bit.* Este último se carga en la FPGA a través del *software* Adept, que además de permitir la programación de la tarjeta ofrece al programador una alternativa para la realización de chequeos de la misma. Cabe anotar, además, que la tarjeta es alimentada y programada a través del puerto USB, al tiempo que es posible conectarla a una batería externa (mediante el conector *Battery* localizado en la misma).

La tarjeta contiene su propio oscilador interno para la generación de las diversas *entidades* que requieran procesos *secuenciales*. Este tiene una frecuencia de trabajo por defecto de 50 MHz, la cual debe tenerse en cuenta en la creación de los diferentes retrasos que se emplean en la creación de diversos programas con VHDL. Si se requiere un oscilador de diferente frecuencia de trabajo, existe una opción para cambiarla mediante un *jumper*, con el cual es posible seleccionar tres valores: 25, 50 o 100 MHz. De otro lado, si fueren necesarias frecuencias de menor valor, la tarjeta dispone de un *socket* especializado para ello, el cual se muestra en la figura 1.18.

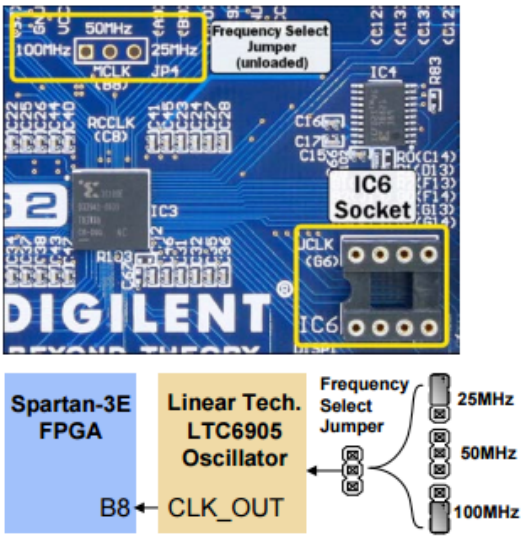


Figura 1.18. Esquema general de osciladores permitidos por la tarjeta Basys 2. Parte superior: configuración de oscilador interno; parte inferior: socket para oscilador externo tipo CMOS.
Fuente: imagen tomada de https://reference.digilentinc.com/_media/reference/programmable-logic/basys-2/basys2_rm.pdf

Otro periférico importante de la tarjeta Basys 2 es el conjunto de *displays* de la misma, los cuales permiten visualizar información alfanumérica. En este sentido, considérese el siguiente esquema general:

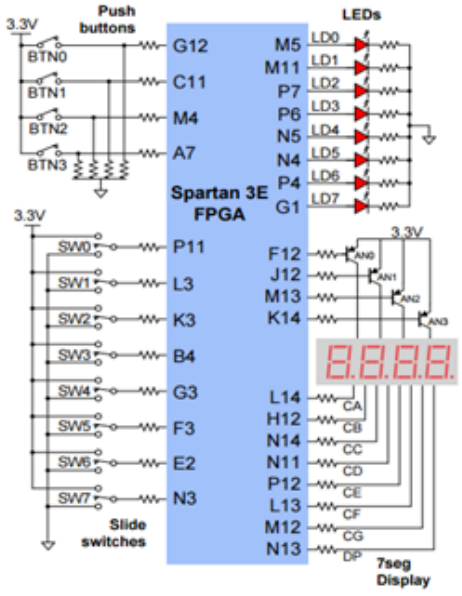


Figura 1.19. Esquema general Displays 7 segmentos y leds para Tarjeta Basys 2.
Fuente: imagen tomada de https://reference.digilentinc.com/_media/reference/programmable-logic/basys-2/basys2_rm.pdf

Nótese que los segmentos denominados CA, CB, CC, CE, CF, CG y DP están puenteados entre sí; además, la activación de un segmento se da a través de un cero lógico (*display* ánodo común). De esta forma, la activación individual de un *display* se da a través de los pines AN0, AN1, AN2 y AN3, con un valor lógico de 0. Cabe resaltar que el proceso de visualización de información debe establecerse a través de *visualización dinámica*: con este, la información es enviada de forma individual a cada *display* en un periodo de tiempo de aproximadamente 8 ms, lo cual permite que los datos se muestren de forma simultánea en todos ellos. Sumado a lo anterior, otros periféricos usados en la tarjeta Basys 2 son los módulos PS2[17] y VGA[18], nombrados anteriormente[19].

Entorno de desarrollo WebPACK ISE

WebPACK ISE[20] (Xilinx Inc, 2012) es un entorno de desarrollo para los productos (FPGA y CPLD) de la empresa Xilinx Inc.[21], por medio del cual es posible crear e implementar entidades a través de VHDL. Cada una de las entidades realizadas en el presente libro se ha desarrollado con la versión más reciente de este *software* (14.7). En la figura 1.20 puede apreciarse la apariencia típica del área de trabajo de esta aplicación.

17 Para más información sobre el protocolo PS2, véase http://www.student.montefiore.ulg.ac.be/~pierards/spe/documentation/PS2-VGA.pdf.

18 Para más información sobre el módulo VGA, véase http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf.

19 En este sentido, esperaríamos que esta obra permita a sus lectores aprender a manejar dichos periféricos a través de la construcción de entidades para tal fin, de acuerdo a las especificaciones de un diseño de hardware determinado.

20 En este sentido, esperaríamos que esta obra permita a sus lectores aprender a manejar dichos periféricos a través de la construcción de entidades para tal fin, de acuerdo a las especificaciones de un diseño de hardware determinado.

21 Existe una versión de evaluación gratuita de WebPACK, la cual dispone de todas las características necesarias para seguir los ejercicios de aprendizaje de lenguaje VHDL propuestos en esta obra.

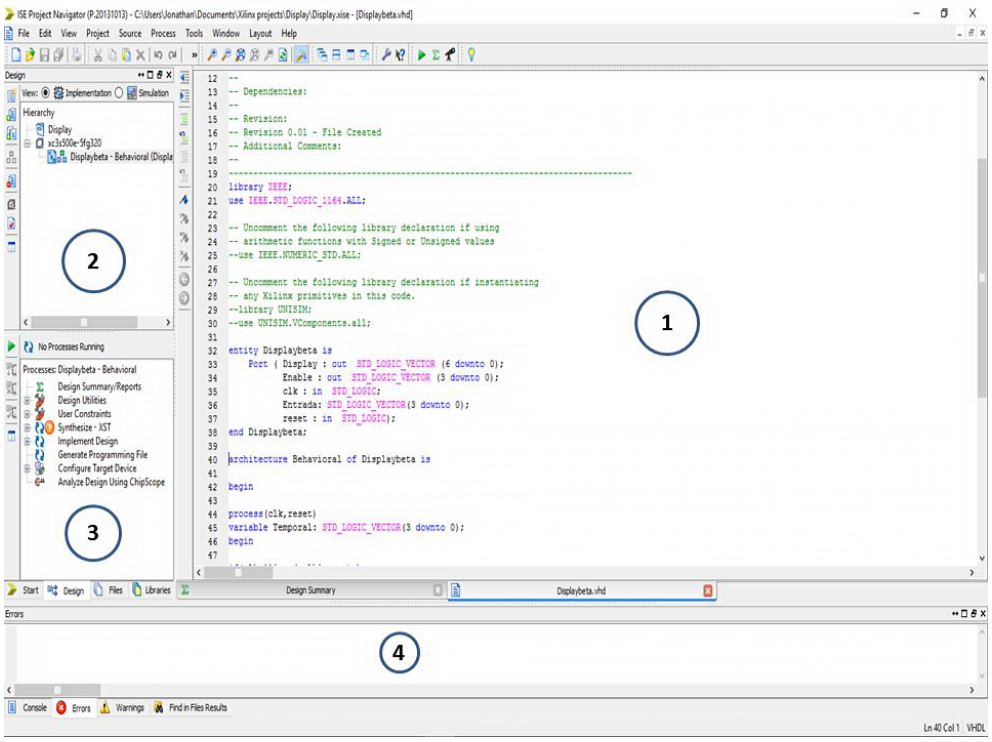


Figura 1.20. Vista general del software WebPACK ISE.
Fuente: elaboración propia.

A continuación, se describirán con detalle las áreas de trabajo numeradas que se muestran en la figura anterior:

1. *Área de programación en VHDL*: en esta área, el diseñador realizará las entidades y describirá su comportamiento a través de lenguaje VHDL. Como se discutirá en el capítulo II, el lenguaje VHDL contiene algunas palabras reservadas que, por lo general, aparecen en color azul o rosa y tienen un significado funcional en este lenguaje. WebPACK ISE establece esta convención para ayudar a disminuir posibles errores que el diseñador pueda tener en relación con estas palabras al construir la entidad. En este sentido, al crear cualquier proyecto, WebPACK ISE establece una plantilla inicial con las librerías que se necesitan inicialmente en el desarrollo de cada entidad.
2. *Área de configuración de dispositivo y visualización de entidad (implementación y simulación)*: en esta área se encontrarán dos

elementos generales. De un lado, se presenta una perspectiva del código realizado en VHDL a través de un dispositivo lógico programable (CPLD o FPGA); en este sentido es importante que, cada vez que se cree un proyecto, se verifique el tipo de dispositivo a ser programado, porque de lo contrario la implementación de la entidad será errónea. Y de otro lado, se encuentra en esta área la visualización de la simulación: en dicha ventana, el diseñador podrá realizar su *test bench* y simularlo para generar el diagrama de tiempos correspondiente a través del *software* de simulación de WebPACK ISE, llamado ISIM, como se dijo anteriormente.

3. *Área de síntesis, asignación de restricciones de usuario e implementación:* en dicho espacio se podrá iniciar el proceso de síntesis de la entidad. Igualmente, se podrán definir los pines de salida y entrada en función con la descripción de la tarjeta Basys 2, discutida anteriormente. La tarjeta contiene una serie de direcciones las cuales están asociadas a las direcciones de la FPGA o CPLD escogido. Por ejemplo, si se observa la figura 1.19, existen algunas direcciones (G12, C11…) que están relacionadas con la FPGA (Xilinx Spartan 3E-100); en el *software*, estas direcciones son establecidas para el proceso de localización y ruteo. Este proceso de asignación en el *software* es conocido como *user constraints*, debido a que al autor establece restricciones en los pines de entrada y salida para el diseño, y el proceso de ruteo y localización optimiza la entidad generada para cumplir con estas especificaciones.
4. *Área de comandos:* en esta área, el usuario podrá visualizar errores, advertencias y el flujo de programa establecido por el entorno de desarrollo en los procesos de codificación, simulación, síntesis e implementación.

Los anteriores son los elementos principales del *software* WebPACK ISE, con cuya estructura es posible familiarizarse mediante la creación de proyectos simples.

Conceptos

En el presente capítulo se brindará una descripción generalizada de los elementos que conforman la estructura del lenguaje VHDL. Sobre este es importante anotar que, en tanto lenguaje de descripción de *hardware*, permite la realización de circuitos de tipos síncrono y asíncrono[22]. Asimismo, sirve como herramienta de especificación de proyectos por su carácter portable y reutilizable, puesto que es independiente de la tecnología usada o del fabricante. Los circuitos descritos en VHDL pueden ser simulados (haciendo uso de ciertos programas diseñados para ello), con el fin de visualizar de forma aproximada la respuesta de los mismos ante una combinación posible de entradas.

Arquitectura, entidad, librería

El lenguaje VHDL está estructurado en unidades, llamadas entidad; arquitectura de una entidad; configuración; declaración de paquete; y cuerpo del paquete (Ashenden, 1990), (Pedroni, 2004). Las tres primeras son básicas para la realización del diseño, mientras que las dos últimas se emplean cuando se desea generar librerías. A continuación, se definen cuatro términos básicos para comprender lo anterior:

- Entidad (*entity*): define la vista externa del modelo. Indica *qué* es el diseño.

22 El concepto de *síncrono* se refiere si el circuito o sistema requiere una señal de reloj para procesar las señales de entrada y generar una determinada salida en función del estado que tenga el sistema en un tiempo establecido t. Los *Flip Flops* o los contadores binarios son ejemplos de sistemas síncronos. Por el contrario, lo sistemas asíncronos no requieren señal de reloj para procesar las entradas del sistema y generar una salida determinada por el diseñador del sistema o circuito.

- Arquitectura (*arquitecture*): define la funcionalidad del modelo. Indica *cómo* es el diseño.

- Paquete (*package*): es un elemento conformado por un conjunto de subprogramas, definición de constantes o tipo de definiciones. Cada paquete consta de dos partes, a saber, *declaración* y *cuerpo*: la primera define la parte visible; mientras que el segundo define la parte solo visible en la programación. Para ilustrar este concepto, considérese el siguiente código en VHDL.

Código fuente 3. Ejemplo de estructura de un paquete realizado mediante VHDL.

```
package ejemplo is
    function maximo(constant a,b: integer) return integer is
      variable num: integer; -- variable local
         begin
           if num > b then
              num := a;  -- a es el máximo
           else

             num := b;  -- b es el minimo
          end if;
          return num;  -- retornar valor máximo
       end;
end ejemplo;
```

Fuente: elaboración propia.

En este caso, el paquete contiene un subprograma llamado "máximo", el cual retorna el máximo de dos números (A, B). Este subprograma puede ser usado en otros códigos, con lo cual se ahorra tiempo en la definición de rutinas en el comportamiento de la entidad a través de la reutilización de paquetes.

- Librería (*library*): listado de ficheros cuyos elementos componen los circuitos.

Es necesario destacar que, para realizar un circuito a través de VHDL, es necesario declarar tanto la entidad como la arquitectura.

Entidad

La entidad es al *hardware* lo que un símbolo esquemático a los diagramas electrónicos, en tanto estos últimos muestran solo las entradas y las salidas previamente definidas o declaradas; una entidad, entonces, puede verse como la estructura del *hardware*, que describe sus entradas y salidas (Haskell, 2009).

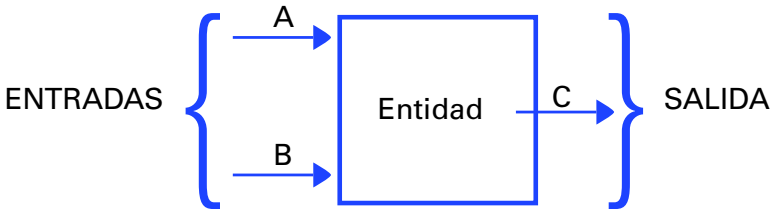


Figura 2.1. Diagrama de entidad en VHDL.
Fuente: elaboración propia.

Los puertos (*ports*) son canales de comunicación que constan de un nombre, el cual será único dentro de la entidad; pueden declararse como entrada (*in*), salida (*out*), entrada-salida (*in-out*) o *buffer*[23]. Los puertos declarados permiten que las señales de entrada, por lo general digitales, se conecten a la entidad; los de salida permiten asignar señales generadas por la entidad; y los de entrada y salida permiten que los datos sean bidireccionales (es decir, que sirvan como entrada o salida). Un ejemplo de entidad mediante VHDL se muestra a continuación.

```
entity nombre_archivo is
Port (A : in STD_LOGIC;
      B: in STD_LOGIC;
      C: out STD_LOGIC);
end nombre_archivo;
```

Modo

Tipo

Figura 2.2. Ejemplo de entidad a través de lenguaje VHDL.
Fuente: elaboración propia.

23 Este tipo de declaración se usa cuando se quiere realimentar una determinada entrada en un diseño de *hardware* establecido.

En el caso anterior, el modo establece si los puertos son declarados como *entrada, salida o entrada-salida*. El tipo se asocia al hecho de que este puerto es bit, std_logic o std_logic_vector, entre otros. Los tipos de datos *permitidos* son los posibles valores que pueden tener los *datos* y los *objetos* en VHDL (Gómez-Pulido, 2012); se describen a continuación.

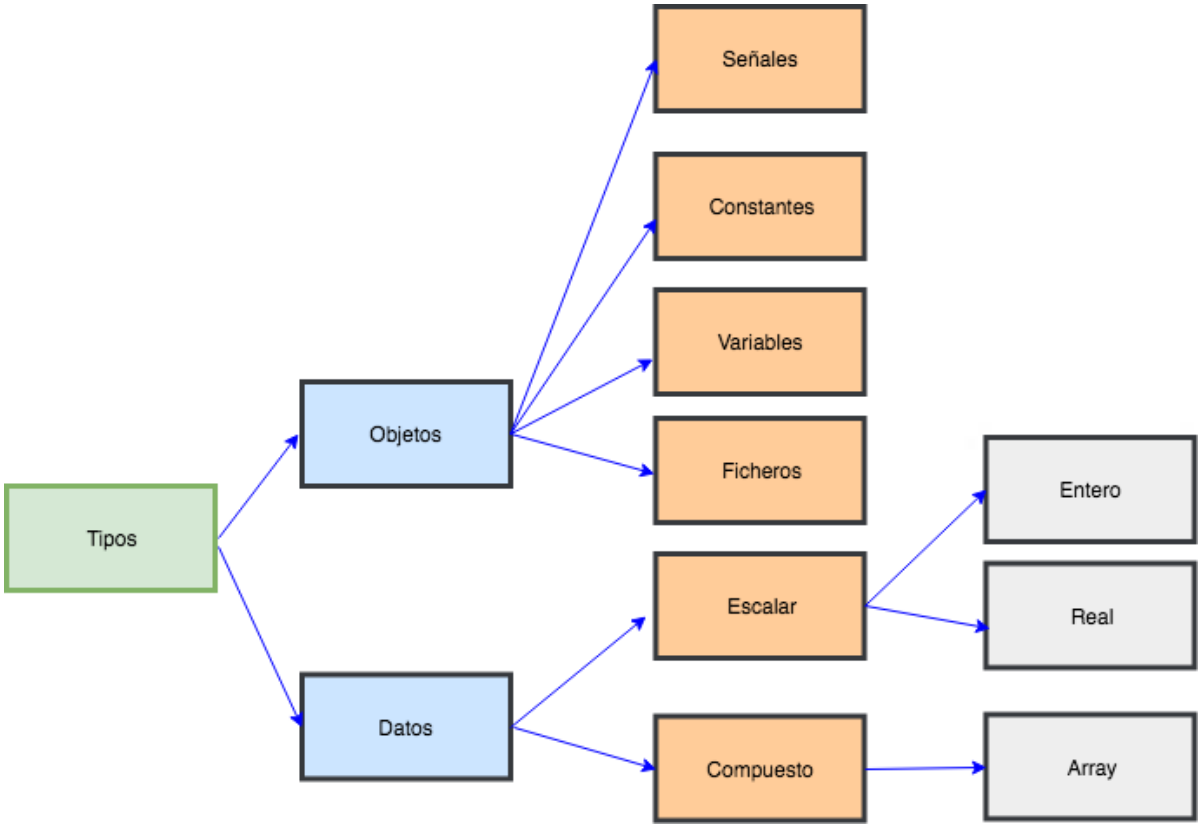


Figura 2.3. Tipos de datos en VHDL.
Fuente: elaboración propia.

Datos

Se consideran datos los posibles valores que se pueden establecer para los puertos de entrada y salida. Estos valores presentan clasificaciones diversas, dentro de las cuales se encuentra aquella que los divide en escalares y compuestos.

Escalares

Este tipo de datos corresponde a los números formados por una sola unidad indivisible, que tienen asociado el conjunto de funciones matemáticas como suma y resta, entre otras. De modo particular, la clase *integer* corresponde a un valor entero codificado de 32 bits.

Tabla 2.1. Tipos de datos escalares.

Tipo	Clase	Ejemplo
Estándar	Bit	´0´, ´1´
Estándar	Boolean	False, True
Estándar	Integer	Range n to m, range m down to n
Estándar	Positive	Integer = 1
Estándar	Natural	Integer = 0
Estándar	Enumerados	Type colors is (white, red, blue)
IEEE.std.logic.1164	std_logic	'U', 'X', '0', 'W', 'L', 'H', 'Z'

Fuente: elaboración propia.

Considérense al respecto los siguientes ejemplos:

```
r :in BIT; -- r es declarado como una señal de un digito tipo bit.
b :in STD_LOGIC; -- b es declarado como señal de un digito (escalar).
DataPort: OUT std_logic_vector (7 downto 0); --DataPort es declarado como un vector de 8 bits.
```

Compuestos

Los datos de este tipo están formados por elementos de otros tipos, lo cual hace posible crear un elemento nuevo. Entre ellos se encuentran vector/matriz y registro. El primero está formado por elementos del mismo tipo —por ejemplo, std_logic o bit—; mientras que el segundo puede componerse de elementos diferentes. Esta clase de datos se declara como "*array*", en tanto objeto de datos que consiste en una "colección" de elementos. La sintaxis de declaración se muestra a continuación:

type nombre is array (rango) of tipo;

La asignación de un valor a una posición del *array* se realiza mediante números enteros.

Objetos

Los objetos en VHDL son elementos que tienen asignado un valor de un tipo determinado; en esta categoría se agrupan los elementos mostrados en la figura 2.3. Las señales, constantes o variables pueden declararse solo en ciertas partes o regiones específicas de un programa en VHDL, las cuales se describen a continuación.

Tabla 2.2. Descripción de objetos y lugar de declaración en un código VHDL.

Señales (*signals*)	**Son declaradas en la *arquitectura* de la entidad. Adicionalmente, las señales se declaran con los tipos, los cuales están relacionados en la tabla 2.1. Los valores de las señales son asignados con el operador "<=".** **Ejemplo:** **signal Se: std_logic;** **Se<='1';**
Variables (*variables*)	Son declaradas dentro de un *proceso* o *subprograma*. Para asignar un valor a una variable, se usa el operador ":=". Ejemplo: **variable** counter: integer **range** 0 **to** 100; counter:=10;
Constantes (*constants*)	Pueden declararse en cualquier región o parte del programa. Su valor no puede ser cambiado una vez se ha inicializado. Ejemplo: **constant** constante: std_logic_vector(2 **downto** 0):= "011";

Fuente: elaboración propia.

Operadores predefinidos y expresiones: VHDL utiliza símbolos especiales que tienen funciones diferentes y específicas, los cuales se utilizan para representar algún tipo de operación aritmética, lógica o de comparación. A continuación, se relacionan estos operadores con su respectiva funcionalidad.

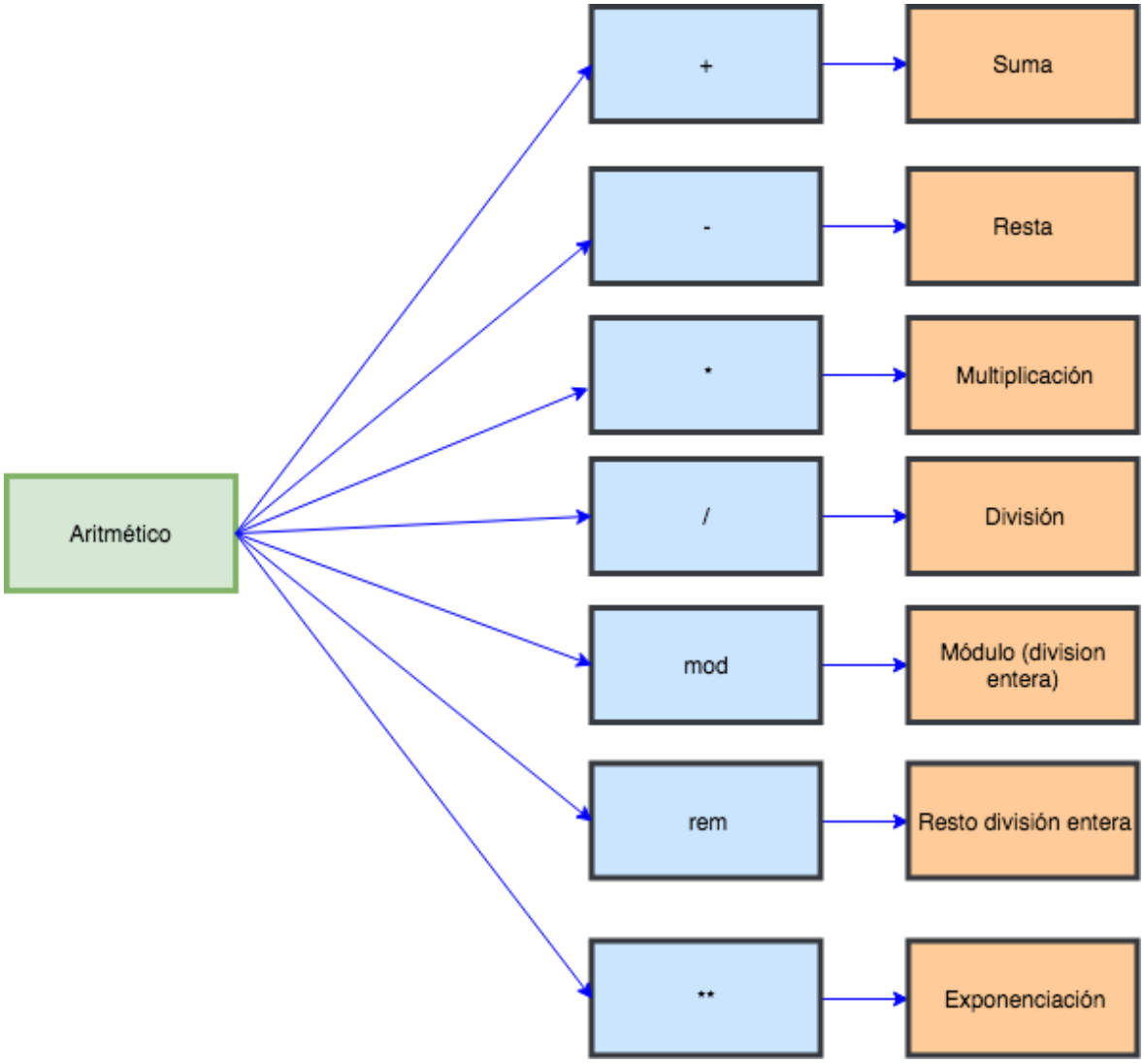


Figura 2.4. Operadores aritméticos.
Fuente: elaboración propia.

Los operadores aritméticos actúan sobre objetivos de tipo entero (*integer*), usado como valor en constantes o valores genéricos, al igual que con vectores; y permiten realizar operaciones de tipo aritmético.

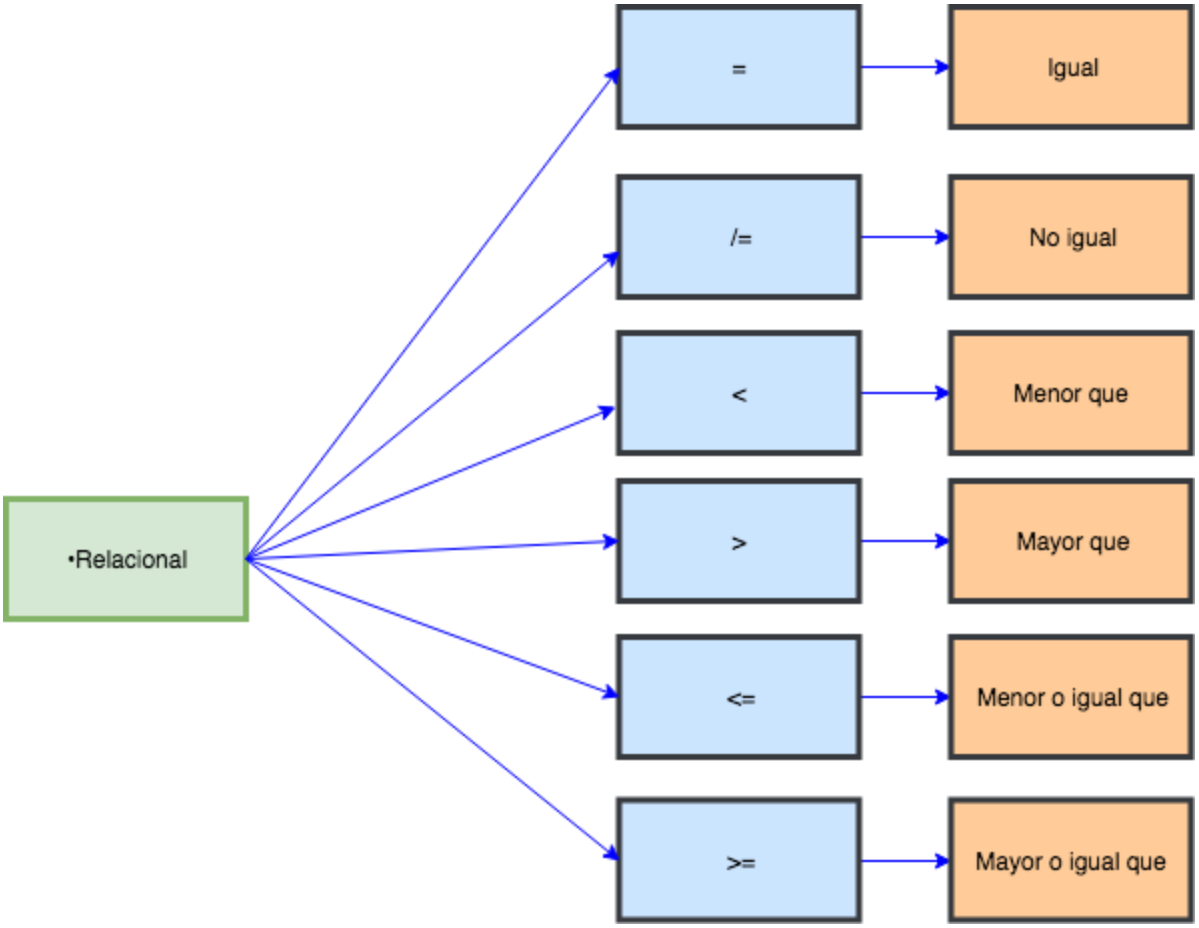


Figura 2.5. Operadores relacionales.
Fuente: elaboración propia.

Los operadores relacionales, por su parte, siempre entregarán un valor booleano al realizar la comparación solicitada. Por ejemplo, si se tiene "=", en el primer caso será verdadero si los operandos son iguales, y falso en caso contrario. Estos operadores se utilizan cuando se requiere realizar comparaciones y tomar acciones puntuales frente a los resultados.

Las operaciones binarias, a su turno, son aquellas establecidas para las compuertas lógicas, las cuales trabajan únicamente con valores uno (1) o cero (0), o valores booleanos[24].

24 Estos están asociados a valores de 1, 0; a nivel binario, estos valores pueden ser interpretados como verdadero o falso, respectivamente.

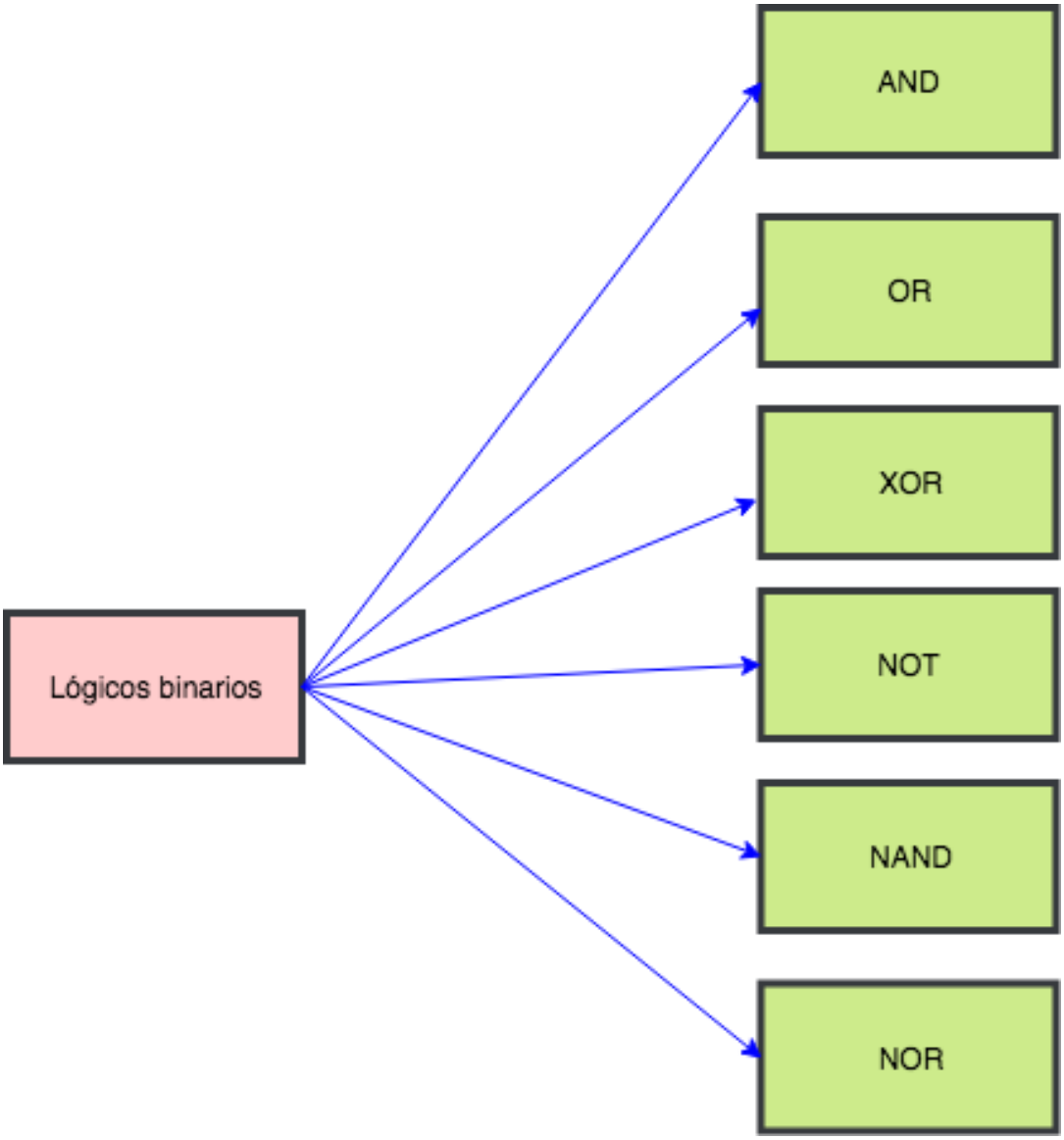


Figura 2.6. Operadores lógicos binarios.
Fuente: elaboración propia.

Arquitectura

La arquitectura define el funcionamiento de la entidad: de modo concreto, indica cómo están funcionando de manera interna el circuito, los procedimientos, las funciones o las señales. La forma general de una arquitectura se describe a continuación:

```
Architecture Descripcion_Entidad of nombre_archivo is
begin
C<= A and B;
end nombre_archivo;
```

Existen tres formas de realizar la descripción de funcionamiento de la entidad, a saber:

- Comportamiento (funcional o algorítmico [*behavioral*]).
- Estructural (*structural*).
- Flujo de datos (*dataflow*).

Comportamiento (funcional o algorítmico [behavioral]): especifica cómo se comportan las salidas con respecto a las entradas. El comportamiento de la entidad se expresa mediante el uso de ejecuciones secuenciales, cuya sintaxis y semántica se asemejan a las usadas en el lenguaje de programación C.

A continuación, se mostrará un ejemplo comportamental de acuerdo a la siguiente condición:

Si c = d entonces a = 1

Si c ≠ d entonces a = 0

Código fuente 4. Comparación entre dos bits descrita a través de comportamiento funcional.

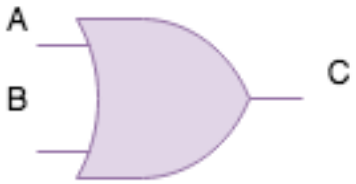
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity compara2bit is
 Port ( c : in BIT_VECTOR (1 downto 0);
 d : in BIT_VECTOR (1 downto 0);
 a : out BIT);
end compara2bit;

architecture Behavioral of compara2bit is

begin
compara: process (c,d)
```

```
        begin
                if c=d then
                 a<='1';
                else
                 a<='0';
                end if;
 end process compara;
end Behavioral;
```

Fuente: elaboración propia.

Flujo de datos (*dataflow*): especifica el circuito como una representación concurrente del movimiento de los datos. Para ello, es posible utilizar instrucciones del tipo *when – else* o ecuaciones booleanas.

A continuación, se explicará el funcionamiento de la compuerta OR utilizando las instrucciones *when – else.* Esta compuerta siempre tendrá a la salida un 1, siempre que en una de sus entradas exista un 1 presente. La salida será 0 solo cuando ambas entradas ostenten dicho valor.



ENTRADA A	ENTRADA B	SALIDA C
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.7. Tabla de verdad y Símbolo Compuerta OR.
Fuente: elaboración propia.

Código fuente 5. Compuerta OR realizada a través de descripción de funcionamiento de flujo de datos.

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

entity ORFLUJO is
 Port ( A : in STD_LOGIC;
 B : in STD_LOGIC;
 C : out STD_LOGIC);
end ORFLUJO;

architecture Behavioral of ORFLUJO is

begin

C <= '0' WHEN (A='0' OR B='0') ELSE '1';

end Behavioral;
```

Fuente: elaboración propia.

Estructural (structural): describe la conexión entre distintos módulos que han sido descritos previamente, tales como compuertas, *flip flops* y sumadores, entre otros.

Librería

Las bibliotecas o librerías (*libraries*) almacenan la información relacionada con el diseño del programa; usarlas hace posible definir los componentes, las funciones y los procedimientos. Las bibliotecas se estructuran en paquetes (*packages*) que permiten clasificar los elementos que las componen.

En VHDL se tienen definidas dos tipos de librerías: IEEE y WORK. En la primera está el paquete std_logic_1164, mientras que en la segunda se encuentran numeric_std, std_arith y gatespkg. Para este lenguaje se debe tener en cuenta que, cuando se habla de realizar una declaración, lo que se hace es hacerla visible. Para ello se tienen dos elementos: por un lado, el nombre de la librería; y por otro, el paquete de la librería a utilizar.

La declaración de una librería se ciñe a la siguiente sintaxis:

```
Library Mibiblioteca;- - Hace visible la librería
Use Nombre_libreria.Paquete1all

library IEEE; -- En este caso IEEE es la librería
use IEEE.STD_LOGIC_1164.ALL; -- std_logic_l164 es el paquete que proporciona los tipos de señales como tipo lógico y vector
```

En lo que respecta a los paquetes, se requieren como mínimo tres para un programa sencillo:

- IEEE.STD_LOGIC_1164
- *Standard*, de la librería "std"
- *Work*, de la librería "work"

Para entender mejor este concepto, se describirán las diferentes librerías que permite usar VHDL, sus aplicaciones y características principales. Las librerías *std* y *work* son visibles por defecto, por lo que no es necesario invocarlas al momento de la programación.

El paquete estándar está predefinido en el compilador, dentro del cual se encuentran incluidos los siguientes tipos de variables: *bit, bit_vector, integer, natural positive, boolean, string character, real time* y *delay_length*.

A continuación, se presentan los diferentes tipos de librería disponibles y sus características:

- IEEE.std_logic_textio.all: contiene tipos y funciones para el acceso a ficheros de texto. Funciona para la escritura y lectura de archivos. Está definida para texto de entrada/salida, usado en tipo texto de archivos de cadena (un archivo de longitud variable guardado en ASCII).
- IEEE.std_logic_arith.all: es un grupo de funciones aritméticas para conversión, comparación, representación de números con signo y sin signo, integración, vectores, y operaciones aritméticas. Posee funciones para realizar conversiones entre tipos de datos.

- IEEE.numeric_bit.all: está definida para tipos de funciones numéricas y aritméticas, tales como herramientas de síntesis. Con esta librería se usan dos tipos definidos —*unsigned* y *signed*— y permite la detección de funciones con reloj.
- IEEE.numeric_std.all: tiene definidas operaciones matemáticas, tales como suma y resta, para el tipo std_logic, el std_logic_vector y el tipo integer.
- IEEE.std_logic_signed.all: posee funciones para trabajar con vectores como si estas fueran datos con signo.
- IEEE.std_logic_unsigned.all: permite realizar operaciones con vectores como si estas fueran datos sin signo.
- IEEE.math_real.all: es usada como interfaz para introducir los coeficientes en formato real.
- IEEE.math_complex.all: permite describir modelos en VHDL para usar constantes complejas en funciones matemáticas y operadores.

Estructura básica

En cualquier lenguaje de programación se pueden tener subprogramas para estructurar el diseño, dentro de los cuales se encuentran funciones y procedimientos: dicha estructura se puede aplicar también en VHDL.

La figura 2.8 es un diagrama de bloque por medio del cual se explicitan los elementos que conforman un archivo fuente en VHDL: la ubicación de las librerías utilizadas en el programa; la entidad donde están definidos los puertos de entrada y de salida; y la arquitectura del mismo. Seguidamente, y para efectos de comparación, se muestra un ejemplo práctico (código fuente 6).

Figura 2.8. Estructura básica de archivo fuente VHDL.
Fuente: elaboración propia.

Código fuente 6. Estructura básica compuerta OR.

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

entity compuertaor is
Port (
a : in STD_LOGIC;
b : in STD_LOGIC;
c : out STD_LOGIC);
end compuertaor;

architecture Behavioral of compuertaor is
begin
C <= a or b;
end Behavioral;
```

Encabezamiento

Entidad

Arquitectura

Fuente: elaboración propia.

Palabras reservadas

En VHDL se utilizan palabras que permiten identificar funciones, puertos o señales, las cuales reciben el nombre de "reservadas". Se requiere que dichas palabras proporcionen información que ayude a describir su función; igualmente, no deben ser muy cortas (es decir, que no indiquen mucho) ni muy largas (complicadas de usar). Además, es importante indicar que deben empezar por un carácter alfabético y no pueden terminar con el carácter de raya al piso ("_"). Las palabras reservadas no pueden emplearse para denominar variables de entrada o de salida ni para nombrar un proceso, toda vez que el lenguaje de programación no lo permitiría.

Tabla 2.4. Palabras reservadas en VHDL.

abs	*access*	*after*	*alias*	*all*
and	*architecture*	*array*	*assert*	*atribute*
begin	*block*	*body*	*buffer*	*case*
component	*configuration*	*constant*	*disconnect*	*downto*
else	*elsif*	*end*	*entity*	*exit*
file	*for*	*function*	*generate*	*generic*
guarded	*if*	*in*	*inout*	*is*
label	*library*	*linkage*	*loop*	*map*
mod	*nand*	*new*	*next*	*nor*
not	*null*	*of*	*on*	*open*
or	*others*	*out*	*package*	*port*
procedure	*process*	*range*	*record*	*register*
rem	*report*	*return*	*select*	*severity*
signal	*subtype*	*then*	*to*	*transport*
type	*units*	*untill*	*use*	*variable*
wait	*when*	*while*	*with*	*xor*

Fuente: elaboración propia.

Declaración de variables

En VHDL existen dos tipos de declaraciones —*secuenciales* y *concurrentes*—, cuyas características se abordarán a continuación.

Declaraciones concurrentes

Al hablar de este tipo se declaraciones se entiende que el conjunto de instrucciones se ejecuta de manera simultánea, sin importar el orden en el cual se escriban. Comúnmente, se asocian las declaraciones concurrentes a todas las declaraciones dentro de una arquitectura. En este tipo de declaración es común hacer uso de los condicionales *when*, *else*, o bien emplear ecuaciones booleanas.

Código fuente 7. Compuerta AND usando declaración concurrente.

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

entity concurrente is
Port( a: in STD_LOGIC_VECTOR (1 downto0);
b :in STD_LOGIC_VECTOR (1 downto 0);
c :out STD_LOGIC);
end concurrente;
architecture Behavioral of concurrente is
begin
c<='1' when (a=b) else '0';
end Behavioral;
```

Encabezamiento

Entidad

Arquitectura

Fuente: elaboración propia.

Declaraciones secuenciales

Las instrucciones de este tipo se ejecutan una tras otra —de allí su nombre— y se encuentran siempre dentro de un proceso o de un subprograma.

Este tipo de declaraciones es bastante común en procesos que requieren una secuencia para la correcta ejecución del proceso que se desea. Como ejemplo, tómese el momento en que se digita la clave en el inicio en una cerradura electrónica: si tal contraseña no es correcta, se envía un mensaje y se solicita digitarla nuevamente; y si falla nuevamente por tres veces consecutivas, la cerradura quedará bloqueada. En este tipo de sistema, a nivel de programación es común usar los condicionales *if* (si), *then* (entonces) y *else* (sino).

De acuerdo con el anterior planteamiento, a continuación se establece un ejemplo en VHDL en el cual se usan declaraciones secuenciales.

Código fuente 8. Declaración secuencial (compara si dos números son iguales).

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

entity secuencial is
Port( a: in STD_LOGIC_VECTOR (1 downto 0);
b : in STD_LOGIC_VECTOR(1 downto 0);
c : out STD_LOGIC);
end secuencial;

architecture Behavioral of secuencial is
begin
process (a,b)
begin
if a = b then
c <='1';
else
c <='0';
end if;
end process;
end Behavioral;
```

Encabezamiento

Entidad

Arquitectura

Fuente: elaboración propia.

Nótese que, en este caso, la sentencia condicional está agregada dentro de la sentencia *process*, que se discutirá a continuación.

Proceso (*process*)

Un proceso en VHDL permite representar una *estructura secuencial* a través de las sentencias mencionadas anteriormente. Un proceso puede tener un nombre asociado dentro de la estructura de programa realizada en VHDL.

Los procesos tienen la siguiente sintaxis en VHDL:

```
p1: process(clk)—Definición de proceso

--Descripción de la secuencia a realizar.

end process;
```

En la sentencia *process* se agregan las *entradas síncronas* y *asíncronas,* indispensables para el funcionamiento adecuado de la entidad.

Tipos de diseño en VHDL

En lenguaje VHDL es posible describir los sistemas de acuerdo a su comportamiento (funcionalidad) o estructura (construcción): la pregunta "¿qué hace la entidad?" hace referencia a la primera; mientras que el cuestionamiento "¿cómo está compuesta?" alude a la segunda.

La construcción está asociada a los componentes e interconexiones propios de la entidad; y la funcionalidad, como se dijo, está asociada a su comportamiento, el cual puede ser descrito a través de una forma algorítmica o de flujo de datos.

Paquete (package)

Los paquetes constan de subprogramas que pueden utilizarse en diseños posteriores. Estos son realizados por el diseñador y pueden contener constantes y tipos de uso interno. Así, la declaración formal de un paquete es la siguiente:

```
package epackage_name is
  ... exported constant declarations
  ... exported type declarations
  ... exported subprogram declarations
end package_name;
```

La utilización del paquete creado requiere hacerlo "visible" para la unidad de diseño, mediante la instrucción "use".

```
use library_name.package_name.all
```

En el próximo capítulo se presentarán ejercicios que permitirán entender de una mejor forma los conceptos establecidos hasta ahora.

Primeros programas en VHDL

En este capítulo se hará uso de los conceptos de electrónica digital para mostrar la construcción de programas en VHDL a través de los conceptos abordados en el capítulo anterior. Los dos primeros ejemplos se realizarán mediante diseño estructural. Para asignar valores a una señal se utiliza el operador "<="; si se utiliza una constante o una variable, se emplea ":=".

Ejemplo 1: comprobación de tabla de verdad, compuerta OR

Este primer programa tiene como propósito hacer una introducción a la programación en VHDL. Lo primero que debe recordarse es la tabla de verdad de la compuerta.

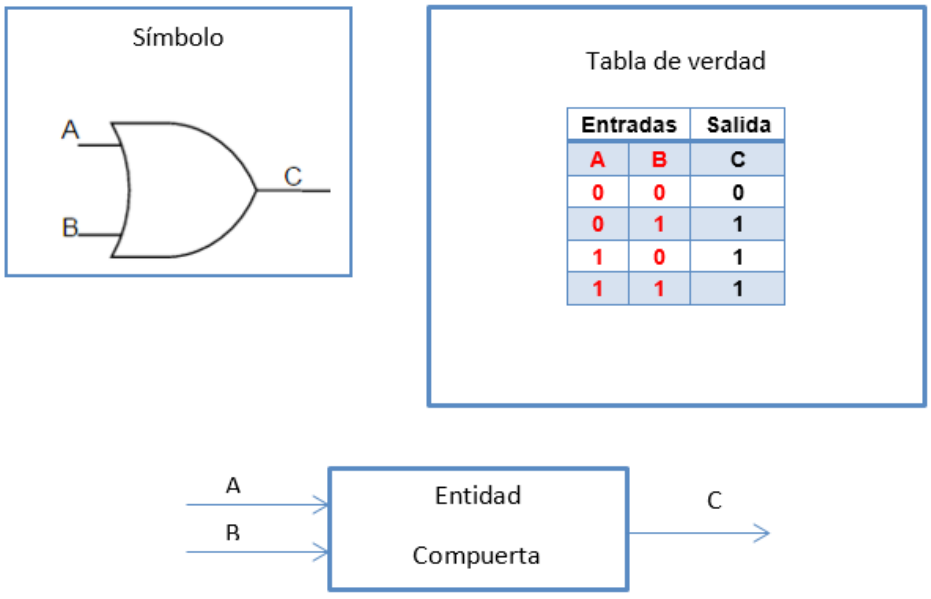


Figura 3.1. Símbolo, Tabla De Verdad Y Entidad.
Fuente: elaboración propia.

Conocidas las entradas y las salidas, se hace uso de palabras reservadas: para este caso, los operadores lógicos binarios, dentro de los cuales está la compuerta OR. Se declaran las variables a y b como entradas y la variable c del tipo estándar lógico. Con el uso del operador relacional, la salida c será igual que a **or** b.

Código fuente 9. Compuerta OR.

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

entity compuertaor is
port(

a: in STD_LOGIC;
b : in STD_LOGIC;
c : out STD_LOGIC);
end compuertaor;
architecture Behavioral of compuertaor is
begin
c <= a or b;  -- Se asigna el valor a c el resultado de a or b.

end Behavioral;
```

Encabezamiento

Entidad

Arquitectura

Fuente: elaboración propia.

Ejemplo 2: variables declaradas como entradas y entrada-salida

El siguiente ejemplo permitirá entender el concepto de la declaración de variables como solo entrada, y la diferencia entre estas y las de entrada-salida. Cualquiera sea el método utilizado, el ejercicio tendrá el mismo resultado.

En la primera parte del ejemplo se cuenta con tres entradas (A, B, C) y una salida (E).

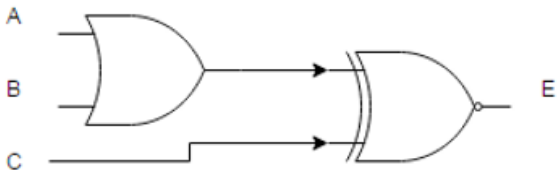


Tabla de verdad

Entradas			Salida
A	B	C	E
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

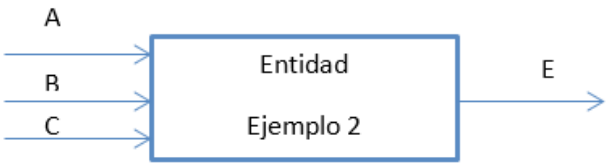


Figura 3.2. Símbolo, tabla de verdad y entidad del ejemplo.
Fuente: elaboración propia.

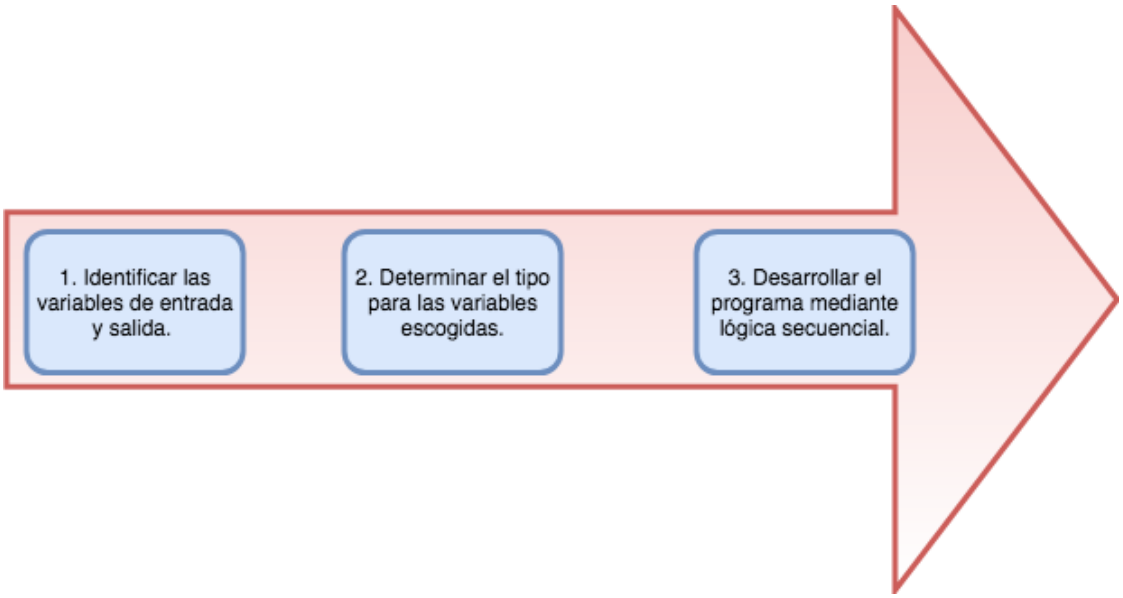


Figura 3.3. Pasos a seguir para el desarrollo de un programa.
Fuente: elaboración propia.

Este programa se desarrollará de la misma forma como se realizaría en electrónica digital. La salida E dependerá exclusivamente de los valores de entrada de las variables A, B y C.

Código fuente 10. Ejemplo 2: circuito digital de tres entradas y una salida.

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

entity ejemplo2 is
Port(
A:  in STD_LOGIC;
B : in STD_LOGIC;
C : in STD_LOGIC;
E : out STD_LOGIC);
End ejemplo2;

architecture Behavioral of ejemplo2 is

begin

E <=((A OR B) XNOR C);

end Behavioral;
```

Encabezamiento

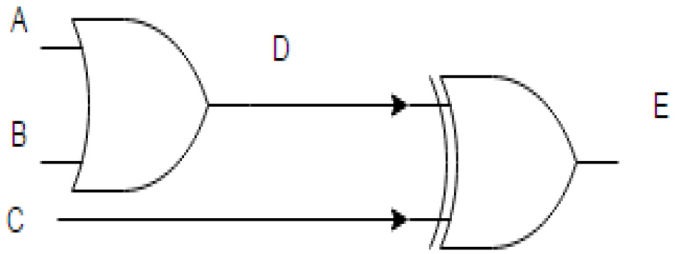
Entidad

Arquitectura

Fuente: elaboración propia.

En la segunda parte de este ejemplo se cuenta con el mismo circuito, pero con la diferencia de contar con la variable D. Esta ya no será solo de entrada o de salida, sino de tipo *in/out*, por cuanto será de salida para la primera compuerta y de entrada en la segunda. Obsérvese que la tabla de verdad y la entidad no sufren ninguna variación.

Código fuente 11. Ejemplo en el que se emplean tres entradas, una variable (entrada-salida) y una sola salida.




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

Encabezamiento

```
entity ejemplo2 is
Port(
A:  in STD_LOGIC;
B : in STD_LOGIC;
C : in STD_LOGIC;
D : inout STD_LOGIC;
E : out STD_LOGIC);
end ejemplo2 or;

architecture Behavioral of ejemplo2 is

begin

D <= A OR B;
E <= D XOR C;

end Behavioral;
```

Entidad

Arquitectura

Fuente: elaboración propia.

Ejemplo 3: números primos

En una tabla con números del 0 al 15, se quiere que se ilumine un led únicamente cuando se tenga un número primo (i. e. aquel que solo puede ser dividido por sí mismo o 1) según la combinación de las variables de entrada. Para escribir el número se requieren 4 bits o dígitos binarios, los cuales serán representados por las letras A, B, C y D.

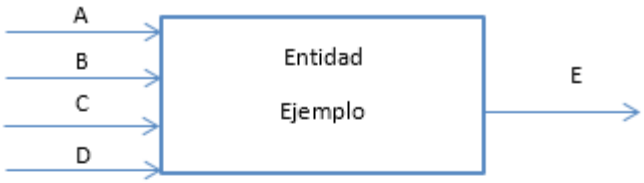


Tabla de verdad

Entradas				Salida
A	B	C	D	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
1	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Figura 3.4. Entidad y tabla de verdad para ejemplo 3.

Fuente: elaboración propia. La ecuación lógica de salida será igual a: $E = \overline{A}D + B\overline{C}D + \overline{B}\overline{C}D + \overline{A}\overline{B}C$

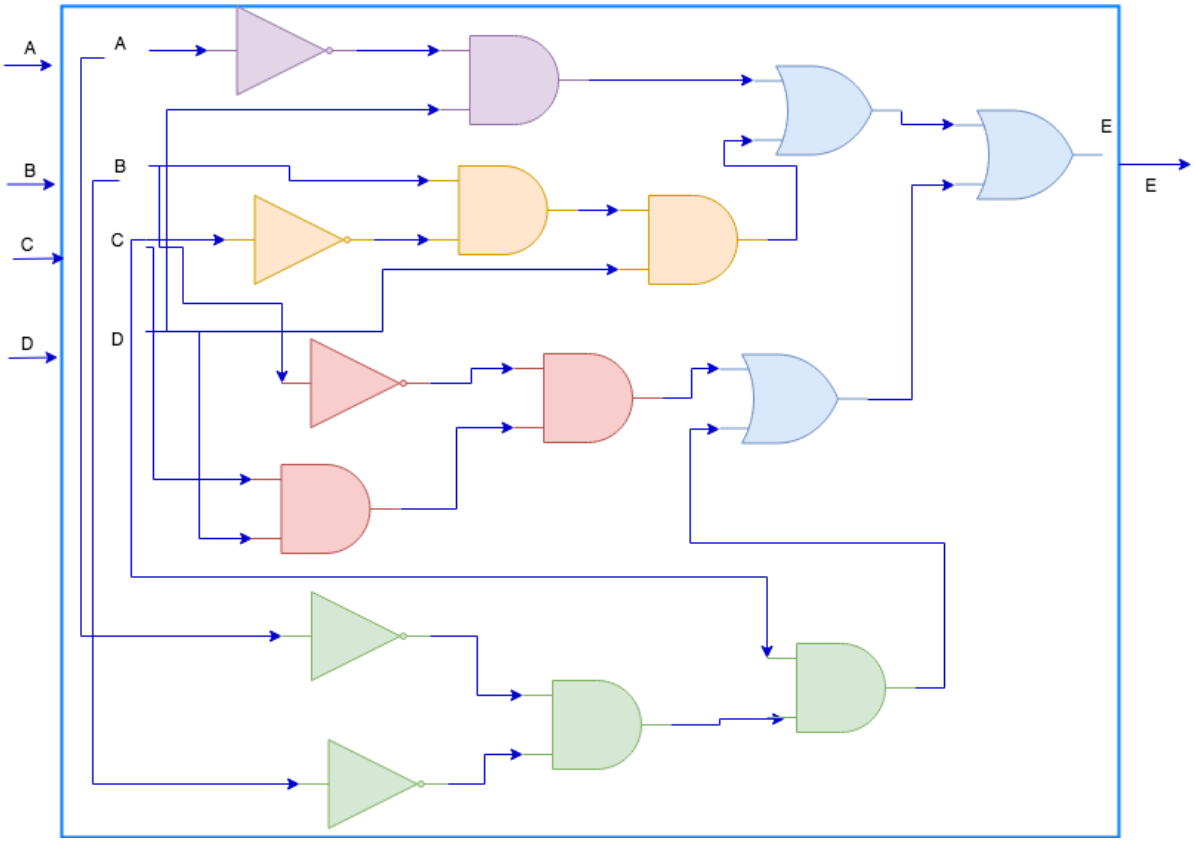


Figura 3.5. Representación esquemática de los números primos de 0 a 15.
Fuente: elaboración propia.

A continuación, se mostrará la declaración mediante ecuaciones booleanas desarrolladas a través de VHDL.

Código fuente 12. Números primos usando ecuaciones booleanas.

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

Encabezamiento

```
entity compara is
Port( A: in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
igual: out STD_LOGIC;
mayor: out STD_LOGIC;
menor: out STD_LOGIC);
end compara;

architecture Behavioral of compara is
```

Entidad

```
begin
        process(A,B)
        BEGIN
        igual<='0';
        mayor<='0';
        menor<='0';
        IF (A=B) THEN IGUAL<='1';
                ELSIF (A>B) THEN MAYOR <='1';
                ELSE MENOR <='1';
        END IF;
        END PROCESS;
end Behavioral;
```

Arquitectura

Fuente: elaboración propia.

Para este caso se ha incluido la simulación (diagrama de tiempos) del comportamiento de la entidad usando un *test bench* (figura 3.6).

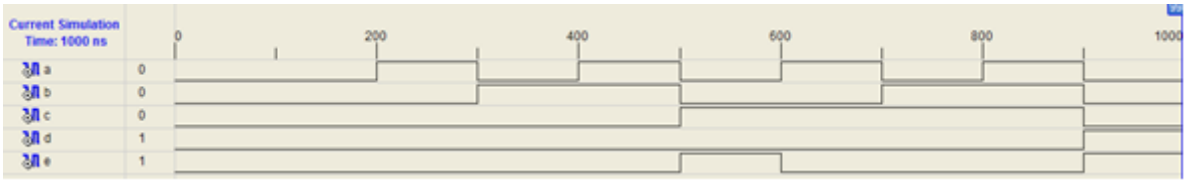


Figura 3.6. Test bench para ejemplo 3; se puede visualizar el funcionamiento adecuado de acuerdo a los criterios establecidos.

Fuente: elaboración propia.

Ejemplo 4: números primos usando sentencia condicional "when"

El planteamiento inicial de este ejemplo es igual al anterior: se quiere que se ilumine un led únicamente cuando se tenga un número primo en una tabla del 0 al 15, según la combinación de las variables de entrada. Asimismo, las condiciones son exactamente iguales a las del ejemplo, con la única diferencia de la estructura de programa: la salida será uno (1) únicamente cuando ("*when*") cuando los números 1, 2, 3, 5, 7, 11 y 13 sean introducidos; en los demás, la salida será cero (0) y, por ende, el led no encenderá.

Se hará uso en este ejemplo del tipo STD_LOGIC_VECTOR, el cual indica que se utilizará una entrada correspondiente a un valor de más de 1 bit. Si se tiene un valor de 2 bit, se mostrará el tipo STD_LOGIC_VECTOR 1 downto 0; si es de 3 bit, será STD_LOGIC_VECTOR 2 downto 0, y así sucesivamente hasta un máximo de 8 bits de entrada. Se realizará la declaración mediante uso de *declaraciones concurrentes*, lo cual permitirá observar la diferencia en la programación aunque el resultado sea igual.

Código fuente 13. Números primos de 0 a 15 usando declaraciones concurrentes.

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

entity primosconwhen is
Port( R: in STD_LOGIC_VECTOR (3 downto 0);
S : out STD_LOGIC);
end primosconwhen;

architecture Behavioral of primosconwhen is

begin
with R select
S <=    '1' when"0001",
        '1' when"0010",
        '1' when"0011",
        '1' when"0101",
        '1' when"0111",
        '1' when"1011",
        '1' when"1101",
        '0' when others;

end Behavioral;
```

Encabezamiento

Entidad

Arquitectura

Fuente: elaboración propia.

A continuación, se muestra la simulación a través de *test bench* de la entidad de números primos mediante el condicional *when*.

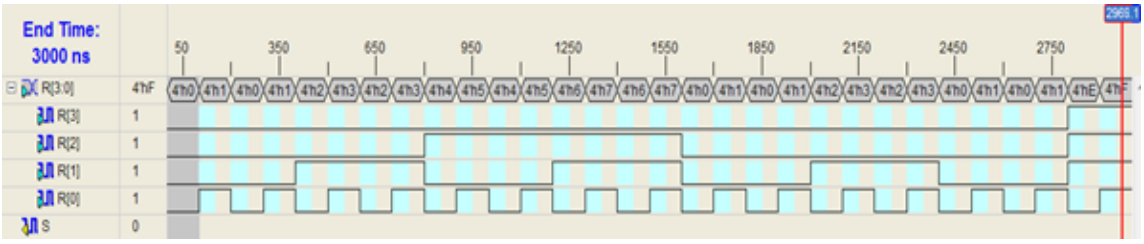


Figura 3.7. Test bench para el ejemplo 4; se puede visualizar el funcionamiento adecuado de acuerdo a los criterios establecidos.
Fuente: elaboración propia.

Ejemplo 5: comparar dos números de cuatro bits mediante sentencia condicional "if, elsif, else"

La sentencia *if* (si) es una declaración secuencial y ejecuta una operación según el valor de la condición. Puede contener, de manera opcional, una parte *else* (sino) que corresponde a la parte negativa de la condición *elsif*, que como su nombre lo sugiere, es una combinación de *if* y *else*; en este caso, la expresión condicional se puede evaluar como cierta. El formato general de la estructura es el siguiente:

```
if<condition> then
<statement>
elsif<condition> then
<statement>
else
<statement>
end if;
```

En este ejemplo se realizará una comparación entre dos números para mostrar si uno de ellos es mayor, menor o igual. Así, se contará con dos números, R y S, cada uno de los cuales tiene cuatro bits. Las variables de entrada serán del tipo vector de cuatro bits, y se tendrán tres señales de salida diferentes que representen las diferentes opciones (mayor, menor o igual).

Figura 3.8. Entidad circuito comparador.
Fuente: elaboración propia.

Código fuente 14. Comparación de dos números a través de VHDL para ejemplo 5.

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

Encabezamiento

```
entity compara is
Port( A: in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
igual: out STD_LOGIC;
mayor: out STD_LOGIC;
menor: out STD_LOGIC);
end compara;

architecture Behavioral of compara is
```

Entidad

```
begin
        process(A,B)
        BEGIN
        igual<='0';
        mayor<='0';
        menor<='0';
        IF (A=B) THEN IGUAL<='1';
                ELSIF (A>B) THEN MAYOR <='1';
                ELSE MENOR <='1';
        END IF;
        END PROCESS;
end Behavioral;
```

Arquitectura

Fuente: elaboración propia.

Se realiza la comparación de diferentes valores a lo largo de la línea de tiempo, como se muestra en la gráfica de *test bench*.

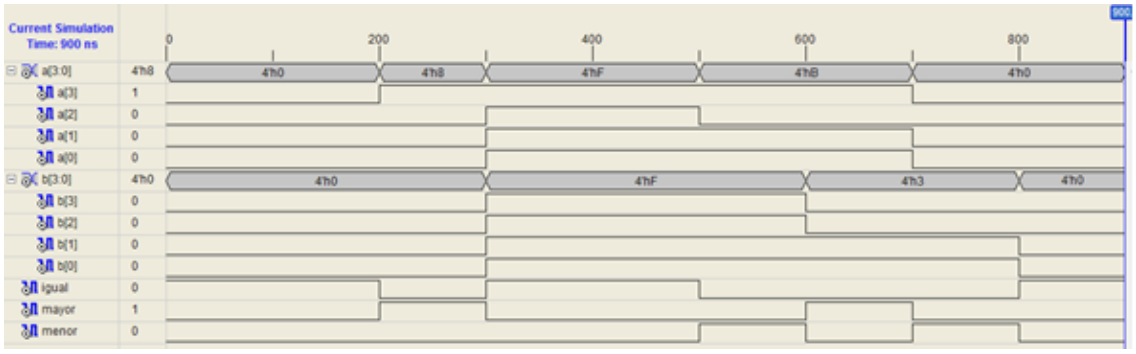


Figura 3.9. Diagrama de tiempo generado mediante test bench para ejemplo 5; se puede visualizar el funcionamiento adecuado de la comparación entre dos números según los criterios establecidos.
Fuente: elaboración propia.

Ejemplo 6: utilización de sentencia condicional "when-else"

La sentencia "*when-else*" permite seleccionar una alternativa entre varias presentes en una expresión, de acuerdo a una combinación de entradas o al valor de una variable especifica. En este ejemplo se plantea contar con tres entradas, nombradas *a, b* y *c,* con una salida llamada *f*. La salida de la entidad funcionará únicamente cuando las entradas *a, b* y *c* estén en los números 0, 2, 6 y 7. El funcionamiento de la entidad se muestra en la figura 3.10.

Entradas			**Salida**
a	**b**	**C**	**f**
0	**0**	**0**	**1**
0	**0**	**1**	**0**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

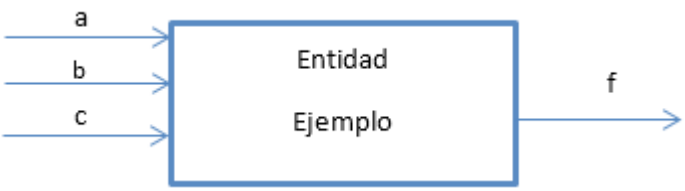


Figura 3.10. Tabla de verdad y diagrama de entidad para el ejercicio propuesto.
Fuente: elaboración propia.

Código fuente 15. Programa de ejemplo en VHDL usando sentencias condicionales "*when-else*".

```
library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_ARITH.ALL;
Use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Encabezamiento

```
Entity ejercicicicocase is
Port( a: in STD_LOGIC;
b :in STD_LOGIC;
c :in STD_LOGIC;
f :out STD_LOGIC);
end ejercicicicocase;
```

Entidad

```
architecture Behavioral of ejercicicicocase is

begin
f <=            '1' when (a='0' and b='0' and c='0') else
                '1' when (a='0' and b='1' and c='0') else
                '1' when (a='1' and b='1' and c='0') else
                '1' when (a='1' and b='1' and c='1') else
                '0';

End Behavioral;
```

Arquitectura

Fuente: elaboración propia.

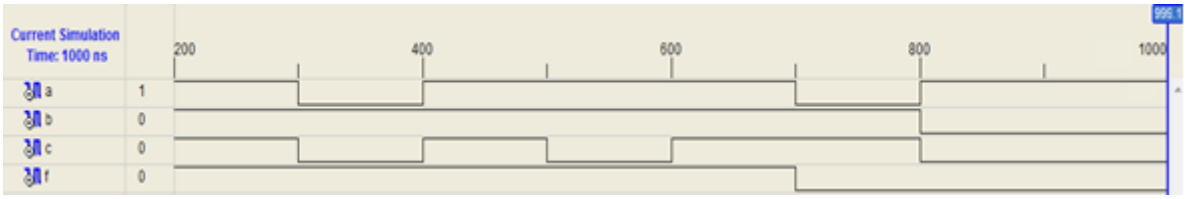


Figura 3.11. Test bench para ejemplo 6; se puede visualizar el funcionamiento adecuado de la comparación entre dos números según los criterios establecidos.
Fuente: elaboración propia.

Ejemplo 7: conversión de hexadecimal a siete segmentos

En este ejemplo se mostrará, de manera sencilla, cómo se pueden representar los números en binario ingresados a partir de los interruptores y visualizar su valor correspondiente en decimal, utilizando los *displays* para tal efecto.

Los *displays* están conformados por segmentos, los cuales poseen una letra que los identifica (figura 3.12). La entrada será declarada como un vector de 4 valores (3 *downto* 0) y la salida será otro vector de 7 valores (6 *downto* 0). En la tabla 3.1, a su turno, se podrán observar los valores comprendidos entre 0000 y 1111. Debe tenerse en cuenta que, según las conexiones internas de la tarjeta, los segmentos se encienden con un "0" en el pin correspondiente. En este ejercicio se encenderán de manera conjunta los cuatro *displays,* mostrando el mismo número.

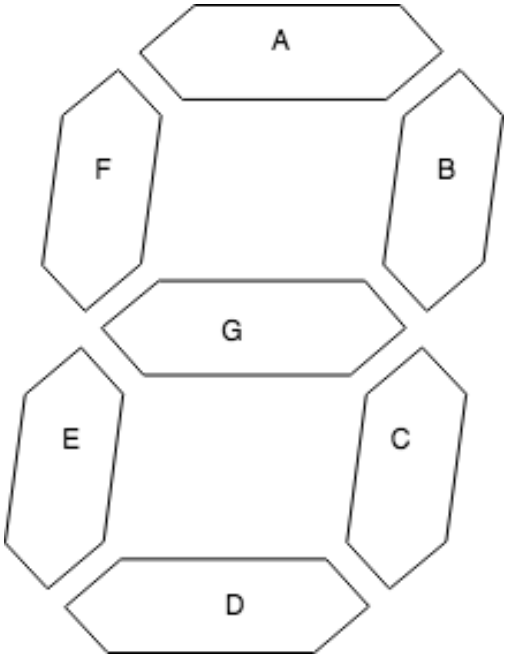


Figura 3.12. Nomenclatura de segmentos en el display.
Fuente: elaboración propia.

Tabla 3.1. Equivalencia entre sistemas numéricos y representación en display siete segmentos.

Número en decimal	Número en binario	Número en hexadecimal	Representación en el *display*						
			G	F	E	D	C	B	A
0	0000	0	1	0	0	0	0	0	0
1	0001	1	1	1	1	1	0	0	1
2	0010	2	0	1	0	0	1	0	0
3	0011	3	0	1	1	0	0	0	0
4	0100	4	0	0	1	1	0	0	1
5	0101	5	0	0	1	0	0	1	0
6	0110	6	0	0	0	0	0	1	0
7	0111	7	1	1	1	1	0	0	0
8	1000	8	0	0	0	0	0	0	0
9	1001	9	0	0	1	1	0	0	0
10	1010	A	0	0	0	1	0	0	0
11	1011	B	0	0	0	0	0	1	1
12	1100	C	1	0	0	0	1	1	0
13	1101	D	0	1	0	0	0	0	1
14	1110	E	0	0	0	0	1	1	0
15	1111	F	0	0	0	1	1	1	0

Fuente: elaboración propia.

Código fuente 16. Descripción de entidad display mediante sentencia condicional with-select.

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

entity sietesegmentos is
Port( D: in STD_LOGIC_VECTOR (3 downto0);
S : out STD_LOGIC_VECTOR (6 downto0));
end sietesegmentos;

architecture Behavioral of sietesegmentos is

begin
with D select
```

Encabezamiento

Entidad

```
S <=    "1000000" when  "0000",
"1111001" when  "0001",
"0100100" when  "0010",
"0110000" when  “0011",
"0011001" when  "0100",
"0010010" when  "0101",
"0000010" when  "0110",
"1111000" when  "0111",
"0000000" when  "1000",
"0011000" when"1001",
"0001000" when"1010",
"0000011" when"1011",
"1000110" when"1100",
"0100001" when"1101",
"0000110" when"1110",
"0001110" when"1111",
"1111111" when others;
end Behavioral;
```

Arquitectura

Fuente: elaboración propia.

A continuación, se muestra el diagrama de tiempos generado a partir del *test bench*:

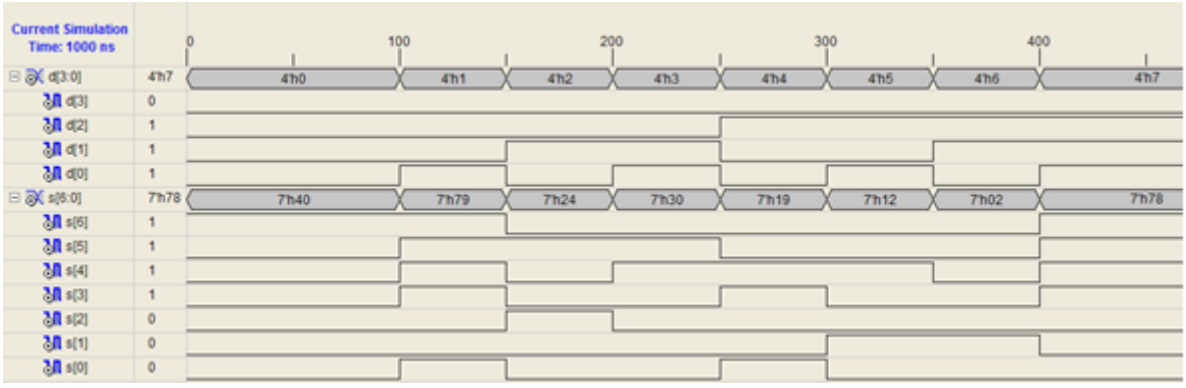


Figura 3.13. Diagrama de tiempos para el ejemplo 7; se puede visualizar el funcionamiento adecuado, por el cual se visualizan los números en los *displays* según los criterios establecidos.
Fuente: elaboración propia.

Ejemplo 8: sentencia *process*

La sentencia *process* es una de las más utilizadas, puesto que permite describir *hardware* y los procesos llevados a cabo dentro de la programación. Este ejemplo tiene como propósito utilizar el reloj interno para visualizar en dos *displays* el incremento progresivo desde el 00 hasta el número 99 y continuar el conteo. En este caso se ha usado el concepto de *visualización dinámica*, el cual permite que la información sea enviada a los *displays* de manera diferenciada (*multiplexada*), compartiendo un solo puerto de conexión. Como se observó en el capítulo 1, los *displays* de la tarjeta Basys 2 comparten un puerto de datos único pero la activación de cada uno está diferenciada, con lo cual la información debe ser enviada a cada *display* en una unidad de tiempo. La tasa de envío de datos (tasa de refresco) es de 10 ms.

Código fuente 17. Contador 0-99 con visualización dinámica a través de la sentencia process.

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

Encabezamiento

```
Entity Contadordosdigitos is
Port( Display: out STD_LOGIC_VECTOR (6 downto 0);
Enable : out STD_LOGIC_VECTOR (3 downto 0);
clk: in STD_LOGIC;
Reset : in STD_LOGIC);
End Contadordosdigitos;

architecture Behavioral of Contadordosdigitos is

begin
```

Entidad

```
process (clk,Reset)--Proceso para contador
variable Delay: integer Range 0 to 50000001;
--Rango de variable para retraso. el reloj es de 50Mhz.
variable Contador: integer Range 0 to 10;
--Contador rango 0 a 99.
Variable Visualizacion: integer Range 0 to 50000000;
--Contador para visualización dinamica en Display.
variable Decenas: integer range 0 to 10;
--Unidades y decenas a ser visualizadas.
variable Unidades: integer Range 0 to 9;
variable Temporal: integer Range 0 to 9;
--Variable temporal que tendrá decenas ó unidades.
```

Variables

```
begin
if (clk='1' and clk'event) then
--Detección de evento de reloj en este caso flanco de subida.

If (Reset='1') then
--Si reset entonces reiniciar variables.
Contador:=0;
Decenas:=0;
Unidades:=0;

end if;
```

Señal de reloj para process.

```
Visualizacion:=Visualizacion+1;
--Incrementar contador Display en una unidad.
Delay:=Delay+1;

If (Delay<500000) then
Delay:=Delay+1;-- Incrementar contador en una unidad.
else
Contador:=Contador+1;
Delay:=0;
end if;

if (Contador>9) then
--Si contador mayor a 9 entonces incrementar decenas
Decenas:=Decenas+1;
Contador:=0;
else
Unidades:=Contador;
end if;

if (Decenas>9) then--ReiniciarDecenas.
Decenas:=0;
end if;

if (Visualizacion<250000) then
Temporal:=Decenas;
Enable<="1110"; -- Activar Display1
else
Temporal:=Unidades;
Enable<="1101"; --Activar Display 2.
end if;
```

Bloque de visualización dinámica.

```
if (Visualizacion>500000) then
Visualizacion:=0;--Reiniciar contador.
end if;

case Temporal is—segmentos en orden: gfedcba
when 0=>Display<="1000000";
when 1=>Display<="1111001";
when 2=>Display<="0100100";
when 3=>Display<="0110000";
when 4=>Display<="0011001";
when 5=>Display<="0010010";
when 6=>Display<="0000010";
when 7=>Display<="1111000";
when 8=>Display<="0000000";
when 9=>Display<="0011000";
when others=>null;
end case;

end if;
end process;

end Behavioral;
```

Conversión

de digito a equivalente

7 segmentos.

Fuente: elaboración propia.

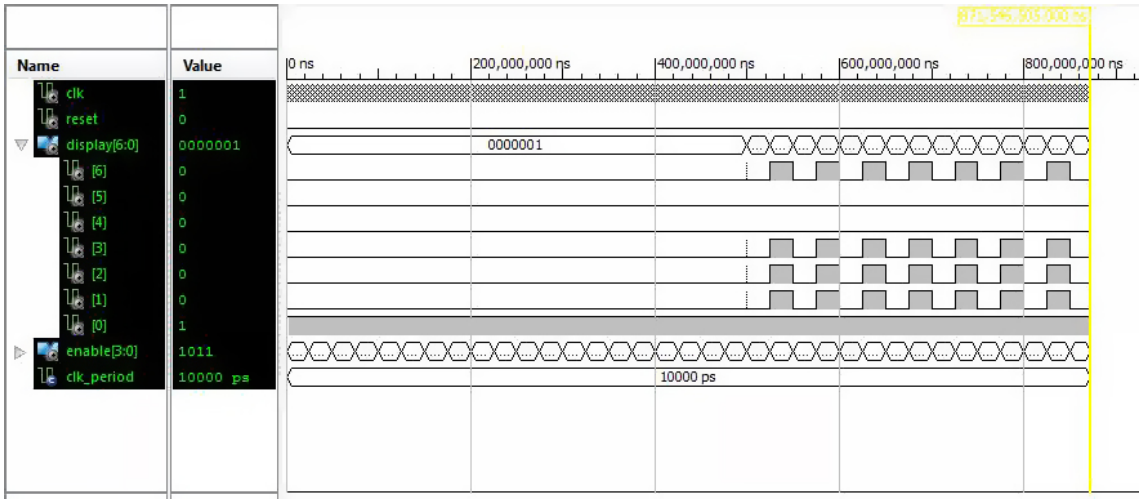


Figura 3.14. Diagrama de tiempos para ejemplo 8; es posible visualizar el funcionamiento adecuado del contador 0-99.

Fuente: elaboración propia.

Programas adicionales en VHDL

Este capítulo busca propiciar el aprendizaje de las diversas metodologías que permiten la construcción de programas en VHDL, mediante varios ejemplos referidos al manejo de puertos y a la creación de periféricos necesarios para la transferencia de información entre procesos.

Ejemplo 9: manejo de LCD

Este ejemplo busca que el lector se familiarice con el funcionamiento de algunos dispositivos visualizadores de información —LCD, por ejemplo—, a través de la construcción de entidades en lenguaje VHDL. Como elemento inicial, considérese el LCD que aparece en la figura 4.1 compatible con el controlador Hitachi 44780u (Hitachi Ltd, 1999).

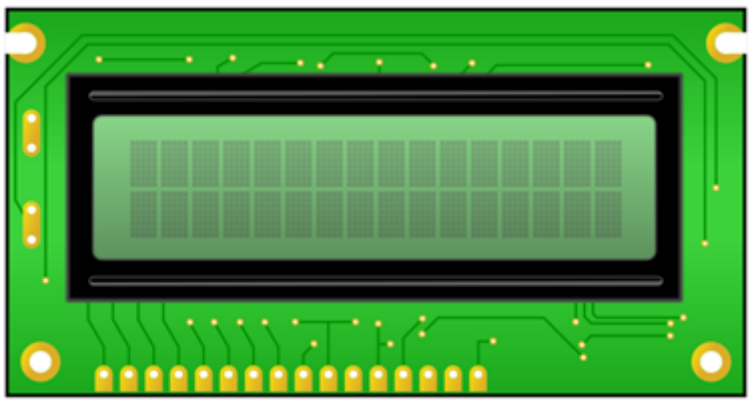


Figura 4.1. Dispositivo visualizador LCD de 2 líneas y 16 caracteres.
Fuente: elaboración propia.

La información que se desea visualizar se muestra habilitando los pixeles necesarios sobre la pantalla, y se presenta debido a la interacción entre la polarización de luz y la estructura de dicha pantalla. Esta contiene un conjunto de pines de configuración con una determinada función para la visualización de la información requerida por el usuario. Este conjunto de pines de configuración se relaciona en la siguiente tabla, en la cual se establece la función de cada uno de ellos.

Tabla 4.1. Pines de configuración de LCD de 2 lineas-16 caracteres.

Número de pin	Función
Pin 1	Vss (conexión a tierra)
Pin 2	Vdd (conexión de 5 voltios)
Pin 3	Vee (ajuste de contraste)
Pin 4	RS (pin de habilitación comando o carácter)
Pin 5	RW: pin de lectura-escritura de la memoria CGROM
Pin 6	E: *enable*, pin de habilitación del LCD en los modos comando y carácter.
Pin 7-Pin 14	Pines de datos D0-D7. A través de estos pines se envía la información hacia el LCD.
Pin 15-Pin 16	*Backlight*: Iluminación de fondo de pantalla para LCD, estos dos pines se polarizan de igual forma que un LED.

Fuente: elaboración propia.

En general, este LCD tiene dos modos de funcionamiento: carácter y comando. En el primero se envía información para ser plasmada sobre la pantalla del LCD; mientras que en el segundo se configura el LCD según algunas especificaciones, tales como número de líneas de datos, rotación de la información, tamaño de representación de un carácter y limpieza del LCD, entre otras. En este ejemplo se quiere realizar una entidad en VHDL para el manejo de mensajes sobre el dispositivo LCD, cuyos pasos de creación se resumen en la figura 4.2.

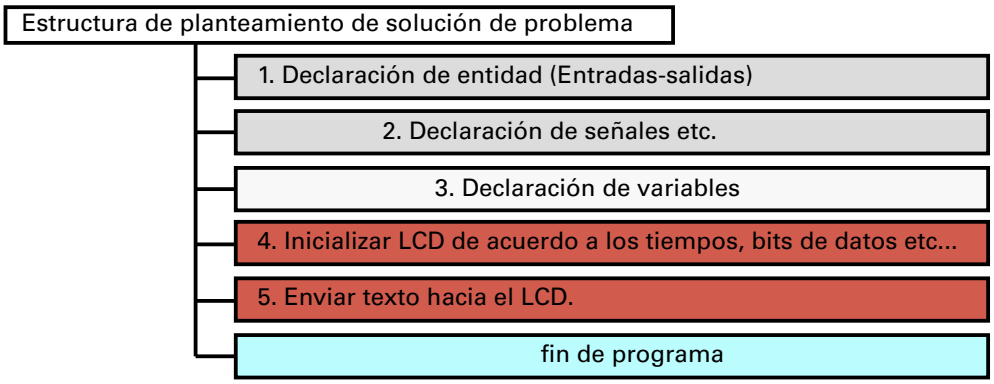


Figura 4.2. Estructura de desarrollo del programa propuesto.
Fuente: elaboración propia.

Declaración de la entidad (entradas-salidas): una de las primeras cosas a tomar en cuenta al realizar esta entidad corresponde a sus entradas y salidas. Por ello, es preciso tener presente que la interfaz a utilizar para el LCD es de 8 bits; con esto, se tendrían 8 bits de datos, además de los pines de activación (*enable*) y RS (definición de modo carácter o comando), es decir, la entidad debería tener diez salidas inicialmente. Igualmente, esta debe tener una señal de entrada de reloj por cuanto el sistema será síncrono, además de la entrada de *reset* o reinicio. El esquema de la estructura de esta entidad se muestra a continuación:

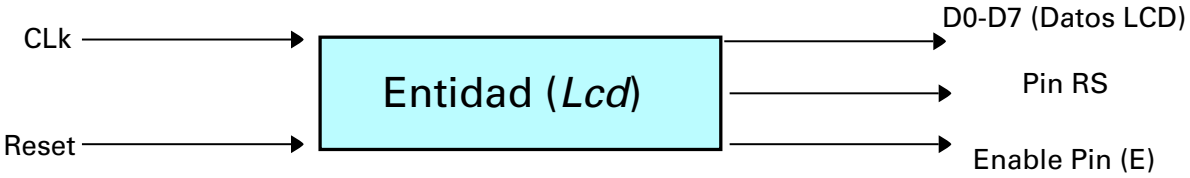


Figura 4.3. Esquema de la estructura de la entidad para el ejemplo 9.
Fuente: elaboración propia.

Después de esta definición de entradas y salidas se pasa a la construcción del código. Para el caso que nos ocupa, considérese el siguiente ejemplo realizado en el programa Xilinx WebPACK ISE 14.7.

La estructura de inicialización del LCD se resume en los siguientes pasos que se explicitan en la figura 4.4.

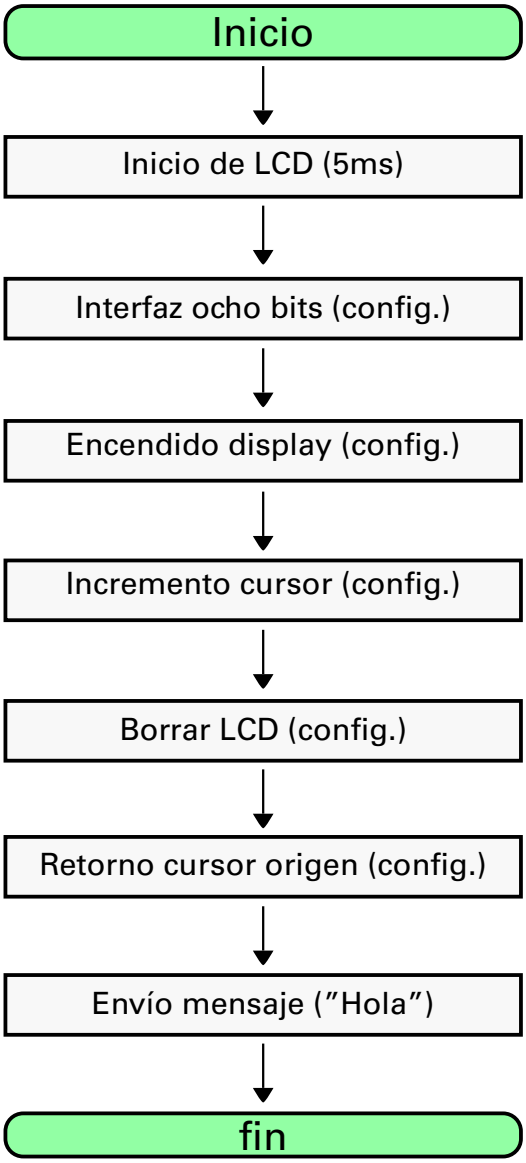


Figura 4.4. Pasos de configuración de LCD 2*16 líneas para el ejemplo 9.
Fuente: elaboración propia.

Para el proceso de configuración, cada instrucción puede tener un tiempo mínimo de ejecución de 1 ms, de acuerdo a las especificaciones del LCD. El envío de cada carácter debe hacerse tomando en cuenta este mismo tiempo y en el programa se relaciona este

retraso para cada una de las instrucciones mencionadas. El código VHDL que relaciona los anteriores pasos se muestra a continuación.

Código fuente 18. Entidad para manejo de LCD, configuración y escritura de caracteres.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity LCDinit is
Port( DataPort :out STD_LOGIC_VECTOR(7 downto 0);
      EN : out STD_LOGIC;
      RS : out STD_LOGIC;
      clk : in STD_LOGIC;
Reset: in STD_LOGIC);
end LCDinit;

architecture Behavioral of LCDinit is
begin
process(clk, Reset)--clk es el reloj del la entidad LCD.

variable CicloLCD: integer range 0 to 5000;--Esta variable permite generar el ciclo de reloj.
variable CicloEN: integer range 0 to 5001;--R
variable Proceso: integer range 0 to 14;--Cantidad de estados posibles para LCD.Mensaje 1 para
enviar a LCD.

begin
--Este ciclo permite generar un reloj de 1ms
      if (clk='1' and clk'event) then
            if (Reset='1') then
            CicloLCD: =0;--Reiniciar variables
            Proceso: =0;
            Act<='0';
            end if;

            CicloLCD:=CicloLCD+1;--incremento de LCD
```

Definición de librerías.

Definición de entidad (entradas-salidas)

Variables de proceso

Reset de entidad

```
------------------Bloque de Reloj.--------------------------

                        If (CicloLCD>=5000) then
                            Proceso:=Proceso+1;-- Los 5 primeros
                            --milisegundos                    permiten el "PowerUp" del LCD
                        CicloLCD: =0;--Reiniciar variables
                        end if;

-------------------------------------------------------------
                        if (Proceso>=5) then
                        Act<='1';
                        else
                        Act<='0';
                        En<='0';
                        DataPort<="00000000";
                        end if;

                        --Si proceso es mayor a 5 es decir 5ms envié comandos.

                        if (Proceso>13) then
                        Proceso:=14;
                        end if;
                        --Si proceso es mayor a 13

                        if(Proceso>=10) then--Envió de caracteres
                    RS<='1';--Inicialización de RS modo carácter.
                        else
                        RS<='0';--Inicialización de RS Pin para modo comando.
                    end if;

if (Act='1') then
case Proceso is
when 5=>Dataport<="00111000";--Interfaz de 8 bits.
When 6=>Dataport<="00001100";--Display on.
When 7=>Dataport<="00000110";--Incremento de cursor.
When 8=>Dataport<="00000001";--Borrar LCD.
When 9=>Dataport<="00000010";--Return to home.
When 10=>Dataport<=std_logic_vector(to_unsigned(character'pos('h'),8 ));--Enviar caracteres a -
----DisplayPort
When 11=>Dataport<=std_logic_vector(to_unsigned(character'pos('o'),8));
When 12=>Dataport<=std_logic_vector(to_unsigned(character'pos('l'),8));
When 13=>Dataport<=std_logic_vector(to_unsigned(character'pos('a'),8));
When 14=>Dataport<="00000000";--Restaurar valor de PORTLCD a 0's
when others=>null;
end case;
        if(Proceso<14) then
                if(CicloLCD>=25000) then
                EN<='0';--Inicialización de pin Enable
                else
                EN<='1';
                Act<='0';
                end if;
        else
                RS<='0';--Inicialización de RS Pin para modo comando.
```

Bloque de escritura de carácteres en el LCD.

```
                EN<='0';--Inicialización de pin Enable
            end if;
            end if;

            end if;
            end process;

    end Behavioral;
```

Fuente: elaboración propia.

En la primera parte del código se definen las librerías a usar para el manejo del dispositivo LCD. En este sentido, una librería típica que contiene la estructura del lenguaje VHDL es IEEE.STD_LOGIC_1164.ALL, la cual se establece dentro de la primera parte del código. Asimismo, se tiene IEEE.NUMERIC_STD.ALL, librería que permite realizar las operaciones de conversión de las letras del mensaje a un equivalente binario de ocho bits. Una vez estas librerías son definidas, se pasa a definir la entidad —entradas-salidas—: en este ejemplo, las entradas *físicas* del proceso son el reloj (*clk*), y el *reset* definido en cualquier sistema electrónico; y las salidas corresponden a los bits de datos (en este caso, 8 para el dispositivo LCD) y los bits de control del LCD, definidos como EN (*enable)* y RS.

Todo lo anterior, permite pasar a la construcción del *código de definición de la entidad.* Para el caso estudiado, esta última se ha realizado dentro de un proceso secuencial: de ahí la necesidad de usar la sentencia *process* en la definición de las entradas síncronas y asíncronas que participan en el funcionamiento de tal entidad (como se anotó, estas son el reloj y el *reset*). Cada instrucción debe tener un tiempo mínimo de 1 ms, de acuerdo a las características de funcionamiento del LCD[25]. La variable CicloLCD ajusta este tiempo realizando un retraso de 5.000 unidades, equivalente a 1 ms, con un reloj de la tarjeta Basys 2 cuya frecuencia asciende 50 MHz. Debe

25 Para más información de las características del controlador del dispositivo LCD (Hitachi 44780u), véase https://www.sparkfun.com/datasheets/LCD/HD44780.pdf.

resaltarse que, para este caso, corresponde usar un buffer que permita la conversión entre las señales de 3,3 V y las de 5 V, necesarias para el funcionamiento del LCD; para este fin se hace necesario el uso de un conversor de nivel (74HC245).

Es necesario percatarse de que la configuración de este integrado —por ejemplo, el pin DIR—, indica la dirección de envió de datos; de esta forma, las entradas de este integrado pueden ser los pines A0...A7 o B0…B7, según el valor lógico de este pin. Otro pin importante es el denominado OE, el cual es la activación del integrado: en este caso, el pin debe estar conectado a tierra. Esta interfaz es necesaria para hacer que las señales de la tarjeta Basys 2 sean compatibles con los niveles lógicos del controlador de LCD. El esquema de conexión final usando el integrado anteriormente mencionado se describe en la figura 4.5.

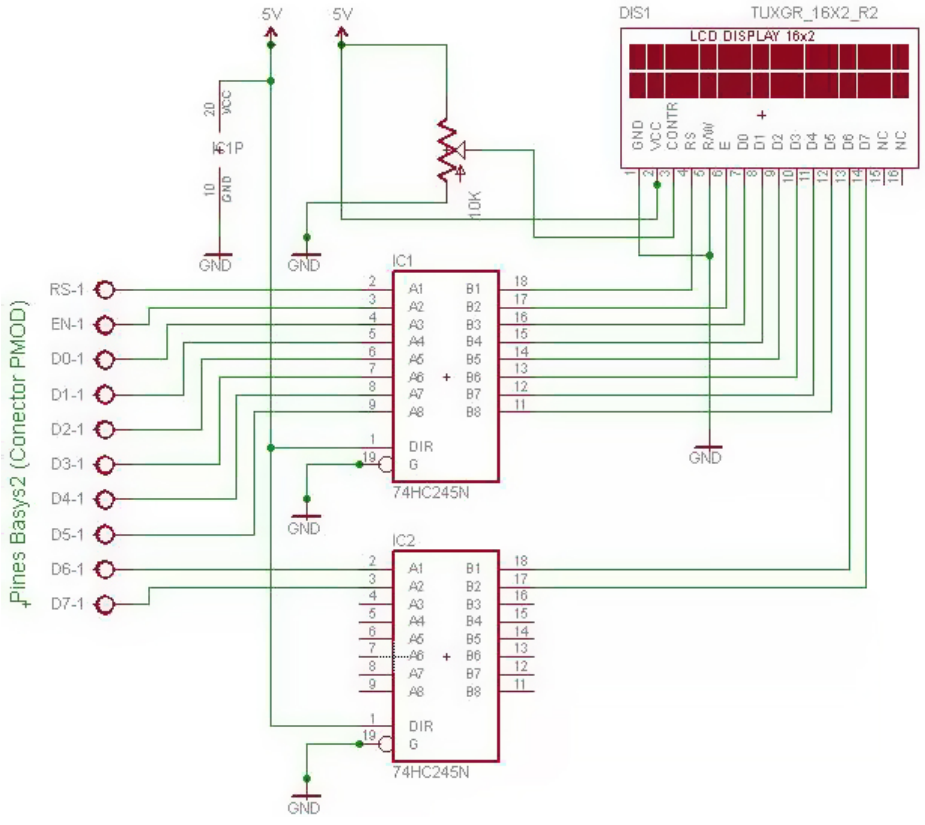


Figura 4.5. Esquema de conexión final para conexión entre LCD y tarjeta Basys 2.
Fuente: elaboración propia.

Para ajustar el contraste del LCD, el programador debe conectar los pines de la tarjeta Basys 2 desde el conector PMOD en el orden establecido en la figura anterior.

Al inicio del encendido del LCD se presenta, por especificación del fabricante, un retraso de 5 ms; luego, se comienzan a enviar los comandos de configuración del LCD, como se observa en la estructura *case* del programa. Un contador es incrementado y se realiza una *Máquina de Estados Finita (FSM)*, cada valor del contador de proceso permite realizar una determinada operación en LCD, por ejemplo, las operaciones de configuración van desde el número 5 al número 9, las operaciones de escritura de texto van desde el número 10 al número 14. Para la escritura de caracteres y en función de que el LCD *únicamente* recibe datos ASCII, cada una de las letras es tomada como un carácter y convertida a un formato de ocho bits para ser enviada hacia el LCD.

En este punto es posible modificar el código mediante la introducción de letras adicionales, de acuerdo al mensaje que se desee transmitir. Esta estructura de código puede resultar muy útil para mostrar información a un usuario según la funcionalidad de la entidad. Para este caso, se ha creado un archivo *test bench* que simula el comportamiento de la misma, mostrado en la figura 4.6. El código equivalente, por su parte, se muestra a continuación[26].

Código fuente 19. Entidad para manejo de LCD, configuración y escritura de caracteres.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY TestBench IS
END TestBench;
```

26 Para visualizar los códigos mostrados en el desarrollo del libro, véase https://github.com/VHDLUniminuto/VHDL.

```
ARCHITECTURE behavior OF TestBench IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT LCDinit
PORT(
 DataPort : OUT std_logic_vector (7 downto 0);
 EN : OUT std_logic;
 RS : OUT std_logic;
 clk : IN std_logic;
 clkout : OUT std_logic;
 Reset : IN std_logic
);
ENDCOMPONENT;


--Inputs
signal clk :std_logic:= '0';
signal Reset :std_logic:= '0';

          --Outputs
signal DataPort :std_logic_vector(7 downto 0);
signal EN :std_logic;
signal RS :std_logic;
signal clkout :std_logic;

-- Clock period definitions
constant clk_period :time:=10 ns;
constant clkout_period :time:=10 ns;

BEGIN

            -- Instantiate the Unit Under Test (UUT)
   uut: LCDinit PORTMAP(
          DataPort => DataPort,
   EN => EN,
          RS => RS,
          clk => clk,
   clkout => clkout,
          Reset => Reset
   );

   -- Clock process definitions
```

Señales de conexión a LCD (RS, EN, *Databits*) y reset de la entidad

```
  clk_process : process
begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;

  clkout_process : process
begin
                clkout <= '0';
                wait for clkout_period/2;
                clkout <= '1';
                wait for clkout_period/2;
end process;

-- Stimulus process
  stim_proc: process
begin
-- hold reset state for 100 ns.
Wait for 100 ns;
Wait for clk_period*1000;
-- insert stimulus here
wait;
endprocess;

END;
```

Definición de la señal de reloj de la entidad (*Proceso sincrónico*).

Inserción de señales de estímulo para entidad, en este caso únicamente se necesita la señal de reloj.

Fuente: elaboración propia.

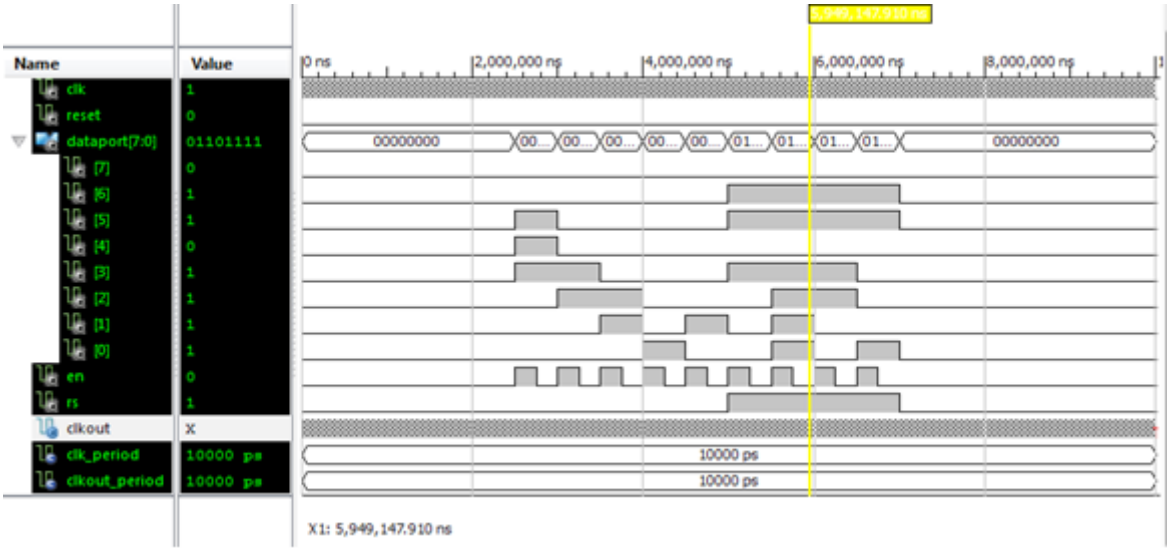


Figura 4.6. Diagrama de tiempos generado a través de test bench para entidad realizada de LCD; se aprecian las formas de onda que permiten observar el comportamiento de la misma.

Fuente: elaboración propia.

Se debe resaltar que el *test bench* fue construido con base en la señal de reloj CLK. El lector podrá cambiar las especificaciones de este reloj según las características de simulación que desee de su entidad. Como se mostró en la figura 4.4, existen varios valores de salida, los cuales corresponden a los datos de comando y caracteres enviados hacia el LCD en varios periodos de tiempo.

Ejemplo 10: máquina de estado (Finite State Machine, FSM)

Una máquina de estado es un *proceso secuencial* o *autómata* en el cual pueden intervenir las entradas o el *estado actual Q(t)* de este sistema. En general, existen dos tipos de máquinas de estado, llamadas de Moore y de Mealy, cada una con características propias de manejo: por ejemplo, en las primeras la salida depende únicamente del estado actual *Q(t)*; mientras que en las segundas depende de este último y de las entradas que la máquina tenga en un momento establecido.

Podemos definir los estados *Q(t)* como un conjunto de variables dentro del sistema que contribuyen a la evolución del mismo; si los estados son conocidos, se pueden conocer la evolución y la salida de la máquina de estado en un momento dado. Para construir dicha máquina se dispone de un gráfico que muestra su evolución, conocido como *diagrama de estado.* Cada uno de los estados puede cambiar o evolucionar en función de las entradas y del estado, en el caso de la máquina de Mealy. Para este ejemplo se tomará en cuenta este tipo de máquina, común en los sistemas secuenciales. Considérese, entonces, el esquema de la figura 4.7:

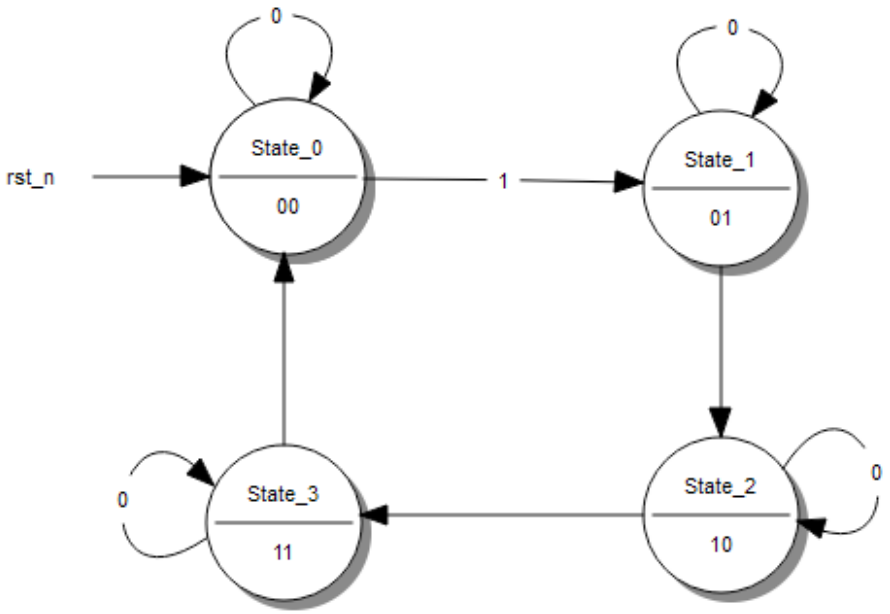


Figura 4.7. Máquina de estado tipo Mealy.
Fuente: elaboración propia.

Nótese en este caso que la máquina tiene cuatro estados, cada uno de los cuales ostenta dos salidas y evoluciona hacia otro en función del valor del estado y de entrada del sistema; un paso intermedio entre estados es conocido como *transición.* En el caso que nos ocupa, si el *reset* es 0, el primer estado es 0, con el cual la salida de la máquina es 0; si la entrada del sistema (x) es 1 (equivalente binario), el sistema cambia *o evoluciona* hacia el estado 1, en el cual la salida de la máquina de estado es 01. La evolución de esta última continúa en función de estos valores hasta llegar de nuevo al estado inicial (0).

Con esto, lo primero que debe establecerse es una gráfica de estado que permita conocer la evolución de la máquina en función de las necesidades del diseño. En este ejemplo se diseñará la máquina de estado mostrada en la figura 4.7 mediante la utilización de lenguaje VHDL, para lo cual se utilizará un concepto nuevo: la sentencia *type*; esta permite definir un tipo específico de dato que puede usarse en el desarrollo de la máquina de estado. Para conocer el manejo de esta sentencia, considérese el siguiente ejemplo:

```
type estado is (cero,uno,dos,tres);
signal es_actual,es_futuro:estado;
```

En este caso, el tipo de dato que se usará se denomina estado: para este caso, definimos el estado *cero, uno, dos, tres*. La sentencia tiene asociada un conjunto de señales que establecen la evolución de la máquina de estado (estados actual y futuro). A continuación, se muestra la entidad en términos de entradas y salidas del sistema.

Figura 4.8. Esquema general de la entidad FSM generada en VHDL.
Fuente: elaboración propia.

A partir de estas premisas se presenta a continuación el código de esta máquina de estado. Nótese que en el mismo se toman en cuenta las entradas, salidas y estados del sistema, así como las sentencias condicionales para validar los estados actual y futuro.

Código fuente 20. Entidad de máquina de estado de figura 4.7, realizada a través de VHDL.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

Librerías

```
entity FSM is
Port( clk : in STD_LOGIC;
        salida : out STD_LOGIC_VECTOR (1 downto 0);
        x : in STD_LOGIC);
end FSM;
```

Declaración de entradas y salidas

```
architecture Behavioral of FSM is
type estado is (cero,uno,dos,tres);
signal es_actual,es_futuro:estado;
begin
```

Declaración de estados

```
process(clk)--Cambio de estado (Retraso).
variable counter: integer range 0 to 100000001;

begin
if (clk='1' and clk'event) then
counter:=counter+1;--contador.
if (counter>100000000) then--El incremento de la
variable debe validarse siempre de manera sincrónica.
   es_actual<=es_futuro;
counter:=0;
end if;
end if;
```

Cambio de estado (cada segundo)

```
end process;

process(x,es_actual)--maquina de estado
begin

case es_actual is
when cero=>
salida<="00";
if (x='1')then
      es_futuro<=uno;
else
      es_futuro<=cero;
end if;
when uno=>
      salida<="01";
es_futuro<=dos;
when dos=>
      salida<="10";
      es_futuro<=tres;
when tres=>
      salida<="11";
      es_futuro<=cero;
end case;


end process;

end Behavioral;
```

Validación de cada estado en función de las entradas y del estado actual *Q(t)*

Fuente: elaboración propia.

Como se dijo, cada estado se describe al inicio del programa mediante la sentencia *type*. La validación de los estados se da en función de las entradas y del estado actual de la máquina de estado, y la evolución hacia un estado posterior se realiza a través de un retraso —en este caso, de dos segundos— de acuerdo a la variable *counter*, la cual se incrementa cuando existe un evento de reloj (en este ejemplo, un flanco de subida). Al estado actual se le asigna el estado futuro cada vez que hay una transición, de acuerdo a las condiciones establecidas. La simulación a través del archivo *test bench* respectivo se muestra a continuación:

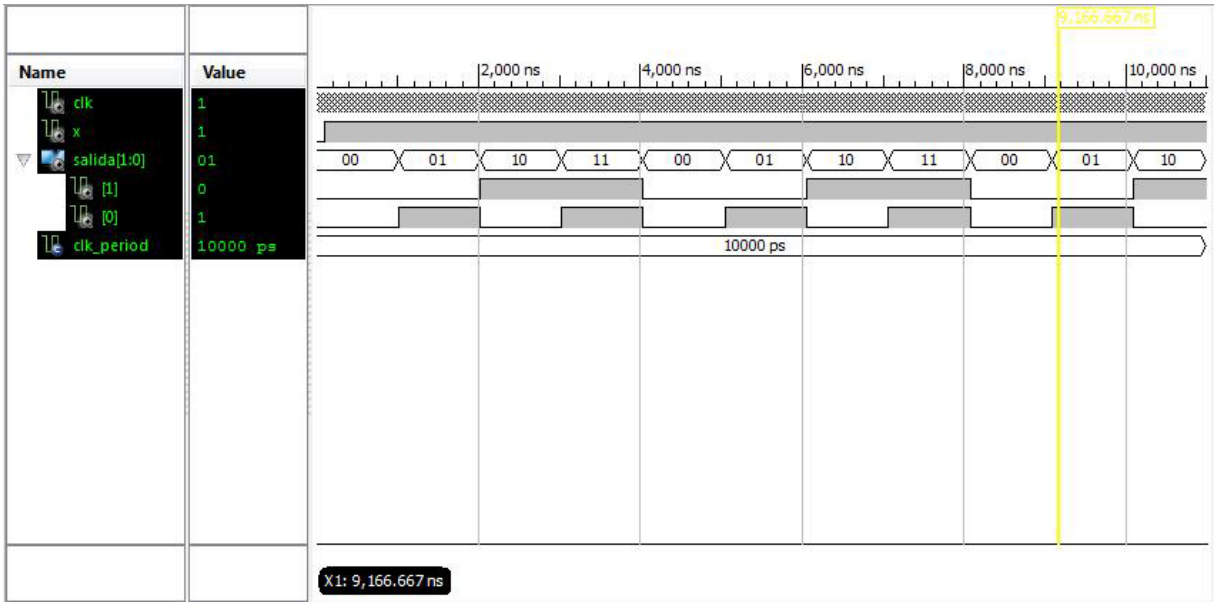


Figura 4.9. Simulación generada mediante test bench para entidad del ejemplo 2.
Fuente: elaboración propia.

Obsérvese que, en este caso, el valor de x=1 inicializa la máquina para ejecutar la secuencia establecida.

Ejemplo 11: secuencia neumática mediante VHDL

En este ejemplo se hace uso de las nociones de máquina de estado mencionadas; el objeto del ejercicio es realizar una secuencia neumática como parte de un proceso industrial. Para lograrlo, se propone el siguiente esquema neumático:

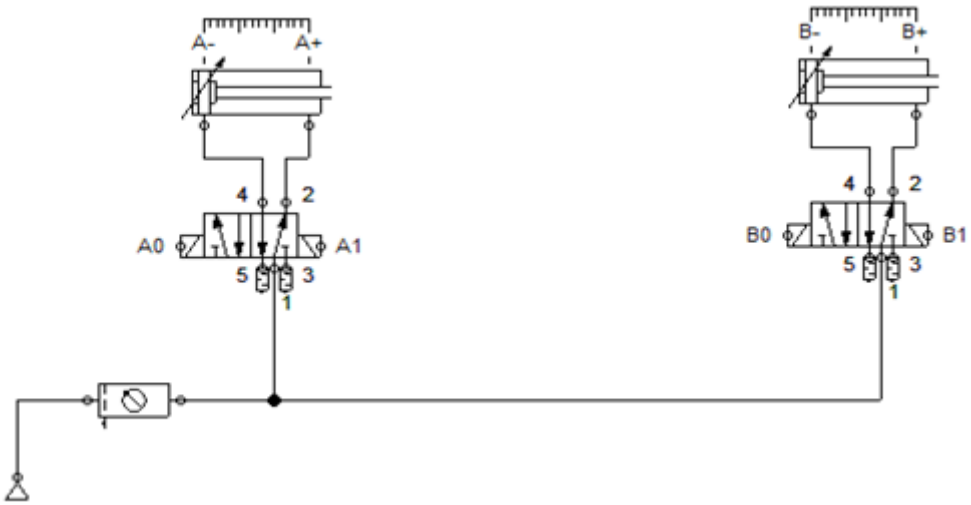


Figura 4.10. Esquema general neumático propuesto en el ejemplo 3.
Fuente: elaboración propia.

La secuencia propuesta para este ejemplo se describe mediante A+B+B-A-. En este caso, el símbolo "+" indica la salida del vástago de cada cilindro: así, el primer cilindro sale, seguido del segundo; luego entra el segundo, y por último entra el primero. La secuencia se repite de forma continua. Cabe resaltar que cada válvula neumática tiene un accionamiento eléctrico establecido mediante los solenoides definidos como *A0, A1, B0* y *B1.* Si uno de ellos es activado, la válvula cambia de estado debido a su carácter *biestable,* o de dos estados.

La detección de las posiciones de los cilindros (mínima y máxima) se realiza con base en finales de carrera colocados en los puntos mínimos y máximos de cada uno. Dicho sistema se puede modelar

con una máquina de estado que cumpla la secuencia definida. Esta última tiene cuatro estados y la transición entre cada uno depende de la posición de los sensores.

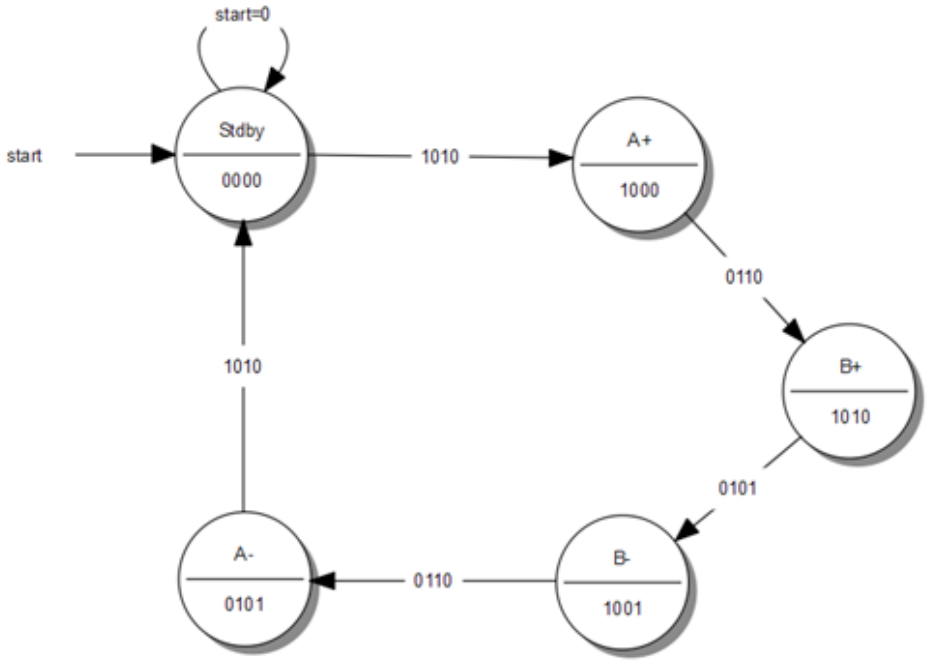


Figura 4.11. Diagrama de estado para ejemplo 11.
Fuente: elaboración propia.

En este caso, la máquina de estado tiene una señal de entrada denominada *start*. Si este bit de entrada a la máquina de estado es 1, el proceso comenzará con la realización de la secuencia estipulada. Las transiciones entre cada estado corresponden a los valores de los sensores (*finales de carrera*) colocados en los cilindros del sistema neumático; así entonces, un valor de 1 indica que el vástago se encuentra en esta posición. Por ejemplo, si se considera la transición entre los estados 0 —*standby*— y 1 —A+—, los sensores deben estar en la posición A- y B-, con lo que se indica que ambos vástagos están en las posiciones iniciales (figura 4.11). Una vez se han dado las consideraciones iniciales en el manejo del sistema neumático, se presentan la entidad y el código VHDL que permiten implementar la máquina de estados establecida en la figura 4.12.

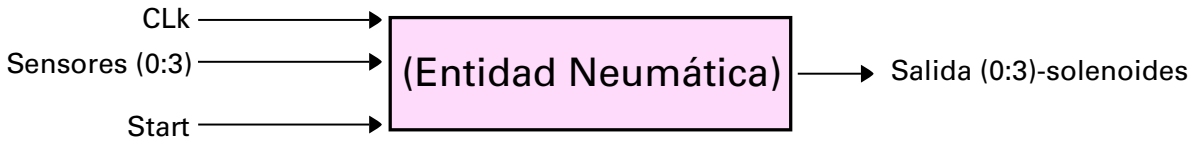


Figura 4.12. Entidad para máquina de estado de secuencia neumática de ejemplo 11.
Fuente: elaboración propia.

Código fuente 21. Entidad de máquina de estado de figura 4.11, realizada a través de VHDL.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration
--if instantiating.
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Neumatica is
Port( Sensores : in STD_LOGIC_VECTOR (3 downto 0);
      Start : in STD_LOGIC;
      Salida : out STD_LOGIC_VECTOR (3 downto 0);
      clk :in STD_LOGIC);
end Neumatica;

architecture Behavioral of Neumatica is
type estado is (standby,amas,bmas,bmenos,amenos);--Definición de los estados de la máquina.
signal estado_actual, estado_futuro: estado;--Definición de estado actual y estado futuro.

begin

process(clk,Start)--Definición de las entradas de Mealy, para este
--caso clk y start.
begin
if (clk'event and clk='1') then--Evento de reloj, flanco de subida.
If (Start='1') then
estado_actual<=estado_futuro;--Cambio de estado asociado a transiciones.
else
estado_actual<=standby;--Si el bit de start es cero, el estado es Standby.
end if;
end if;
end process;

process (estado_actual,Sensores)
begin
```

Definición de librerías

Definición de entidad

```
case estado_actual is--Definición de estados y transiciones de máquina de
--estado.
when standby=>
        Salida<="0000";
If (Sensores="1010") then
estado_futuro<=amas;
else
estado_futuro<=standby;
end if;
when amas=>
Salida<="1000";
If (Sensores="0110") then
estado_futuro<=bmas;
else
        estado_futuro<=amas;
end if;
when bmas=>
Salida<="1010";
If (Sensores="0101") then
        estado_futuro<=bmenos;
else
        estado_futuro<=bmas;
end if;
when bmenos=>
        Salida<="1001";
If (Sensores="0110") then
        estado_futuro<=amenos;
else
        estado_futuro<=bmenos;
end if;
when amenos=>
if (Sensores="1010")then
estado_futuro<=standby;
else
estado_futuro<=amenos;
end if;
end case;


end process;
end Behavioral;
```

Definición de estados y transiciones (en este caso, a+, b+,b-,a-, como se definió en el diagrama de estado).

Fuente: elaboración propia.

En este programa, las entradas de la entidad corresponden a las señales de reloj, sensores e inicio, mientras que las señales de salida manejan directamente los solenoides de las válvulas neumáticas. Para establecer este accionamiento, considérese el siguiente circuito de aislamiento entre la parte de control y la parte de potencia, que en este caso corresponde a los solenoides de las válvulas (figura 4.13). Debe resaltarse que estos últimos trabajan a 24 voltios.

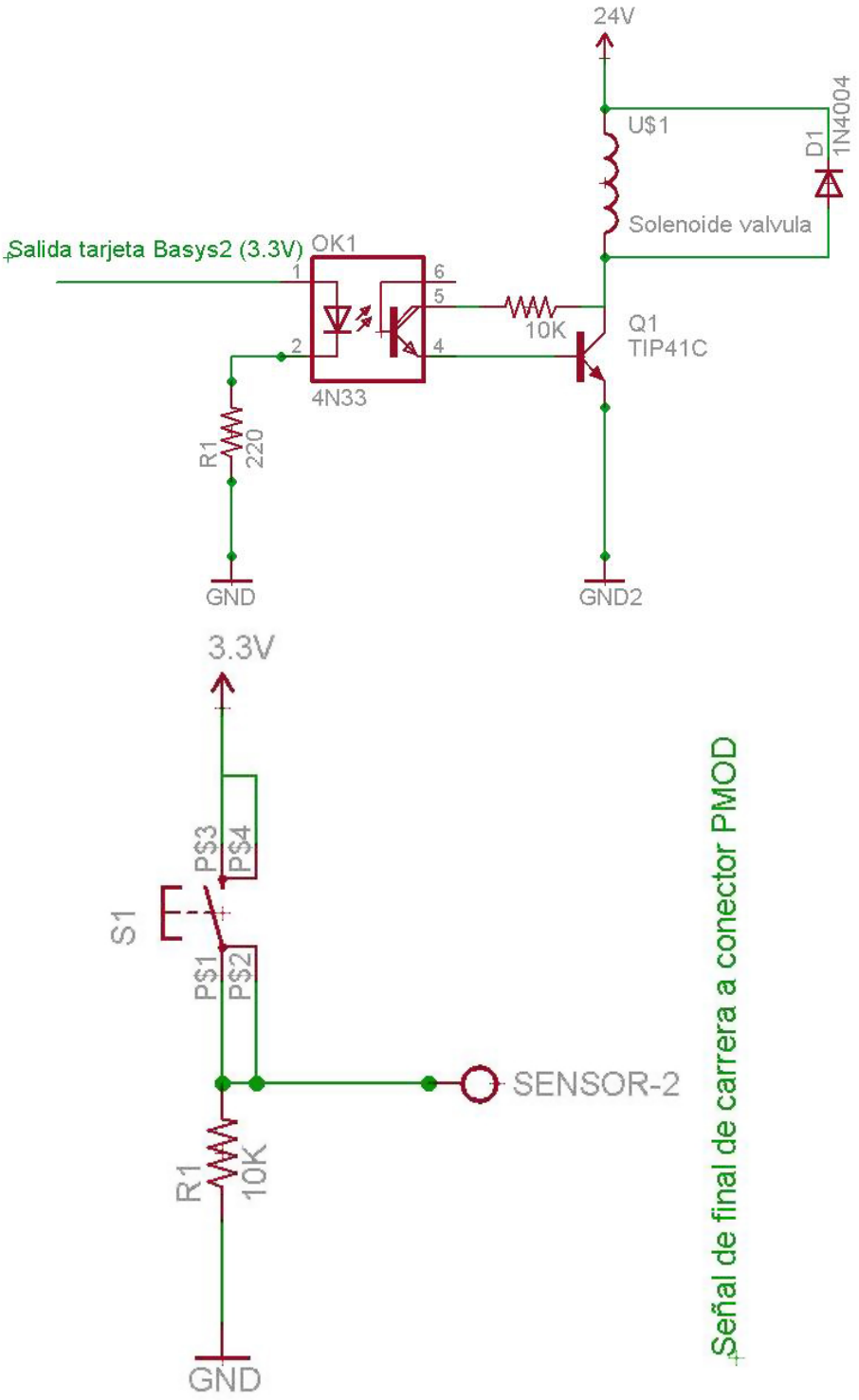


Figura 4.13. Esquema de acondicionamiento de etapa de potencia (arriba) y esquema de conexión de final de carrera (abajo) hacia tarjeta Basys 2.

Fuente: elaboración propia.

La etapa contiene un optoacoplador y un transistor que conmuta el solenoide de la válvula neumática, permitiendo que esta cambie de estado. Este circuito hace posible una interfaz adecuada entre las señales de la tarjeta Basys 2 de 3,3 voltios y los solenoides (cuyo voltaje de operación en este caso es de 24 voltios, como se mencionó). Para la conexión de los sensores (finales de carrera) se establece el esquema mostrado en la parte inferior de la figura 4.13; y el *switch* es presionado cuando el vástago se encuentra en la posición, con lo que genera un 1 lógico sobre el pin seleccionado de la tarjeta. Esta máquina de estado provee una funcionalidad adecuada para sistemas industriales, hecho que posibilita otro tipo de usos para una FPGA. El resultado del archivo de simulación *test bench* se muestra en el código fuente 22; al respecto, nótese que se establece la máquina de estado de acuerdo al código definido anteriormente.

Código fuente 22. Esquema de *test bench* para ejemplo 11, secuencia neumática.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY test IS
END test;

ARCHITECTURE behavior OF test IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT Neumatica
PORT(
        Sensores : IN std_logic_vector(3 downto 0);
        Start : IN std_logic;
        Salida : OUT std_logic_vector(3 downto 0);
        clk : In std_logic
);
ENDCOMPONENT;
```

```
--Inputs
signal Sensores :std_logic_vector(3 downto 0):=(others=> '0');
signal Start :std_logic:= '0';
signal clk :std_logic:= '0';

          --Outputs
signal Salida :std_logic_vector(3 downto 0);

-- Clock period definitions
constant clk_period :time:=10 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
uut: Neumatica PORTMAP(
        Sensores => Sensores,
Start => Start,
        Salida => Salida,
        clk => clk
);

-- Clock process definitions
  clk_process :process


begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;


-- Stimulus process
  stim_proc:process
begin
-- hold reset state for 100 ns.
wait for 100 ns;
```

Periodo de reloj de 10 ns; el lector lo puede ajustar a sus necesidades.

Definición de las señales de la máquina de estado de proceso neumático

```
wait for clk_period*10;
Start<='1';
Sensores<="1010";
wait for clk_period*10;
Sensores<="0110";
wait for clk_period*10;
Sensores<="0101";
wait for clk_period*10;
Sensores<="0110";
wait for clk_period*10;
Sensores<="1010";
-- insert stimulus here

wait;
end process;

END;
```

Estimulo de la entidad. En este punto se realiza el cambio de las señales que participan en la evolución de la máquina de estado (en este caso, las señales *start* y sensores).

Fuente: elaboración propia.

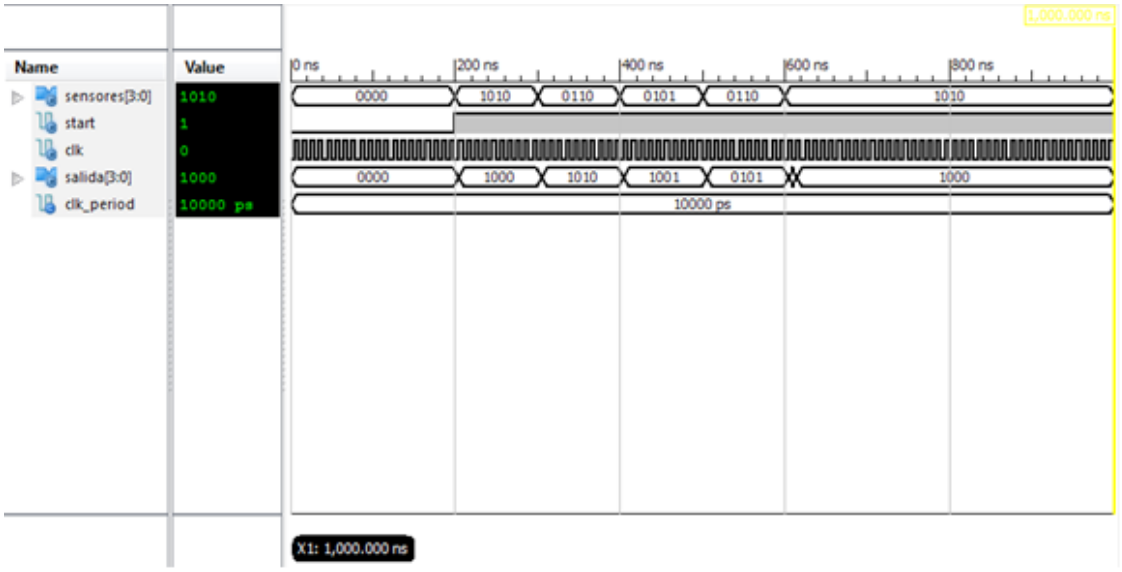


Figura 4.14. Diagrama de tiempos generado a partir del test bench del código fuente 22; se puede visualizar el funcionamiento adecuado de la máquina de estado de acuerdo a los criterios de diseño establecidos.
Fuente: elaboración propia.

Ejemplo 12: termómetro

Este ejemplo pretende servir como introducción al manejo de dispositivos de Conversión Analógica-Digital (ADC).

Muchas de las señales provenientes de los procesos físicos coexisten de forma analógica. Estas son convertidas en señales eléctricas a través del proceso conocido como transducción, por medio de un sensor de la variable establecida. Para este ejemplo se hará uso del conversor análogo digital ADC0804, junto con el sensor de temperatura LM35 para la creación de un termómetro digital; y la visualización se realizará a través del *display* de la tarjeta Basys 2. El conversor ADC0804, por su parte, es un dispositivo que permite la conversión de una señal análoga en un equivalente digital de 8 bits. Es decir, si consideramos el voltaje de referencia en un valor de 5 voltios, el valor de cada bit estará definido mediante el concepto de resolución establecido por la siguiente ecuación:

$$Res = \frac{5V}{2^8 bits} = \frac{19.5mV}{bit} \quad (1)$$

Lo anterior significa que por cada bit se obtiene un valor de 19,5 mV: en este caso, tal resolución es suficiente para el sistema a implementar.

El proceso de conversión de un valor de análogo a digital se da por medio de tres pasos, a saber muestreo, codificación y cuantificación.

- Muestreo: la señal es *muestreada,* es decir, se toma una muestra para ser enviada hacia el bloque de codificación.
- Codificación: se asigna un valor binario a la señal.
- Cuantificación: se asigna un valor *correspondiente* entre el valor de la muestra tomada y el código binario establecido.

Típicamente, los conversores ADC tienen la configuración de aproximaciones sucesivas (SAR) y están definidos de acuerdo al número

de bits que pueden manejar —8, 10, 12, 24—; dicho de otro modo, la señal de salida se da en un total de bits definido previamente. Para la implementación de este sistema, considérese el siguiente esquema general del dispositivo ADC0804.

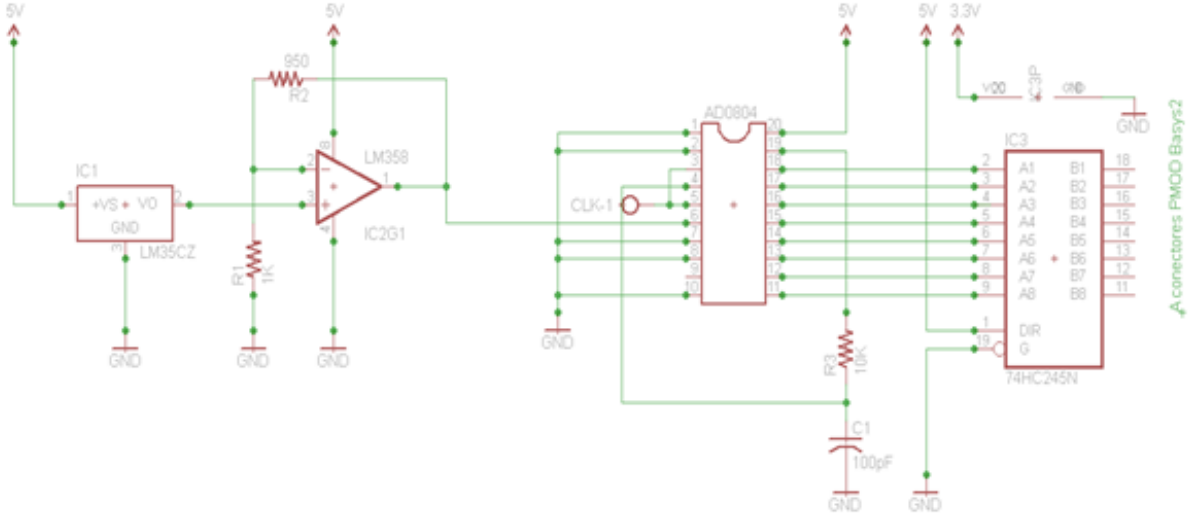


Figura 4.15. Esquema de conexión de integrado ADC0804 y sensor LM35.
Fuente: elaboración propia.

Para este caso se ha dado una ganancia al sensor LM35 de 1,95, lograda a través del amplificador operacional LM358, que permite establecer una relación directa entre el equivalente de voltaje y el equivalente de código digital; por ejemplo, si la temperatura percibida es 26 °C, el valor de salida del conversor análogo-digital será 26 en formato binario, es decir, 00011010. Este equivalente es detectado por la entidad y convertido al formato adecuado para representarse mediante el conjunto de *displays* de 7 segmentos de la tarjeta Basys 2. El conversor análogo-digital necesita una señal de actualización para la conversión, garantizada con una frecuencia de 1 KHz y generada por la entidad creada mediante código VHDL. De igual forma, se dispone un cambiador de nivel de señales de 5V a 3,3 V a través del integrado 74HC245. El diagrama de flujo para este programa se muestra en la siguiente figura:

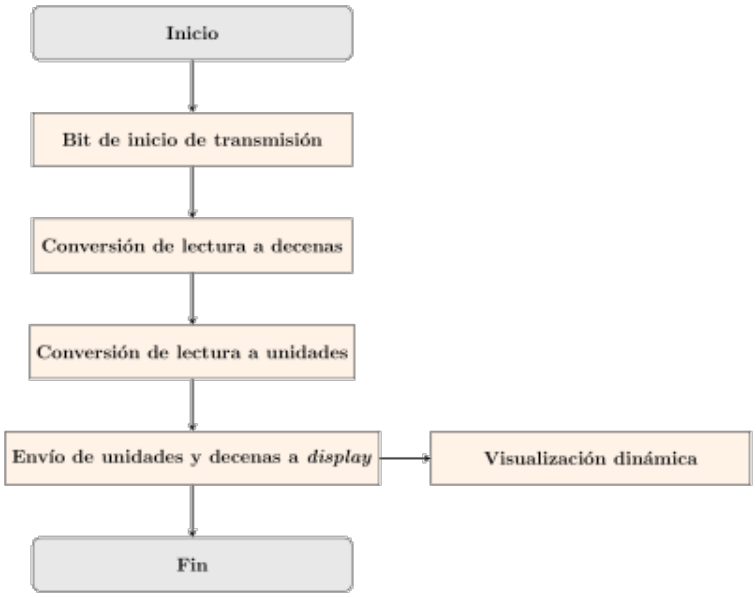


Figura 4.16. Diagrama de flujo de ejemplo propuesto.
Fuente: elaboración propia.

El código de la entidad, por su parte, se muestra a continuación.

Código fuente 23. Descripción de entidad para termómetro, de acuerdo a la estructura mostrada en la figura 4.16.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Termometer is
Port( Display : out STD_LOGIC_VECTOR (6 downto 0);
Enable : out STD_LOGIC_VECTOR (3 downto 0);
                ValTemperatura: in STD_LOGIC_VECTOR (7 downto 0);
--Este es el valor de temperatura previamente convertido del
--AD0804.
clk : in STD_LOGIC;
        Reset : in STD_LOGIC;
        clkAD : out STD_LOGIC);
end Termometer;

architecture Behavioral of Termometer is

begin

P1:process(clk, reset)--Proceso principal de entidad.
variable Temperatura: integer range 0 to 150:=0;--Temperatura máxima sobre LM35
variable Unidades: integer range 0 to 9;--Variable que guarda los valores de temperatura.
variable Decenas:  integer range 0 to 9;--Variable que guarda decenas de temperatura
```

Declaración de pines de entrada y salida (entidad)

```
variable Centenas: integer range 0 to 9;--Variable que guarda decenas de temperatura
variable IteracionC: integer range 0 to 1;--Iteración indica los procesos que deben realizarse
Centenas.
variable IteracionD: integer range 0 to 1;--Iteración indica los procesos que deben realizarse -------
-- Centenas.
variable Visualizacion: integer range 0 to 50000000;--Contador para visualización dinámica en ----
--Display.
variable Temporal: integer range 0 to 12;--Variable temporal que tendrá decenas o unidades.
variable cicloADC: integer range 0 to 50001;--Variable para ciclo de reloj de conversor Análogo----
--Digital.
begin

if (clk='1' and clk'event) then--Permite establecer un sistema de lógica secuencial validando el
estado del reloj.
        if (Reset='1') then--Reinicio de entidad.
         Temperatura:=0;
         Centenas:=0;
         Decenas:=0;
         Unidades:=0;
         Temporal:=0;
        end if;

--Bloque de conversión de conversor Análogo-Digital 0804. A este se   debe asignar un ciclo de
--reloj para convertir el dato de entrada.

        CicloADC:=CicloADC+1;
        If (CicloADC>25000) then
        clkAD<='1';
        else
        clkAD<='0';
        end if;

        if (CicloADC>50000) then
        CicloADC:=0;
        end if;

        Temperatura:=to_integer(unsigned(ValTemperatura));--Convertir el valor de temperatura --
--en variable.
--Estos bucles permiten indicar el valor de temperatura discriminado en unidades, decenas y --------
--centenas.
        if (Temperatura<100 and Temperatura>=10) then--Pregunta el rango entre 10 y 99
        --IteracionD:=1;
          bucle2: for b in 1 to 10 loop--Esta iteración permite establecer las centenas de ----------
---temperatura.
                Temperatura:=Temperatura-10;
                Decenas:=Decenas+1;
                if Temperatura<10 then
                exit bucle2;
                end if;
                end loop;
                end if;

        if (Temperatura>=0 and Temperatura<10) then
```

Señal de actualización para ADC 0804 con frecuencia de 1 KHz

```
            Unidades:=Temperatura;
            end if;
--Fin de bucle.
--Inicio de bucle de visualización dinámica.
Visualizacion:=Visualizacion+1;--Incrementar contador Display  en una unidad.
            If (Visualizacion<250000and Visualizacion>0) then
                        Temporal:=Decenas;
                        If (Decenas=0) then
                        Enable<="1111";-- Activar Display1
                        else
                        Enable<="1110";-- Activar Display1
                        end if;
                        Decenas:=0;
                        end if;

            if (Visualizacion<500000and Visualizacion>250000) then
                        Temporal:=Unidades;
                        Enable<="1101";--Activar Display 2.
                        Unidades:=0;
                        end if;

            if (Visualizacion<750000and Visualizacion>500000) then
                        Temporal:=10;
                        Enable<="1011";--Activar Display 2.
                        end if;

            if (Visualizacion<1000000and Visualizacion>750000) then
                        Temporal:=11;
                        Enable<="0111";--Activar Display 2.
                        end if;

            if (Visualizacion>1000000) then
                                    IteracionD:=0;
                                    IteracionC:=0;
                                    Visualizacion:=0;--Reiniciar contador.
                                    Temperatura:=0;
                                    end if;

            case Temporal is--abcdefg
                                    when 0=>Display<="0000001";
                                    when 1=>Display<="1001111";
                                    when 2=>Display<="0010010";
                                    when 3=>Display<="0000110";
                                    when 4 =>Display<="1001100";
                                    when 5=>Display<="0100100";
                                    when 6=>Display<="1100000";
                                    when 7=>Display<="0001111";
                                    when 8=>Display<="0000000";
                                    when 9=>Display<="0001100";
                                    when 10=>Display<="0011100";
                                    when 11=>Display<="0110001";
                                    when others=>null;
                                    end case;

end if;
end process;

end Behavioral;
```

Conversión de valor binario leído en variable temperatura a *display* de 7 segmentos

Fuente: elaboración propia.

Para la visualización de la temperatura se ha convertido la variable en unidades y decenas, según el caso. Para este proceso, la variable temperatura se resta en un total de 10, siempre y cuando se encuentre en el rango de las decenas; es decir, si la variable de temperatura está en el rango de 10 a 99, el proceso de resta de 10 continúa. El número de veces que este proceso se realiza es el número de las decenas: por ejemplo, si se considera el número 35 como valor leído de temperatura, el proceso de resta se realiza tres veces, lo cual equivale al número en la casilla de decenas; el valor restante, 5 en este caso, correspondería a las unidades, es decir, 35-10-10-10-**5**, donde el número de veces del valor 10 indica las decenas y el valor de 5, el de unidades.

Los valores anteriores se envían por separado para visualizarse a través del mencionado proceso de visualización dinámica. Este último permite *multiplexar* el valor de tiempo con el fin de enviar el dato de los valores de unidades y decenas de forma individual en un periodo de tiempo establecido (debido a que los puntos correspondientes a los segmentos del *display* están conectados entre sí), con un punto de activación (*enable*) diferente para cada *display*. También debe resaltarse que las señales de este *display* se activan mediante ceros lógicos debido a su naturaleza de ánodo común. Se resalta en este proceso el uso de la sentencia secuencial *case*, además de *for-loop.* Una vez el número de veces del bucle de iteración de decenas se ha completado, la sentencia *exit* fuerza la salida del bucle, lo que hace posible continuar con el proceso de unidades y de visualización dinámica. El archivo de simulación *test bench* respectivo se muestra a continuación.

Código fuente 24. Descripción test bench para entidad para termómetro.

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY test IS
END test;
ARCHITECTURE behavior OF test IS

-- Component Declaration for the Unit Under Test (UUT)
COMPONENT Termometer
PORT(
        Display : OUT std_logic_vector (6 downto 0);
        Enable : OUT std_logic_vector (3 downto 0);
        ValTemperatura : IN std_logic_vector (7 downto 0);
        clk : IN std_logic;
        Reset : IN std_logic;
        clkAD : OUT std_logic;
        AlarmaPin : OUT std_logic;
        Alarma : OUT std_logic

);
END COMPONENT;


--Inputs
signal ValTemperatura : std_logic_vector (7 downto 0):=(others=> '0');
signal clk : std_logic:= '0';
signal Reset : std_logic:= '0';

            --Outputs
signal Display : std_logic_vector (6 downto 0);
signal Enable : std_logic_vector (3 downto 0);
signal clkAD : std_logic;

-- Clock period definitions
constant clk_period :time:=10 ns;
constant clkAD_period :time:=10 ns;

BEGIN

            -- Instantiate the Unit Under Test (UUT)
   uut: Termometer PORTMAP(
          Display => Display,
Enable => Enable,
          ValTemperatura => ValTemperatura,
          clk => clk,
          Reset => Reset,
          clkAD => clkAD
```

Declaración de señales de entidad

```
-- Clock process definitions
  clk_process :process
                begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;

  clkAD_process :process
                begin
                clkAD <= '0';
                wait for clkAD_period/2;
                clkAD <= '1';
                wait for clkAD_period/2;
end process;
-- Stimulus process
  stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 100 ns;

wait for clk_period*10;
ValTemperatura<="00011010";
wait for clk_period*1000;

 -- insert stimulus here
 wait;
 end process;
 END;
```

Definición de reloj de sistema

Cambio de señales necesarias en el comportamiento de la entidad

Fuente: elaboración propia.

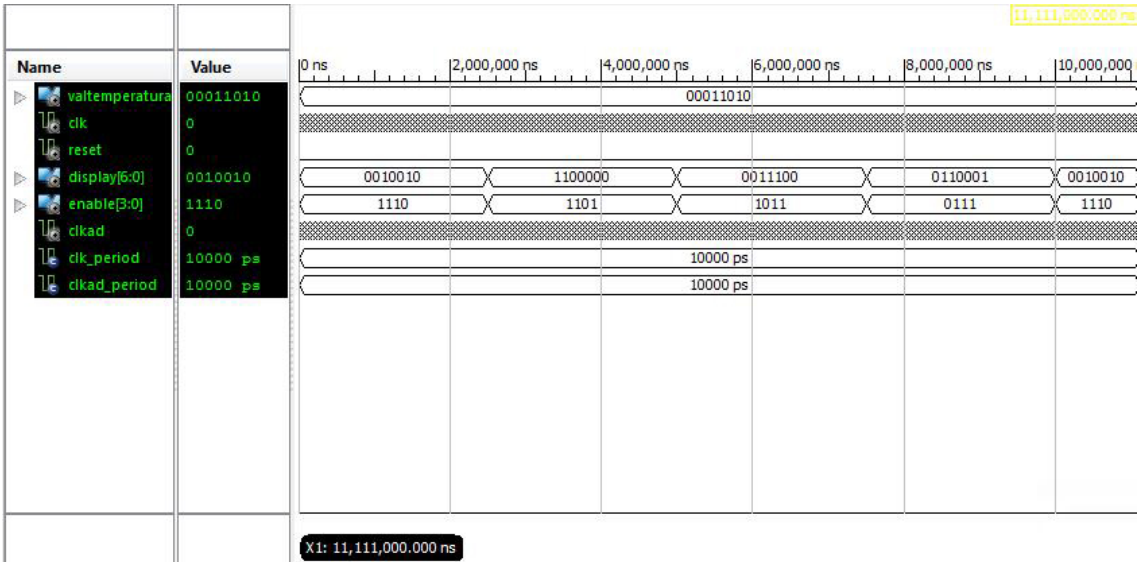


Figura 4.17. Diagrama de tiempos generado a partir de *test bench* de código fuente 22; se puede observar el valor establecido de temperatura (26 °C) sobre los *displays* de siete segmentos.

Fuente: elaboración propia.

Obsérvese que, en el ejemplo que nos ocupa, un valor de entrada de 26°C se visualiza en orden en los *displays* según el valor del vector *enable.* En este caso, y de acuerdo a la visualización dinámica, cada valor se envía de forma individual: por ejemplo, en un *display* de ánodo común, el valor binario 0010010 equivale al número 2 (comenzando con el segmento a, b, c…g de izquierda a derecha); 1100000 corresponde al valor de 6; 0011100 corresponde al valor "0"; y 0110001 corresponde a un valor de C. Así, estos valores se plasman sobre los cuatro *displays* de la tarjeta Basys 2, generándose con ello el dato de temperatura. En el ejemplo, la visualización se realiza en una escala de 1 °C sin valores decimales.

Ejemplo 13: Universal Asynchronous Receiver Transmitter (UART)

Uno de los estándares de comunicaciones de uso común en los ámbitos industrial, de domótica y académico, entre otros, es RS-232, en el cual la información se envía de manera *serial.* Cada trama de datos es enviada a una velocidad específica cuya unidad es el baudio, o símbolos por segundo. Los datos se envían a través de dos pines específicos de transmisión y recepción, y a esa cadena se adicionan algunos bits para el chequeo de errores en transmisión o recepción —conocidos como *bits de paridad*—. El esquema general de una trama de datos en este estándar se puede observar en la siguiente figura:

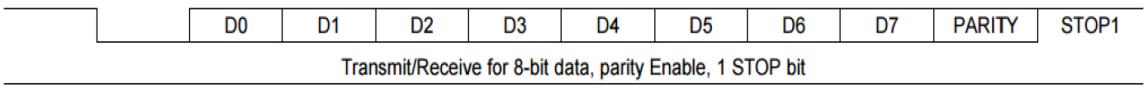


Figura 4.18. Esquema de trama de datos en estándar RS-232.
Fuente: elaboración propia.

Para el inicio de la transmisión de datos, el bus cambia su estado de un 1 lógico a un 0 lógico en un tiempo determinado por la velocidad de transmisión, que recibe el nombre de *tiempo de bit*. Una vez este bit es 0, el receptor comprende que una trama de datos está por transmitirse. Generalmente, los datos se envían desde el bit menos significativo (LSB) hacia el más significativo (MSB), denominados en la figura 4.18 como D0 y D7, respectivamente. Además, se puede agregar un bit de paridad para indicar al receptor posibles errores en la transmisión de datos. La secuencia termina con un bit de parada, en el cual el bus retorna a su estado original de 1 lógico. La transmisión y recepción de datos se realiza normalmente en un formato de 8 bits; sin embargo, existen métodos de transmisión de 7 y 9 bits. En este último modo, la transmisión y recepción se realiza a dispositivos *direccionados*, es decir, aquellos que dentro de una red contienen una dirección fija.

Cada trama de datos de 8 bits se almacena de forma temporal en una unidad conocida como *First in-First Out* (FIFO), en la cual el primer dato que entra a la unidad es el primero en salir; esta se establece a partir de *registros de desplazamiento.* Así, un módulo UART está formado a partir de tres elementos generales:

- Unidad FIFO
- Pines de transmisión (TX) y recepción (RX)
- Generador de tasa de baudios (velocidad de transmisión-recepción de datos)

A partir de estas premisas, este ejemplo considerará la construcción de una entidad que permita la transmisión de datos a una tasa especifica de 9.600 baudios: esto significa que el tiempo de bit es $T_{bit} = \frac{1}{9600 baudios} = 104\mu S.$

Otra consideración consiste en el desplazamiento de los datos para ser enviados de manera serial: al respecto, VHDL tiene un conjunto de instrucciones que permiten desplazamientos de bits a la derecha y a la izquierda: *shift right logical* (srl) y *shift left logical* (sll), respectivamente. Este tipo de operaciones se dan para datos de tipo *unsigned*, con lo cual la declaración del vector de datos debe hacerse de acuerdo a este último. Así, la creación de la entidad debe tener en cuenta la elaboración de una unidad para la generación de la velocidad de transmisión de datos y una unidad FIFO que permita el desplazamiento de los datos de manera serial, respetando el tiempo de bit ya descrito. Para la generación de la tasa de velocidad en baudios se tomará un retraso de 5208, determinado con base en la siguiente expresión (tomando el reloj de la Basys 2 con una frecuencia de 50 MHz):

$$Valor\ retraso = \frac{\frac{1}{9600 baudios}}{\frac{1}{50MHz}} = 5208 \quad (2)$$

Para la transmisión de datos estos son convertidos a formato ASCII usando la sentencia **to_unsigned (**carácter '**pos(**'H'**),**8**),** en la cual la letra entre comillas es convertida en formato ASCII, para ser enviada al receptor en este caso un computador. Para este caso se recomienda para la transmisión de datos usar un conversor serial a *USB*, por ejemplo, el módulo CP2102 o el módulo FT232RL.

La estructura general de la entidad UART se muestra a continuación:

Figura 4.19. Estructura de entidad UART para transmisión de datos.
Fuente: elaboración propia.

Para este caso, el diagrama de flujo de este programa tendrá la estructura que se muestra en la figura 4.20.

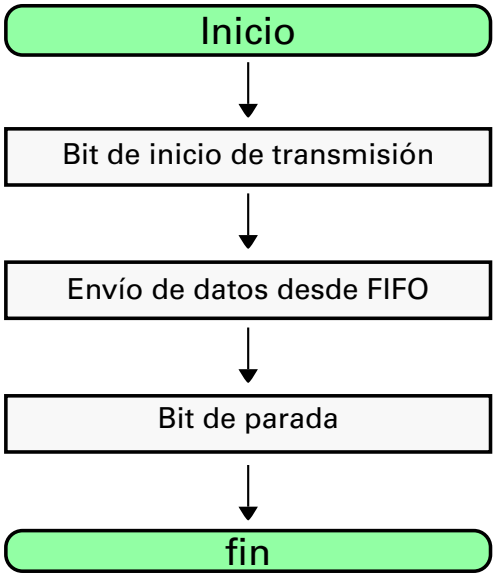


Figura 4.20. Diagrama de flujo de programa UART en VHDL.
Fuente: elaboración propia.

El código general de esta entidad mediante VHDL se muestra a continuación.

Código fuente 25. UART en VHDL.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity UARTmodule is
Port( clk : in STD_LOGIC;
      BaudRatePin : out STD_LOGIC;
      Reset : in STD_LOGIC;
      TX : out STD_LOGIC);
end UARTmodule;
```

Definición de pines de la la entidad.

```
architecture Behavioral of UARTmodule is
signal Comando: unsigned (7 downto 0);--Vector de datos a ser enviado. comando debe ser -------
---unsigned.
begin
P1:process(clk, Reset)--Proceso de logica secuencial.
variable Contadorbits: Integer range 0 to 13:=0;--Variable de conteo de bits para FIFO. Los bits de
inicio y stop .
variable CheckTransmision: integer range 0 to 1:=0;--Variable que indica si se esta estableciendo
--una transmisión
variable Baudrate: integer range 0 to 5208;--Delay para un Baudrate de 9600 bits, Formula ---------
--(1/9600)/(1/50Mhz)=5208
variable Delay: integer range 0 to 26000001:=0;--Delay transmisión. 500ms
variable ContadorPalabras: integer range 0 to 10:=0;--Contador rango 0 a 10.
begin

if (clk='1' and clk'event) then
if (Reset='1') then--Reiniciar todas las variables de
--proceso
CheckTransmision:=0;
Comando<="00000000";
TX<='1';--Estado inicial del proceso
Contadorbits:=0;
Delay:=0;
end if;
--Bloque de espera de 50ms para comenzar transmision
If (Delay>2500000) then--Delay inicial. 50ms startup
CheckTransmision:=1;
Delay:=0;--Reset delay
case ContadorPalabras is
when 0=>Comando<=to_unsigned (character 'pos('H'),8);
when 1=>Comando<=to_unsigned (character 'pos('o'),8);
when 2=>Comando<=to_unsigned (character 'pos('l'),8);
when 3=>Comando<=to_unsigned (character 'pos('a'),8);
when 4=>Comando<=to_unsigned (character 'pos(' '),8);
when 5=>Comando<=to_unsigned (character 'pos('M'),8);
when 6=>Comando<=to_unsigned (character 'pos('u'),8);
when 7=>Comando<=to_unsigned (character 'pos('n'),8);
when 8=>Comando<=to_unsigned (character 'pos('d'),8);
when 9=>Comando<=to_unsigned (character 'pos('o'),8);
when 10=>Comando<=to_unsigned (10,8);
when others=>null;
end case;

ContadorPalabras:=ContadorPalabras+1;
else
if (CheckTransmision=0) then
Delay:=Delay+1;--Incremento de Delay
CheckTransmision:=0;
end if;
end if;
```

Estado inicial de bus para transmisión de datos.

Datos a ser Transmitidos, Palabra ***"Hola mundo"***.

Contador de dígitos para este caso 1 bit de inicio, 10 bits de datos, 1 bit de stop, 1 bit de restablecimiento De bus.

```
--Generación de transmisión
If (CheckTransmision=0) then--Chequear si hay transmisión en curso.
TX<='1';--Estado inicial del proceso
BaudRatePin<='0';--Clk output stop.
else
if (Baudrate<=2604) then--Salida de reloj
--para comprobación de Baudrate.
--Reloj 50% de duty cycle.
BaudRatePin<='1';
          else
           BaudRatePin<='0';
          end if;
  Baudrate:=Baudrate+1;
--Incrementar baudrate.
          If (Baudrate>5208) then--delay de 104us
Baudrate:=0;
           Contadorbits:=Contadorbits+1;
--Incremento de contador en una unidad.
          If (Contadorbits>2and Contadorbits<11) then
          Comando<=Comando srl1;
--Rotación a la derecha, el bit de rotación
--se guarda en comando posición 0, las operaciones
--sll, srl solamente funcionan con tipos signed o unsigned.
                              end if;
                              if (Contadorbits=13) then
                              Contadorbits:=0;
                              CheckTransmision:=0;
                              Delay:=0;
                              end if;
                              Baudrate:=0;--Reiniciar Baudrate.
                    end if;

                    case Contadorbits is
                    when 1=>TX<='0';--Iniciación de bit de inicio de transmisión
                    when 2 to10=>TX<=Comando(0);--iniciación de bit de inicio de transmisión
                    when 11=>TX<='1';--bit de parada en este caso solo 1.
                    when 12=>TX<='1';--Restablecer línea de comunicación serial.
                    when others=>TX<='1';
                    end case;
end if;
end if;
end process;
end Behavioral;
```

Comprobación de generador de velocidad de transmisión. Esta señal se puede validar por medio de osciloscopio.

Validación de ***Tbit*** cada 104 µs. Los datos se envían en este periodo de tiempo.

Fuente: elaboración propia.

Este capítulo ha constituido una contextualización sobre la diversidad de sistemas de *hardware* que pueden describirse mediante VHDL; derivado de esto último, esperamos que se haya mostrado al lector un panorama general respecto de las posibles aplicaciones de este lenguaje en proyectos de uso común, y que con ello se propicie la generación de soluciones a partir de los elementos abordados.

Glosario

A

- ***Adaptative Logic Module (ALM):*** es el componente básico de las familias de dispositivos compatibles y está diseñado para maximizar el rendimiento y el uso de recursos. Cada ALM puede soportar hasta ocho entradas y ocho salidas, y contiene dos o cuatro celdas de lógica de registro y dos celdas lógicas combinacionales, dos sumadores completos dedicados, una cadena de transporte, una cadena de registro y una LUT con una máscara de 64 bits (Altera, Adaptive Logic Module (ALM), 2016).

B

- ***Bipolar Junction Transistor (BJT):*** "dispositivo semiconductor de tres capas [...] que consta de dos capas de material tipo n y una de material tipo *p*, o bien de dos capas de material tipo *p* y una de material tipo *n*. [...] El término 'bipolar' refleja el hecho de que huecos y electrones participan en el proceso de inyección hacia el material polarizado de forma opuesta" (Boylestad & Nashelsky, 2009).

- **Bloque combinacional:** agrupación de diferentes elementos electrónicos dentro de un solo grupo o bloque. La combinación o unión de estos elementos produce una salida que depende exclusivamente del valor existente en las entradas en ese momento preciso (Wakerly, 2001).

- **Bloque secuencial:** agrupación de diferentes elementos electrónicos dentro de un solo grupo o bloque. Dentro de los elementos allí utilizados se cuenta con elementos de memoria, los cuales almacenan la información de manera temporal. Las salidas están determinadas no solo por los valores de las entradas sino también por la secuencia de las mismas, que conducen al estado presente, y el siguiente valor (Wakerly, 2001).

- **Bloque lógico configurable (*Configurable Logic Block,* CLB):** componente fundamental de una FPGA, que permite al usuario implementar virtualmente cualquier funcionalidad lógica dentro del chip (Digilent, 2016).

C

- ***Complementary Metal Oxid Semiconductor* (CMOS):** dispositivo semiconductor de metal óxido complementario. Usa menos elementos internos, lo cual permite circuitos de tamaño reducido y fabricación más simple. El rango de voltaje de trabajo puede variar entre +3 y +18 V.

- **Compuerta lógica:** dispositivo electrónico básico, que permite representar funciones de tipo lógico describiendo la relación entre las entradas y la salida, obteniendo solamente una posible respuesta "0" o "1", sobre el cual se elaboran o construyen componentes de mayor complejidad. Dentro de las compuertas lógicas se encuentran AND "&", OR "≥1", XOR "=1" y la inversora, o NOT.

- ***Complex PLD* (CPLD):** chip que puede emular el comportamiento de miles de puertas lógicas interconectadas entre sí y junto con registros o *flip-flops*. Esto lo clasifica como un dispositivo no volátil, puesto que el diseño de interconexión le permite mantener la información allí almacenada, aun si la energía es removida. Otra ventaja es que requiere poca energía, aun cuando tiene gran cantidad de compuertas lógicas (Sandige & Sandige, 2012)

D

- ***Display:*** palabra inglesa que puede ser traducida como sustantivo ('monitor'), o bien como verbo ('exhibir' o 'mostrar'). En electrónica, esta palabra se utiliza para identificar los *displays* de siete segmentos, que muestran los números del 0 al 9; o bien los Liquid Cristal Displays (LCD), esto es, pantallas delgadas y planas formadas por filas y columnas (2x16) que permiten mostrar letras, símbolos y números de manera simultánea, según la programación escrita.

E

- **Expresiones lógicas o booleanas:** en 1854, el matemático George Boole escribió el libro *Una investigación sobre las leyes del pensamiento,* en el cual describía la manera en que realizamos decisiones lógicas con base en circunstancias verdaderas o falsas. Los métodos que describió se conocen actualmente como lógica booleana, y el sistema de usar símbolos y operadores para describir estas decisiones se denomina álgebra booleana (Tocci, Widmer, & Moss, 2007).

- ***Electrically Erasable Programmable Read Only*** **Memory (EEPROM):** memoria programable y borrable eléctricamente de solo lectura, también conocida como E^2PROM.

F

- ***Flip-Flop:*** este dispositivo, comúnmente conocido por la abreviatura "FF", se fabrica mediante la utilización de distintos arreglos de compuertas. Puede permanecer en un estado posible "1" o "0", siempre y cuando no reciba una señal externa que lo obligue a cambiar. Con base en esta cualidad, puede mantener o almacenar un valor durante un tiempo indeterminado (Tocci, Widmer, & Moss, 2007).

- **Field Programmable Gate Array (FPGA):** "[es] la tecnología de arreglos de compuertas programables en campo […]. Desde que Xilinx los inventó en 1984, los FPGAs [sic] han pasado de ser sencillos chips de lógica de acoplamiento a reemplazar a los circuitos integrados de aplicación específica (ASICs) [sic] y procesadores para procesamiento de señales y aplicaciones de control". (Instruments, 2016)

I

- **Institute of Electrical and Electronics Engineers (IEEE):** este instituto, creado en la ciudad de Nueva York en 1884, asocia principalmente a profesores y estudiantes de carreras de ingeniería; es una entidad sin ánimo de lucro cuya sede principal se encuentra hoy en Piscataway (Estados Unidos) y cuenta con subsedes alrededor del mundo. Para más información, véase http://www.ieee.org.co/.

L

- **Lógica de transistor – transistor (*Transistor-transistor Logic,* TTL):** esta ha sido la familia principal de circuitos integrados (CIS) digitales bipolares durante más de 30 años. Ha sido desplazada por la tecnología CMOS, debido a que su desempeño no es el mejor. Su rango de voltaje de trabajo oscila entre 0 y +5 V.

- ***Lookup Tables* (LUT):** son usadas para implementar generadores de funciones en los CLB. Cuatro entradas independientes son proporcionadas para cada dos generadores de funciones (F1 – F4 y G1 – G4). Los generadores de funciones pueden implementarse en cualquiera de las cuatro entradas de funciones booleanas definidas arbitrariamente. (Xilinx inc, 2016).

- ***Low Voltage CMOS* (LVCMOS):** se utilizan en aplicaciones CMOS puras donde las salidas tienen cargas menores de 100 µA.

P

- **Producto de sumas (*Product of Sums,* POS):** los métodos de simplificación y de diseño en circuitos lógicos digitales generan una expresión lógica donde intervienen las diferentes variables. Consta de dos o más términos OR y se opera con AND. Dicha expresión puede ilustrarse de la siguiente forma:

$$Salida = (A + B + \overline{C}).(\overline{A.} + C)$$

- **Protocolo *Joint Test Action Group* (JTAG):** grupo de la industria formado en 1985 para desarrollar métodos que permitieran la programación y depuración de circuitos impresos (PCB). La interfaz JTAG consiste de 5 pines: *Test Clock* (TCK), *Test Mode Select* (TMS), *Test Data In* (TDI), *Test Data Out* (TDO) y *Test Reset* (TRST) (opcional). El funcionamiento del protocolo consiste en ubicar elementos de memoria entre los pines externos de un circuito integrado y la lógica interna del dispositivo.

R

- **RAM estática (*Static RAM,* SRAM):** consiste, básicamente, en *latches* internos que guardan la información binaria. Si la memoria deja de tener energía de alimentación, la información allí almacenada se perderá.

T

- **Transistor de efecto de campo (*Field-Effect Transistor,* FET):** dispositivo semiconductor, controlado por voltaje. Las cargas presentes establecen un campo eléctrico, el cual controla la ruta de conducción del circuito de salida sin que se requiera un contacto directo entre las cantidades de control y las controladas. (Boylestad & Nashelsky, 2009).

Lista de referencias

Altera, C. (1998). *CPLDs vs FPGAs Comparing High-Capacity Programmable Logic.* Recuperado de http://extras.springer.com/1998/978-1-4615-5827-9/lit/pib/pib18_01.pdf

Altera, C. (2 de Diciembre de 2016). *Adaptive Logic Module (ALM).* Recuperado de http://quartushelp.altera.com/15.0/mergedProjects/reference/glossary/def_alm.htm

Ashenden, P. J. (1990). *The VHDL Cookbook.* Adelaide, Australia: University of Adelaide.

Bhasker, J. (1991). *A VHDL Premier.* New Jersey: Prentice Hall.

Boylestad, R. L., & Nashelsky, L. (2009). *Electrónica: Teoria de circuitos y dispositivos electrónicos.* México: Pearson educación.

Digilent. (2 de Diciembre de 2016). *Diligent Blog.* Recuperado de https://blog.digilentinc.com/fpga-configurable-logic-block/

Farooq U., M. Z. (2012). *FPGA Architectures: An Overview. In: Tree-based Heterogeneous FPGA Architectures.* New York: Springer.

Gómez-Pulido, J. (2012). *Introduction to FPGA design.* Cáceres, España: University of Extremadura.

Haskell, R. (2009). *Introduction to Digital Design.* Michigan, USA: LBE Books.

Hitachi Ltd. (1999). *HD44780U (LCD-II).* Recuperado de https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Instruments, N. (3 de Diciembre de 2016). *Introducción a la Tecnología FPGA: Los Cinco Beneficios Principales.* Recuperado de http://www.ni.com/white-paper/6984/es/

Pedroni, V. (2004). *Circuit Design with VHDL.* Lomdon, England: MIT Press .

Sandige, R. S., & Sandige, M. L. (2012). *Digital and Computer Design with VHDL.* New York: Mc Graw Hill.

Tocci, R. J., Widmer, N. S., & Moss, G. L. (2007). *SISTEMAS DIGITALES Principios y Aplicaciones.* México: Pearson Educación.

Toronto, U. d. (2000). *Architecture of FPGAs and CPLDs: A Tutorial.* Recuperado de http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf

Wakerly, J. (2001). *Diseño digital: principios y prácticas.* México: Pearson Education.

Xilinx inc. (24 de Enero de 2002). XC9500XL High-Performance CPLD. Recuperado de http://www1.cs.columbia.edu/~sedwards/classes/2006/4840/ds054.pdf

Xilinx inc. (1 de Diciembre de 2008). *Virtex-4 FPGA User's manual.* Recuperado de http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

Xilinx Inc. (2012). *ISE In-Depth Tutorial.* Recuperado de http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf

Xilinx inc. (2013). *Caracteristicas FPGA Spartan 3E.* Recuperado de http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

Xilinx inc. (1 de Diciembre de 2016). *Look-up Table (LUT) (definition) [FPGA Editor].* Recuperado de http://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/html/fe_d_lut.htm

Reseña autores

Sergio González Gil es Ingeniero Electrónico de la Universidad Antonio Nariño, Especialista en Telecomunicaciones móviles de la Universidad Distrital de Bogotá, Especialista en Pedagogía y Docencia Universitaria y Gestión de proyectos de desarrollo de la Universidad la Gran Colombia, Magíster en Administración de Empresas con Especialidad en Dirección de Proyectos de la Universidad Viña del Mar de Chile, con una experiencia de más de 15 años en el área de la ingeniería experiencia de más de 13 años en diferentes áreas como Auditor Externo, Ingeniero de Soporte y Docencia. Actualmente se desempeña como docente de tiempo completo del programa de Tecnología en Electrónica de la Corporación Universitaria Minuto de Dios - UNIMINUTO.

Jonathan Álvarez Ariza es Tecnólogo en Electrónica de la Corporación Universitaria Minuto de Dios - UNIMINUTO, Ingeniero Electrónico de la Universidad Central de Colombia y candidato a Magíster en Docencia de la Universidad de La Salle. Ha escrito varios artículos internacionales sobre educación en ingeniería en especial en las áreas de programación y control automático y ha sido ponente en eventos internacionales entre los cuales se encuentran la conferencia ICEED (International Conference in Engineering Education) y Frontiers in Education (FIE), eventos organizados por IEEE. Actualmente se desempeña como docente y líder del semillero de control automático (SeCon) del programa de Tecnología en Electrónica de la Corporación Universitaria Minuto de Dios - UNIMINUTO.

Digital 2018
En su composición se utilizó el tipo Universe LT Std
Primera edición 2018
Bogotá, D.C., 2018 - Colombia

UNIMINUTO
Corporación Universitaria Minuto de Dios
Sede Principal