UNIVERSIDAD
LASALLISTA BENAVENTE

ESCUELA DE INGENIERÍA EN COMPUTACION CON ESTUDIOS INCORPORADOS A LA
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
CLAVE: 8793-16

"PLD'S: CARACTERÍSTICAS, APLICACIONES, PRÁCTICAS Y PROGRAMACIÓN"

TESIS

QUE PARA OBTENER EL TITULO DE:
INGENIERA EN COMPUTACIÓN

PRESENTA:
IDALIA MARTÍNEZ MONDRAGÓN

ASESOR:
ING. ANSELMO RAMÍREZ GONZÁLEZ

CELAYA, GTO. FEBRERO DEL 2007

Agradecimientos

A Dios

Por darme la vida, la sabiduría, entendimiento y fuerza para lograr mis metas en especial la tesis.

A mis Padres

A ellos por ser los padres perfectos y maravillosos por amarme, ser comprensivos, tener paciencia y ayudarme en los malos y buenos momentos de mi vida, gracias por darme la oportunidad de terminar mis estudios y darme la fortaleza para llegar hasta el día de hoy. Gracias por ser mis padres.

A mis Hermanos

A Rubén Jonás, Emmanuel y Alberto por apoyarme, ser unos buenos hermanos y gracias por estar siempre a mi lado.

A mis Profesores

Gracias a todos por su apoyo, por transmitirme sus conocimientos y experiencias, por ser excelentes maestros y amigos.

A mi Asesor

Ing. Anselmo Ramírez Gonzáles por su apoyo y orientación.

A mis Tíos y Abuelos

Gracias por ayudarme y apoyarme.

A mis Compañeros

Lalo, Francisco Javier, Eulogio, Faby, Héctor, Lolita, Isma, Zoby, Carlos y Manuel.

A mi mejor Amiga Zoobeida

A ella por ser una excelente amiga, por apoyarme y estar siempre a mi lado.

A mi novio Alex

Por ser mi amigo y compañero, por apoyarme, comprenderme y formar parte de mi vida.

ÍNDICE

Capítulo IV SINTAXIS DEL LENGUAJE

Capítulo V MANUAL DE PRÁCTICAS

Introducción

Los avances tecnológicos en el área de la electrónica nos han forzado a mantenernos vigentes en las materias relativas a los circuitos digitales. En los últimos 30 años hemos visto surgir el circuito integrado y su creciente capacidad de integración, lo cual ha traído al mercado, nuevos y mejores dispositivos cada vez más complejos, mientras que ya son una realidad los circuitos con más de diez millones de transistores en un solo substrato. A pesar de que es bien sabido que la tecnología de integración tiene límites físicos, aún nos queda por ver mucho de los avances que serán realizados en los próximos años. Esta creciente complejidad en la escala de integración permite que los diseñadores modernos puedan incorporar cada vez mas funciones en una sola pastilla y con un poder de desempeño que nadie se había imaginado.

Mientras que en los años 70s, con el nacimiento del circuito integrado, se pudieron reducir significativamente las dimensiones físicas de una computadora, los 80s vieron el nacimiento de las computadoras personales y para los 90s el que, prácticamente, cualquier persona tuviera acceso a una de estas unidades. Esta realidad es cotidiana en nuestros tiempos a los albores del siglo XXI donde no solamente se tiene acceso a una computadora, sino que estamos rodeados de sistemas digitales para la realización de las más diversas tareas que van desde los controles industriales para los procesos de manufactura, hasta los productos domésticos como televisores, radios, sistemas de video, televisión por satélite, hornos de microondas, lavadoras programables, juguetes, etc. Esta masificación en el uso de los sistemas digitales como cerebro de los aparatos y equipos que nos rodean ha sido posible gracias al avance en las escalas de integración y los sistemas de una sola pastilla SOC (System On a Chip). Así vemos teléfonos móviles de tamaño increíblemente pequeños, televisores portátiles, computadoras portátiles, videojuegos personales y un sinnúmero de aparatos que hacen uso del SOC para reducir tamaño. A la par con la tecnología que permite una integración tan alta en dispositivos de una sola pastilla, se han desarrollado sistemas de soporte para poder hacer uso de estas técnicas; tan solo hay que imaginar cómo es posible trabajar en un proyecto con una complejidad de millones de transistores.

Humanamente no es posible manejar transistor por transistor un diseño tan complejo y es evidente que se requieren herramientas que faciliten el trabajo.
Estas herramientas hacen uso del diseño jerárquico y permiten al diseñador automatizar la mayor parte del trabajo repetitivo, simplificándolo de una manera descriptiva. Por consiguiente, los lenguajes descriptivos de circuitos HDL (Hardware description Language) han tomado especial importancia en nuestros días.
En conjunto se presentan la teoría de la electrónica digital y sistemas digitales al mismo tiempo que se hace uso del lenguaje descriptivo VHDL para la síntesis de los circuitos.
Los PLD's se utilizan en casi todos los nuevos equipos electrónicos de control, industriales, de consumo, de oficina, de comunicaciones, etc.
Los dispositivos lógicos programables de última generación presentan un rendimiento tanto en capacidad como velocidad, el presente trabajo aborda el diseño de sistemas con PLD's con las siguientes ventajas:

a) La utilización de componentes individuales, en lugar de un gran número de componentes discretos para realizar una determinada función.
b) La utilización de módulos reconfigurables que se adapten a un diseño de sistema no único, sino diferente de acuerdo a los requerimientos especificados en cada caso.
c) La reducción de tiempo de producción de un sistema y reducción de gastos.

Con este trabajo se comprenderá qué son los PLD's y conocer sus características, aplicaciones y la programación de estos**, c**on el fin de comprobar el rendimiento tanto en capacidad como velocidad de los dispositivos lógicos programables.

En el primer capítulo se conocerá los conceptos fundamentales y la historia de los dispositivos lógicos programables.
En el segundo capítulo se presenta los diferentes tipos de PLD's que existen en el mercado, compararlos y ver cual nos conviene más.

En el tercer capítulo es un complemento del segundo capítulo también se comparara y se conocerá las arquitecturas de los PLD’s.
En el cuarto capítulo se conocerá y se analizara el lenguaje VHDL para la programación de dispositivos lógicos programables.
Y en el quinto capítulo se programara los diferentes circuitos implementando el leguaje VHDL.

CAPÍTULO I

ANTECEDENTES DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES

1.1 HISTORIA DE LOS PLD's

A pesar de que la electrónica es una ciencia relativamente nueva, con un siglo de existencia, la teoría sobre la cual está fundamentada fue desarrollada por Aristóteles cuando formalizó la lógica. Esto permaneció sin ningún cambio durante 2000 años hasta que el matemático inglés George Boole introdujo elementos algebraicos a la lógica.

La búsqueda de la naturaleza del átomo y las partículas sub-atómicas, realizada por físicos a finales del siglo XIX y principios del XX, genero el desarrollo de una nueva técnica, denominada electrónica, la cual se fundamentó en los principios de la física cuántica y tuvo como aplicaciones principales las comunicaciones y el desarrollo armamentista. A finales de la segunda guerra mundial, un grupo de científicos comenzó con el desarrollo de una máquina de cálculo que utilizaba dispositivos electrónicos y cuya finalidad era automatizar el cómputo de las trayectorias de misiles, las cuales anteriormente se elaboraban manualmente. Esta máquina, vino a ser la primera computadora del mundo y utilizando en su operación los principios del álgebra Boole, con lo que nace la electrónica digital. Denominada digital debido a que utilizaba los estados verdadero y falso para realizar sus operaciones en dígitos binarios, conocidos desde entonces como bit.

Con el nacimiento de los semiconductores, las computadoras pudieron fabricarse con un mayor poder de procesamiento y a la vez con un menor tamaño y costo. Las mejores tecnologías no tardaron en hacer posible la integración de varios dispositivos semiconductores y crear así los primeros circuitos integrados los cuales hicieron que las computadoras se hicieran más poderosas, más reducidas y con un menor costo.

Mientras que los primeros circuitos integrados solamente incorporaban unos cuantos transistores, los avances en las técnicas de integración permitieron colocar cada vez más dispositivos sobre un substrato y rápidamente se pudieron fabricar circuitos integrados con una complejidad equivalente a un módulo de una computadora lo cual permitió la creación de computadoras más accesibles para un número mucho mayor de usuarios.

Desde finales de la década de los sesenta, los equipos electrónicos digitales se han construido utilizando circuitos integrados de función lógica fija, realizados en pequeña o mediana escala de integración.

Para las realizaciones muy complejas que exigirían un número elevado de circuitos integrados (CI) de función fija, se utilizan circuitos diseñados a medida que sólo sirven para una aplicación.

La lógica programable se usa como una manera de "personalizar" los diseños lógicos, es decir, se diseñan pensando en el hardware propio. Los primeros PLD's se programaron por máscara[1] y se desarrollaron por fabricantes de computadoras, a principios de los sesenta llegó la lógica programable por fusibles y desde entonces ésta técnica, estuvo disponible tanto para pequeños como para grandes usuarios. La tarea del diseñador lógico se simplifica con el uso del software, existen programas con los que actualmente se realizan tareas de diseño y minimización de lógica secuencial, esto es, representación Booleana de alto nivel, el software facilita la selección de dispositivos y ayuda a generar vectores de prueba para la simulación del circuito antes de su implementación final.

Durante la primera mitad de los 90 la industria electrónica experimentó una explosión en la demanda de computadoras, teléfonos móviles, dispositivos de comunicación de gran ancho de banda. Estos y otros productos exigían sistemas con pocos circuitos integrados, aunque muy complejos, en pequeñas placas de circuitos impresos (PCB).
Dos de estas tecnologías que se van a analizar en los siguientes capítulos: los dispositivos lógicos programables PLD y los lenguajes de descripción hardware, en particular se analizara VHDL.

[1] **Programación por máscara** la hace el fabricante durante el último proceso de fabricación de la unidad. Esta programación se utiliza especialmente en las memorias ROM. El procedimiento para la fabricar de una ROM, requiere que el usuario llene la tabla de verdad en función de lo que desea que realice la ROM. El fabricante hace la máscara correspondiente para que los caminos produzcan unos y ceros de acuerdo a la tabla de verdad del usuario. Este procedimiento es muy costoso, razón por la cual sólo es conveniente si se van a fabricar grandes cantidades con el mismo tipo de configuración.
Véase en www.google.com.mx/search?hl=es&q=define%2A=mascara&programa

En los últimos quince años la industria electrónica ha tenido una gran evolución en el desarrollo de sistemas digitales; desde computadoras personales, sistemas de audio y vídeo hasta dispositivos de alta velocidad para las comunicaciones.
Productos elaborados con una alta tecnología que permite aumentar la funcionalidad, disminuir costos, mejorar el aprovechamiento de la energía, así como una marcada tendencia hacia la miniaturización.

Esto ha sido posible gracias a la implementación de herramientas de diseño asistidos por computadora, conocidas como herramientas CAD (**C**omputer **A**ided **D**esign), aunque específicamente se hace uso de herramientas EDA (**E**lectronic **D**esign **A**utomation), que es el nombre que se le da a todas las herramientas CAD para el diseño de sistemas electrónicos.

Este software de diseño electrónico cada vez mas sofisticado y, además, contamos con computadoras más veloces y con mayor capacidad de procesamiento, facilitando a los ingenieros el desarrollo de circuitos. Ambos, hardware y software, constituyen actualmente herramientas muy importantes que simplifican el trabajo de diseño electrónico. Además de facilitar el trabajo, el uso de herramientas EDA también aceleró los procesos de diseño. Esta situación condujo a adoptar nuevas metodologías para el diseño y evaluación de los circuitos electrónicos.

El uso de las herramientas EDA junto con los dispositivos lógicos programables, que pueden ser utilizados en diferentes aplicaciones e inclusive ser reprogramados, cambió bastante el concepto de diseño de circuitos digitales. VHDL es un lenguaje que se creó para el diseño, modelado y documentación de circuitos complejos. Actualmente se le utiliza para la síntesis de circuitos digitales utilizando dispositivos lógicos programables. Es así como los dispositivos lógicos programables y VHDL, constituyen los elementos fundamentales para estas nuevas metodologías de diseño.

En los últimos años se ha producido un gran desarrollo tecnológico en el área de la microelectrónica, llegando a una gran densidad de integración de componentes sobre un mismo substrato. Con ello se han obtenido circuitos cada vez más complejos, que ejecutan un mayor número de operaciones y que precisan de gran cantidad de terminales de conexión.

El campo de la electrónica se ha desarrollado grandemente en los últimos años, es así como nuevos procedimientos y tecnologías en el diseño de sistemas electrónicos han sido probados para poder construir sistemas complejos con el menor riesgo de fallas.

Además, la mayor densidad de componentes necesaria para implementar dichos sistemas y la necesidad de diseños compactos originó el desarrollo de dispositivos tales como los circuitos integrados de aplicación específica (ASIC's) y de dispositivos lógicos programables (PLD's).

1.2 CONCEPTOS FUNDAMENTALES

La fabricación de dispositivos de lógica programable se basa en los siguientes dos conceptos.

- FUNCIONALIDAD COMPLETA

  La cual se fundamenta en el hecho de que cualquier función lógica se puede realizar mediante una suma de productos.
- CELDAS DE FUNCIONES UNIVERSALES

  También denominadas generadores de funciones, son bloques lógicos configurables para procesar cualquier función lógica, similares en su funcionamiento a una memoria. En estas celdas se almacenan los datos de salida del circuito combinacional en lugar de implementar físicamente la ecuación booleana.

1.3 DISPOSITIVOS LÓGICOS PROGRAMABLES

Un dispositivo lógico programable, o PLD cuyas características pueden ser modificadas y almacenadas mediante programación. El dispositivo programable más simple es el PAL. El circuito interno de un PAL consiste en una matriz de conexiones, una matriz de compuertas AND y un arreglo de compuertas OR. Una matriz de conexiones es una red de conductores distribuidos en filas y columnas con un fusible en cada punto de intersección, mediante la cual se seleccionan cuales entradas del dispositivo serán conectadas al arreglo AND cuyas salidas son conectadas al arreglo OR.

1.3.1 ¿Que son los PLD's?

Los PLD's son dispositivos digitales que se pueden configurar por el usuario para implementar una amplia variedad de funciones lógicas en sistemas. Estos dispositivos tienen pines de entrada, un arreglo lógico programable y pines de entrada y salida. Muchos PLD's tienen salidas programables que incrementan su flexibilidad haciéndolos útiles para una gran variedad de aplicaciones (fig. 1.1).

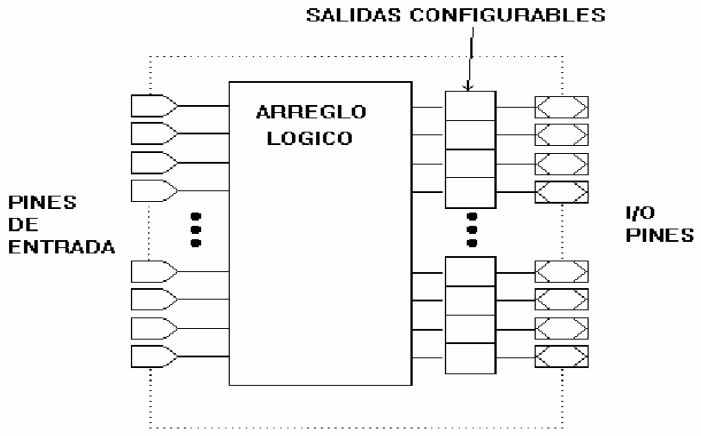


Figura 1.1 Diagrama a bloques de los PLD's[2]

[2] Véase en www.virtual.unal.edu.co/cursos/ingenieria/2000477/lecciones/040101.htm

1.3.2 Estructura interna de un PLD

La mayoría de los PLD's están formados por una matriz de conexiones, una matriz de compuertas AND, y una matriz de compuertas OR y algunos con registros. Las matrices pueden ser fijas o programables.

Con estos recursos se implementan las funciones lógicas deseadas mediante un software especial y un programador de dispositivos. El tipo más sencillo de matriz programable, que data de los años 60, era una matriz de diodos con un fusible en cada punto de intersección de la misma. En la figura 1.2 se muestran los circuitos básicos para la mayoría de los PLD's.

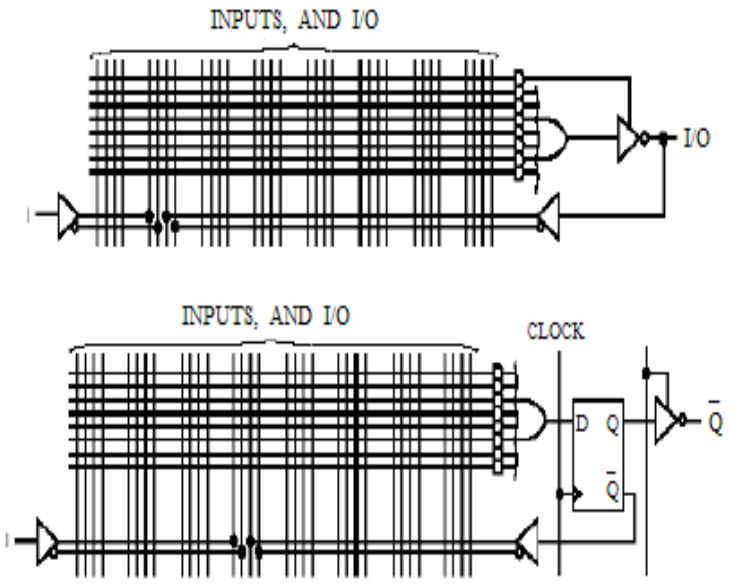


Figura 1.2 Estructura comúnmente utilizadas en PLD's[3]

Estructura de un PLD

La estructura básica de un *PLD* está formada por un arreglo de compuertas *AND* y *OR* interconectadas a través de fusibles.

[3]Véase en www.uag.mx/214/II_DISPOSITIVOS_LOGICOS_PROGRAMABLES.pdf

Matriz AND

La matriz *AND* está formada por una red de compuertas *AND* conectadas a través de conductores y fusibles en cada punto de intersección. Cada punto de intersección entre una fila y una columna se denomina celda. La figura 1.3 muestra un arreglo de compuertas no programado.

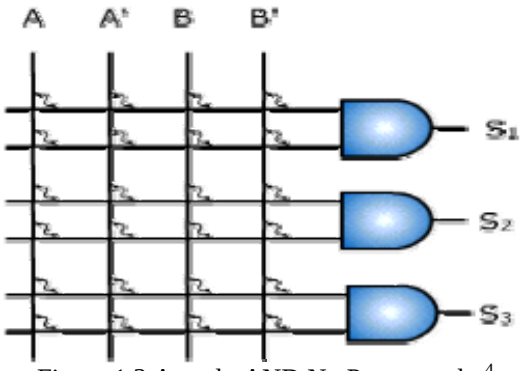


Figura 1.3 Arreglo AND No Programado.[4]

Cuando se requiere una conexión entre una fila y una columna, el fusible queda intacto y en caso de no requerirse la conexión, el fusible se abre en el proceso de programación. La figura 1.4 muestra un arreglo *AND* programado.

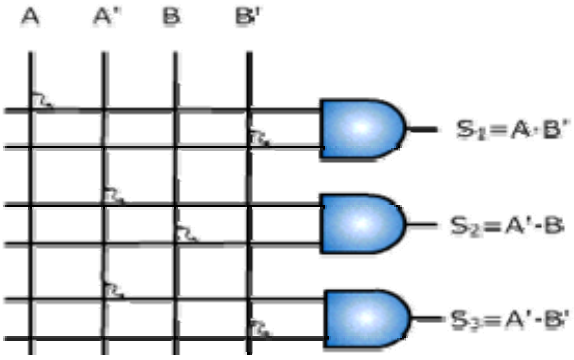


Figura 1.4 Arreglo AND Programado.[5]

[4] Véase en www.virtual.unal.edu.co/PLD.htm

[5] Véase en www.virtual.unal.edu.co/PLD.htm

Matriz OR

La matriz OR está formada por una red de compuertas *OR* conectadas a través de conductores y fusibles en cada punto de intersección. La figura 1.5 muestra un arreglo de compuertas no programado.

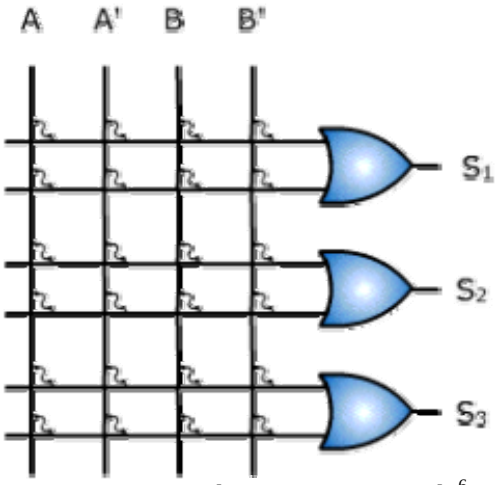


Figura 1.5 Arreglo OR No Programado[6]

La figura muestra 1.6 un arreglo *OR* programado.

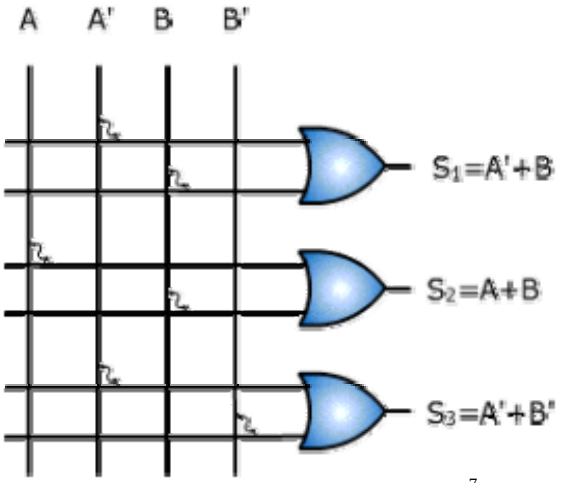


Figura 1.6 Arreglo OR Programado[7]

[6] Véase en www.virtual.unal.edu.co/PLD.htm

[7] Véase en www.virtual.unal.edu.co/PLD.htm

Memoria programable de sólo lectura PROM (Programable Read Only Memory). La *PROM* está formada por un conjunto fijo (no programable) de puertas *AND* conectadas como decodificador y una matriz programable *OR* como se muestra en la figura 1.7. La *PROM* se utiliza como una memoria direccionable y no como un dispositivo lógico.

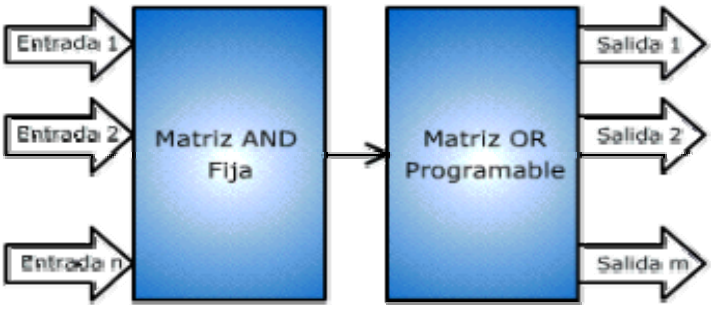


Figura 1.7 Diagrama de bloques de una PROM[8]

Arreglo Lógico Programable PLA (Programmable Logic Array). El *PLA* es un *PLD* formado por una matriz *AND* programable y una matriz *OR* programable como se muestra en la figura 1.8. La *PLA* ha sido desarrollada para superar algunas de las limitaciones de las memorias *PROM* .

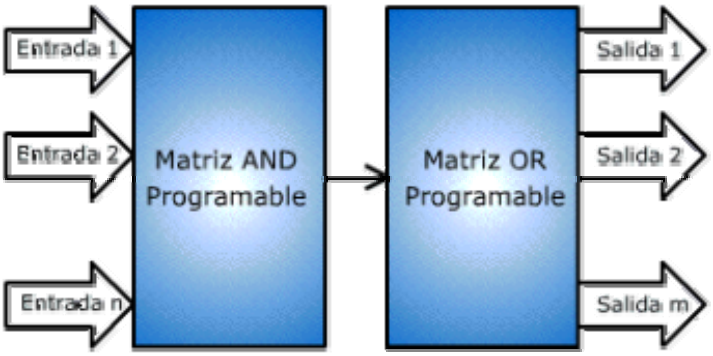


Figura 1.8 Diagrama de bloques de una PLA[9]

[8] Véase en www.ilustrados.com/diagramas/ingenieria.pdf

[9] Véase en www.ilustrados.com/diagramas/ingenieria.pdf

En la actualidad existen soluciones con dispositivos lógicos programables complejos que combinan arquitectura superior y software de gran alcance, ofreciendo un nivel sin precedente en la flexibilidad del diseño.

1.4 CARACTERÍSTICAS DE LOS PLD's

El trabajo con PLD's proporciona: facilidad de diseño, atributos, fiabilidad, economía y seguridad.

a) Facilidad de diseño

Las herramientas de soporte al diseño con PLD's facilitan enormemente este proceso. Las hojas de codificación que se utilizaban en 1975 han dejado paso a los ensambladores y compiladores de lógica programable (PALASM, AMAZE, ABEL, CUPL, OrCAD/PLD, etc.). Estas nuevas herramientas permiten expresar la lógica de los circuitos utilizando formas variadas de entrada tales como; ecuaciones, tablas de verdad, procedimientos para máquinas de estados, esquemas, etc.

La simulación digital posibilita la depuración de los diseños antes de la programación de los dispositivos. Todo el equipo de diseño se reduce a un software de bajo costo que corre en un PC, y a un programador.

b) Atributos

Los PLD's TTL que hay en el mercado tienen tiempos de conmutación tan rápidos como los circuitos integrados de función fija más veloces. Los PLD's ECL son todavía más rápidos. Sin embargo, el incremento de velocidad obtenido con los dispositivos CMOS, que ya han igualado o superado en prestaciones a los dispositivos TTL, está provocando el abandono de la tecnología bipolar por parte de los fabricantes. En cuanto al consumo de potencia, los PLD's generalmente consumen menos que el conjunto de chips a los que reemplazan.

c) Fiabilidad

Cuanto más complejo es un circuito, más probabilidades hay que alguna de sus partes falle. Los PLD’s reducen el número de chips en los sistemas, la probabilidad de un fallo disminuye. Los circuitos impresos con menor densidad de CI son más fáciles de construir y más fiables. Las fuentes de ruido también se reducen.

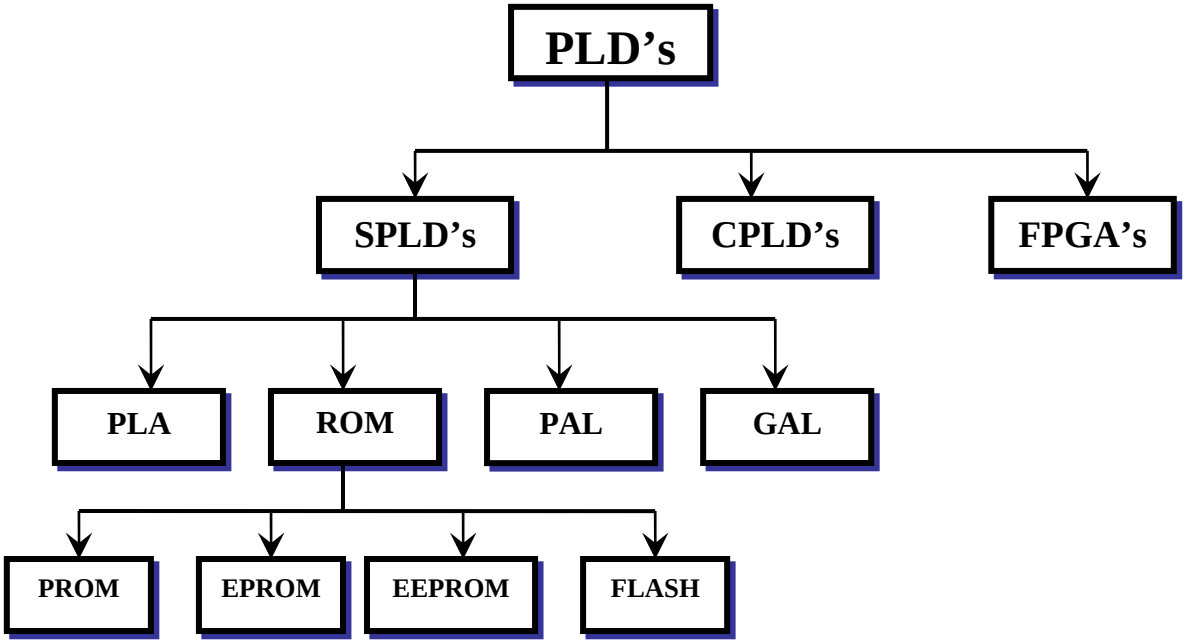
d) Economía

Los costos de pérdida de mercado por una introducción tardía de un producto. Otros son más claros, por ejemplo, la reducción del área de las placas de circuito impreso obtenida gracias a que cada PLD sustituye a varios circuitos integrados de función fija. Muchas veces se consigue reducir el número de placas de circuito impreso economizándose en conectores. La reducción de artículos en almacén también aporta ventajas económicas.

e) Seguridad

Los PLD’s tienen fusibles de seguridad que impiden la lectura de los dispositivos programados, protegiendo los diseños frente a copias.

1.5 CLASIFICACIÓN DE LOS PLD's



1.5.1 PLD: Dispositivo Lógico Programable

Conjunto de circuitos integrados formados por cierto número de puertas lógicas y/o módulos básicos y/o biestables[10] cuyas conexiones puedan ser personalizadas o programables, bien sea por el fabricante o por el usuario. La gran ventaja de estos dispositivos es que los fabricantes pueden realizar grandes cantidades de estos CI lo que disminuye su costo de producción y los usuarios posteriormente pueden personalizar sus diseños en sus propios laboratorios sin grandes inversiones.

[10] Biestable capaz de permanecer en un estado determinado puede ser set (activación) y reset (desactivación)
Véase en www.monografias.com

1.5.2 SPLD's: Dispositivo Lógico Programable Simple

La mayoría de los SPLD's están formados por matrices de conexiones: una matriz de compuertas AND y una matriz de compuertas OR como se observa en la tabla 1.1. Con estos recursos se implementan las funciones lógicas deseadas mediante un software especial y un programador. Las matrices pueden ser fijas o programables. Una matriz de conexiones es una red de conductores distribuidos en filas y columnas con un fusible en cada punto de intersección, mediante el cual se seleccionan cuales entradas serán conectadas a las compuertas AND/OR.

	AND	OR
PLA	Programables	Programables
PAL,GAL	Programables	Fijas
PROM	Fijas	Programables

Tabla 1.1 Clasificación de los SPLD's

1.5.3 ROM: Memoria de solo lectura

Es una memoria de sólo lectura que se programa mediante máscaras. Es decir, el contenido de las celdas de memoria se almacena durante el proceso de fabricación para mantenerse después de forma irrevocable. Desde el instante en que el fabricante grabó las instrucciones en el chip, por lo tanto la escritura de este tipo de memorias ocurre una sola vez y queda grabado su contenido aunque se le retire la energía.

Se usa para almacenar información vital para el funcionamiento del sistema: en la gestión del proceso de arranque, el chequeo inicial del sistema, carga del sistema operativo y diversas rutinas de control de dispositivos de entrada/salida suelen ser las tareas encargadas a los programas grabados en ROM.

Estos programas forman la llamada BIOS (Basic Input Output System). Junto a la BIOS se encuentra el chip de CMOS donde se almacenan los valores que determinan la configuración hardware del sistema, como tipos de unidades, parámetros de los discos duros, fecha y hora del sistema; esta información no se pierde al apagar la computadora. Estos valores se pueden modificar por medio del SETUP.

Si tenemos idea de cómo se fabrican los circuitos integrados, sabremos de donde viene el nombre. Estos se fabrican en placas de silicio (obleas) que contienen varias decenas de chips. Estas obleas se fabrican a partir de procesos fotoquímicos, donde se impregnan capas de silicio y oxido de silicio, y según convenga, se erosionan al exponerlos a la luz.

Como no todos los puntos han de ser erosionados, se sitúa entre la luz y la oblea una mascara con agujeros, de manera que donde deba incidir la luz, esta pasará. Con varios procesos similares pero más complicados se consigue fabricar los transistores y diodos micrométricos que componen un chip.

Las PC’s vienen con una cantidad de ROM, donde se encuentras los programas de BIOS (Basic Input Output System), que contienen los programas y los datos necesarios para activar y hacer funcionar el computador y sus periféricos.

La ventaja de tener los programas fundamentales en la computadora almacenados en la ROM es que están allí implementados en el interior de la computadora y no hay necesidad de cargarlos en la memoria desde el disco de la misma forma en que se carga el DOS. Debido a que están siempre residentes, los programas en ROM son muy a menudo los cimientos sobre los que se construye el resto de los programas. Dentro de un proceso de elaboración de datos de una computadora, no es posible grabar ningún dato en las memorias ROM. Son memorias perfectas para guardar microprogramas, sistemas operativos, tablas de conversión, generación de caracteres etc.

Las características fundamentales de las memorias ROM son:

1. Alta densidad: La estructura de la celda básica es muy sencilla y permite altas integraciones.

2. No volátiles: El contenido de la memoria permanece si se quita la alimentación.

3. Costo: La programación se realiza a nivel de máscaras durante el proceso de fabricación, resultan baratas en grandes cantidades, de modo que el costo de fabricación se reparte en muchas unidades y el costo unitario es baja.

4. Sólo lectura: Únicamente son programables a nivel de máscara durante su fabricación. Su contenido, una vez fabricada, no se puede modificar.

Hay muchos tipos de ROM:

Siglas	Descripción
ROM	Memoria de solo lectura
PROM	Memoria de solo lectura programable
EPROM	Memoria de solo lectura programable y borrable
EEPROM	Memoria de solo lectura programable y borrable electrónicamente

1.5.3.1 PROM: Memoria de solo lectura programable

Se entiende por dispositivo programable aquel circuito de propósito general que posee una estructura interna que puede ser modificada por el usuario final para implementar una amplia gama de aplicaciones. El primer dispositivo que cumplió estas características era una memoria PROM, que puede realizar un comportamiento de circuito utilizando las líneas de direcciones como entradas y las de datos como salidas. Hay dos tipos básicos de PROM:

1. Programables por máscara (en la fábrica).
2. Programables en el campo (field) por en usuario final. Son las EPROM y EEPROM.

Son menos costosas para volúmenes pequeños de producción y se pueden programar de manera inmediata.

Una PROM se compone internamente de dos grupos de compuertas: un grupo de compuertas AND y un grupo de compuertas OR como se muestra en a figura 1.9.

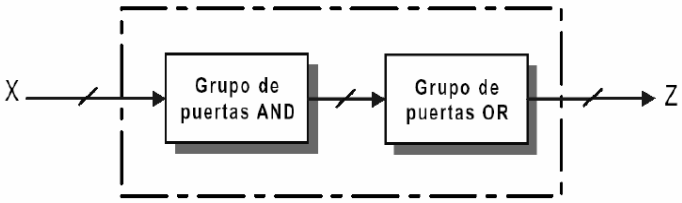


Figura 1.9 Componente internos de una PROM[11]

El grupo de compuertas AND están programadas de antemano y conectadas de forma inalterable, mientras que el grupo de compuertas OR son programables por el usuario.

En la figura 1.9 se muestra la arquitectura de la mayoría de las PROM consiste generalmente en un número fijo de términos AND que alimenta una matriz programable OR. Se usan principalmente para decodificar las combinaciones de entrada en funciones de salida.

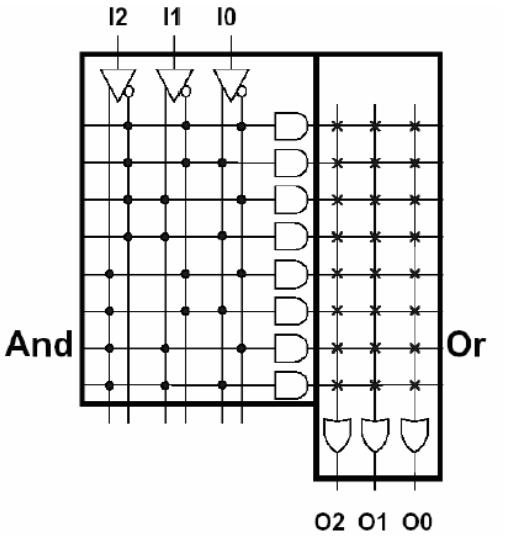


Figura 1.10 Arquitectura de una PROM[12]

[11] Véase en www.ate.uniovi.es/fernando/Doc2003/EI/BAC_PLDs%20(3).pdf
[12] Véase en www.lattice.com/memorias/ing.pdf

Las aplicaciones más importantes son:

a. Microprogramación
b. Programas de sistema
c. Tablas de función

Una alternativa para proyectos pequeños es el uso de una de las memorias de sólo lectura programables o **PROM**, memoria basada en semiconductores que contiene instrucciones o datos. Éstas existen en muchas variantes, pero todas permiten que el usuario programe el dispositivo por si mismo, ahorrándose el alto costo de la producción de la máscara. En la memoria PROM los contenidos pueden ser leídos pero no modificados por un programa de usuario.

Sus contenidos no se construyen, como la ROM, directamente en el procesador cuando éste se fabrica, sino que se crean por medio de un tipo especial "programación", ya sea por el fabricante, o por especialistas técnicos de programación del usuario. El proceso de programación es destructivo: una vez grabada, no se puede hacer cambios.

Las operaciones muy importantes o largas que se habían estado ejecutando mediante programas, se pueden convertir en microprogramas y grabarse permanentemente en una pastilla de memoria programable sólo de lectura. Una vez que están en forma de circuitos electrónicos, estas tareas se pueden realizar casi siempre en una fracción del tiempo que requerían antes. La flexibilidad adicional que se obtiene con la PROM puede convertirse en una desventaja si en la unidad PROM se programa un error que no se puede corregir.

CAPÍTULO II

VENTAJAS Y DESVENTAJAS

2.1 VENTAJAS Y DESVENTAJAS DE LOS PLD’s

VENTAJAS

Conjunto de circuitos integrados formados por cierto número de puertas lógicas y/o módulos básicos y/o biestables cuyas conexiones pueden ser personalizadas o programadas, bien sea por el fabricante o por el usuario.

La gran ventaja de estos dispositivos reside en que los fabricantes pueden realizar grandes modelos de estos CI lo que abarata sus costes de producción y los usuarios posteriormente pueden personalizar sus diseños en sus propios laboratorios sin grandes inversiones:

- Consumos medios, aunque hay familias especializadas en bajo consumo energético
- Fiabilidad Alta
- Tiempo de desarrollo muy bajo, sin dependencia de terceros
- Metodología sencilla
- Equipamiento sencillo
- Aumentan la confidencialidad de las placas
- Bajo costo para circuitos medios complejos
- Debido a su integración, es más fácil almacenarlos por el espacio que ocupan
- Las placas de circuitos impresos de las distintas aplicaciones existentes tengan un tamaño más pequeño
- Reducción en el tiempo de diseño
- Menos espacio sobre la tarjeta, lo que significa menos tarjetas de circuito impreso y gabinetes más pequeños.
- Menores requerimientos de potencia (fuentes de alimentación más pequeñas).
- Procesos de ensamble más rápido, menos conexiones de circuitos donde puedan ocurrir fallas.
- Procedimientos de detección de fallas más sencillos

El reciente desarrollo de los dispositivos lógicos programables (PLD) ofrece a los diseñadores lógicos una manera de reemplazar varios CI estándares con un solo CI. Un PLD es un CI que contiene un número muy grande de compuertas, y registros que están conectados entre sí dentro del CI. Sin embargo muchas de las conexiones son del tipo fusibles.

Se dice que el CI es programable porque la función específica que este realice en una determinada aplicación está determinada por la interrupción selectiva de algunas de las conexiones, mientras que al mismo tiempo se dejan otras intactas.

DESVENTAJAS

- Reducida potencia de salida
- Limitación en los voltajes de funcionamiento
- Mayor costo para circuitos muy simples
- Herramientas específicas del fabricante

2.2 CONOCER LOS DIFERENTES TIPOS DE PLD’s

2.2.1. *C*lases de Dispositivos Lógicos Programables.

2.2.1.1. Circuitos integrados a medida.

Los Circuitos Integrados a Medida (Full Custom), se diseñan a petición de un cliente para que resuelvan una determinada aplicación. Conllevan un alto costo de desarrollo y su empleo sólo se justifica para volúmenes de producción muy elevados. El tiempo necesario para la construcción de un CI a medida es considerable ya que puede oscilar de unos meses a unos años.

2.2.1.2. Matrices de puertas.

Las Matrices de puertas (Gate Arrays) son pequeños trozos de silicio pendientes de algún proceso de metalización que defina las conexiones entre un importante número de puertas o transistores que poseen en su interior. Las matrices de puertas proporcionan densidades superiores a las 100.000 puertas, con un aprovechamiento del 80 al 90 por 100 para los dispositivos pequeños y del 40 por 100 para los grandes. Los fabricantes de silicio ponen a disposición de sus potenciales clientes abundante documentación sobre estos Gate Arrays, con una serie de macros que pueden utilizar de forma inmediata y otras que pueden construirse ellos mismos.

Los macros son agrupaciones de un número de células básicas que realizan funciones comunes como; sumadores; puertas NOT, AND, NAND, NOR XOR, etc.; latches y flip-flops S-R, J-K, D; buffer; osciladores; registros, decodificadores, multiplexores, etc.

Junto a esta documentación, los fabricantes aportan un software que contabiliza el número de células básicas utilizadas por todas las macros, sugiere el Gate Array adecuado para la aplicación, calcula la potencia disipada por el Gate Array que alojará el diseño del cliente, proporciona información sobre los tiempos de propagación de las señales y permite verificar el funcionamiento del circuito.

Una vez superadas todas las etapas previas, el cliente envía la documentación generada al fabricante para que éste ultime los procesos de metalización y fabrique un primer prototipo. El diseño con Gate Arrays puede durar semanas o meses. Requiere un volumen alto de circuitos para justificar sus costos.

2.2.3. FPIC's.

Los FPIC's (Field Programmable Integrated Circuits): son chips programables por el usuario mediante programadores comerciales.

El término FPIC también incluye a los CI no destinados a las aplicaciones lógicas. Son las memorias, los microcontroladores, los PLD (Programmable Logic Device), las FPGA (Field Programmable Gate Array) y los ASPLD (Aplication Specific Programmable Logic Devices).

Los FPIC ofrecen soluciones de bajo coste, de tiempo de desarrollo corto y con menor riesgo que los circuitos a medida, las matrices de puertas y las células normalizadas.

2.2.3.1. PLD's.

Los PLDs (Programmable Logic Devices) son pequeñas ASIC's configurables por el usuario capaces de realizar una determinada función lógica. La mayoría de los PLD consisten en una matriz de puertas AND seguida de otra matriz de puertas OR. Mediante esta estructura, puede realizarse cualquier función como suma de términos productos.

Aunque las memorias PROM, EPROM y EEPROM son PLD's, muchas veces se las excluye de esta denominación debido a que su contenido se define utilizando elementos de desarrollo propios de microprocesadores, tales como; ensambladores, emuladores y lenguajes de programación de alto nivel.

Otras veces, cuando estas memorias se usan para realizar una función lógica y no para guardar un programa de un microprocesador, se las incluye dentro del término PLD.

2.2.3.2. ASPLD's.

Los ASPLD's (Application Specific Programmable Logic Devices) son PLD's diseñados para realizar funciones específicas como, decodificadores de alta velocidad, secuenciadores, interfaces para buses particulares, periféricos programables para microprocesadores, etc.

Partes del ASPLD son programables permitiendo la adaptación del circuito a una aplicación determinada, pero manteniendo su función básica; así, por ejemplo, un decodificador lo personaliza el usuario, pero sigue siendo un decodificador. Estos circuitos están muy optimizados para la función para la que han sido diseñados.

2.2.3.3. FPGA's.

Las FPGA's (Field Programmable Gate Arrays) contienen bloques lógicos relativamente independientes entre sí, con una complejidad similar a un PLD de tamaño medio. Estos bloques lógicos pueden interconectarse, mediante conexiones programables, para formar circuitos mayores. A diferencia de los pld's, no utilizan arquitectura de matriz de compuertas AND seguida de la matriz de puertas OR y necesitan un proceso adicional de ruteado del que se encarga un software especializado.

La primera FPGA la introdujo Xilinx en el año 1985. La programación de las FPGAs de Xilinx basadas en RAM estática es diferente a la programación de los PLD's.

Los PLD tienen entre 100 y 2000 puertas, las FPGA's tienen desde 1200 a 20.000 puertas y la tendencia es hacia un rápido incremento en la densidad de puertas. El número de flip-flops de las FPGA generalmente supera al de los PLD. Sin embargo, la capacidad de la FPGA para realizar lógica con las entradas suele ser inferior a la de los PLD.

2.3 APLICACIONES DE LOS DIFERENTES DISPOSITIVOS LÓGICOS PROGRAMABLES

Electrónica automotriz

Xilinx expande su línea XA de dispositivos lógicos programables para aplicaciones en la industria automotriz.

En el marco del congreso internacional de electrónica automotriz celebrado recientemente en París, Xilinx informó la expansión de su familia XA de dispositivos lógicos programables (PLD) para la industria automotriz.

Spartan-3E es una familia de PLD's que ofrece versiones de entre 100,000 y 1.6 millones de compuertas, que además incluye una serie de características que pueden ser atractivas en aplicaciones automotrices, como por ejemplo la compatibilidad con 18 estándares de entrada/salida. Por otra parte, la tecnología Virtex-4 integra un PLD con un procesador PowerPC embebido y multiplicadores DSP, en un solo empaquetado.

Xilinx es una compañía dedica al desarrollo de circuitos integrados programables, que cuenta con más de 7,500 clientes entre los que destacan Alcatel, Cisco, Ericsson, Fujitsu, HP, IBM, Lucent, Motorola, NEC, Nokia, Nortel, Samsung, Siemens, Sony y Toshiba. La compañía ha desarrollado, desde hace algún tiempo, diversos dispositivos para la industria automotriz. Hacia finales del 2004, la línea XA de dispositivos lógicos programables (PLD), se convirtió en la primera de la compañía en obtener la certificación del consejo de electrónica automotriz AEC-Q100, satisfaciendo los requerimientos de la industria automotriz.

En la tabla 2.1 se encuentra algunas aplicaciones típicas SOC y la complejidad de sus elementos.

Aplicación portátil	Bloques funcionales	Transistores
Reloj digital	Generador de base de tiempo Contadores	1,000
Videojuego	Microprocesador Circuitos de interfaz Memoria	50,000
Radio	Sintetizador de frecuencia Preamplificador Demodulador Amplificador	10,000
Teléfono	Preamplificador Sintetizador de frecuencia Demodulador Codificador Modulador Amplificador	100,000

Tabla 2.1 Amplificaciones típicas SOC

2.3.1 Campos de aplicación de la lógica programable

La lógica programable es una herramienta de diseño muy poderosa, que se aplica en el mundo industrial y en proyectos universitarios en todo el mundo.

En la tabla 2.2 se encuentra resumido el nivel de integración que se ha venido dando con el desarrollo de la electrónica.

Escala de integración	Siglas	Número de transistores	Ejemplos de aplicación
Pequeña escala de integración	SSI	Hasta 100	Circuitos básicos (compuertas AND, OR, NAND, NOT, etc.)
Mediana escala de integración	MSI	De 100 a 1,000	Registros y contadores
Alta escala de integración	LSI	De 1,000 a 10,000	Microprocesadores y memorias
Muy alta escala de integración	VLSI	Más de 10,000 a 100,000	Microprocesadores completos
Ultra alta escala de integración	ULSI ó VVLSI	Más de 100,000	Microprocesadores múltiples, incluyendo memoria, puertos de entrada y salida

Tabla 2.2 Escalas de integración

Los primeros circuitos integrados incorporaban una complejidad equivalente a unas cuantas compuertas y funciones lógicas básicas, no llegando a superar los 100 transistores; este nivel de complejidad se conoció como pequeña escala de integración (SSI, Small Scale Integration). Funciones más complejas se pudieron integrar, como algunas unidades aritméticas y registros, cuando la escala de integración aumentó en un orden de magnitud dando llegada a los circuitos de mediana escala de integración. Los primeros microprocesadores de 8 bits y sus circuitos periféricos pudieron ser desarrollados gracias a la alta escala de integración con una complejidad de 1,000 a 10,000 transistores. En este punto se marcó el inicio del cambio en el enfoque de diseño en electrónica ya que ahora era posible el contar con soluciones complejas con solo unos cuantos componentes.

La tecnología no se detuvo y pronto se aumentó el orden de magnitud de la integración dando cabida a los circuitos integrados de muy alta escala de integración (VLSI). Microprocesadores más rápidos y más poderosos pudieron diseñarse y el mundo de las microcomputadoras comenzó a extenderse con gran rapidez. También en este punto se comenzaron a introducir los primeros circuitos lógicos programables cuya finalidad era proporcionar la interfaz en los sistemas de microprocesador. Al mismo tiempo surgen los primeros lenguajes descriptivos para permitir al diseñador un uso adecuado de los nuevos circuitos complejos.

En este momento ya existen circuitos integrados con más de 10 millones de transistores y se está trabajando para poder realizar el primer circuito con más de 100 millones de transistores. Estos niveles de complejidad permiten que la mayor parte de las aplicaciones de la electrónica puedan ser presentadas como un sistema en una sola pastilla SOC (System On a Chip) la cual es la tendencia moderna del diseño.

En la actualidad se usan desde los PLD más sencillos (como el GAL, PAL, PLA) como reemplazos de circuitos LSI y MSI, hasta los potentes CPLD y FPGA, que tienen aplicaciones en áreas como telecomunicaciones, computación, redes, medicina, procesamiento digital de señales, multiprocesamiento de datos, microondas, sistemas digitales, telefonía celular, filtros digitales programables, entre otros.

En general, lo CPLD son recomendables en aplicaciones donde se requieren muchos ciclos de sumas de productos, ya que pueden introducirse en el dispositivo para ejecutar al mismo tiempo, lo que conduce a pocos retrasos. En la actualidad, los CPLD son muy utilizados a nivel industrial, ya que resulta fácil convertir diseños compuestos por múltiples PLD sencillos en un circuito CPLD.

Por otro lado, los FPGA son recomendables en aplicaciones secuenciales que no suponen grandes cantidades de términos producto. Por ejemplo, los FPGA desarrollados por la compañía ATMEL ofrecen alta velocidad en cómputo intensivo, aplicaciones en procesadores digitales de señales (DSP) y en otras fases del diseño lógico, debido a la gran cantidad de registros con los que cuentan sus dispositivos (de 1024 a 6400).

Desarrollos recientes

Existen desarrollos realizados por diversas compañías (Tabla 2.3) cuyo funcionamiento se basa en un PLD; por ejemplo una tarjeta basada en un FPGA de la familia XC4000 de Xilinx Corporation. Este desarrollo permite el procesamiento de datos paralelos a alta velocidad, lo que reduce los problemas de procesamiento de datos intensivos.
Otro ejemplo es una aplicación realizada en un dispositivo CPLD de la familia Flex10K de Altera Corporation (nivel de integración de 7000 compuertas). La función de esta tarjeta es permitir diversas aplicaciones en tiempo real, como el filtrado digital y muchas otras propias del campo del procesamiento digital de señales.

Compañía	Productos desarrollados con lógica programable
Andraka Consulting Group http://users.ids.net/-randraka/Inc	• Procesadores digitales de señales (DSP) • Comunicaciones digitales • Procesadores de audio y video
Code Logic http://home.intekom.com/codelogic	• Lógica configurable • Control embebido
Boton Line http://www.bltinc.com/	• Módems de alta velocidad • Audio, video, adquisición de datos y procesamiento de señales en general • Aplicaciones militares: Criptografía, seguridad en comunicaciones, proyectos espaciales.
Comit's Services http://www.comit.com/	• Redes: aplicaciones en protocolos TCP/IP • Multimedia: Comprensión de Audio/Video • Aplicaciones en tiempo real
New Horizonts GU http://www.netcomuk.co.uk/-newhoriz/index.html	• Digitalizadores, Cámaras de Video (120Mbytes/sec) • Video en tiempo real • Puertos paralelos de comunicaciones para PC
Design Service Segments http://www.smartech.fi/	• Diseño de microprocesadores complejos • Dispositivos para telecomunicaciones, DSP • Aplicaciones en diseños para control industrial

Tabla 2.3 Compañías que incorporan lógica programable en sus diseños

2.3.2 Aplicación de los PLD's en la industria

EN COMPUTADORAS

- Altera ha introducido la familia Stratix, la familia de PLDs más rápida de la industria. La arquitectura Stratix logra reducciones en el tamaño de un 35%, con 8 veces más RAM, logrando hasta el doble de rapidez que cualquier PLD de la industria.
- Para el aumento de la velocidad de los procesadores

EN BRAZOS MECÁNICOS

- Los brazos mecánicos existen en el mercado desde hace varios años, por regla general los adquieren las universidades, los centros de investigación y las industrias, estos se controla a través de la programación de arreglos lógicos genéricos (GAL).
- La velocidad de los motores se controla con un oscilador que se conecta al reloj del GAL.

EN LA INDUSTRIA AUTOMOTRIZ

- Xilinx expande su línea XA de dispositivos lógicos programables para aplicaciones en la industria automotriz. En el marco del congreso internacional de electrónica automotriz celebrado recientemente en París, Xilinx informó la expansión de su familia XA de dispositivos lógicos programables (PLD) para la industria automotriz. Es una familia de PLD's que ofrece versiones de entre 100,000 y 1.6 millones de compuertas, que además incluye una serie de características que pueden ser atractivas en aplicaciones automotrices, como por ejemplo la compatibilidad con 18 estándares de entrada/salida.

Figura 2.1 Ejemplo de industria automotriz [32]

[32] Véase en www.monografias.com/trabajos18/memorias-programables/memorias-programables.html

- La electrónica ha encontrado un campo de operación cada vez más amplio y lucrativo en la industria automotriz. La utilización de radios, televisiones, seguros y vidrios eléctricos, alarmas y sistemas electrónicos de control se ha vuelto común en la industria automotriz actual.

  Recientes estudios han señalado que tan solo el campo de los dispositivos lógicos programables (PLD) para sistemas de audio para automóviles.

Figura 2.2 Sistema de audio[33]

- **Conexión con microcontroladores**. Puesto que puede funcionar en modo autónomo, resulta muy útil para el desarrollo de periféricos para microcontroladores: controladores de sensores, coprocesadores para hacer ciertas operaciones más rápidamente, etc.

EN LA ROBÓTICA

- Para mover servos, temporizadores, controladores para sensores de ultrasonidos, de distancia, etc.

Figura 2.3 Ejemplo de robot[34]

[33] Véase en www.monografias.com/trabajos18/memorias-programables/memorias-programables.html
[34] Véase en www.monografias.com/robotica.shtml

EN EL CONTROL DE SEMAFOROS

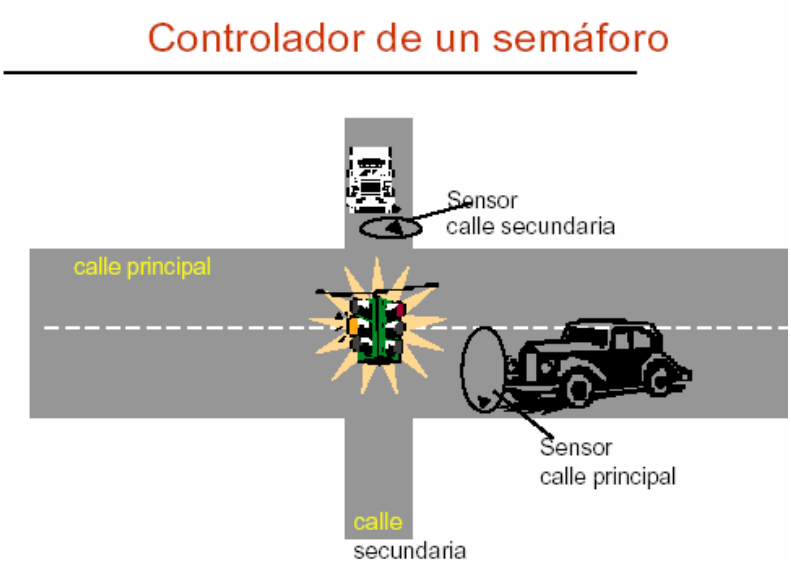


Figura 2.4 Control de semáforos[35]

[35] Véase en www.monografias.com/semaforos.html

CAPÍTULO III

CARACTERÍSTICAS DEL DISEÑO DE LOS PLD’s

3.1 CARACTERÍSTICAS DEL DISEÑO DE LOS PLD's

El trabajo con PLD's proporciona:

- ❖ Las herramientas de soporte al diseño con PLD's facilitan la programación, estas herramientas permiten expresar la lógica de los circuitos utilizando formas variadas de entradas tales como: ecuaciones, tablas de verdad, procedimientos para máquinas de estado, esquemas, etc.
- ❖ Bajo costo por que el equipo de diseño se reduce a un software que corre en una PC y un programador.
- ❖ Tiempo de desarrollo muy bajo
- ❖ Variedad de técnicas de diseño
- ❖ Proporcionan seguridad
- ❖ Reducción en el número de chips
- ❖ Los PLD's tienen fusibles de seguridad que impiden la lectura de los dispositivos programados, protegiendo los diseños frente a copias.
- ❖ Facilidad de diseño

Etapas básicas en el proceso de diseño

El proceso de diseño se puede dividir en seis etapas bien definidas

- ➢ Definición de los requerimientos del diseño
- ➢ Descripción del diseño en VHDL
- ➢ Simulación del código fuente
- ➢ Síntesis, optimización del diseño
- ➢ Programación del dispositivo

Definición de los requerimientos del diseño

Antes de empezar a escribir líneas de código, deberíamos tener una idea clara de los objetivos y requerimientos del diseño (especificaciones): ¿Qué funcionalidad debe tener el diseño?; esto es, ¿Para que sirve?, ¿Cuáles son los tiempos requeridos para la inicialización "setup" o la relación reloj-salida?, ¿Cuál es la frecuencia máxima de operación?, ¿Cuál son los caminos críticos?. Responder de forma adecuada a estas y otras preguntas ayudarán a elegir una metodología de diseño y una arquitectura de dispositivo adecuada para empezar a trabajar.

Descripción del diseño en VHDL

Formulación del diseño

A partir de las especificaciones se puede tener la tentación de empezar a escribir líneas de código, pero es recomendable decidir una **metodología** de diseño. Esto es, teniendo una idea de cómo será descrito, tenderemos a escribir de forma más eficiente.

Existen tres tipos de metodología: "top-down", "bottom-up" y "flat". Las dos primeras implican el crear estructuras jerárquicas, mientras que la última ve el diseño como un todo.

- La metodología "top-down" divide el diseño en componentes funcionales. Cada componente tiene entradas y salidas específicas y desarrolla una determinada función. Cada componente se describe mediante cajas y existen diferentes niveles. Los niveles permiten clarificar la interconexión entre los diferentes componentes.
- La metodología "bottom-up" supone exactamente lo contrario: definiendo y diseñando componentes individuales, se van uniendo para componer el diseño completo.
- La metodología "flat" es aquella en la que los detalles de los componentes son definidos en el mismo nivel que las interconexiones entre ellos. Es la que se considera más apropiada para diseños pequeños, donde el disponer de los detalles de la estructura interna de un componente funcional no distrae del diseño global a nivel de chip.

Codificación del diseño

Seguidamente a la decisión de la metodología a aplicar, podemos describir nuestros diagramas de flujo, diagramas de bloque, etc. con el lenguaje de descripción elegido. Hay que ser muy cuidadoso con la **sintaxis** y con la **semántica**.

Un modo de trabajo utilizado por muchos ingenieros consiste en editar un ejemplo y adaptarlo a las necesidades del diseño concreto.

Simulación del Código Fuente

La simulación de código permite depurar errores funcionales antes de la implementación (síntesis).

Síntesis, Optimización del diseño

Síntesis

Se puede definir como la traducción de la descripción de un diseño a una representación de circuito de bajo nivel. En otras palabras, es el proceso por el cual creamos ecuaciones a partir de descripciones de diseño, en principio abstractas.

El proceso de síntesis depende de la tecnología empleada. El paso de una descripción en VHDL hacia un conjunto de ecuaciones es diferente de un dispositivo a otro. El proceso de síntesis convierte el diseño a unas estructuras de datos internas, traduciendo el "comportamiento" descrito en alto nivel a una descripción de nivel de transferencia de registros (register-transfer level: RTL). La descripción RTL especifica registros, señales de entrada y salida y la lógica combinacional entre ellas. Algunas herramientas de síntesis traducen estructuras de datos en funciones lógicas optimizadas según la arquitectura elegida (ej. algunos dispositivos disponen de puertas XOR o bloques operacionales más complejos; entonces estas herramientas buscan parte de la lógica diseñada que se puede sustituir por estas estructuras).

Optimización

El proceso de optimización depende de tres variables (en inglés constraints):

- La forma de las expresiones booleanas
- El tipo de recursos disponibles
- Las directivas de síntesis utilizadas (tanto automáticas como propias de usuario).

Algunas formas funcionales se implementan mejor en unos recursos que en otros. Por ejemplo, mientras que una suma de términos productos se pueden implementar eficientemente en una PAL, para una expresión canónica se puede utilizar o un multiplexor o una estructura RAM.

La optimización de estructura PLD o CPLD implica la simplificación de las expresiones lógicas a una suma mínima de términos producto, además también se optimiza el número de literales. Para ello se utilizarán técnicas de simplificación de la forma canónica en una suma de términos producto.

La optimización para FPGA's típicamente requiere que la lógica se exprese en otra forma. Se busca entonces factores comunes que se puedan utilizar en diferentes partes del diseño.

Programación del dispositivo

Después de completar la descripción, la síntesis y la simulación del circuito con éxito, el siguiente paso es generar el archivo que nos permite implementar físicamente nuestro diseño en un dispositivo programable. Todos los programas de VHDL para síntesis generan un archivo llamado JEDEC con el que podemos programar el dispositivo.

Proceso de diseño

Un proceso típico para diseñar con los PLD es el siguiente:

1. La lógica a ser implementada en un PLD se expresa en un archivo fuente usando un lenguaje de diseño. Estos incluyen ecuaciones booleanas, tablas de verdad y sintaxis de maquina de estado.

2.El archivo se procesa por un compilador lógico para generar un archivo JEDEC del diseño. El compilador configura los bits en el archivo JEDEC los cuales determinan las conexiones que se harán en el arreglo lógico y configurar así las salidas. La figura 3.8 es una muestra de este proceso.

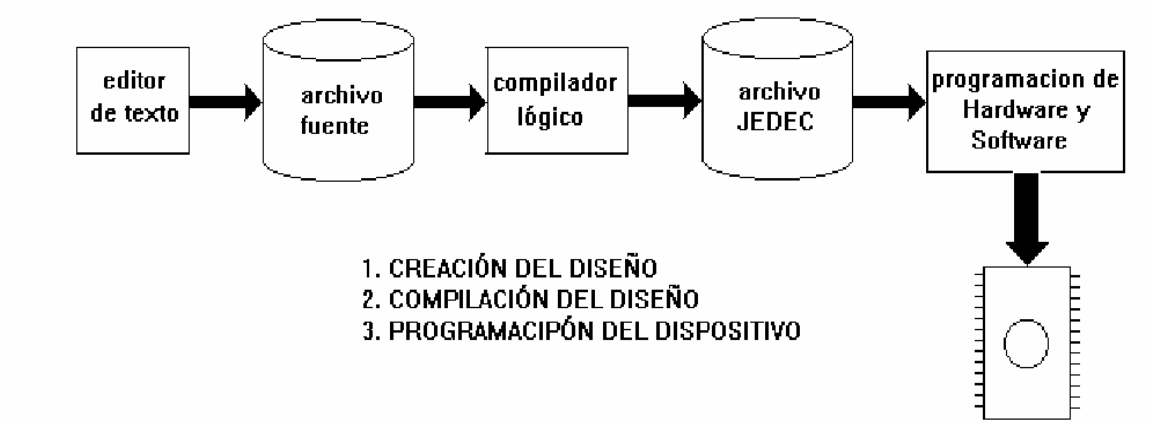


Figura 3.1 Proceso típico de diseño para PLD's[36]

Al final se realiza la programación de hardware y software y se utiliza una PC, un programador de PLD's y después se inserta el CI al programador (figura 3.2).

Figura 3.2 Programación de hardware y software[37]

[36] Véase en www.gte.us.es/creacióndedispositivos.pdf
[37] Véase en www.ElectronicosOnline.com

3.2 DIFERENTES ARQUITECTURAS DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES

ARQUITECTURA BASICA DE LOS PLD's

Existen en la actualidad infinidad de arquitecturas diferentes de PLD's y su número se incrementa día a día. Aunque resulta casi imposible hacer una referencia completa de todos los tipos de PLD's en el mercado, en este trabajo sólo se presentarán algunas de las más comunes y una amplia lista de las distintas PLD's que podemos encontrar en el mercado.

1. CLASES DE PLD's.

Los PLD's disponen de muchas entradas y resultaría muy complicado mostrarlas en un dibujo, se utiliza una representación simplificada, según la cual, para las puertas AND sólo se dibuja una línea de entrada llamada línea producto. Esta línea se cruza con dos líneas por cada entrada (entrada directa y entrada invertida), pudiendo existir un fusible en cada intersección. Aunque sólo se dibuja una línea de entrada por cada puerta AND, en realidad esta puerta tiene tantas entradas como intersecciones de la línea producto.

Si en una intersección hay una X, significa que el fusible está intacto; sí no hay una X, el fusible esta fundido y no existe la conexión. En ocasiones, las puertas OR también se dibujan con una sola entrada. En el diagrama simplificado de la figura 3.3 aparece una matriz de puertas AND de seis entradas, cuyas salidas están conectadas a una puerta OR. La intersección de las líneas producto con las líneas de entrada forman una matriz de puertas AND programable de 6x3 fusibles. El circuito está programado para realizar la función OR -exclusiva entre las entradas A y B.

La puerta AND inferior está marcada con una X. Significa que todos sus fusibles están intactos y que su salida es 0. Cuando se funden todos los fusibles de una línea producto, la salida de la puerta AND asociada es 1.

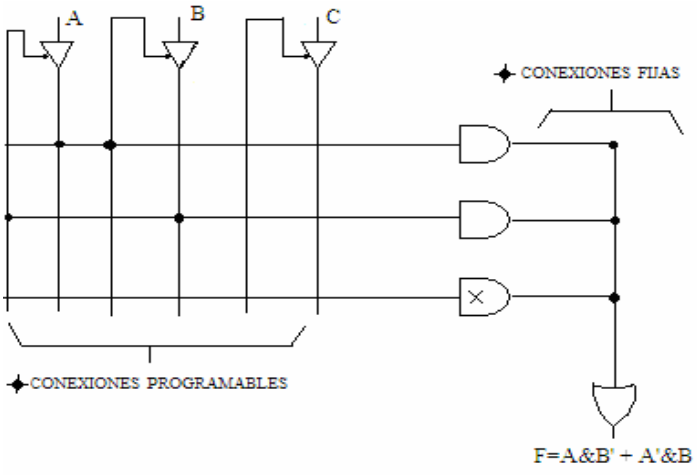


Figura 3.3 Representación simplificada de una función[38]

PAL (Programmable Array Logic). También llamados PLA's, son un tipo de PLD's en las que se pueden programar las uniones en la matriz de puertas AND, siendo fijas las uniones en la matriz de puertas OR (Figura 3.3).

Los dispositivos con arquitectura PAL son los más populares y los más utilizados, razón ésta por la que dedicamos el siguiente capítulo, para analizarlos más a fondo.

[38] Véase en www.uag.mx/214/II_DISPOSITIVOS_LOGICOS_PROGRAMABLES.pdf

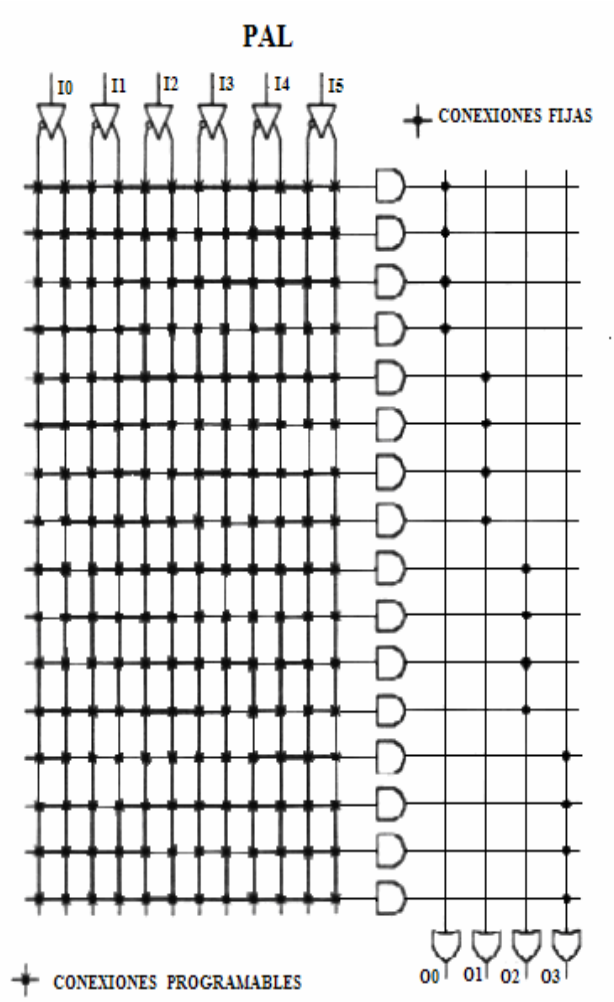


Figura 3.4 Estructura de una PAL[39]

FPLA (Field Programmable Logic Array). Es un PLD en el que se pueden programar las uniones en ambas matrices (Figura 3.5). Son los dispositivos más flexibles, pero resultan penalizados en tamaño y en velocidad debido a los transistores adicionales en la matriz de puertas OR. Se utilizan fundamentalmente para construir máquinas de estados. Para otras aplicaciones, las PAL resultan más efectivas. Las PAL y las FPLA son sistemas combinacionales incompletos porque teniendo n entradas, disponen de menos de 2n términos producto.

[39] Véase en www.latticesemi.com

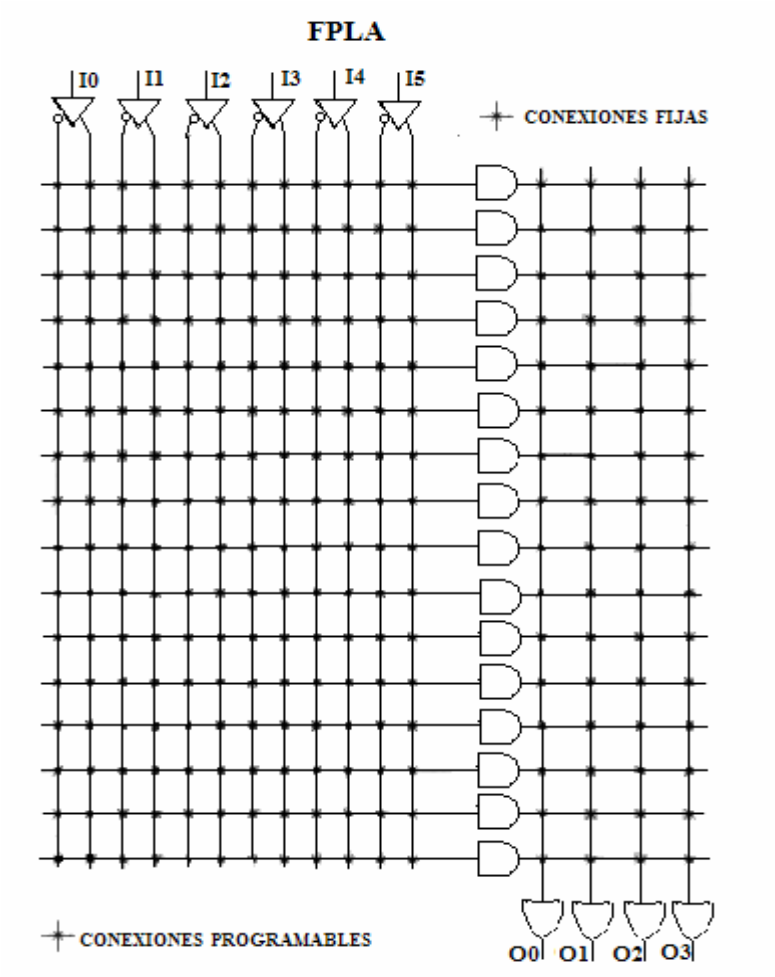


Figura 3.5 Estructura de una FPLA[40]

PROM (Programmable Read Only Memory). Es un PLD en el que las uniones en la matriz de puertas AND es fija, siendo programables las uniones en la matriz de puertas OR como se muestra en la figura 3.6.

Una PROM es un sistema combinacional completo que permite realizar cualquier función lógica con las n variables de entrada, ya que dispone de 2n términos productos. Están muy bien adaptadas para aplicaciones tales como: tablas, generadores de caracteres, convertidores de códigos, etc.

[40] Véase en www.latticesemi.com

Generalmente las PROM tienen menos entradas que las PAL y FPLA. Se pueden encontrar PROM con capacidades potencia de 2, que van desde las 32 hasta las 8192 palabras de 4, 8 o 16 bit de ancho.

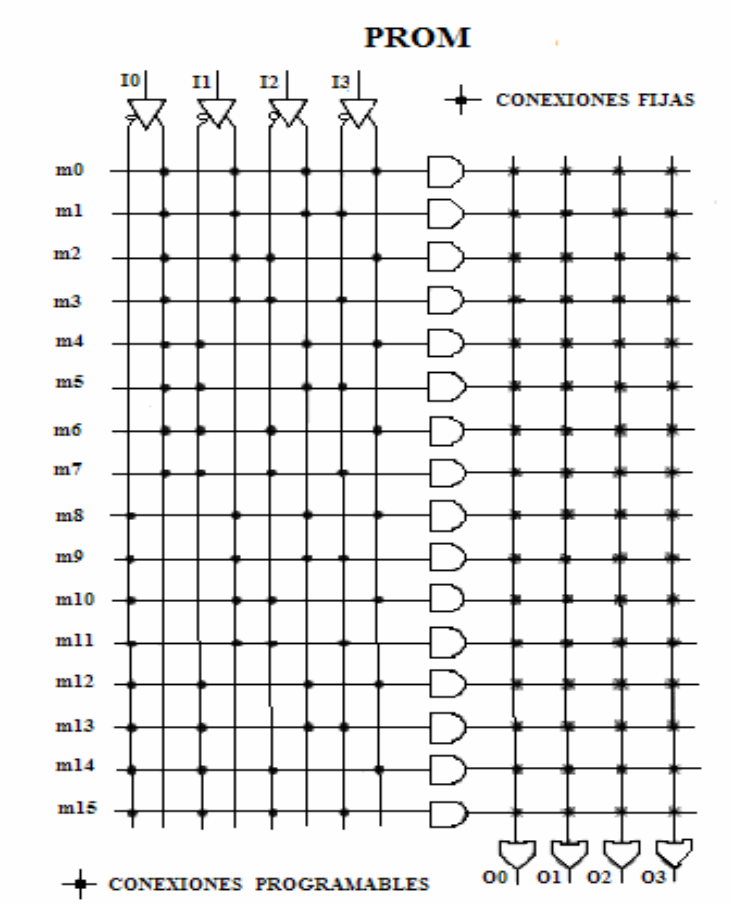


Figura 3.6 Estructura de una PROM[41]

2. COMO SE CATALOGAN LOS PLD's.

Si consultamos las hojas de datos de una PALCE16V8H-20, encontramos claves que permiten extraer valiosa información del nombre del dispositivo. La información incluida en el nombre indica:

[41] Véase en www.latticesemi.com

PAL	Programmable Array Logic.
CE	C-MOS Electrically Erasable.
16V8	16 Entradas a la matriz de puertas AND y ocho salidas.
H	Half Power = 90 mA
20	Tiempo de propagación = 20 nsg.

Las entradas del PLD entran al arreglo lógico los cuales son hechos de columnas y filas. La figura 3.7 muestra tal arreglo. Cada par de columnas representa la entrada negada o complementada y la misma entrada sin negar, cada fila constituyente un termino AND.
Las conexiones lógicas se establecen entre diferentes columnas y filas en el arreglo para determinar cual combinación de entradas llevaran al termino AND a un nivel alto.
Mas de un termino AND alimenta una compuerta OR. La salida es la suma de productos.

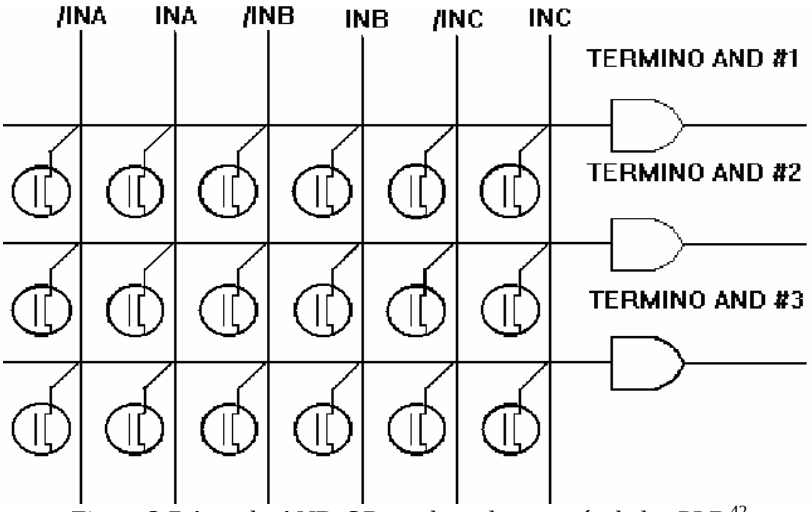


Figura 3.7 Arreglo AND-OR usado en la mayoría de los PLD[42]

[42] Véase en www.ilustrados.com

¿PORQUE EL USO DE LOS PLD's?

La flexibilidad y programación de los PLD's hacen que el diseño con ellos sea mucho mas rápido que diseñar con lógica discreta. Esto es, se puede utilizar el PLD para implementar la mayoría de las funciones hechas con los cientos de dispositivos de la familia lógica "7400". También cabe recalcar que se toma menos espacio sobre el circuito impreso que con los dispositivos discretos.

Una vez tomada la decisión de cambiar de lógica discreta a PLD, la siguiente pregunta es: ¿Cuál PLD? lo más flexible de un PLD es lo utilizable que es, ya que un diseñador puede implementar cualquier ecuación lógica. Hay que escoger PLD's que sean compatibles con los otros dispositivos que se estén utilizando.

Hay que tomar en consideración la potencia que se requiere que se gaste, ya que hay unos que requieren menos potencia para trabajar que otros. La estabilidad es otro factor importante.

3.3 COMPAÑIAS FABRICANTES DE PLD's

Existen diversas compañías internacionales que fabrican o distribuyen dispositivos lógicos programables. Algunas ofrecen productos con características generales y otras introducen innovaciones a sus dispositivos.

Altera Corporation

Altera es una de las compañías más importante de producción de dispositivos lógicos programables y también es la que más familias ofrece, ya que tiene en el mercado ocho familias: APEX™20K, FLEX®10K, FLEX 8000, FLEX 6000, MAX®9000, MAX7000, MAX5000 y Classic™. La capacidad de integración en cada familia pueden variar desde 300 hasta 1000000 de compuertas utilizables por dispositivos, además de que todas tienen la capacidad de integrar sistemas complejos.

Las características generales más significativas de los dispositivos Altera son las siguientes:

- Frecuencia de operación del circuito superior a los 175 Mhz y retardos pin a pin de menos de 5 ns.
- La implementación de bloques de arreglos integrados (EAB), que se usan para realizar circuitos que incluyan funciones aritméticas como multiplicadores, ALU y secuenciadotes. También se aplican en microprocesadores, microcontroladores y funciones complejas con DSP (procesadores digitales de señales).
- La programación en sistemas (ISP), que permite programar los dispositivos montados en la tarjeta.
- Mas de cuarenta tipos y tamaños de encapsulados, incluyendo el TQFP, el cual es un dispositivo delgado, de forma cuadrangular y plano, que permite ahorrar un espacio considerable en la tarjeta.
- Operación multivoltaje, entre los 5 y 3.3 volts, para máximo funcionamiento y 2.5V en sistemas híbridos.

Cypress semiconductor

La compañía Cypress Semiconductor ofrece una amplia variedad de dispositivos lógicos programables complejos (CPLD), que se encuentran en las familias Ultra 37000™ y FLASH370i™. Cada una de estas familias ofrece la reprogramación en sistemas (ISR), la cual permite reprogramar los dispositivos las veces que se quiera dentro de la tarjeta.

Todos los dispositivos de ambas familias trabajan con voltajes de operación de 5 o de 3.3V y en su interior contienen desde 32 hasta 128 macroceldas.

En lo que respecta al software de soporte, Cypress ofrece su poderoso programa Warp, el cual se basa en VHDL.

Este programa permite simular de manera gráfica el circuito programado, generando un archivo de mapa de fusibles (jedec) que puede ser programado directamente en cualquier PLD, CPLD o FPGA de Cypress o de otra compañía que sea compatible.

Clear logic

La compañía Clear Logic introdujo en noviembre de 1998 los dispositivos lógicos procesados por láser (LPLD), tecnología que provee reemplazos de los dispositivos de la Compañía Altera, pero a un costo y tamaño menores. La tecnología LPLD puede disponer de arriba de un millón de transistores para la construir alrededor de 512 macroceldas.

Sustituye al dispositivo MAX7212A de Altera y reduce el tamaño más de 60% respecto al chip original. Las primeras familias introducidas por Clear Logic son CL7000 y CL7000E, las cuales tienden a crecer en un futuro.

Motorola

Motorola, empresa líder en comunicación y sistemas electrónicos, ofrecen también dispositivos FPGA y FPAA (Field Programmable Array Analog: campos progrmables de arreglos analógicos). La FPAA son los primeros campos programables para aplicaciones analógicas, utilizados en las áreas de transporte, redes, computación y telecomunicaciones.

Xilinx

Xilinx es una de las compañías líder en soluciones de lógica programable, incluyendo circuitos integrados avanzados, herramientas en software para diseño, funciones predefinidas y soporte de ingeniería. Xilinx fue la compañía que inventó los FPGA y en la actualidad sus dispositivos ocupan más de la mitad del mercado mundial de los dispositivos lógicos programables.

Los dispositivos de Xilinx reducen de manera significativa el tiempo requerido para desarrollar aplicciones en las áreas de computación, telecomunicaciones, redes, control industrial, instrumentación, aplicaciones militares y para el consumo general.

Las familias de CPLD XC9500 y XC9500XL proveen una larga variedad de dispositivos programables con características que van desde los 5 a 3.3 volts de operación, 36 a 288 macroceldas, 34 a 192 terminales de entrada y salida, y programación en sistema.

Los dispositivos de las familias XC4000 y XC1700 de FPGA manejan voltajes de operación entre los 5 y 3.3 V, una capacidad de integración arriba de las 40 000 compuertas y programación en sistema.

En lo que se refiere al software, Xilinx desarrolló una importante herramienta llamada Foundation Series, que soporta diseños estándares basados en ABEL~HDL y en VHDL. Existe una amplia y variada gama de dispositivos lógicos programables disponibles en el mercado. La elección de uno u otro depende de los recursos con que cuenta el diseñador y los requerimientos del diseño.

Futuro de la lógica programable

Debido al auge actual de la lógica programable, no es difícil suponer que se pretende mejorar a futuro las herramientas existentes con el fin de extender su campo de aplicación a más áreas. Algunas compañías buscan mejorar la funcionalidad e integración de sus circuitos a fin de competir con el mercado de los ASIC. Esto mejoraría el costo por volumen, el ciclo de diseño y se disminuiría el voltaje de consumo. Otra característica que se pretende mejorar es la reprogramación de los circuitos, debido a que su implementación requiere muchos recursos físicos y tecnológicos.

Por esta razón se busca cambiar la metodología de diseño para incluir sistemas reprogramables por completo.

Algunos desarrollos cuentan con memoria RAM o microprocesadores integrados en la tarjeta de programación. La tendencia de algunos fabricantes es integrar estos recursos en un circuito. En la tabla 3.1.

Compañía	Productos de hardware	Herramientas software
Altera	FPGA: Familias APEX 20K, FLEX 6000, MAX 9000, MAX 7000, MAX 5000 y CLASSIC	MAX+PLUS II: Soporta VHDL, Verilog y entrada esquemática.
Chip Express	LPGA(Laser Program Gate Array): CX3000, CX2000 y QYH500	QuICk Place&route: Diseños en base a vectores de prueba (CTV, ChipExpress Test Vector) y VHDL
Clear Logic	LPLD (Laser-processed Logic Device): CL7000, CL7000E, CL7000S	Desarrollos basados en herramientas de Altera
Cypress Semiconductors	PLD: GAL22V10 CPLD:Ultra37000,FLASH370i	WARP: Soporta VHDL, Verilog y esquemáticos
Motorola	FPAA(Field Programmable Analog Array): MPAA020	Easy Analog: herramienta de diseño interactiva exclusiva para diseño con FPAA
Mantis	FPGA: Familias MACH4 y MACH5A	MACHXL: Verilog y VHDL
Quick Logic	PAsic(Asic Programable) y la familia QL de FPGA	Quick Works: herramienta de soporte para VHDL, Verilog y captura esquemática
Xilinx	CPLD: Familia XC9500 y XC9500XL FPGAs Familia XC400 y XC1700	Xilinx Foundation Series: Soporta ABEL-HDL, VHDL y esquemáticos

Tabla 3.1 Compañías fabricantes de PLD's

CAPÍTULO IV

SINTAXIS DEL LENGUAJE

4.1 AMBIENTE DE DESARROLLO DE LÓGICA PROGRAMABLE

Una de las grandes ventajas al diseñar sistemas digitales mediante dispositivos lógicos programables radica en el bajo costo de los recursos requeridos para el desarrollo de estas aplicaciones. De manera general, el soporte básico se encuentra formado por una computadora personal, un grabador de dispositivos lógicos programables y el software de aplicación que soporta las diferentes familias de circuitos integrados PLD.

En la actualidad, diversos programas CAD (Diseño asistido por computadora), como PALASM, OPAL, PLP, ABEL, CUPL, entre otros, se encuentran disponibles para la programación de dispositivos lógicos.

Descripción de compiladores lógicos para PLD

Compilador lógico	Características
PALASM (PAL Assembler: ensamblador de PAL)	• Creado por la compañía Advanced Miro Devices (AMD). • Desarrollado únicamente para aplicaciones con dispositivos PAL. • Acepta el formato de ecuaciones booleanas. • Utiliza cualquier editor que grabe en formato ASCII.
OPAL (Optimal PAL language: lenguaje de optimización para arreglos programables)	• Desarrollado por Nacional Semiconductors • Se aplica en dispositivos PAL y GAL • Formato para usar lenguajes de máquinas de estado, ecuaciones booleanas de distintos niveles, tablas de verdad, o cualquier combinación entre ellas. • Disponible en versión estudiantil y profesional (OPAL Jr y OPAL Pro). • Genera ecuaciones de diseño partiendo de una tabla de verdad.

PLPL (Programable Logic Prpgramming Language: Lenguaje de programación de lógica programable)	• Creado por AMD • Introduce el concepto de jerarquías en sus diseños • Formatos múltiples (ecuaciones booleanas, tablas de verdad, diagramas de estado y las combinaciones entre éstos). • Aplicaciones en dispositivos PAL y GAL.
ABEL (Advanced Bolean Expression Language: Lenguaje avanzado de expresiones booleanas)	• Creado por Data I/O Corporation • Programa cualquier tipo de PLD (Versión 5.0). • Proporciona tres diferentes formatos de entrada: Ecuaciones booleanas, tablas de verdad y diagramas de estado. • Es catalogado como un lenguaje avanzado HDL (Lenguajes de descripción en hardware).
CUPL (Compiler Universal Programmable Logic: Compilador universal de lógica programable)	• Creado por AMD para desarrollo de diseños complejos. • Presenta una total independencia del dispositivo. • Programa cualquier tipo de PLD. • Facilita la generación de descripciones lógicas de alto nivel. • Al igual que ABEL, también es catalogado como HDL.

Estos programas conocidos también como **compiladores lógicos** tienen una función en común: procesar y sintetizar el diseño lógico que se va a introducir en un PLD mediante un método específico (ecuaciones booleanas, diagramas de estado, tablas de verdad).

Método tradicional de diseño con lógica programable

La manera tradicional de diseñar con lógica programable, parte de la representación esquemática del circuito que se requiere realizar y luego define la solución del sistema por el método adecuado (ecuaciones boolenas, tablas de verdad, diagramas de estado, etc.).

Por ejemplo, en la figura 4.1 se observa un diagrama que representa a un circuito construido con compuertas lógicas AND y OR. En este caso se eligió el método de ecuaciones booleanas para representar su funcionamiento, aunque se pudo usar también un también un tabla de verdad.

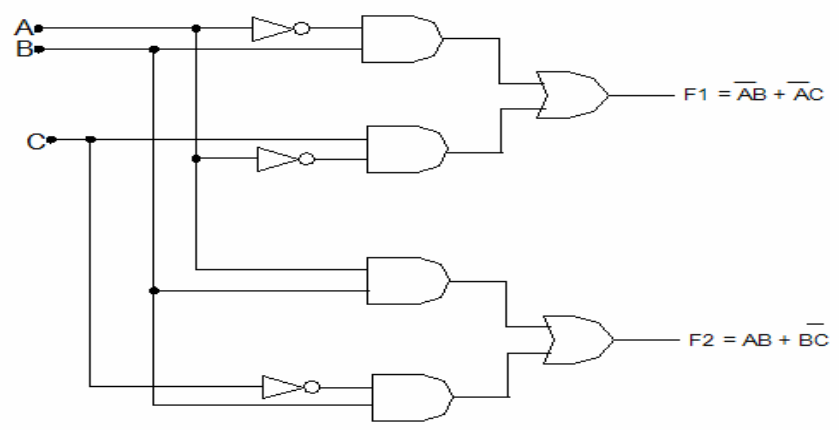


Figura 4.1 Obtención de las ecuacioes booleanas [43]

Como se puede observar, las ecuaciones que rigen el comportamiento del sistema se encuentran derivadas en función F1 y F2 del circuito. Una vez que se obtienen estas ecuaciones, el siguiente paso es introducir en la computadora el archivo fuente o de entrada; es decir, el programa que contiene los datos que permitirán al compilador sintetizar la lógica requerida. Típicamente se introduce alguna información preliminar que indique datos como el nombre del diseñador, la empresa, fecha, nombre del diseño, etc. Luego se especifica el tipo de dispositivo PLD que se va a utilizar, la numeración de los pines de entrada y salida, y las variables del diseño. Por último, se define la función lógica en forma de ecuaciones booleanas o cualquier formato que acepte el compilador.

El siguiente paso consiste en la compilación del diseño, el cual radica básicamente en localizar los errores de sintaxis o de otro tipo, cometidos durante la introducción de los datos en el archivo fuente.

[43] Véase en www.lapham25.tripod.com/archivos/plds.pdf

El compilador procesa y traduce el archivo fuente y minimiza las ecuaciones. En este paso, el diseño se ha simulado utilizando un conjunto de entradas y sus correspondientes valores de salida conocidos como **vectores de prueba.**
Durante este proceso se comprueba que el diseño funcione correctamente antes de introducirlo al PLD. Si se detecta algún error en la simulación, se depura el diseño para corregir este defecto.
Una vez que el diseño no tiene errores, el compilador genera un archivo conocido como **JEDEC** (Joint Electronic Device Engineering Council) o mapa de fusibles.
Este archivo indica al grabador cuáles fusibles fundir y cuáles activar, para que luego se grabe el PLD (de acuerdo con el mapa de fusibles) en un grabador típico.

Ciertos PLD (PROM, PAL, GAL) se programan empleando un grabador de dispositivos lógicos. Algunos otros PLD, como los CPLD y FPGA, presentan la característica de ser programables dentro del sistema (ISP, In- System Programmable); es decir, no hay que introducirlos al grabador, ya que por medio de elementos auxiliares se pueden programar dentro de la tarjeta de circuito integrado.

El método de diseño con lógica programable reduce de manera considerable el tiempo de diseño y permite al diseñador mayor control de los errores que se pudieran presentar, ya que la corrección se realiza en el software y no en el diseño físico.

4.2 LA LÓGICA PROGRAMABLE Y LOS LENGUAJES DE DESCRIPCIÓN EN HARDWARE (HDL)

Como consecuencia de la creciente necesidad de integrar un mayor número de dispositivos en un solo circuito integrado, se desarrollaron nuevas herramientas de diseño que auxilian al ingeniero a integrar sistemas de mayor complejidad.

Esto permitió que en la década de los cincuenta aparecieran los lenguajes de descripción en hardware (HDL) como una opción de diseño para el desarrollo de sistemas electrónicos elaborados.
Un lenguaje HDL para descripción de hardware (HDL: Hardware Description Language) es una herramienta formal para describir el comportamiento y la estructura de sistemas. Estos lenguajes alcanzaron mayor desarrollo durante los años setenta, lapso en que se desarrollaron varios de ellos como IDL de IBM, TI-HDL de Texas Instruments, Zeus de General Electric, etc., todos orientados al área industrial, así como los lenguajes en el ámbito universitario (AHPL, DDL, CDL, ISPS, etc.).

Los primeros no estaban disponibles fuera de la empresa que los manejaba, mientras que los segundos carecían de soporte y mantenimiento adecuado que permitieran su utilización industrial. El desarrollo continuó y en la década de los ochenta surgieron lenguajes como VHDL, Verilog, ABEL 5.0, AHDL, etc., considerados lenguajes de descripción en hardware porque permitieron abordar un problema lógico a nivel funcional (descripción de un problema sólo conociendo las entradas y salidas), lo cual facilita la evaluación de soluciones alternativas antes de iniciar un diseño detallado.

Tipos de lenguajes de descripción en hardware

- **De bajo nivel:** Permiten definir un circuito a nivel de arquitectura (FlipFlops, compuertas básicas, ecuaciones lógicas).
  - **PALASM, CUPL, ABEL**
- **De nivel medio:** Permite definir un circuito en modo jerárquico (funciones aritméticas, maquinas de estado).
  - **AHDL** (Altera Hardware descripción language)
- **De alto nivel:** No solo posibilitan mayor nivel de abstracción, sino que también son usados para la simulación, para la síntesis del generador de estímulos (diagramas de estado).
  - **VHDL, VERILOG HDL**

Una de las principales características de estos lenguajes radica en su capacidad para describir en distintos **niveles de abstracción** (funcional, transferencia de registros RTL y lógico o nivel de compuertas) cierto diseño. Los niveles de abstracción se emplean para clasificar modelos HDL según el grado de detalle y precisión de sus descripciones. Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

- **Algorítmico:** Se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.
- **Transferencia de registro (RT):** Consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.
- **Lógico o de compuertas:** El circuito se expresa en términos de ecuaciones lógicas o de compuertas.

4.3 VHDL, LENGUAJE DE DESCRIPCIÓN EN HARDWARE

En la actualidad, el lenguaje de descripción en hardware más utilizado a nivel industrial es VHDL (Hardware Description Language), que apareció en la década de los ochenta como un lenguaje estándar, capaz de soportar el proceso de diseño de sistemas electrónicos complejos, con propiedades para reducir el tiempo de diseño y los recursos tecnológicos requeridos. El Departamento de Defensa de Estados Unidos creó el lenguaje VHDL como parte del programa “Very High Speed Integrated Circuits” (VHSIC), a partir del cual se detectó la necesidad de contar con un medio estándar de comunicación y la documentación para analizar la gran cantidad de datos asociados para el diseño de dispositivos de escala y complejidad deseados; es decir, VHSIC debe entenderse como la rapidez en el diseño de circuitos integrados.

En 1983, IBM, Intermetrics y Texas Instruments empezaron a trabajar en el desarrollo de un lenguaje de diseño que permitiera la estandarización, facilitando con ello, el mantenimiento de los diseños y la depuración de los algoritmos, para ello el IEEE propuso su estándar en 1984.

Tras varias versiones llevadas a cabo con la colaboración de la industria y de las universidades, que constituyeron a posteriori etapas intermedias en el desarrollo del lenguaje, el IEEE publicó en diciembre de 1987 el estándar IEEE std 1076-1987 que constituyó el punto firme de partida de lo que después de cinco años sería ratificado como VHDL.
Esta doble influencia, tanto de la empresa como de la universidad, hizo que el estándar asumido fuera un compromiso intermedio entre los lenguajes que ya habían desarrollado previamente los fabricantes, de manera que éste quedó como ensamblado y por consiguiente un tanto limitado en su facilidad de utilización haciendo dificultosa su total comprensión. Este hecho se ha visto incluso ahondado en su revisión de 1993. Surgió la necesidad de describir en VHDL todos los ASIC creados por el Departamento de Defensa, por lo que en 1993 se adoptó el estándar adicional VHDL IEEE1164.

Pero esta deficiencia se ve altamente recompensada por la disponibilidad pública, y la seguridad que le otorga el verse revisada y sometida a mantenimiento por el IEEE.
La independencia en la metodología de diseño, su capacidad descriptiva en múltiples dominios y niveles de abstracción, su versatilidad para la descripción de sistemas complejos, su posibilidad de reutilización y en definitiva la independencia de que goza con respecto de los fabricantes, han hecho que VHDL se convierta con el paso del tiempo en el *lenguaje de descripción de hardware* por excelencia.
Hoy en día VHDL se considera como un estándar para la descripción, modelado y síntesis de circuitos digitales y sistemas complejos. Este lenguajes presenta diversas características que lo hacen uno de lo HDL más utilizados en la actualidad. Finalmente la letra “V” dentro de VHDL hace referencia al proyecto (VHSIC).

4.3.1 Ventajas y desventajas del desarrollo de circuitos integrados con VHDL

VENTAJAS

Ventajas que presentan los circuitos integrados con VHDL:

- **Notación formal.** La programación en VHDL cuenta con una notación que permite su uso en cualquier diseño electrónico.
- **Disponibilidad pública.** VHDL es un estándar no sometido a patente o marca registrada aluna, por lo que cualquier empresa o institución puede utilizarla sin restricciones. Además, dado que el IEEE lo mantiene y documenta, existe la garantía de estabilidad y soporte.
- **Independencia tecnológica de diseño.** VHDL se diseño para soportar diversas tecnologías de diseño (PLD, FPGA, ASIC, etc.) con distinta funcionalidad (circuitos combinacionales, secuenciales, síncronos y asíncronos), a fin de satisfacer las distintas necesidades de diseño.
- **Independencia de la tecnología y proceso de fabricación.** VHDL se creó para que fuera independiente de la tecnología y del proceso de fabricación del circuito o del sistema electrónico.

  El lenguajes funciona de igual manera en circuitos diseñados con tecnología MOS, bipolares, BICMOS, etc., sin necesidad de incluir diseño información concreta de la tecnología utilizada o de sus características (retardos, consumos, temperaturas, etc.), aunque esto puede hacerse de manera opcional.
- **Capacidad descriptiva en distintos niveles de abstracción.** El proceso de diseño consta de varios niveles de detalle, desde la especificación hasta la implementación final (niveles de abstracción). VHDL ofrece la ventaja de poder diseñar en cualquiera de estos niveles y combinarlos, con lo cual se genera lo que se conoce como simulación **multinivel.**

- **Uso como formato de intercambio de información.** VHDL permite el intercambio de información a lo largo de todas las etapas del proceso de diseño, con lo cual favorece el trabajo en equipo.
- **Independencia de los proveedores.** Debido a que VHDL es un lenguaje estándar, permite que las descripciones o modelos generados en un sitio sean accesibles desde cualquier otro, sean cuales sean las herramientas de diseño utilizadas.
- **Reutilización del código.** El uso de VHDL como lenguaje estándar permite reutilizar los códigos en diversos diseños, sin importar que hayan sido generados para una tecnología (CMOS, bipolar, etc.) e implementación (FPGA, ASIC, etc.) en particular.
- **Facilitación de la participación en proyectos internacionales.** En la actualidad VHDL constituye el lenguaje estándar de referencia a nivel internacional. Impulsado en sus inicios por el Departamento de Defensa de Estados Unidos, cualquier programa lanzado por alguna de las dependencias oficiales de ese país vuelve obligatorio su uso para el modelado de los sistemas y la documentación del proceso de diseño.

  Este hecho ha motivado que diversas empresas y universidades adopten a VHDL como su lenguaje de diseño.

  En Europa la situación es similar, ya que en nuestros días la mayoría de las grandes empresas del ramo lo ha definido como el lenguaje de referencia en todas las tareas de diseño, modelado, documentación y mantenimiento de los sistemas electrónicos. De hecho, el número de usuarios de VHDL en Europa es mayor que en Estados Unidos, debido en gran parte a que resulta el lenguaje más común en la mayoría de los consorcios.

DESVENTAJAS

Desventajas del desarrollo de circuitos integrados con VHDL

VHDL presenta grandes ventajas; sin embargo, es necesario mencionar también algunas desventajas que muchos diseñadores consideran importantes:

- En algunas ocasiones, el uso de una herramienta propuesta por alguna compañía en especial tiene características adicionales al lenguaje, con lo que se pierde un poco la libertad de diseño. Como método alternativo, se pretende que entre diseñadores que utilizan distintas herramientas exista una compatibilidad en sus diseños, sin que esto requiera un esfuerzo importante en la traducción del código.
- Debido a que VHDL es un lenguaje diseñado por un comité, presenta una alta complejidad, ya que se debe dar gusto a las diversas opiniones de los miembros de éste, por lo que resulta un lenguaje difícil de aprender.

4.4 VHDL: SU ORGANIZACIÓN Y ARQUITECTURA

Introducción

Tal como lo indican sus siglas, VHDL (Hardware Description Language) es un lenguaje orientado a la descripción o modelado de sistemas digitales; es decir, se trata de un lenguajes mediante el cual se puede describir, analizar y evaluar el comportamiento de un sistema electrónico digital.

VHDL es un lenguaje poderoso que permite la integración de sistemas digitales sencillos, elaborados en un dispositivo lógico programable, sea de baja capacidad de integración como un GAL, o de mayor capacidad como los CPLD y FPGA.

En un principio VHDL, al igual que los demás HDL's, nacieron con el propósito de facilitar la labor de los diseñadores de circuitos electrónicos, agilizando su diseño y haciendo más flexible su posterior depuración y mantenimiento.

Por este motivo se dotó a VHDL con abundantes instrucciones más orientadas a la simulación que a la implementación física del diseño. Esto dio la diferenciación del VHDL sintetizable del simulable, siendo este último el más extendido y el que cuenta con más herramientas en los programas. Si trabajamos con VHDL sintetizable, sólo podremos hacer uso de un conjunto de instrucciones válidas.

El lenguaje nos permite describir circuitos complejos manejando todas las sentencias y herramientas de las que dispone, pero no siempre se garantiza que se pueda llegar a grabar en un dispositivo de lógica programable (PLD), ya que ciertas instrucciones no tienen equivalente físico. Como conclusión, se puede decir que todo el código de un programa en VHDL es simulable, pero no siempre será sintetizable.
Dado que el VHDL es el lenguaje estándar, todas las empresas fabricantes de PLD's y FPGA's (Cypress, Xilinx, Altera, Actel, etc.) han desarrollado su propio compilador cada uno con sus propias funciones y características especiales.

A pesar de ser un producto no muy pensado de cara al usuario, la herramienta de Cypress es tan potente como las demás, contando además con un gran soporte técnico vía e-mail. El nombre de la herramienta se llama warp R6.3, que actualmente va por su versión 6.3.
Este conjunto de programas está orientado a la creación de un fichero propio de VHDL (*.vhd), para compilarlo (Galaxy) y posteriormente simularlo (Nova). Para conseguir este software se puede solicitar por correo ordinario o electrónico en la página web de Cypress[44].

[44] Véase en www.cypress.com

4.4.1 Unidades básicas de diseño

La estructura general de un programa en VHDL está formada por módulos o unidades de diseño, cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen, describen, estructuran, analizan y evalúan el comportamiento de un sistema digital.

Existen cinco tipos de unidades de diseño en VHDL:

1) Declaración de entidades (entity declaration)
2) Arquitectura (architecture)
3) Configuración (configuration)
4) Declaración del paquete (package declaration)
5) Cuerpo del paquete (package body)

En el desarrollo de programas en VHDL pueden utilizarse o no tres de los cinco módulos, pero dos de ellos (entidad y arquitectura) son indispensables en la estructuración de un programa.
Las declaraciones de entidad, paquete y configuración se consideran unidades de diseño primarias, mientras que la arquitectura y el cuerpo del paquete son unidades de diseño secundarias porque dependen de una entidad primaria que se debe analizar antes que ellas.

4.4.2 Entidad

Una entidad (entity) es el bloque elemental de diseño en VHDL. Las entidades son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.) que forman de manera individual o en conjunto un sistema digital.

La entidad puede representarse de muy diversas maneras; por ejemplo en la figura 4.2.a) muestra la arquitectura de un sumador completo a nivel de compuertas; ahora bien, esta entidad se puede representar a nivel de sistema indicando tan sólo las entradas (Cin, A y B) y salidas (SUMA y Cout) del circuito: figura 4.2 b).

De igual forma, la integración de varios subsistemas (medio sumador) puede representarse mediante una entidad [figura 4.2 c)]. Los subsistemas pueden conectarse internamente entre si; pero la entidad sigue identificando con claridad sus entradas y salidas generales.

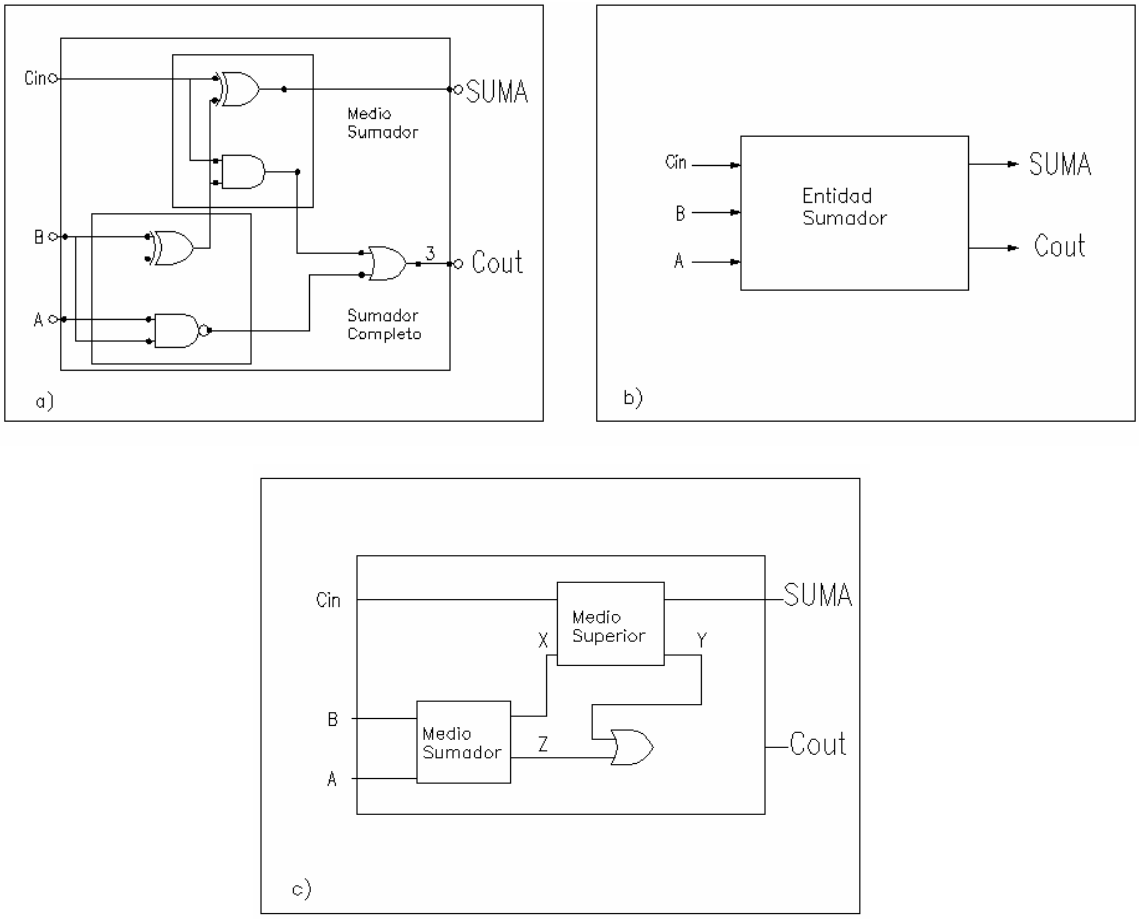


Figura 4.2 a) Descripción a nivel de compuerta b) Símbolo funcional de la entidad c) Diagrama a bloques representativo de la entidad.[45]

[45] Véase en MAXINEZ G., David y Jessica Alcalá Jara, ***VHDL, El arte de programar sistemas digitales,*** 2ª ed., Compañía Editorial Continental, México,2004, p.290

1) Puertos de entrada-salida

Cada una de las señales de entrada y salida en una entidad son referidas como puerto, el cual es similar a una terminal (pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un nombre, un modo y un tipo de dato. El nombre se utiliza como una forma de llamar al puerto; el modo permite definir la dirección que tomará la información y el tipo define que clase de información se transmitirá por el puerto.

Por ejemplo, respecto a los puertos de la entidad que representan a un comparador de igualdad (figura 4.3), las variables a y b denotan los puertos d entrada y la variable c se refiere al puerto de salida.

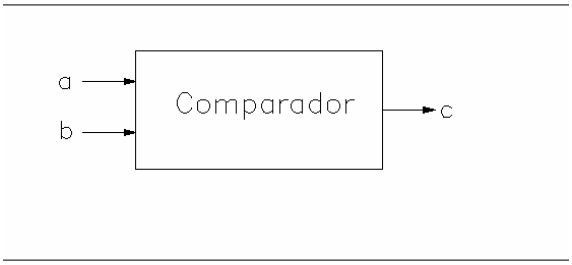


Figura 4.3 Comparador de igualdad[46]

2) Modos

Un modo permite definir la dirección en la cual el dato es transferido a través de un puerto. Un modo puede tener uno de cuatro valores: in (entrada), out (salida), inout (entrada/salida) y buffer como se muestra en la figura 4.4.

- **Modo in**. Se refiere a las señales de entrada. Este sólo es unidireccional y nada más permite el flujo de datos hacia dentro de la entidad.
- **Modo out**. Indica las señales de salida de la entidad.
- **Modo inout**. Permite declarar a un puerto de forma bidireccional es decir, de entrada/salida; además permite la retroalimentación de señales dentro o fuera de la entidad.

[46] Véase en www.monografias.com

- **Modo buffer**. Permite hacer retroalimentaciones internas dentro de la entidad, pero a diferencia del modo inout, el puerto declarado se comporta como una terminal de salida.

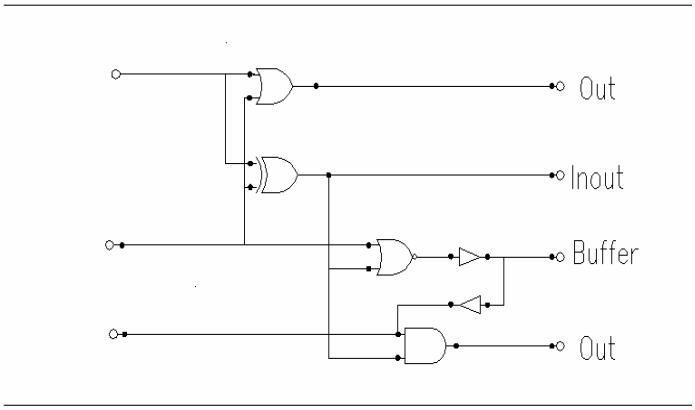


Figura 4.4 Modos y el curso de sus señales[47]

3) Tipos de datos

Los tipos son los valores (datos) que el diseñador establece para los puertos de entrada y salida dentro de una entidad; se asignan de acuerdo con las características de un diseño en particular. Algunos de los tipos más utilizados en VHDL son:

- **Bit**, el cual tiene valores de 0 y 1 lógico.
- **Boolean** (booleano) que define valores de verdadero o falso en una expresión.
- **Bit_vector** (vectores de bits) que representa un conjunto de bits para cada variable de entrada o salida.
- **Integer** (entero) que representa un número entero.

[47] Véase en NAGLE TROY H., ***Análisis y diseño de circuitos lógicos digitales,*** 4ª ed., Editorial Prentice Hall, México 1996, p.200

4.4.3 Declaración de entidades

La declaración de una entidad consiste en la descripción de las entradas y salidas de un circuito de diseño identificado como entity (entidad); es decir, la declaración señala las terminales o pines de entrada y salida con que cuenta la entidad de diseño.
Por ejemplo la forma de declarar la entidad de un circuito sumador se muestra a continuación en la figura 4.5.

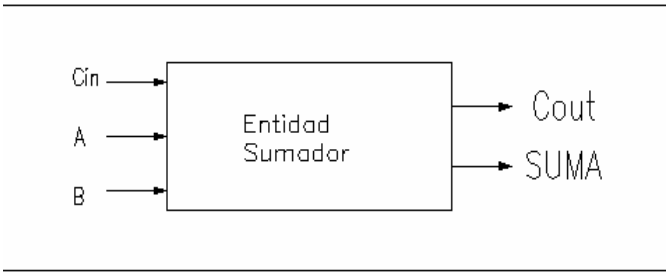


Figura 4.5 Declaración de la entidad sumador[48]

```
1  - -Declaración de la entidad de un circuito sumador
2  entity sumador is
3  port (A, B Cin: in bit;
4        Suma Cin: out bit);
5  end sumador;
```

Los números de las líneas (1, 2, 3, 4, 5) no son parte del código; se usan como referencia para explicar alguna sección en particular. Las palabras en negritas están reservadas para el leguaje de programación VHDL; esto tiene un significado especial para el programador; el diseñador asigna los otros términos.
Ahora comencemos a analizar el código línea por línea.

[48] Véase en www.monografias.com/sumador.html

En la línea 1 inicia con dos guiones (--), los cuales indican que el texto que está a la derecha es un comentario cuyo objetivo es documentar el programa, ya que el compilador ignora todos los comentarios.

En la línea 2 se indica la declaración de la entidad con la palabra reservada **entiry,** seguida del **identificador** o nombre de la entidad (sumador) y la palabra reservada **is.** Los puertos de entrada y salida (**port**) se declaran en las líneas 3 y 4, respectivamente en este caso los puertos de entrada son A, B y Cin, mientras que SUMA y Cout representan los puertos de salida.

El tipo de dato que cada puerto maneja es del tipo **bit**, lo cual indica que sólo pueden manejarse valores de '0' y '1' lógicos. Por último, en la línea 5 termina la declaración de entidad con la palabra reservada **end**, seguida del nombre de la entidad (sumador).

Como cualquier lenguaje de programación, VHDL sigue una sintaxis y una semántica dentro del código, mismas que hay que respetar. En esta entidad conviene hacer notar el uso de punto y coma (;) al finalizar una declaración y de dos puntos (:) al asignar nombres a las entradas y salidas.

Ejemplo 1

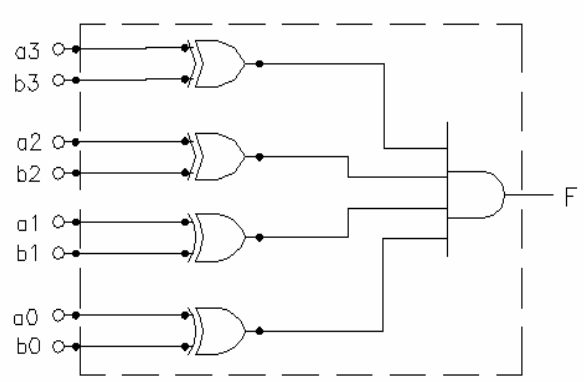


Figura 4.6 Circuito lógico[49]

[49] Véase en www.ilustrados.com

Solución

Como se puede observar, las entradas y salidas del circuito se encuentran delimitadas por la línea punteada. En este caso, a3, b3, a2, b2,….a0, b0 son las entradas y F es la salida.

La declaración de entidad sería de la siguiente forma:

```
1 - -Declaración de la entidad
2    entity circuito is
3       port (a3, b3, a2, b2, a1, b1, a0, b0: in bit;
4                                              F: out bit);
5    end circuito;
```

Identificadores

Los identificadores son simplemente los nombres o etiquetas que se usan para referir variables, constantes, señales, procesos, etc. Pueden ser números, letras del alfabeto y guiones (_) que separen caracteres y no tienen una restricción en cuanto a su longitud. Todos los identificadores deben seguir ciertas especificaciones o reglas para que se puedan compilar sin errores, mismos que aparecen en la tabla 4.1.

Regla	Incorrecto	Correcto
El primer carácter siempre es una letra mayúscula o minúscula	4Suma	Suma4 SUMA4
El segundo carácter no puede ser un guión bajo	S_4bits	S4_bits
Dos guiones juntos no son permitidos	Resta__4	Resta_4_
Un identificador no puede utilizar símbolos	Clear#8	Clear_8

Tabla 4.1 Especificaciones para la escritura de identificadores

4.4.4 Diseño de entidades mediante vectores

La entidad **sumador** usa bits individuales, los cuales sólo pueden representar dos valores (0 o 1). De manera general, en la práctica se utiliza conjuntos (palabras) de varios bits; en VHDL las palabras binarias se conocen como **vectores de bits**, los cuales se consideran un grupo y no como bits individuales.

Como ejemplo los vectores de 4 bits que se muestran a continuación:

vector_A = [A3,A2,A1,A0]
vector_B = [B3,B2,B1,B0]
vector_SUMA = [S3,S2,S1,S0]

En la figura 4.7 se observa la entidad del sumador, las entradas A, B y la salida SUMA incorporan vectores de 4 bits en sus puertos. La entrada Cin y la salida Cout son de un bit.

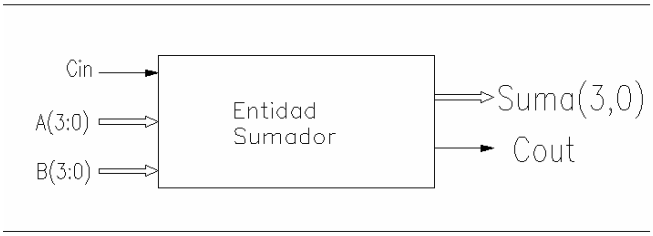


Figura 4.7 Entidad representada por vectores.[50]

La manera de describir en VHDL una configuración que utilice vectores consiste en la utilización de la sentencia **bit_vector**, mediante la cual se especifican los componentes de cada uno de los vectores utilizados. La parte del código que se usa para declarar un vector dentro de los puertos es el siguiente.

[50] Véase en T. L. Floyd, ***Fundamentos de sistemas digitales,*** 2ª ed., Editorial Prentice Hall, México, 2005, p.385

port (vector_A, vector_B: **in** bit_vector (3 **downto** 0));

port (vector_SUMA: **out** bit_vector (3 **downto** 0));

Esta declaración define los vectores (A, B y SUMA) con cuatro componentes distribuidos en orden descendente por medio del comando:

3 **downto** 0 (3 **hacia** 0)

Los cuales se agrupan de la siguiente manera.

vector_A(3) = A3	vector_B(3) = B3	vector_SUMA(3) = S3
vector_A(2) = A2	vector_B(2) = B2	vector_SUMA(2) = S2
vector_A(1) = A1	vector_B(1) = B1	vector_SUMA(1) = S1
vector_A(0) = A0	vector_B(0) = B0	vector_SUMA(0) = S0

Una vez que se ha establecido el orden en que aparecerán los bits enunciados en cada vector, no se puede modificar, a menos que se utilice el comando **to**:

0 **to** 3 (0 **hasta** 3)

que indica el orden de aparición en sentido **ascendente**.

Para describir en VHDL la entidad del circuito sumador representado en la figura 4.7. Observe como la entrada Cin (Carry in) y la salida Cout (Carry out) se expresan de forma individual.

```
entity sumador is
port (A,B: in bit_vector (3 downto 0));
        Cin: in bit;
        Cout: out bit;
        SUMA: out bit_vector (3 downto 0);
end sumador;
```

Declare la entidad del circuito lógico en la figura del ejemplo 4.7, mediante vectores.

```
1 -Declaración de entidades mediante vectores
2  entity circuito is
3  port (
4          a,b: in bit_vector(3 downto 0));
5             F: out bit)
6         );
7  end circuito;
```

4.4.5 Arquitecturas

Una arquitectura (architecture) se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevarán a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas.

La gran ventaja que presenta VHDL para definir una arquitectura radica en la manera en que pueden describirse los diseños; es decir, mediante el **algoritmo de programación** empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.
De manera general, los estilos de programación utilizados en el diseño de arquitecturas se clasifican como:

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

El nombre asignado a estos estilos no es importante, ya que es tarea del diseñador escribir el comportamiento de un circuito utilizado uno u otro estilo que a su juicio le sea el más acertado.

- **Descripción funcional**

En la figura 4.8 se describe funcionalmente el circuito comparador. Se trata de una descripción funcional porque expone la forma en que trabaja el sistema; es decir, las descripciones consideran la relación que hay entre las entradas y las salidas del circuito, sin importar como este organizado en su interior.

Para este caso:

si a = b entonces c = 1

si a ≠ b entonces c = 0

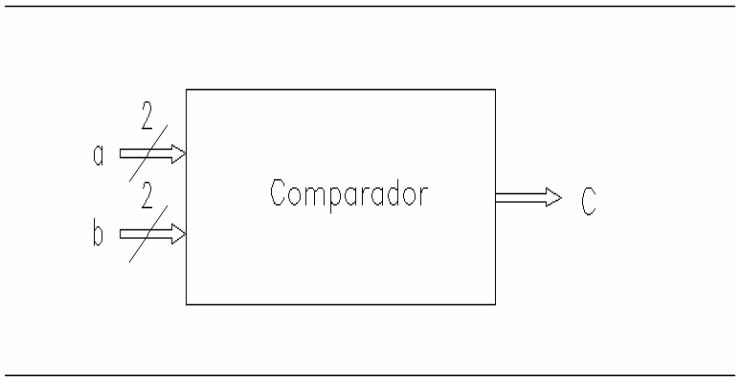


Figura 4.8 Descripción funcional de un comparador de igualdad de dos bits [51]

[51] Véase en T. L. Floyd, OP.CIT. p.190

El código que representa el circuito de la figura 4.8 se muestra en el siguiente listado:

```
1   --Ejemplo de una descripcion funcional
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity comp is
5   port (a,b: in bit_vector (1 downto 0);
6             c: out bit);
7   end comp;
8   architecture functional of comp is
9   begin
10   compara: process (a,b)
11   begin
12          if a = b then
13             c <=  '1';
14          else
15              c<= '0';
16   end if;
17   end process compara;
18   end funcional;
```

La declaración de la entidad (entity) se describen en las líneas de la 1 a la 7; el código ocupa de la línea 8 a la 18, dode se desarrolla el algoritmo (**architecture**) que describe el funcionamiento del comparador.

Para iniciar la declaración de la arquitectura (línea 8), es necesario definir un nombre arbitrario con que se pueda identificar (funcional) además de incluir la entidad con que se relaciona (comp). En la línea 9 es el inicio (**begin**) de la sección donde se comienzan a declarar los procesos que rigen el comportamiento del sistema.

La declaración del proceso (línea 10) se utiliza para la definición de algoritmos y comienza con una etiqueta opcional (compara en este ejemplo), seguida de dos puntos (:), la palabra reservada **process** y une la lista (a y b), que hace referencia a las señales que determinan el funcionamiento del proceso. De la línea 12 a la 17 el proceso se ejecuta mediante declaraciones secuenciales del tipo **if-then-else** (si-entonces-si no). Esto se interpreta como sigue (línea 12): **si** el valor de la señal *a* es igual al valor de la señal *b*, **entonces** '1' se asigna a *c*, **si no** (else) se asigna un '0' (el símbolo <= se lee como "se asigna a"). Una vez que se ha definido el proceso, se termina con la palabra reservada **end process** y de manera opcional el nombre del proceso (compara); de forma similar se añade la etiqueta (funcional) al terminar la arquitectura en la línea 18.

La **descripción funcional** se basa principalmente en el uso de procesos y de declaraciones secuenciales, las cuales permiten modelar la función con rapidez.
Describa mediante declaraciones del tipo **if-then-else** el funcionamiento de la compuerta OR mostrada en la siguiente figura 4.9 con base en la tabla de verdad.

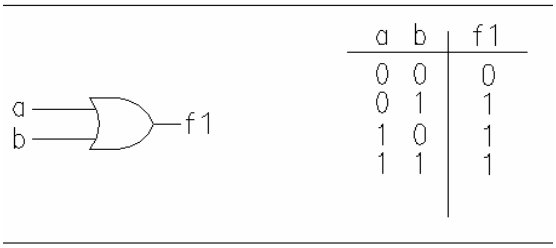


Figura 4.9 Compuerta OR[52]

La declaración de la librería y el paquete se introducen en las líneas 2 y 3, respectivamente. La declaración de la entidad se define entre las líneas 4 a 7. Por último, la arquitectura se describe en las líneas 8 a 17. En VHDL se manejan dos tipos de declaraciones: *secuencial* y *concurrentes*.

[52] Véase en www.ilustrados.com/compuertas/tutorial.pdf

Una declaración secuencial de la forma **if-then-else** se halla en el siguiente listado dentro del proceso, donde su ejecución debe seguir un orden para evitar la pérdida de la lógica descrita.

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

- **Descripción por flujo de datos**

La descripción por flujo de datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales (if-then-else). Este tipo de descripciones permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones. En este tipo de descripción se pueden utilizar dos formatos: mediante instrucciones **when-else** (cuando-si no) o por medio de **ecuaciones booleanas.**

a) Descripción por flujo de datos mediante when-else

En una declaración concurrente no importa el orden en que se ejecutan. Tal es el caso del siguiente listado.

```
1   - -Ejemplo de declaraciones de la entidad de un comparador
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity comp is

5   port(a,b: in bit_vector (1 downto 0);
6           c: out bit);
7   end comp;
8   architecture f_datos of comp is
9   begin
10   c<= '1' when (a = b) else '0'; (asigna a c el valor
11   de 1 cuando a = b si no vale 0).
12   end f_datos;
```

Describa mediante declaraciones del tipo **when-else** el funcionamiento de la siguiente compuerta AND de la figura 4.10.

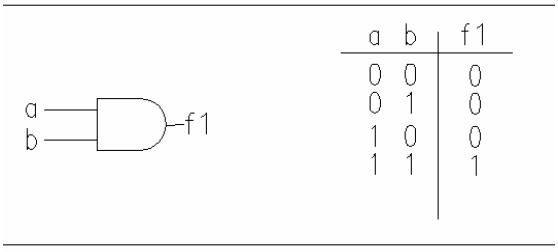


Figura 4.10 Compuerta AND[53]

[53] Véase en www.ilustrados.com/compuertas/tutorial.pdf

```
1   - -Algoritmo utilizando flujo de datos
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity com_and is
5   port (a,b: in std_logic;
6             f: out std_logic);
7   end com_and;
8   architecture compuerta of com_and is
9   begin
10        f <= '1' when (a = '1' and b = '1') else
11              '0';
12   end compuerta;
```

b) Descripción por flujo de datos mediante ecuaciones booleanas

Otra forma de describir el circuito comparador de dos bits es mediante la obtención de sus ecuaciones booleanas figura 4.11.

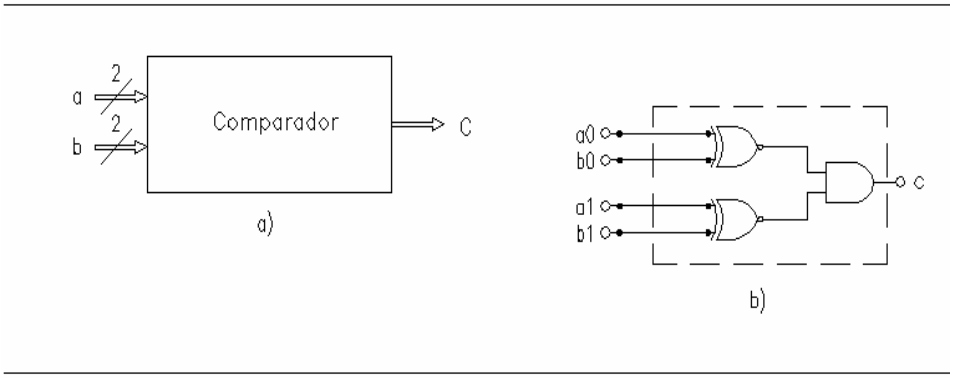


Figura 4.11 a) Entidad del comparador de dos bits
b) Comparador de dos bits realizado con compuertas[54]

[54] Véase en PARDO y Boluda, ***VHDL, El lenguaje para síntesis y modelado de circuitos,*** 2ª ed., Editorial Alfaomega, México, 2002, p.45

El interior del circuito comparador de la figura 4.11a) puede representarse por medio de compuertas básicas figura 4.11b) y este circuito puede describirse mediante la obtención de sus ecuaciones booleanas.

```
1   - - Ejemplo de declaraciones de la entidad de un comparador
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity comp is
5   port (a,b: in bit_vector (1 downto 0);
6             c: out bit);
7   end comp;
8   architecture booleana of comp is
9   begin
10        c <= (a (1)  xnor b (1);
11           and a (0) xnor b(0));
12   end booleana;
```

La forma de **flujo de datos** en cualquiera de sus representaciones describe el camino que los datos siguen al ser transferidos de las operaciones efectuadas entre las entradas a y b a la señal de salida c.

Describa mediante ecuaciones booleanas el circuito mostrado en la siguiente figura 4.12.

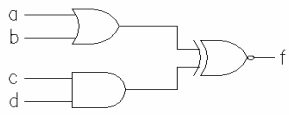


Figura 4.12 Circuito digital[55]

[55] Véase en PARDO y Boluda, OP.CIT. p.74

```
1   - -Declaración mediante ecuaciones booleanas
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity ejemplo is
5   port (a,b,c,d: in std_logic;
6                 f: out std_logic);
7   end ejemplo;
8   architecture compuertas of ejemplo is
9   begin
10        f <= ((a or b) xnor (c and d));
11   end compuertas;
```

- **Descripción estructural**

Una **descripción estructural** basa su comportamiento en modelo lógico establecidos (compuertas, sumadores, contadores, etc.). El usuario puede diseñar estas estructuras y guardalas para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se este utilizando.

En la figura 4.13 se encuentra un esquema del circuito comparador de igualdad de 2 bits, el cual está formado por compuertas nor-exclusivas y una compuerta AND.

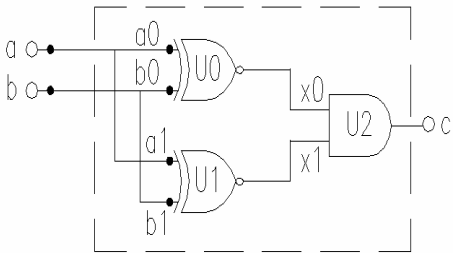


Figura 4.13Representación esquemática de un comparador de 2 bits.[56]

[56] Véase en www.ilustrados.com/compuertas/tutorial.pdf

Comparación entre los estilos de diseño

El estilo de diseño utilizado en la programación del circuito despende del diseñador y de la complejidad del proyecto. Por ejemplo, un diseño puede describirse por medio de ecuaciones booleanas, pero si es muy extenso quizá sea más apropiado emplear estructuras jerárquicas para dividirlo; si se requiere diseñar un sistema cuyo funcionamiento dependa sólo de sus entradas y salidas, es conveniente utilizar la descripción funcional, la cual presenta la ventaja de requerir menos instrucciones y el diseñador no necesita un conocimiento previo de cada componente del circuito.

4.5 INTRODUCCIÓN A LA DESCRIPCIÓN EN VHDL DE CIRCUITOS DIGITALES

Un sistema digital es una combinación de dispositivos diseñada para manipular cantidades físicas o información que esté representadas en forma digital; esto es, que sólo pueden tomar valores discretos. Algunos de los sistemas digitales más conocidos son las computadoras y calculadoras, los relojes digitales y las máquinas de escribir.

Los circuitos digitales presentan ventajas como las siguientes:

- Son fáciles de diseñar
- Presentan facilidad para almacenar información
- Tienen mayor exactitud y precisión
- El ruido puede afectarlos de manera mínima

La limitación más grande de las técnicas digitales es que el mundo real es fundamentalmente analógico.

En los sistemas digitales, la información que se está procesando por lo general se presenta en forma binaria. Dichas cantidades binarias sólo pueden representarse en dispositivos con dos estados de operación y por medio de voltajes (o corrientes) que están representadas en las entradas o salidas de los diversos circuitos digitales.

Los circuitos digitales están diseñados para producir voltajes de salida que se clasifican dentro de los intervalos de voltaje prescritos 0 y 1.

Casi todos los circuitos digitales que se utilizan en los sistemas digitales modernos son circuitos integrados (CI).

Un circuito integrado es un semiconductor pequeño de silicio, llamado pastilla, que contiene componentes eléctricos como transistores, diodos, resistores y capacitores. Los diversos componentes están interconectados dentro de la pastilla para formar un circuito electrónico.

Clasificación de circuitos digitales

Una familia o tecnología de circuitos digitales corresponde al conjunto de circuitos integrados están construidos con determinado tipo de elementos electrónicos, las principales familias y subfamilias con algunas de sus características se presentan en la siguiente tabla.

SIGLA	FAMILIA O SUBFAMILIAS
TTL	Lógica de transistor transistor
CMOS	MOSFET complementario
ECL	Lógica de emisor acoplado
DTL	Lógica de diodo transistor

La tecnología digital se puede manifestar en los siguientes campos:

- Mecánica
- Electromecánico
- Neumático
- Hidráulico
- Electrónico

Los circuitos digitales representan el "hardware" de las computadoras, pero las funciones lógicas también son posibles de realizar por la programación de las computadoras mediante el "Software".

Los circuitos digitales según su función pertenecen a dos grandes clases:

- Circuitos Digitales combinacionales
- Circuitos Digitales secuenciales

Circuito Combinacional

Un circuito combinacional, como su nombre lo sugiere es un circuito cuya salida depende solamente de la combinación de sus entradas en el momento que se está realizando la medida en la salida.

Analizando el circuito, con compuertas digitales, que se muestra a continuación en la figura 6.13, se puede ver que la salida de cada una de las compuertas que se muestra depende únicamente de sus entradas.

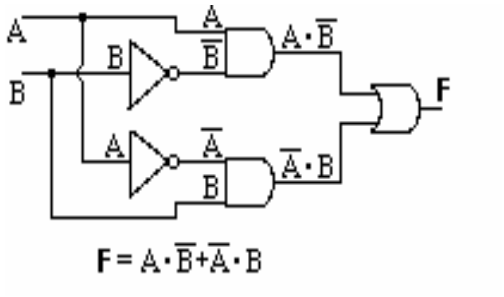


Figura 4.14 Circuito combinacional[57]

La salida F variará si alguna de las entradas A o B o las dos a la vez cambian.

Los circuitos de lógica combinacional son hechos a partir de las compuertas básicas AND, OR, NOT. También pueden ser construidos con compuertas NAND, NOR, XOR, que son una combinación de las tres compuertas básicas.

[57]Véase en www.ilustrados.com/circuitos_secuenciales/secuenciales.pdf

Tabla de verdad:

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Circuito Secuencial

La diferencia principal entre un circuito combinacional y un circuito secuencial es de que en el segundo caso hay una realimentación de una señal de salida hacia la entrada. Ver la siguiente figura 4.15.

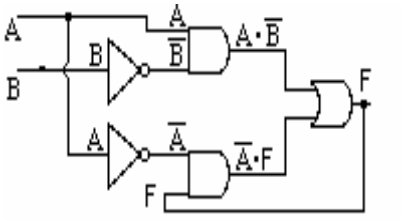


Figura 4.15 Circuito Secuencial[58]

Se puede ver que la salida de la compuerta OR es realimentada y se utiliza como entrada de la compuerta AND inferior. Esto significa que la salida (F) de este circuito digital dependerá de las entradas (A y B), pero también dependerá de la salida F (la salida que se realimenta) que se haya dado, un instante antes.

En otras palabras, la salida F depende de las entradas A y B y del valor, que tenía esta salida, previamente.

[58] Véase en www.ilustrados.com/circuitos_secuenciales/secuenciales.pdf

4.5.1 Multiplexor

Un multiplexor o conmutador permite el paso selectivo de una o varias señales hacia una salida. El multiplexor más sencillo es aquel que permite el paso selectivo de dos señales hacia una salida y su diagrama se muestra en la figura 4.16.

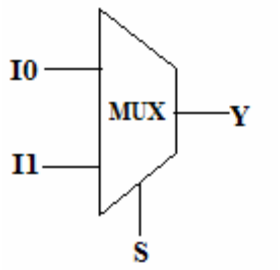


Figura 4.16 Multiplexo[59]

Los multiplexores se diseñan describiendo su comportamiento mediante la declaración **with-select-when** o ecuaciones booleanas.

En la figura 4.17a) se observa que el multiplexor dual tiene como entrada de datos las variables a, b, c y d, cada una de ellas representadas por dos bits (a1, a0), (b1, b0) etc., las líneas de selección (s) de dos bits (s1 y s0) y la línea de salida z (z1 y z0).
En la figura 4.17b) se muestra un diagrama simplificado que resalta la representación mediante vectores de bits.

[59] Véase en www.monografias.com

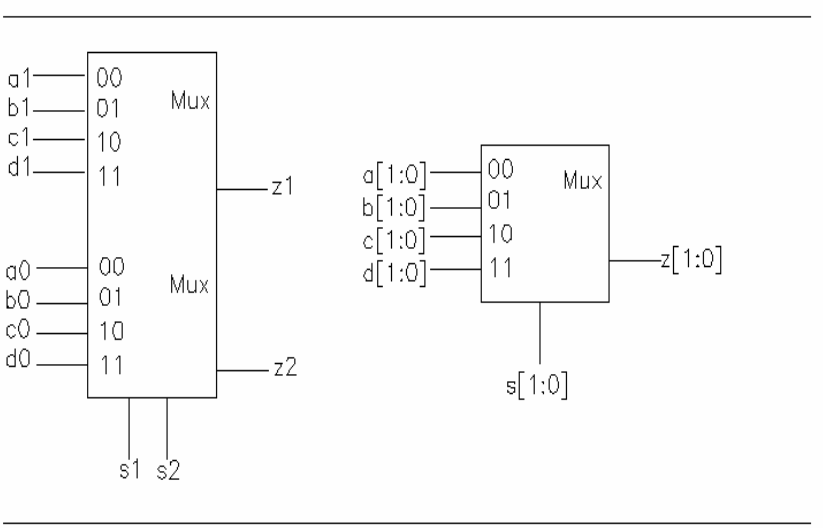


Figura 4.17 a)Multiplexor de 4 bits b)Multiplexor com vectores[60]

En el siguiente listado se muestra la descripción mediante **with-select-when** del multiplexor dual 2 x 4. En este caso la señal *s* determina cuál de las cuatro señales se asigna a la salida *z*. Los valores de *s* están dados como "00", " 01" y "10"; el término **other** (otros) especifica cualquier combinación adicional que pudiera presentarse (que incluy el "11"), ya que esta variable se encuentra definida dentro del tipo std_logic_vector, el cual contiene nueve valores.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity mux is port(
4   a,b,c,d: in std_logic_vector(1 downto 0);
5           s: in std_logic_vector(1 downto 0);
6           z: out std_logic_vector(1 downto 0);
7   end mux;
8   architecture arqmux4 of mux is
9   begin
10   with s select
```

[60]Véase en www.ilustrados.com

```
11  z <= a when “00”,
12        b when “01”,
13        c when “10”,
14        d when others,
15  end arqmux4;
```

Tipos lógicos estándares

Las funciones estándares definidos en el lenguaje VHDL se crearon para evitar que cada distribuidor de software introdujera sus paquetes y tipos de datos al lenguaje. Por esta razón el Instituto de Ingenieros Eléctricos y Electrónicos, IEEE, estableció desde 1987 los estándares std_logic y std_logic_vector.

En cada uno de los estándares se definen ciertos tipos de datos conocidos como tipos lógicos estándares, los cuales se pueden utilizar haciendo referencia al paquete que los contiene (en este caso std_logic_1164).

Descripción de multiplexores mediante ecuaciones booleanas

En el siguiente listado se muestra una solución con ecuaciones booleanas para el multiplexor de la figura 4.16.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity mux is port (
4        a,b,c,d: in std_logic_vector (1 downto 0);
5              s: in std_logic_vector (1 downto 0);
6              z: out std_logic_vector (1 downto 0);
7  end mux;
8  architecture arqmux of mux is
9  begin
```

```
10  z(1) <= (a(1) and not (s(1)) and not (s(0))) or
11       <= (b(1) and not (s(1)) and s(0)) or
12       <= (c(1) and s(1) and not (s(0))) or
13       <= (d(1) and  s(1) and s(0));
14  z(0) <= (a(1) and not (s(1)) and not (s(0))) or
15       <= (b(1) and not (s(1)) and s(0)) or
16       <= (c(1) and not (s(1)) and not (s(0))) or
17       <= (d(1) and s(1) and s(0));
18  end arqmux;
```

4.5.2 Comparadores

Un comparador realiza la prueba entre dos palabras binarias y determina si el resultado es mayor, menor o igual. Al no ser conmutativa la comparación, se debe especificar el orden de la comparación. En la siguiente figura 4.18 se muestra el diagrama de bloques de un comparador de magnitud.

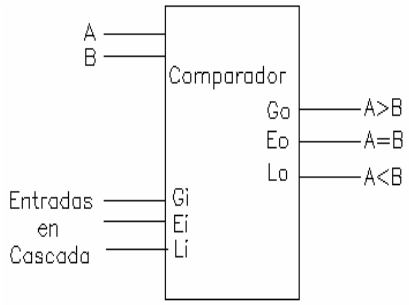


Figura 4.18 Diagrama de bloques de un comparador[61]

[61] Véase en www.monografias.com/trabajos18/comparadores.html

a) Medio comparador

Se analiza primero el medio comparador de 1 bit, es decir, el comparador de palabras de 1 bit que no tiene entradas de comparación en cascada. Este medio comparador se muestra en la figura 4.19

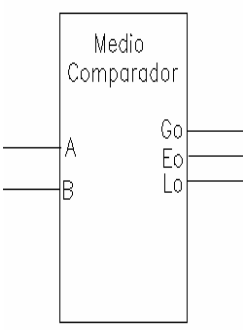


Figura 4.19 Medio comparador[62]

La relación lógica en el medio comparador aparece en la siguiente tabla.

A	B	Go	Eo	Lo
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Las ecuaciones lógicas que rigen el medio comparador son las siguientes.

$$Go = AB'$$

$$Eo = (A\ B)$$

$$Lo = A'B$$

Un medio comparador es de poca utilidad práctica, dado que está muy limitado, sin embargo sirve para definir comparadores más grandes.

[62] Véase en www.monografias.com/trabajos18/comparadores.html

b) Comparador completo

El comparador completo de 1 bit es como el que aparece en la figura 4.18, tomando A y B como palabras de 1 bit. En la siguiente tabla se muestra las relaciones lógicas del comparador completo de 1 bit.

Gi	Ei	Li	A	B	Go	Eo	Lo
1	0	0	x	x	1	0	0
0	0	1	x	x	0	0	1
0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	1
0	1	0	1	0	1	0	0
0	1	0	1	1	0	1	1

Las relaciones lógicas que rigen al comparador completo de 1 bit se encuentran en las siguientes ecuaciones.

$$Go = Gi + EiAB'$$

$$Eo = Ei(A\ B)'$$

$$Lo = Li + EiA'B$$

El comparador completo de 1 bit puede ser conectado en cascada para obtener comparadores de palabras más grandes.

c) Comparadores en VHDL

La descripción VHDL del medio comparador y el comparador completo de 1 bit no tienen mayor problema.

Programación en VHDL de un medio comparador de 1 bit

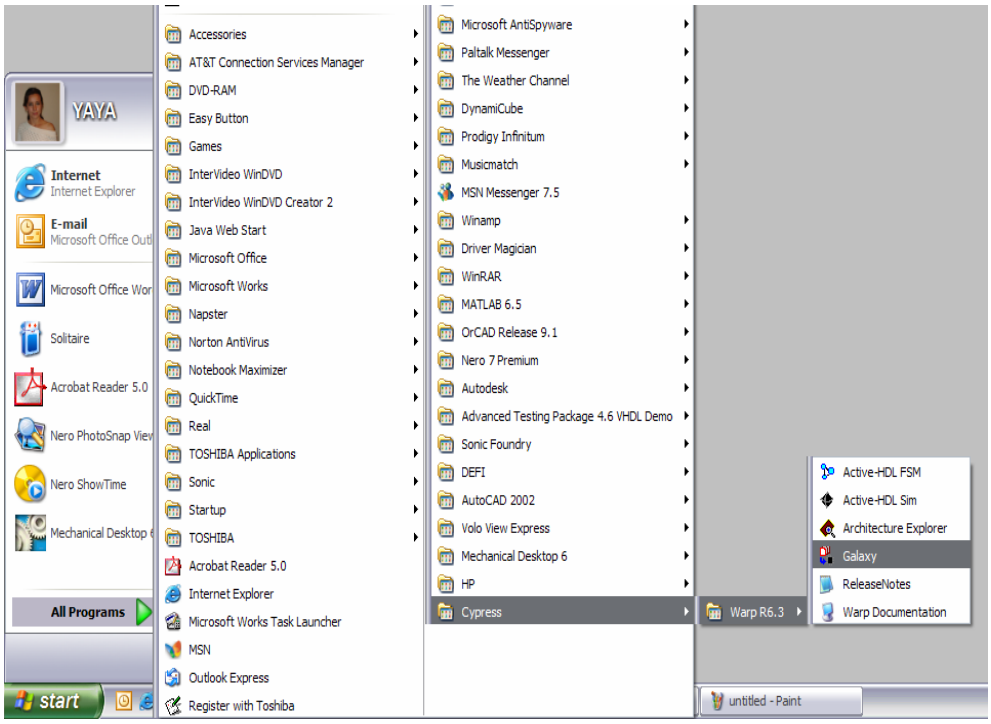
```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   entity Medio_comp_1 is
4       port (
5           A, B: in std_logic;  - - Entradas
6           Go, Eo, Lo: out std_logic  - - Salidas );
7       end Medio_comp_1;
8   architecture Simple of Medio_comp_1 is
9     begin
10        Go < = A AND NOT (B);  - - Mayor
11        Eo < = A XNOR B;          - - Igual
12        Lo < = NOT (A) AND B;   - - Menor
13  end Simple;
```

Programación en VHDL de un comparador de 1 bit

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   entity Comparador_1 is
4       port (
5           A, B: in std_logic;  - - Entradas
6           Gi, Ei, Li: in std_logic; - - Cascada
7           Go, Eo, Lo: out std_logic  - - Salidas
8           );
9       end Comparador_1;
10  architecture Simple of Comparador_1 is
11      begin
12        Go < = Gi OR (Ei AND (A AND NOT (B) ) );  - - Mayor
13        Eo < = Ei AND (A XNOR B);                  - - Igual
14        Lo < = Li  OR (Ei AND (NOT (A) AND B) );   - - Menor
15  end Simple;
```

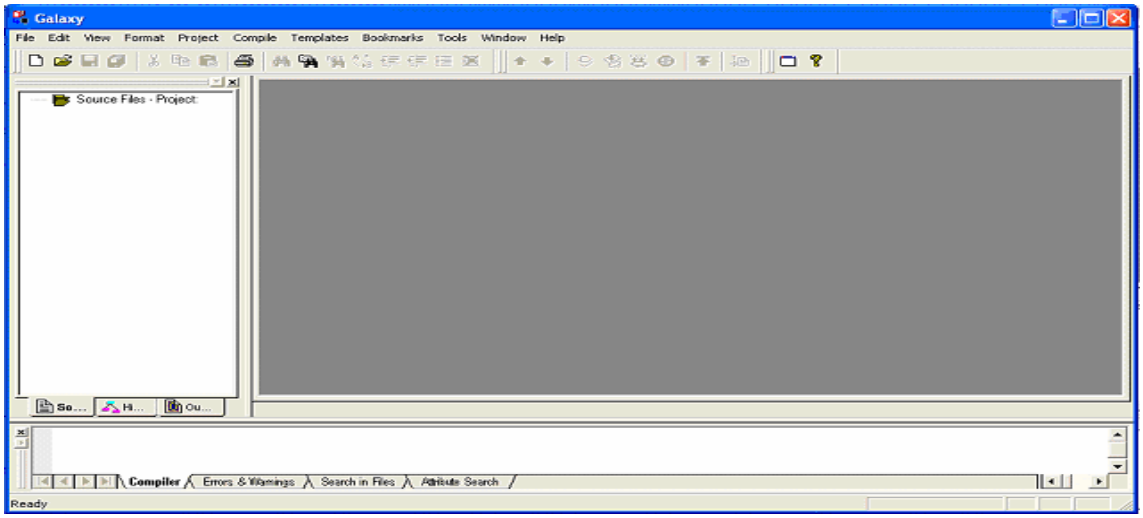
4.6 INTRODUCCIÓN AL LENGUAJE VHDL

El programa Galaxy es el núcleo de la suite WARP R6.3, ya que nos gestiona los programas creados, nos permite editarlos y elegir distintas opciones de compilación. Una vez en Windows haz click en el botón de Start, selecciona All programs > Cypress > Warp R6.3 > Galaxy.



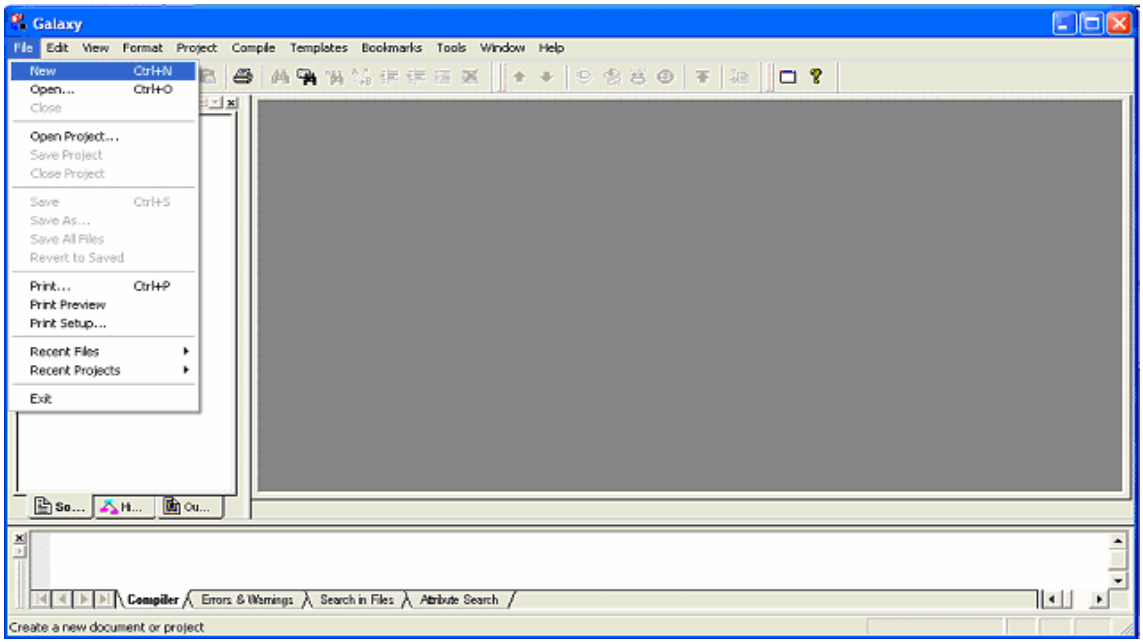
Pantalla 4.1 Localización del programa Galaxy

Su pantalla principal es la siguiente:



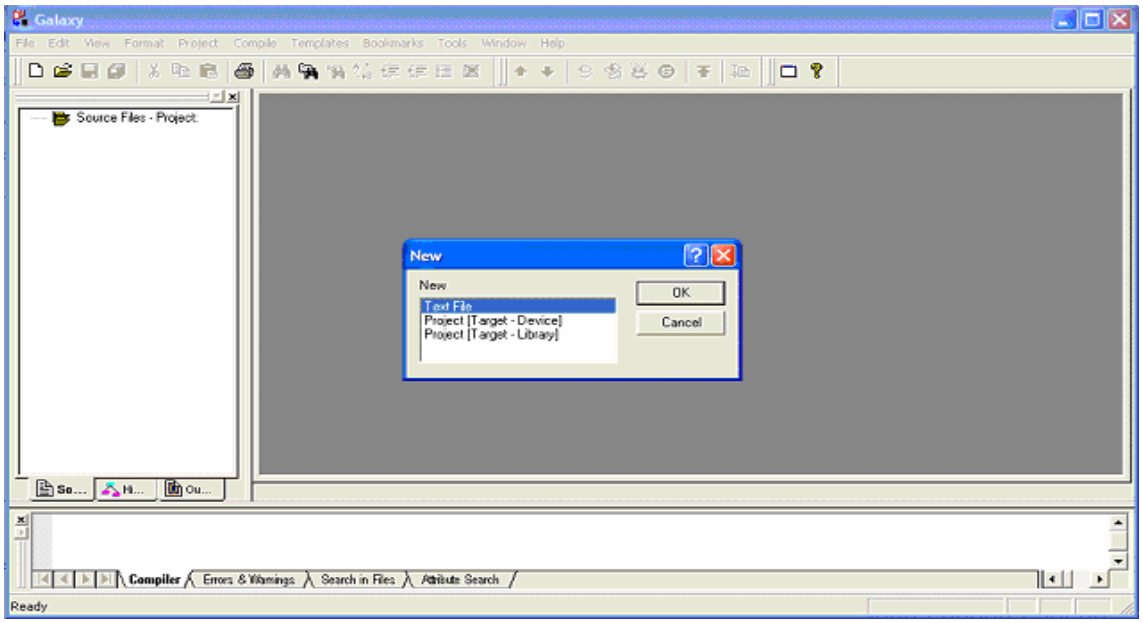
Pantalla 4.2 Pantalla principal

Con este programa se puede describir un circuito en VHDL, compilarlo y simularlo, sin más que usar el menú adecuado. Para realizar un proyecto nuevo haz click en el menú y selecciona *File > New*.



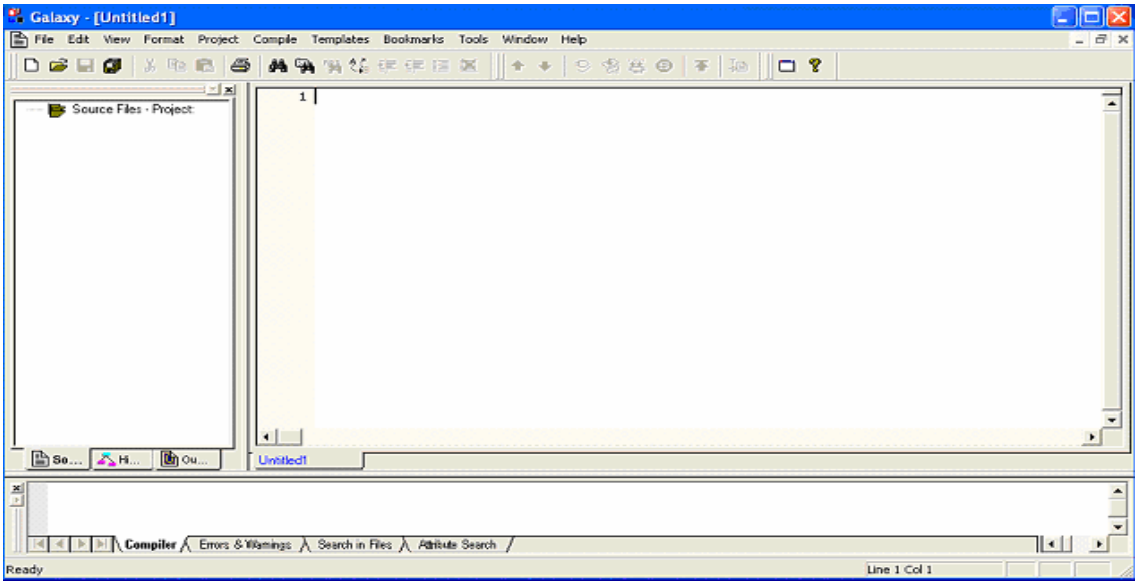
Pantalla 4.3 Hacer un nuevo proyecto

Posteriormente aparece un cuadro de dialogo con el titulo de New con tres opciones, selecciona Text File y después haz click en el botón de OK como se muestra en la siguiente imagen.



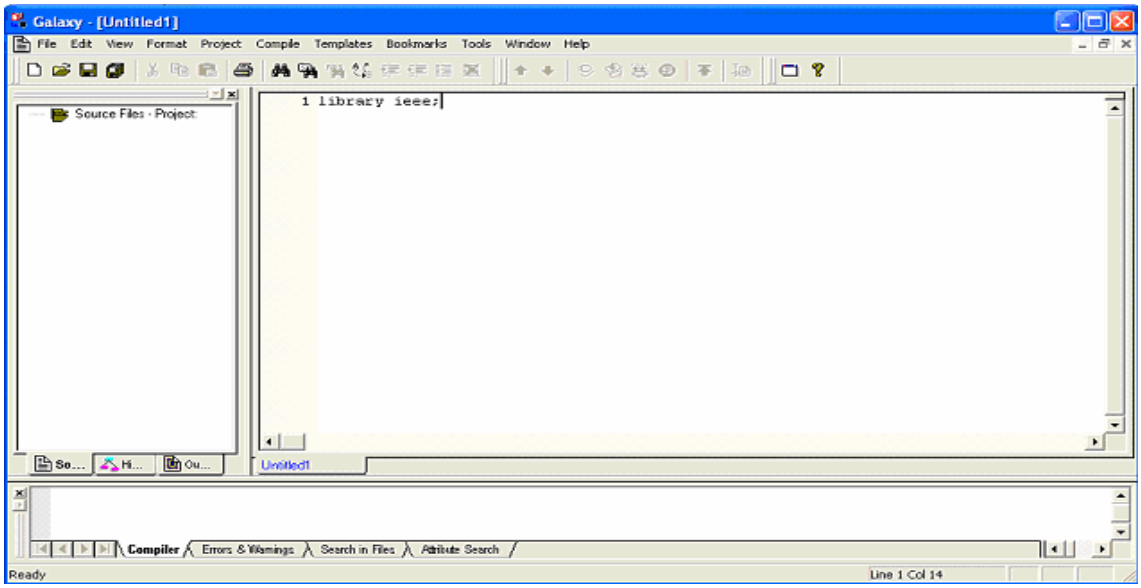
Pantalla 4.4 Elección de tipo de proyecto

Esta es la siguiente pantalla que aparecerá en esta es donde se teclea el código fuente.



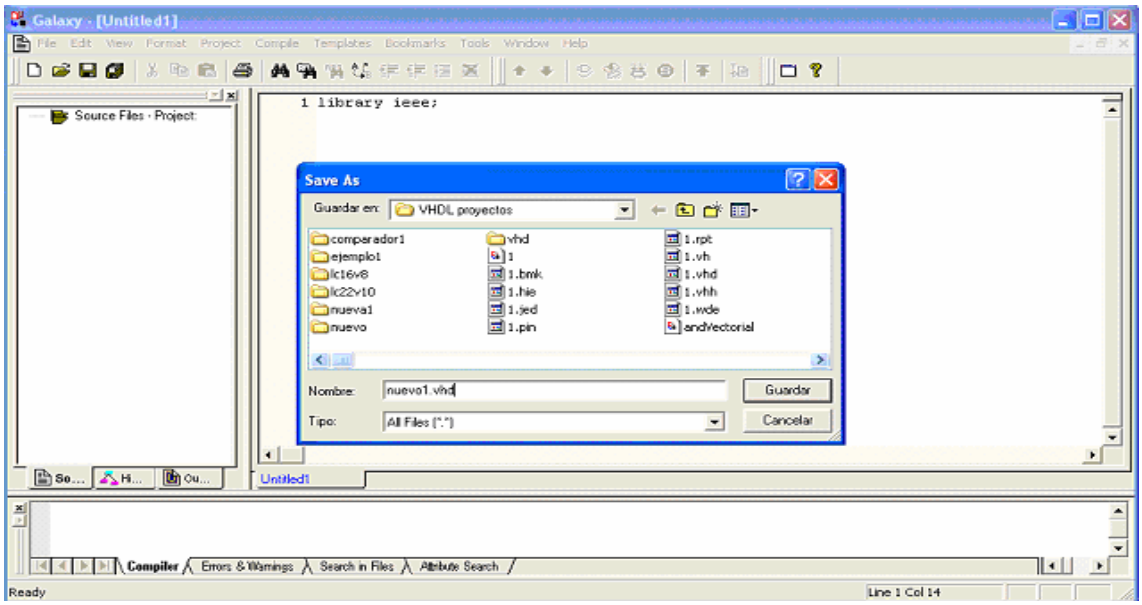
Pantalla 4.5 Iniciar el proyecto

Escribe lo siguiente dentro del archivo de texto: **library** ieee;



Pantalla 4.6 Inicio del programa

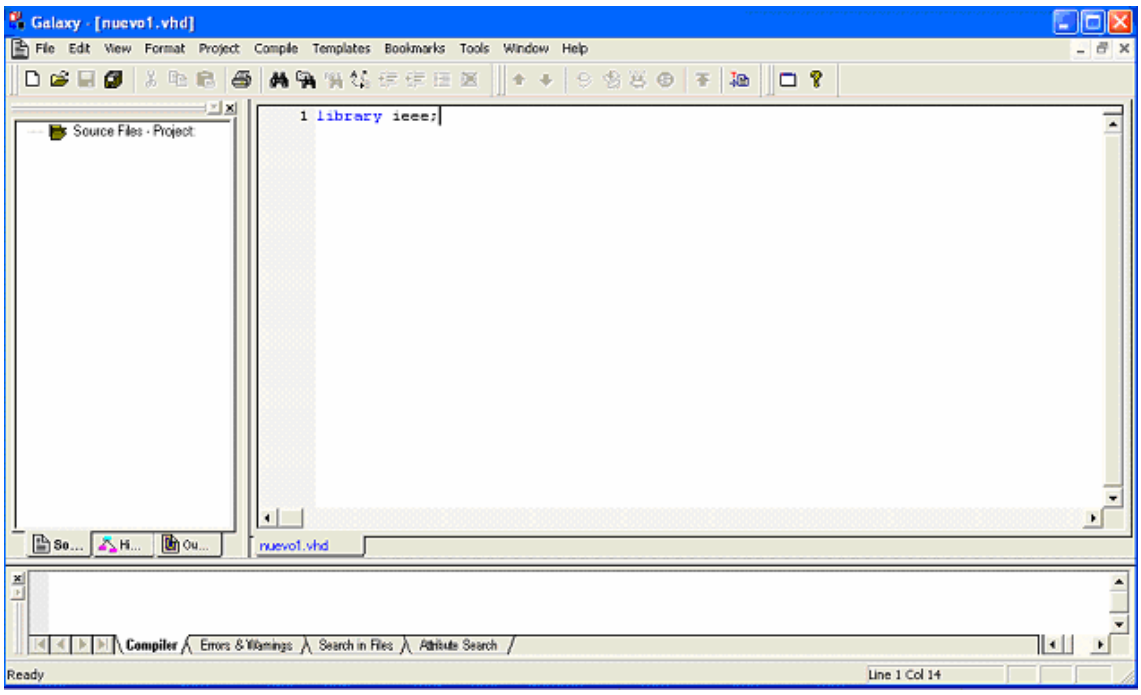
Elige File > Save As y en la opción de Guardar, selecciona el nombre de la carpeta que vas a usar en este caso fue VHDL proyectos, después en la celda de *file name* se escribe el nombre de nuestro programa en este caso es nuevo1, con la extensión vhd, ya que contiene un código en VHDL y haz click en Save.



Pantalla 4.7 Como guardar un archivo

Cierra el archivo de texto.

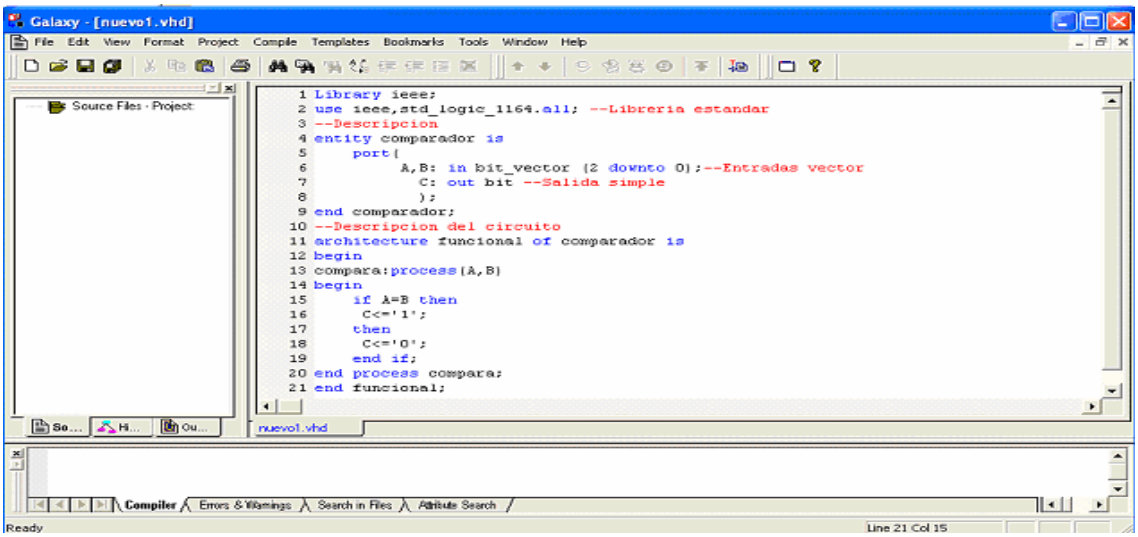
Selecciona Project > Add Files, te aparecerá una ventana que debe de tener tu archivo nuevo1.vhd, haz clic en Add y posteriormente en OK.



Pantalla 4.8 Programa con nombre y extensión

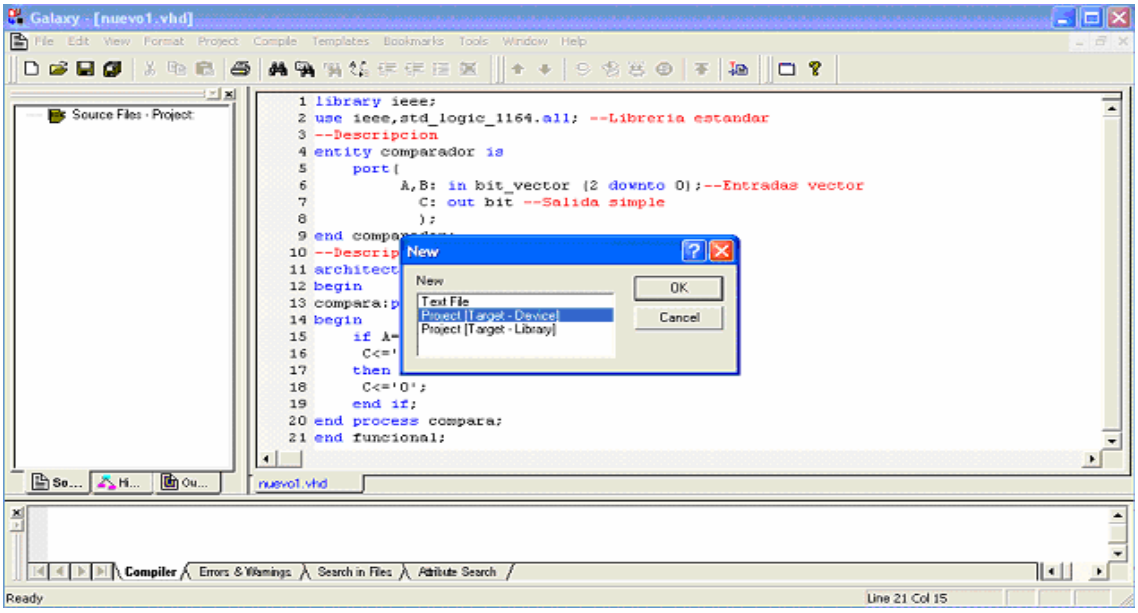
Tu archivo de texto se ha convertido en un archivo de *VHDL* y está dentro de tu proyecto, todo lo que programes, será compilado en el dispositivo que especificaste. En tu ventana de proyecto (si no se encuentra abierta, la puedes activar haciendo *View* > *Project Windows*) debe de aparecer un icono en forma de hoja que tiene el nombre de tu proyecto. Haz click en esta hoja y debe de aparecer un archivo con la instrucción: ***library ieee***; solo que ahora la palabra library está en color azul porque es una palabra reservada. Esto demuestra que haz hecho correctamente el procedimiento.

Si la configuración de colores es la normal, notarás que todas las palabras reservadas se muestran en color azul y los comentarios en color rojo. Ahora ya podemos comenzar a hacer tu descripción en VHDL.



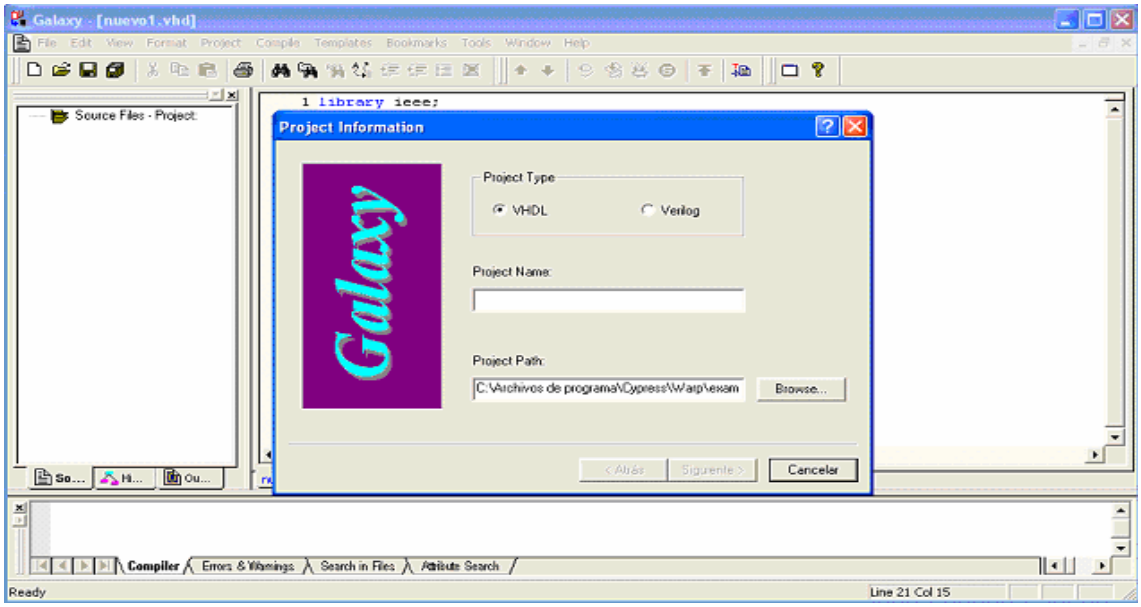
Pantalla 4.9 Teclear el código fuente del programa

Seleccionamos en el menú principal *File>New*, posteriormente debe aparecer un cuadro de dialogo con el titulo de New con tres opciones, seleccionamos Project [Target-Device] y después haz clic en el botón de OK como se muestra en la siguiente pantalla.



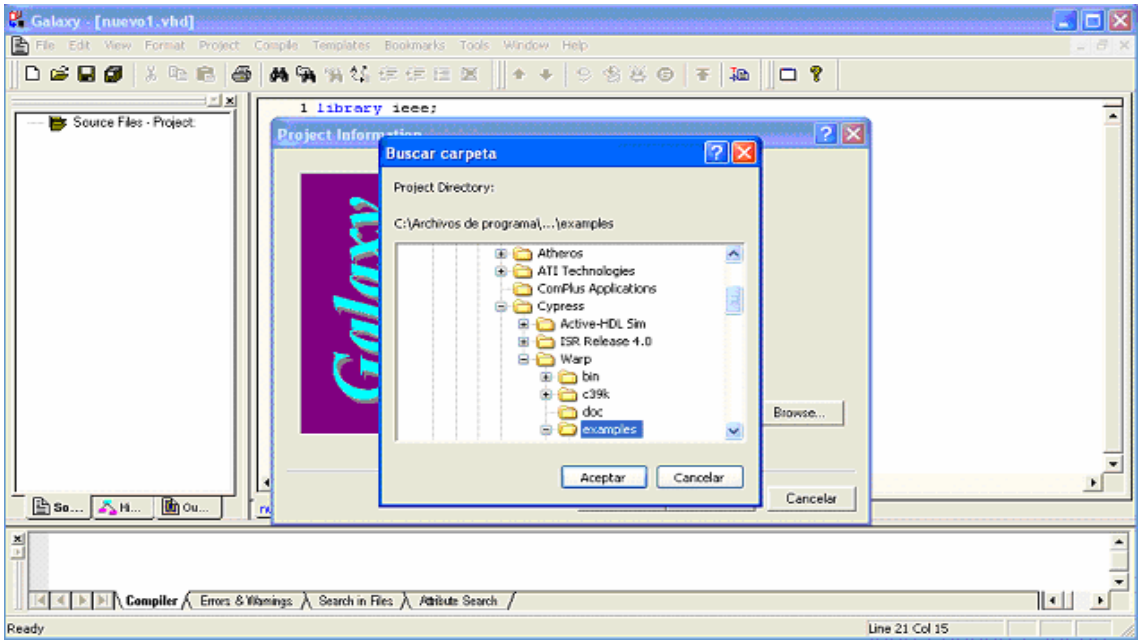
Pantalla 4.10 Crear un proyecto para dispositivo programable

Cuando selecciones la opción de *Project[Target-Device]* aparecerá una ventana con el nombre de Project Information, selecciona VHDL después en Project Name en la celda escribe el nombre que deseas poner al proyecto y en Project Path se indica cual es la ruta donde se guardan los nuestros proyecto.



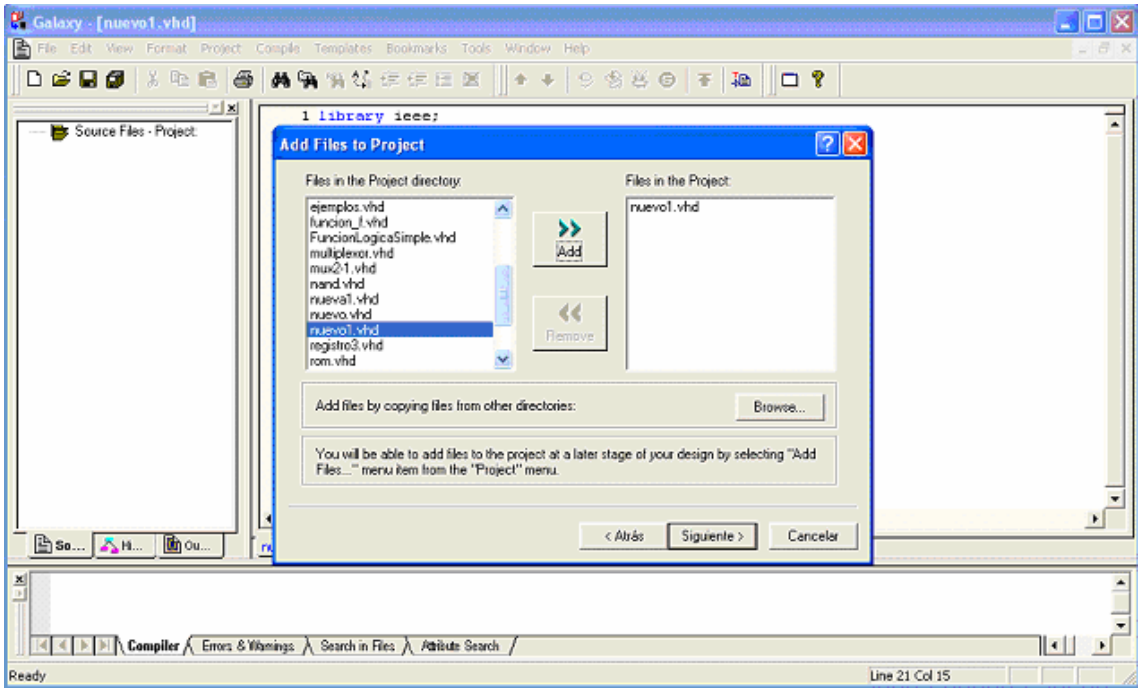
Pantalla 4.11 Información del proyecto

Si la ruta no es la deseada buscamos otra se da clic en el botón de Browse… y aparece la siguiente pantalla y ahí seleccionamos la nueva ruta y se da clic en el botón de OK como se muestra en la siguiente pantalla.



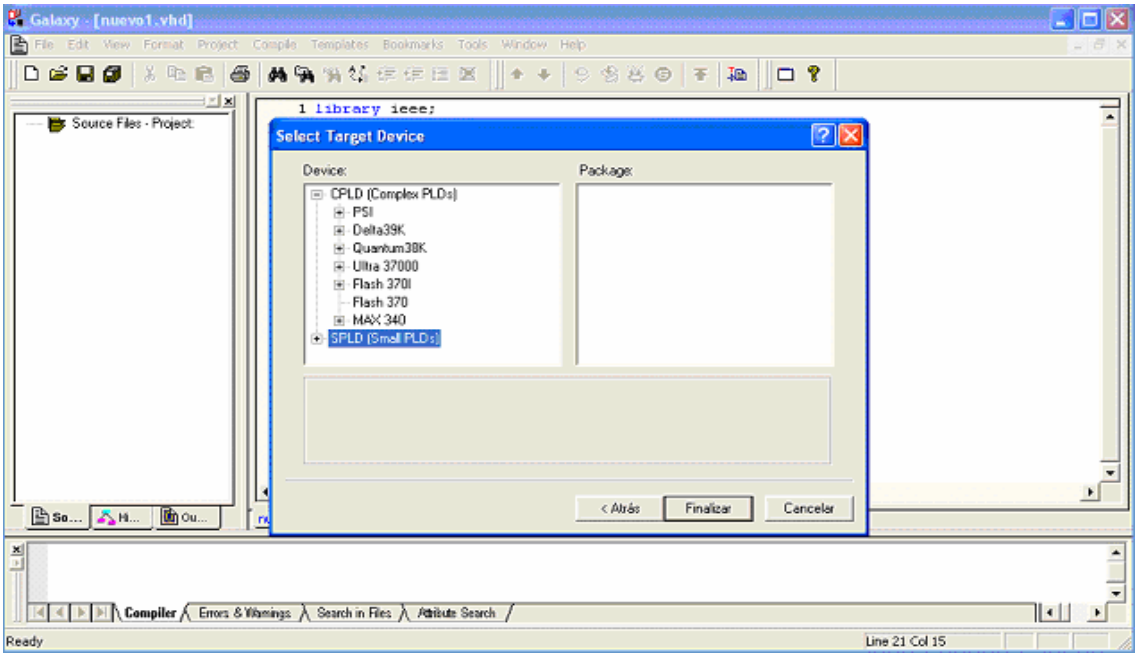
Pantalla 4.12 Buscar carpeta

Cuando hayas terminado aparece una ventan con título *Add Files to Project,* esta ventana te pedirá que archivo con extensión VHD se va usar. En este caso vamos a seleccionar el archivo *nuevo 1*, después haz clic en el botón de *Add*, finalmente se da clic en el botón de Next.



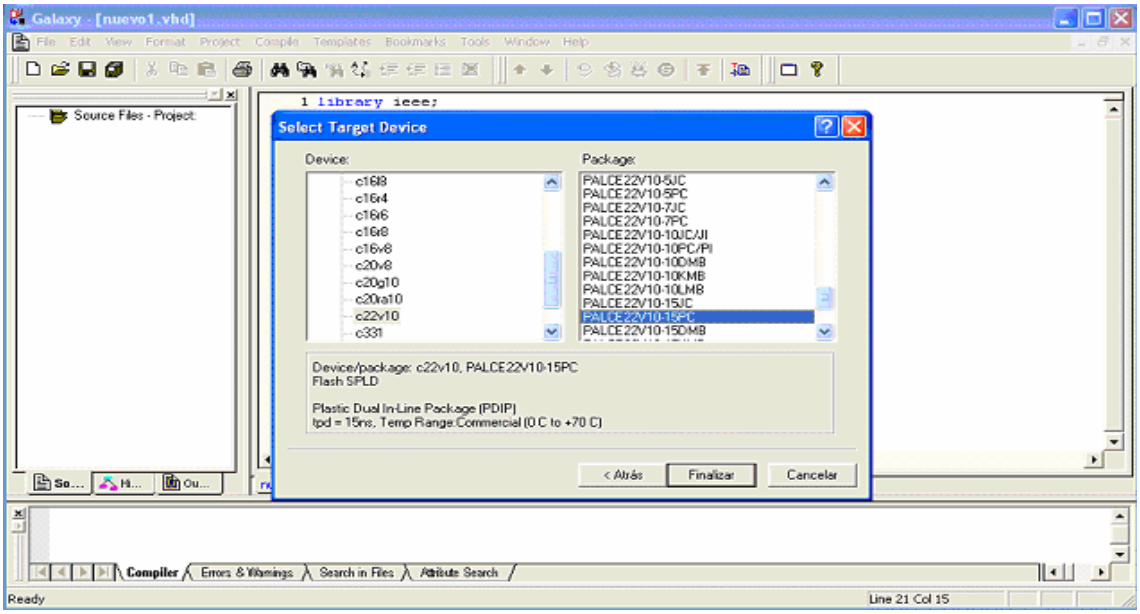
Pantalla 4.13 Agregar proyectos

La siguiente ventana lleva el título de Select Target Device, esta ventana es muy importante ya que es aquí donde debes especificar el PLD en el que vas a trabajar. Los PLD's que se usan cuando se está aprendiendo son los pequeños (generalmente 16v8, 22v8 ó 22v10), estos PLD's están en la ventana como SPLD. Haz doble click sobre este texto, te aparecerá una lista de los SPLD's más comunes, selecciona el SPLD que te interese y nuevamente aparecerá una lista donde hay varios tipos del mismo SPLD, por ejemplo PALCE16V8-10PC/PI. Cuando escojas el dispositivo, estos están en grupos de grandes familias.



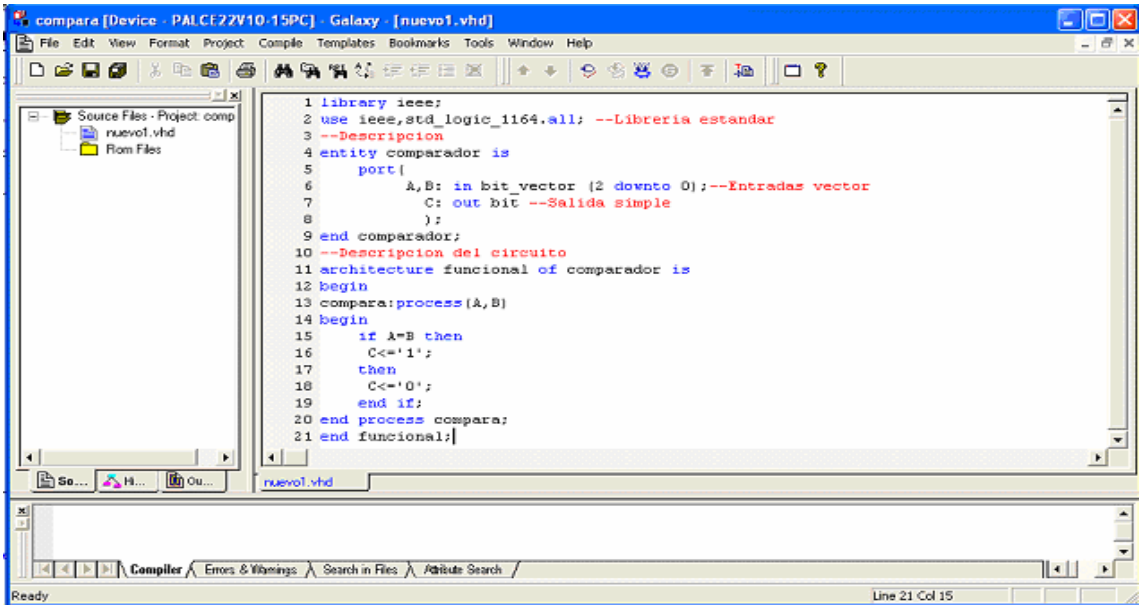
Pantalla 4.14 Selección de tipo familia de PLD's

Para este caso se seleccionará la familia de PLD's, escogiendo como dispositivo *c22v10, PALC22V10-15PC* la cual es compatible con una GAL22V10. Finalmente haz click en *Finish.*



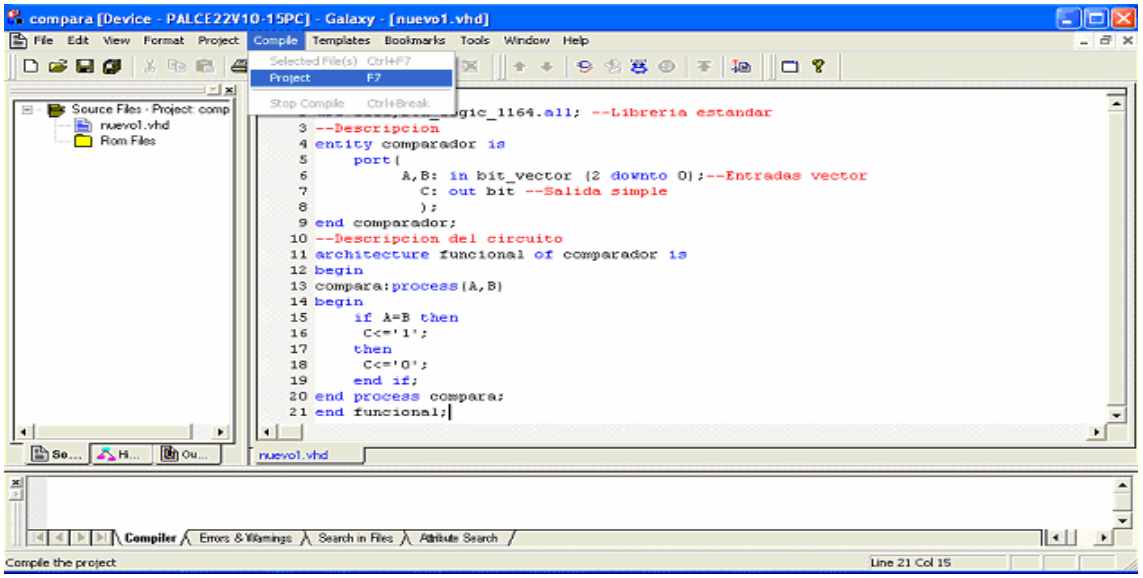
Pantalla 4.15 Seleccionar el dispositivo

Siguiendo correctamente los pasos anteriores se tendrá el programa ya cargado en el nuevo proyecto.



Pantalla 4.16 Programa cargado

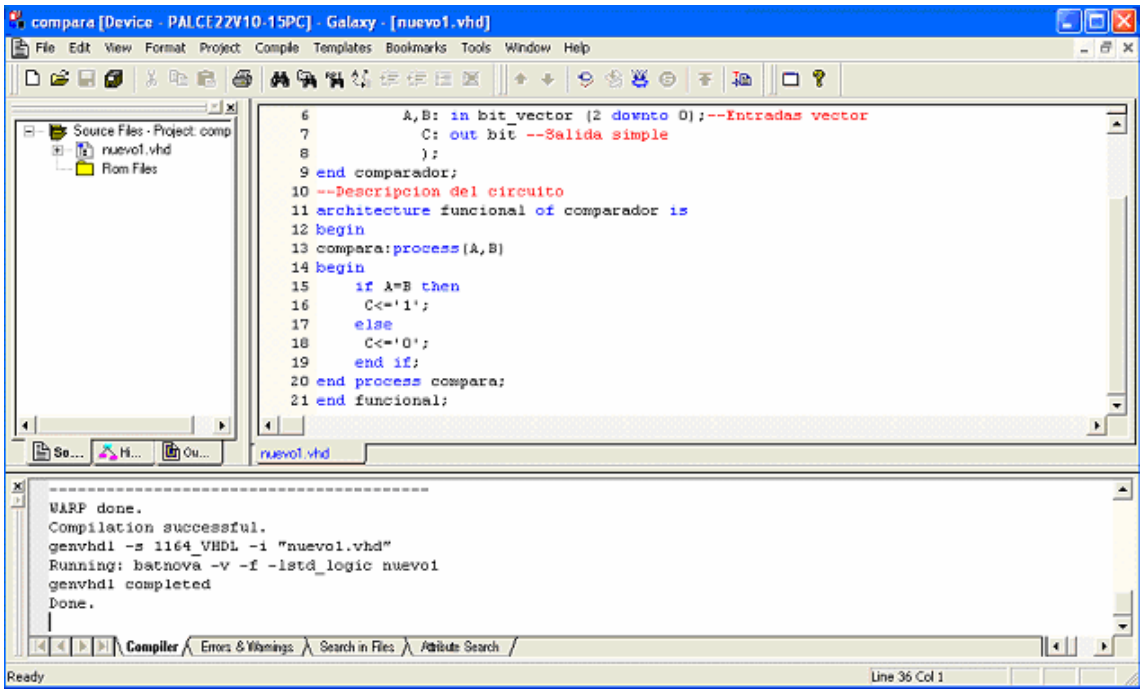
Para compilar el nuevo proyecto y así generar los archivos necesarios para programar el dispositivo, se selecciona "*Compile*" y después "*Project*".



Pantalla 4.17 Compilar el proyecto

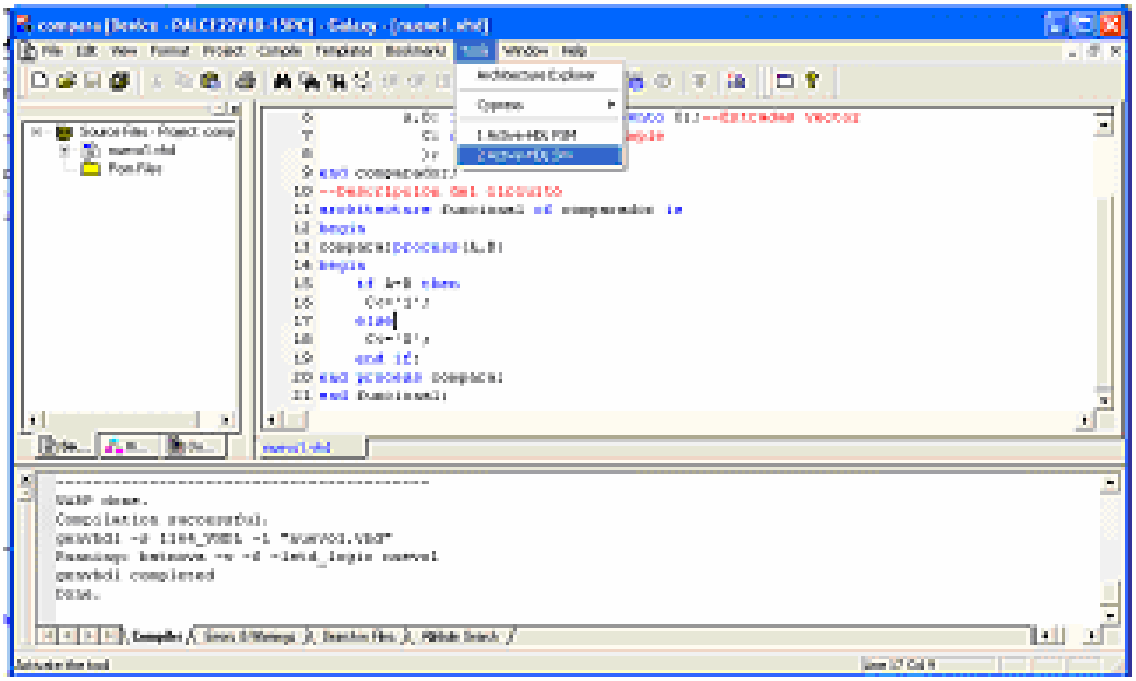
De no estar bien realizado el programa se generaran errores los cuales se mostraran en esta ventana.

En la parte inferior de la pantalla se puede observar que el programa fue compilado exitosamente.



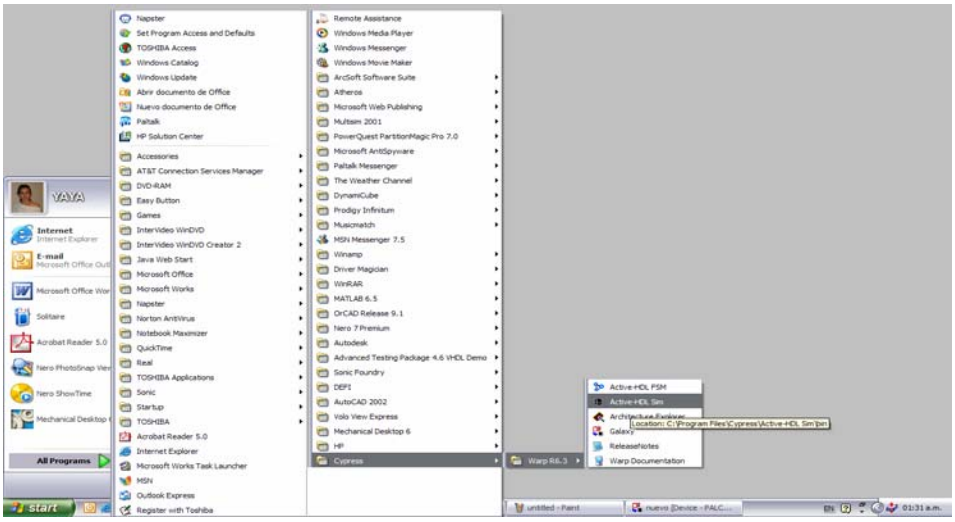
Pantalla 4.18 Detección de errores

Después se activa el Active-HDL Sim se puede acceder de 2 formas, la primera es en Tool > Active-HDL Sim como se muestra en la siguiente pantalla.



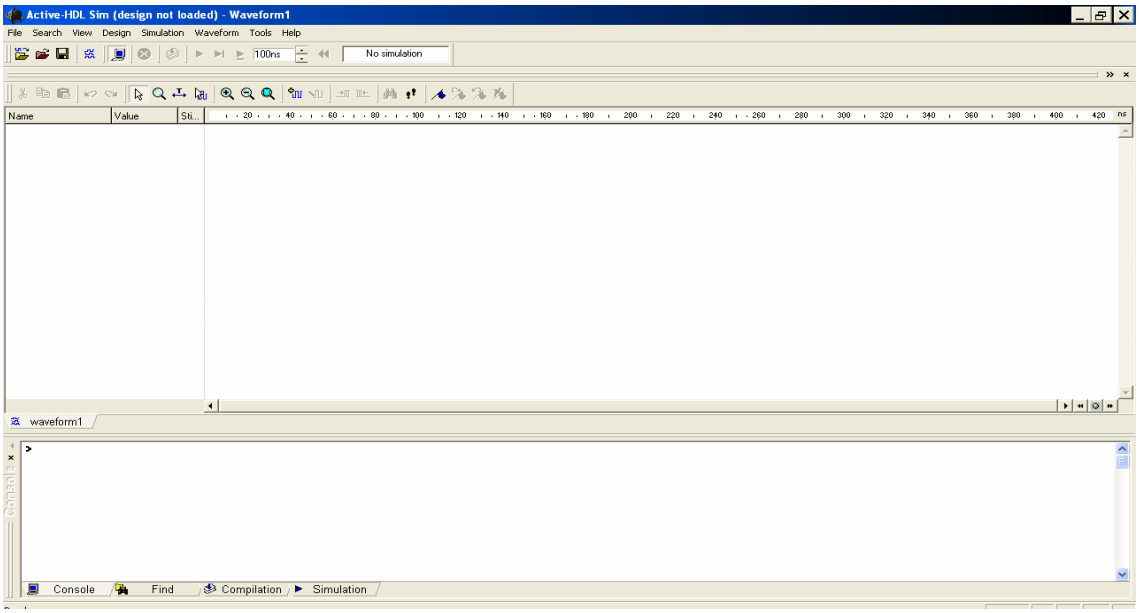
Pantalla 4.19 Activación de Active-HDL Sim

La otra forma es de la siguiente forma abrirá el programa Active – HDL Sim que normalmente se encuentra en Menú Stara > All Programs > Cypress > Warp R6.3 > Active-HDL Sim.



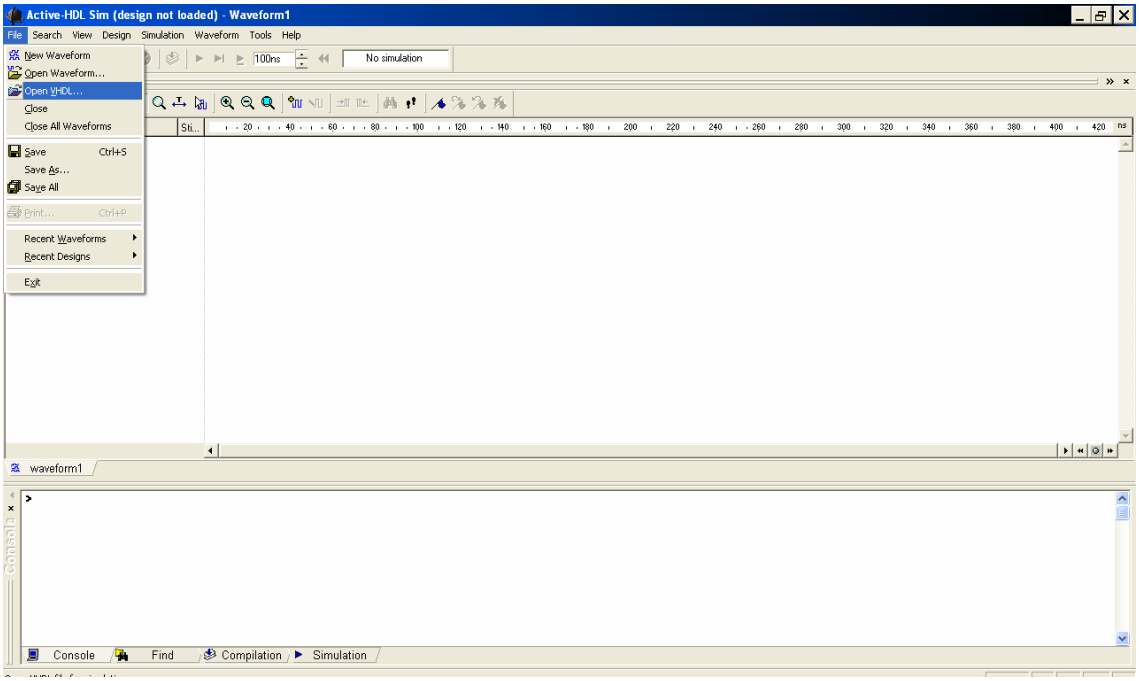
Pantalla 4.20 Segunda forma de acceder a Active-HDL Sim

Después aparece la siguiente pantalla.



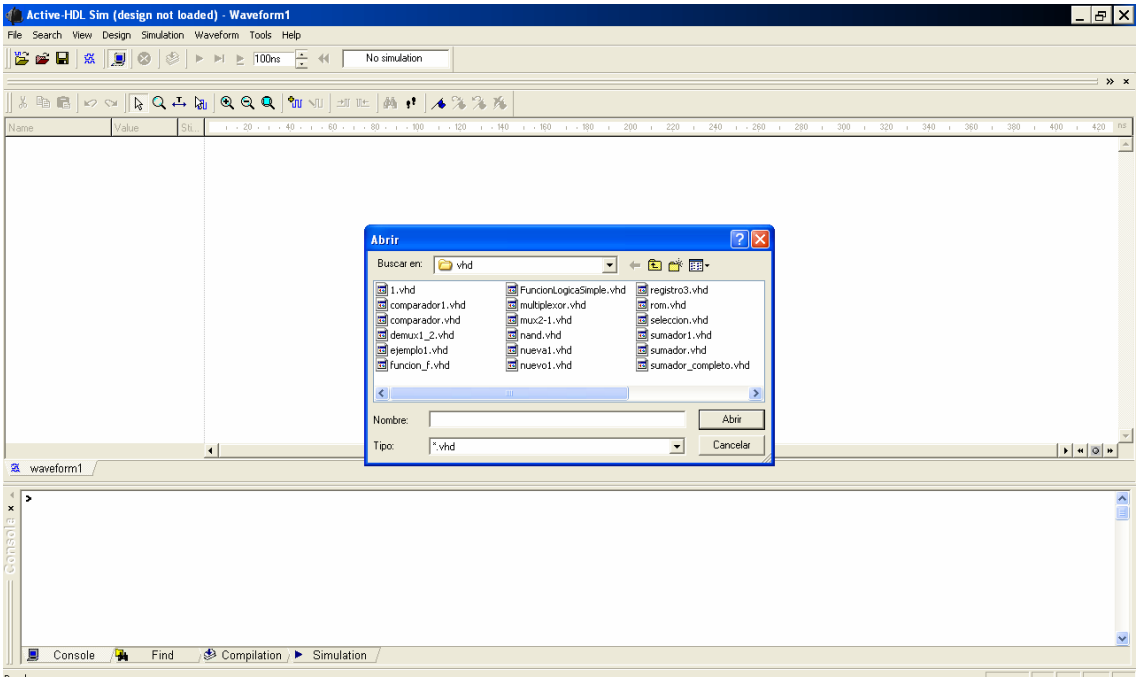
Pantalla 4.21 Pagina principal de Active-HDL Sim

Para cargar el archivo selecciona Open VHDL dentro del Menú File.



Pantalla 4.22 Cargando archivos

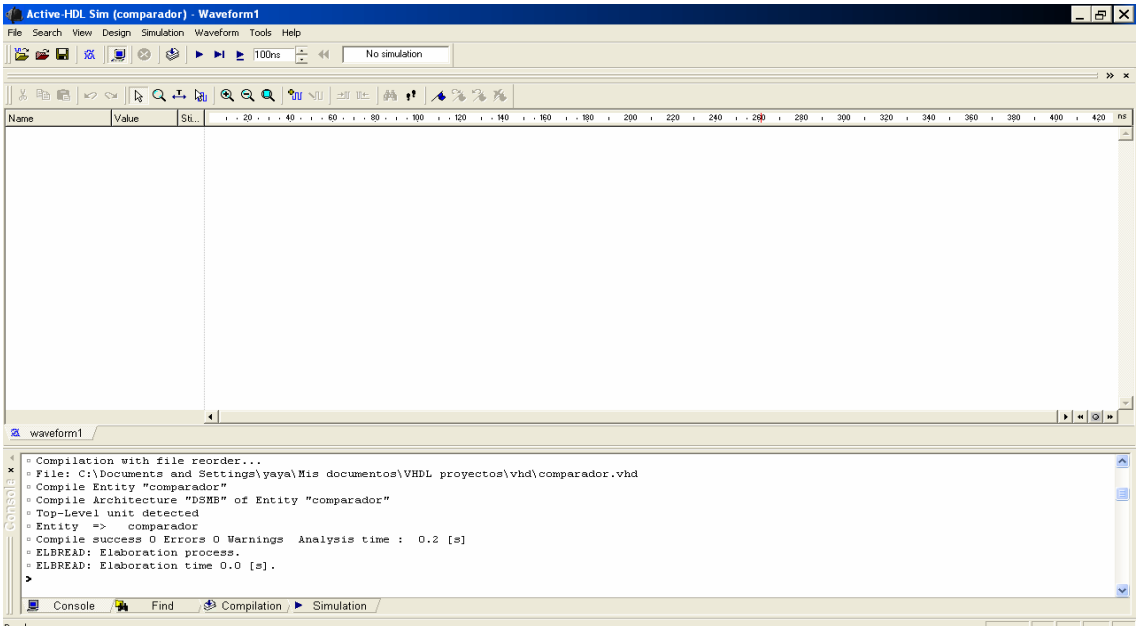
Después aparece la pantalla de Abrir y se elegirá el archivo de comparador.vhd y se da clic en el botón Abrir.



Pantalla 4.23 Abrir el proyecto

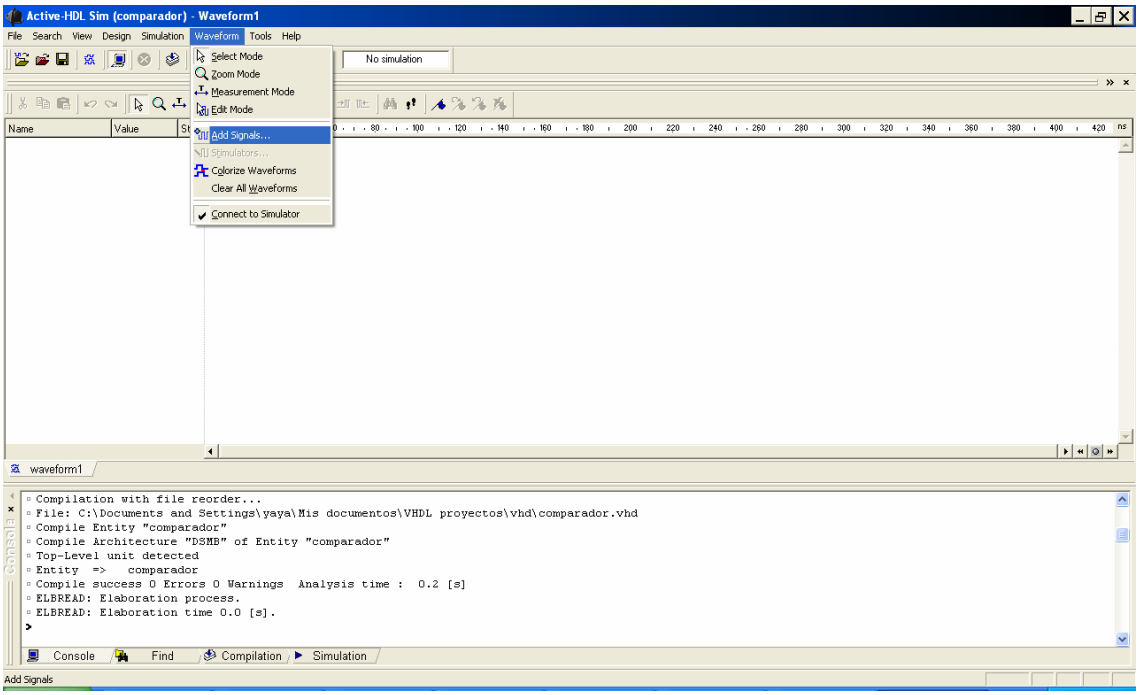
Es importante mencionar que el archivo con el formato 1164/VHDL se crea en el subdirectorio vhd dentro del directorio de trabajo del proyecto una vez que éste es compilado. Si se selecciona por accidente el archivo.vhd creado por el usuario, el compilador del programa generará varios errores y no podrá ser simulado. Una vez que el archivo apropiado es cargado, se desplegará una serie de mensajes dentro de la ventana de compilación. Uno de los mensajes que debería aparecer cuando el archivo es compilado correctamente es el siguiente:

--Compile success 0 Errors 0 Warnings Análisis time: 0.1 [s].



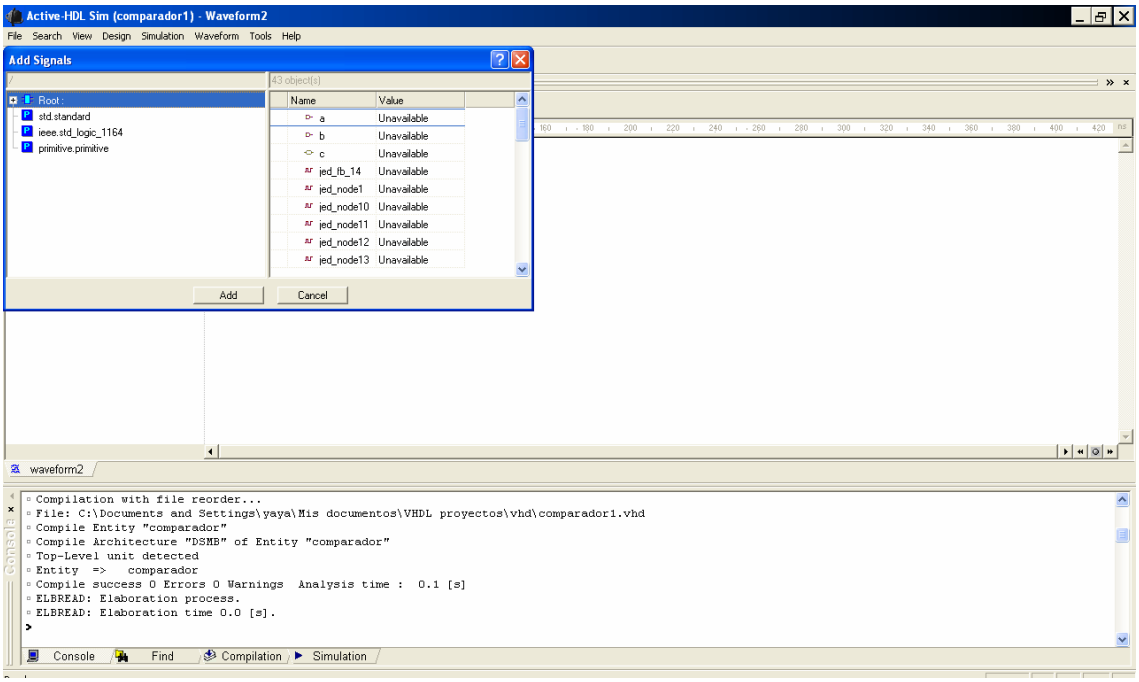
Pantalla 4.24 Compilación correcta

Después se selecciona en Waveform, Add Signals. Para elegir tanto las señales de entrada como las de salida.



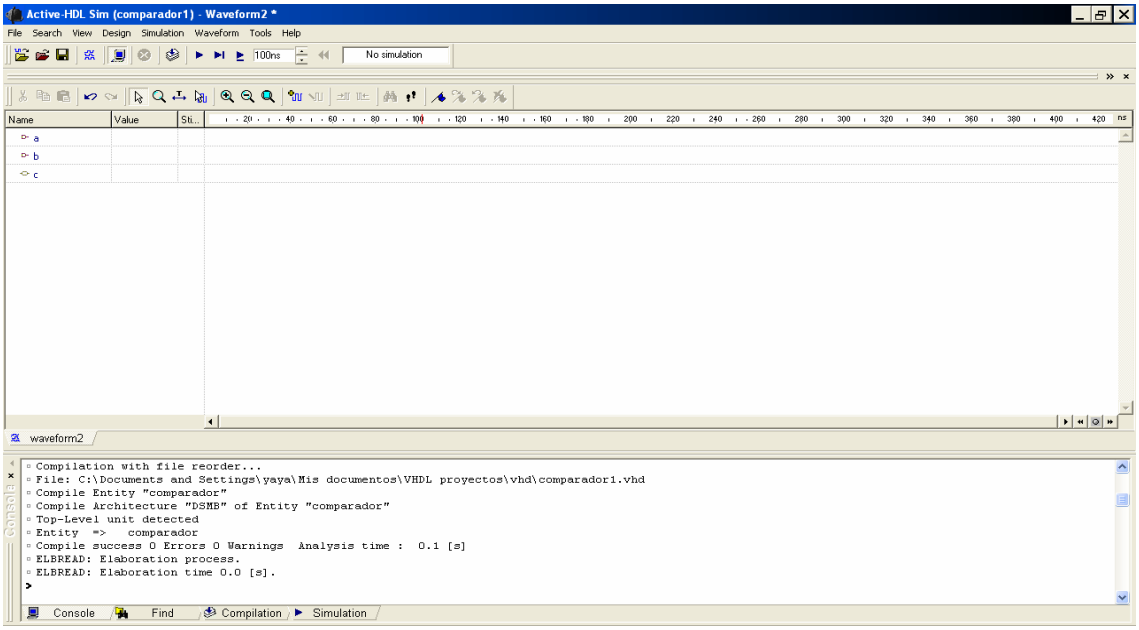
Pantalla 4.25 Activación de Waveform

Se abre la ventana para agregar señales aparecerá con una lista de todas las señales de entrada, salida, entrada/salida y nodos internos de conexión disponibles en el diseño. Para agregar algunas de las señales que se encuentran en esta lista basta con hacer doble clic sobre el nombre de la señal. Si deseas agregar varias señales al mismo tiempo, puedes seleccionarlas mediante la tecla control y haciendo clic sobre cada señal que deseas agregar para después hacer clic sobre el botón *Add* que se encuentra en la parte inferior de la ventana.



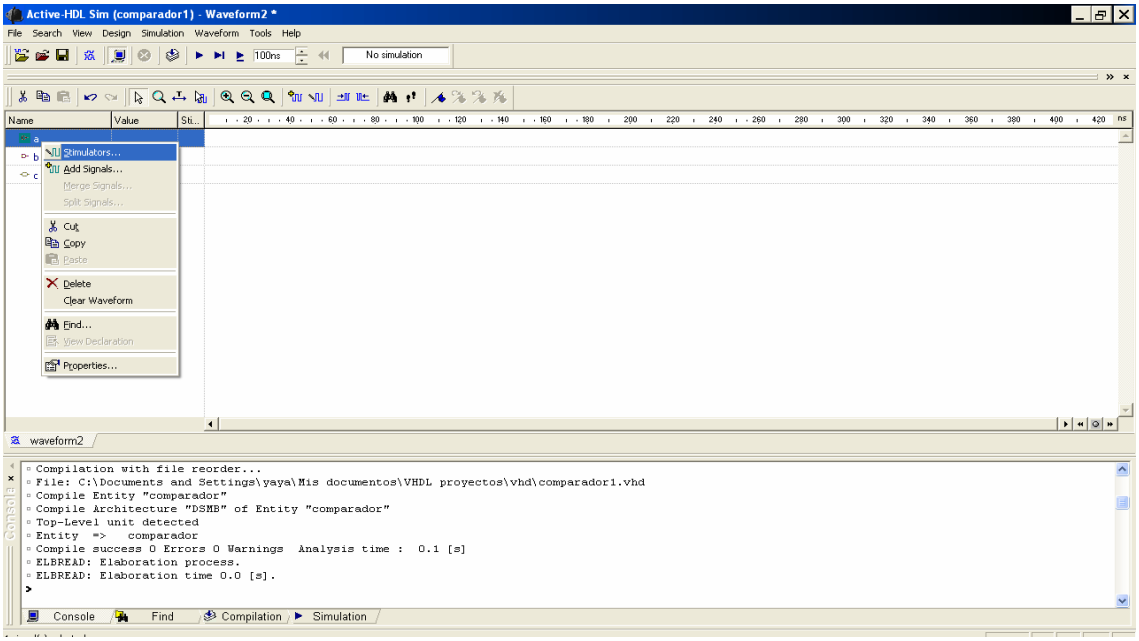
Pantalla 4.26 Agregar una señal

Después aparece la siguiente ventana con las diferentes señales.



Pantalla 4.27 Diferentes señales

Debido a que las señales están en forma de vectores de 3 bits se le pondrán valor a cada variable. Después se selecciona una de las señales se da clic derecho y se abre un menú y se selecciona Stimulators y clic.



Pantalla 4.28 Seleccionar el tipo de señal

Tipos de señales de estimulación:

Clock

Sirve para definir señales de reloj definidas con los siguientes parámetros: frecuencia, valor inicial, ciclo de trabajo y tiempo de inicio.

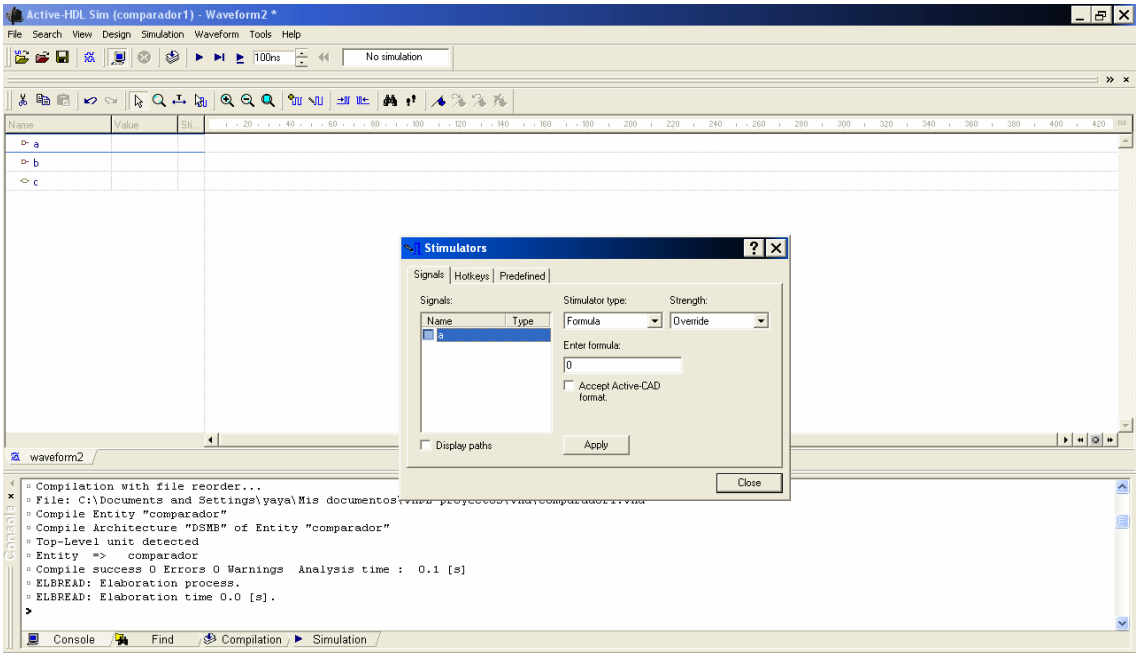
Custom

Un estimulador personalizado se crea editando los valores deseados en la ventana de simulación. Al estimulador de entrada se le puede asignar un estado bajo o un estado alto ‘1’ o ‘0’ respectivamente.

Formula

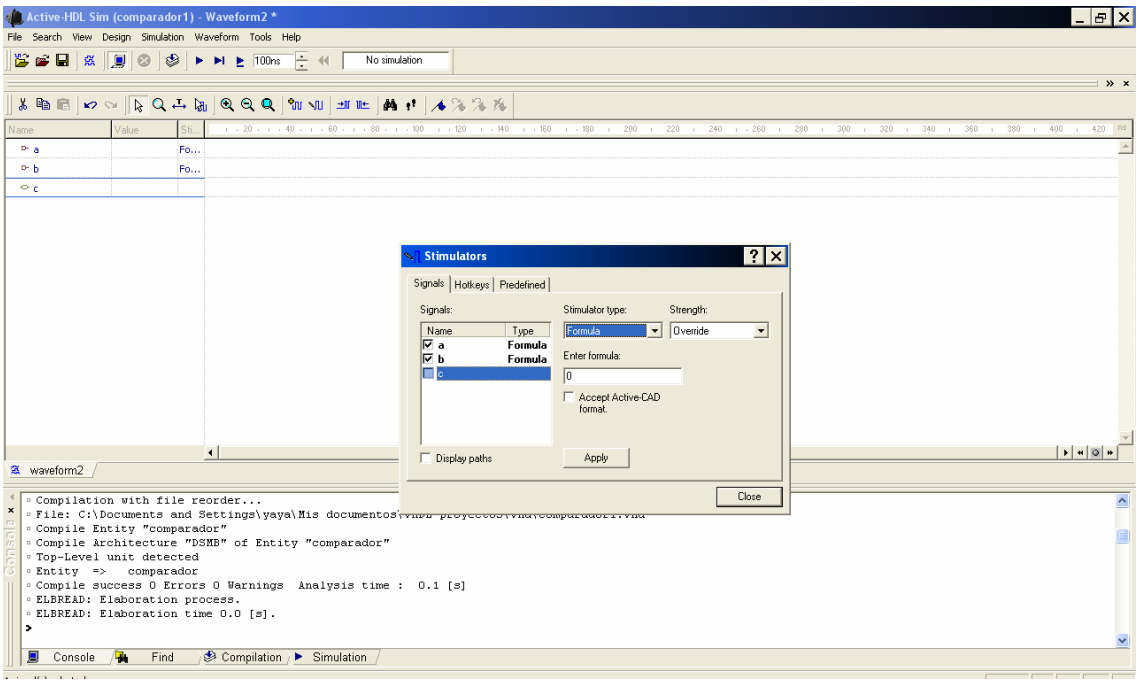
Un estimulador de tipo formula produce una señal definida por una simple sintaxis. La señal es definida como secuencias pares de valor-tiempo. Con la componente tiempo indicamos el momento en el que la señal asume el valor especificado.

Se abre la ventana de Stimulators donde se asigna valor a las señales.



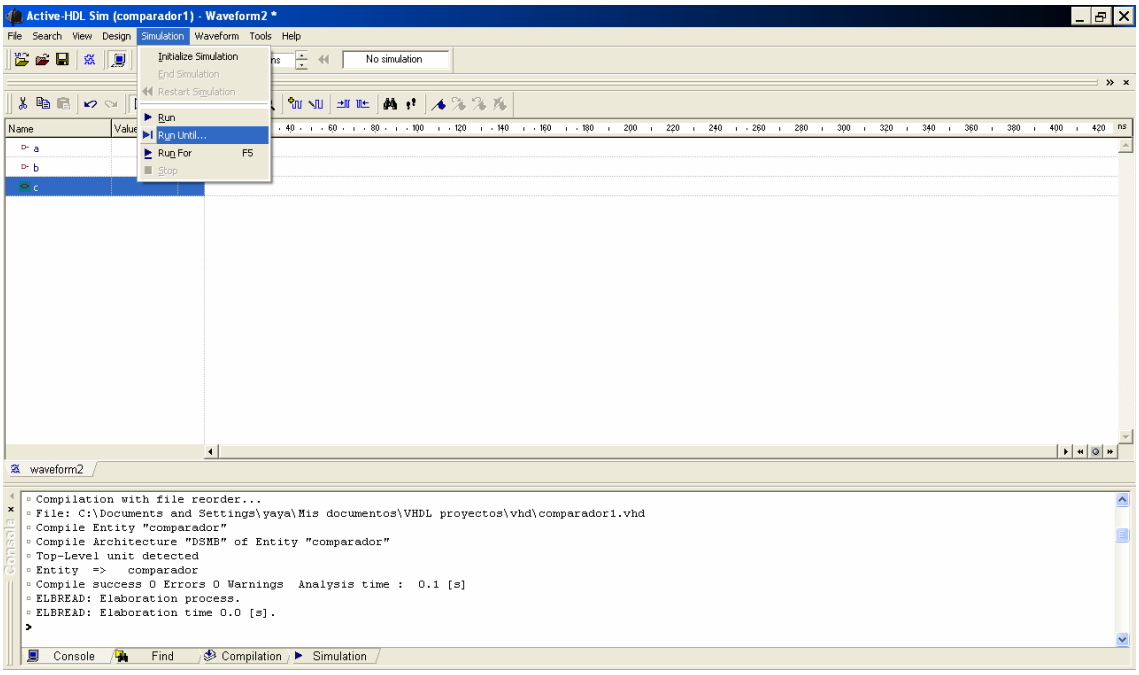
Pantalla 4.29 Asignación de valores

Después de asignar el valor a la primera variable se hace lo mismo para las siguientes variables y se da clic en close.



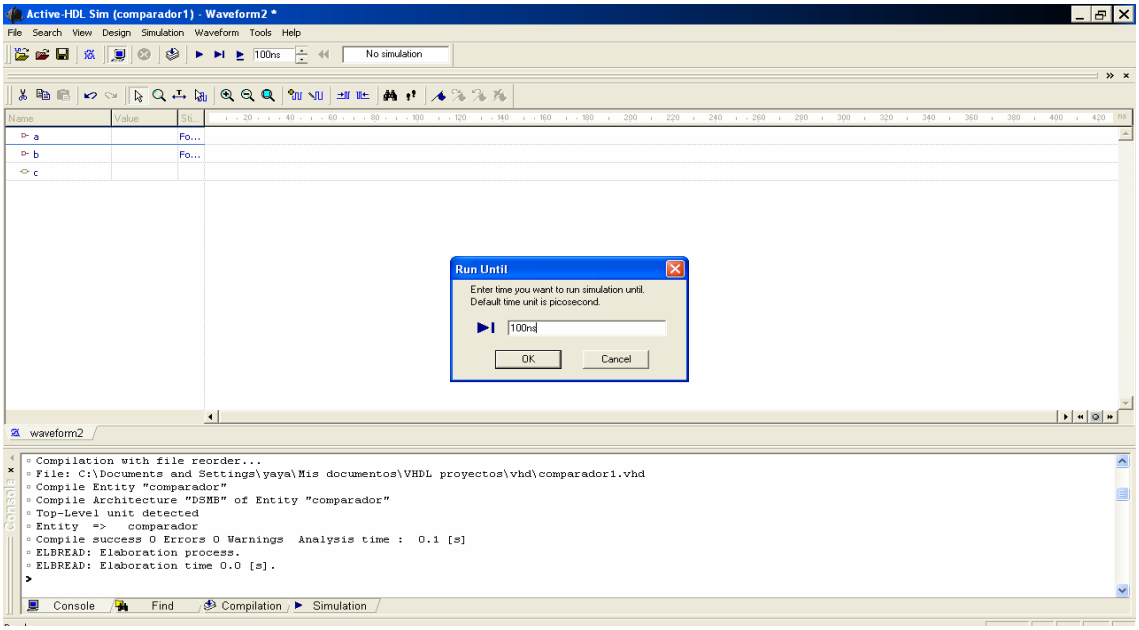
Pantalla 4.30 Ventana de Stimulators

Una vez administrados los valores para las entradas se procede a simular, en la opción “Simulation”, “Run Until”, Para así poder tener lecturas varias en un espacio mas reducido.



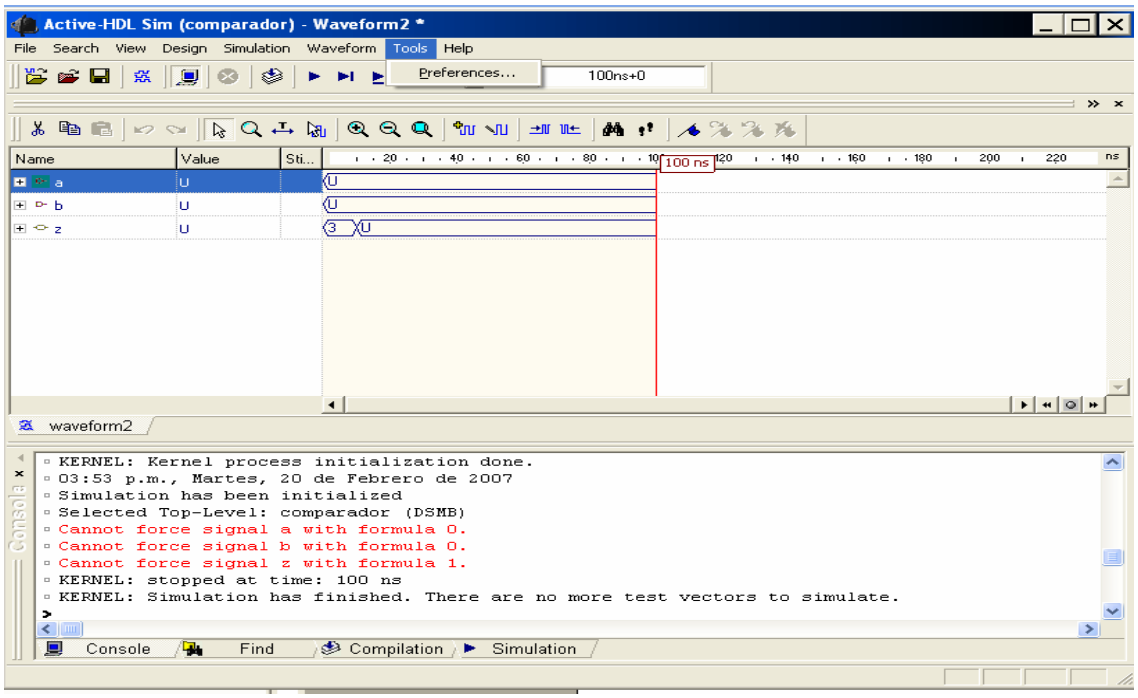
Pantalla 4.31 Simulación de señales

Se selecciona el tiempo de simulación. En este caso será de 100ns y clic en el botón de OK.



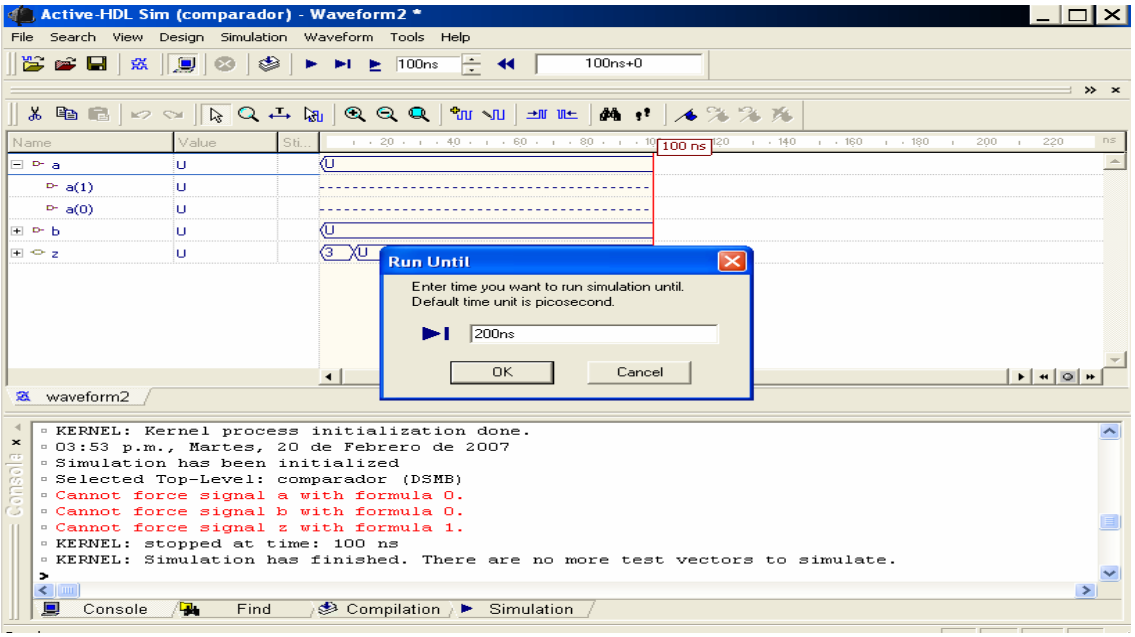
Pantalla 4.32 Tiempo de la primera estimulación

Aquí se muestra que la simulación es de acuerdo al programa, se compara A con B sin son iguales, C devolverá un “1” caso contrario devuelve un “0”.



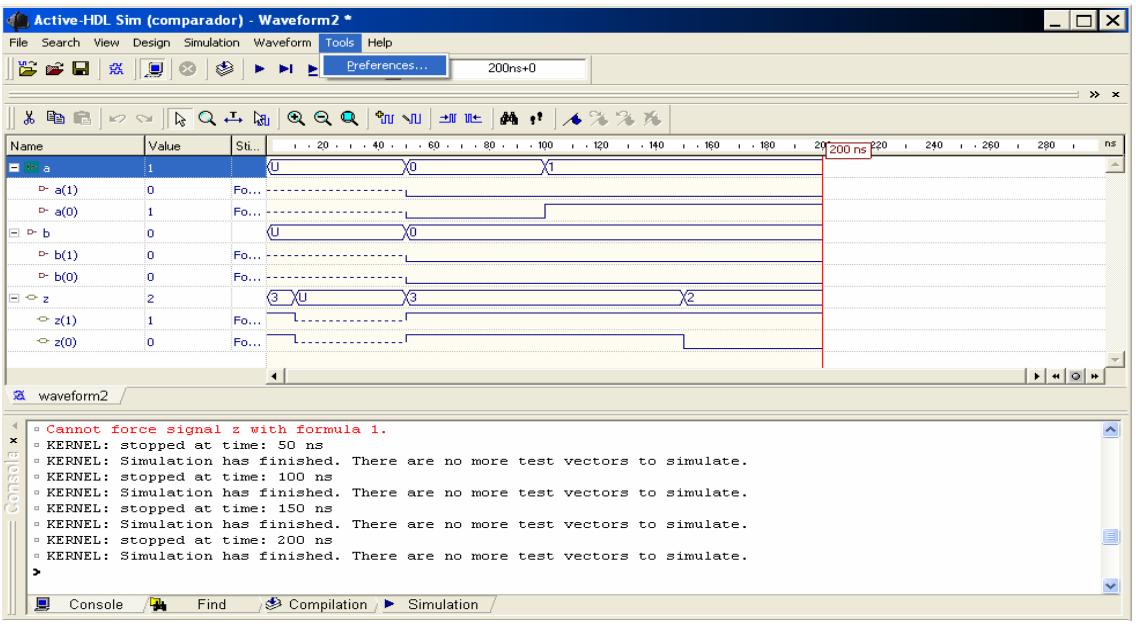
Pantalla 4.33 Simulación de la variables

Se vuelve a correr el programa ahora el tiempo es 200ns.



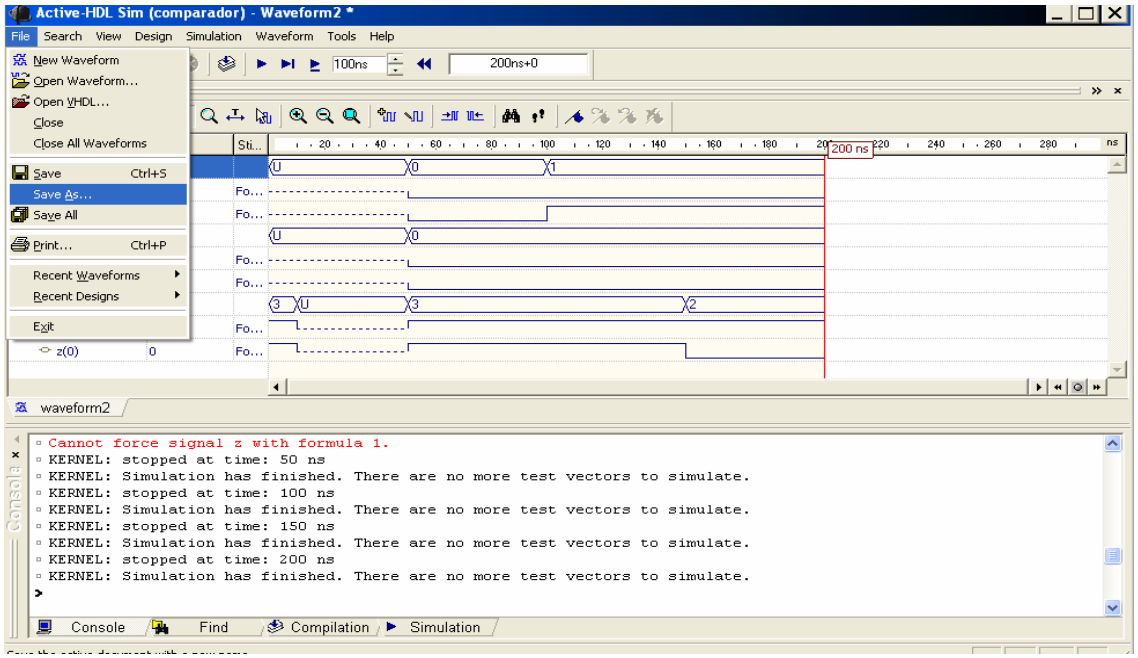
Pantalla 4.34 Nuevo tiempo de simulación

Ahora se muestra en cambio de las variables en el nuevo tiempo.

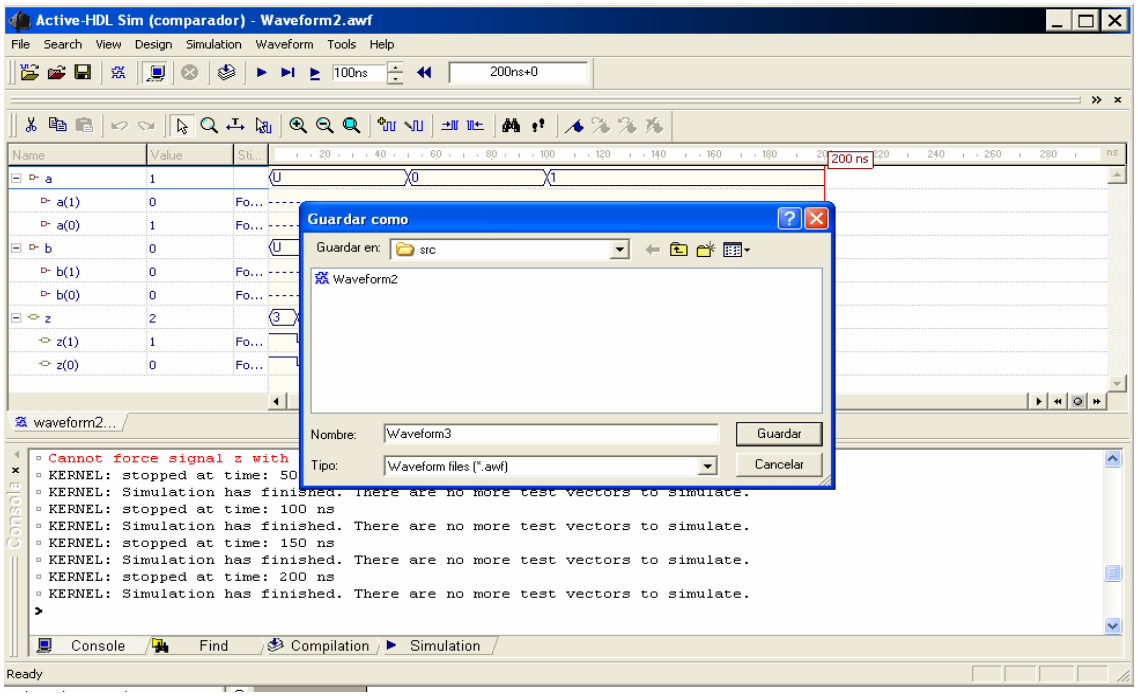


Pantalla 4.35 Tiempo de estimulación

Para guardar el Waveform2* se da clic en file y se selecciona en Save as.



Pantalla 4.36 Guardar un proyecto



Pantalla 4.37 Se guardar el proyecto

CAPÍTULO V

MANUAL DE PRÁCTICAS

5.1 PROGRAMACIÓN DE ESTRUCTURAS BÁSICAS MEDIANTE DECLARACIONES CONCURRENTES

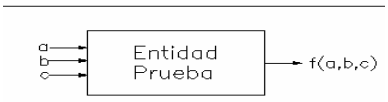
Las declaraciones concurrentes se encuentran fuera de la declaración de un proceso y suelen usarse en las descripciones de flujo de datos y estructural. Esto se debe a que en una declaración concurrente no importa el orden en que se escriban las señales, ya que el resultado para determinada función sería el mismo.

En VHDL existen tres tipos de declaraciones concurrentes:

- Declaración condicionales asignadas a una señal (**when-else**)
- Declaración concurrentes asignadas a señal
- Selección de una señal (**with-select-when**)

1) Declaración condicionales asignadas a una señal (**when-else)**

La declaración **when-else** se utiliza para asignar valores a una señal, determinando así la ejecución de una condición propia del diseño. Por ejemplo la entidad cuyo funcionamiento se define en la tabla de verdad como se muestra en la figura 5.1.



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figura 5.1 Declaración when-else[63]

[63] Véase en www.ate.uniovi.es/fernando/Doc2003/EI/Lenguaje%20VHDL.pdf

La entidad se puede programar mediante *declaraciones condicionales* (when-else), debido a que este modelo permite definir paso a paso el comportamiento del sistema.

```
1   - -Ejemplo combinacional básico
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity tabla is port (
5       a, b, c: in std_logic;
6           f: out std_logic);
7   end tabla;
8   architecture ejemplo of tabla is
9   begin
10      f <= '1' when (a = '0' and b = '0' and c = '0') else
11          '1' when (a = '0' and b = '1' and c = '1') else
12          '1' when (a = '1' and b = '1' and c = '0') else
13          '1' when (a = '1' and b = '1' and c = '1') else
14          '0';
15  end ejemplo;
```

La función de salida f (línea 10) depende directamente de las condiciones que presentan las variables de entrada, además que la ejecución inicial de una u otra condición no afecta la lógica del programa, el resultado es el mismo; es decir, la condición de entrada “111”, visualizad en la tabla de verdad, puede ejecutarse antes que la condición “000” sin alterar el resultado final.

La ventaja de la programación en VHDL en comparación con el diseño lógico puede intuirse considerando que la función de salida f mediante álgebra booleana se representa con:

$$f = \bar{a}\,\bar{b}\,\bar{c} + \bar{a}\,b\,c + a\,b\,\bar{c} + a\,b\,c$$

En el diseño convencional se utilizarían inversores, compuertas OR y compuertas AND; en VHDL la solución es directa utilizando la función lógica *and*. Como ejemplo la línea 10 a la 14 las instrucciones se interpretarían de la siguiente manera:

10 asigna a “f” el valor de 1 cuando a = 0 y b = 0 y c = 0 si no
11 asigna a “f” el valor de 1 cuando a = 0 y b = 1 y c = 1 si no
12 asigna a “f” el valor de 1 cuando a = 1 y b = 1 y c = 0 si no
13 asigna a “f” el valor de 1 cuando a = 0 y b = 1 y c = 1 si no
14 asigna a “f” el valor de 0.

Operadores lógicos

Los operadores lógicos más utilizados en la descripción de funciones booleanas, y definidos en los diferentes tipos de datos bit, son los operadores *and, or, nanad, xor, xnor* y *not*. Las operaciones que se efectúen entre ellos (excepto not) deben realizarse con datos que tengan la misma longitud o palabra de bits.

En el momento de ser compilados los operadores lógicos presentan el siguiente orden y prioridad:

1) Expresiones entre paréntesis
2) Complementos
3) Función AND
4) Función OR

Ecuación	**En VHDL**
q = a + x * y	q = a or (x and y)
$y = a + b * c + d$	y = not (a or (b and not c) or d)

2) Declaración concurrente asignada a señal

En este tipo de declaraciones las funciones de salida mediante la ecuación booleana que describe el comportamiento de cada una de las compuertas. En la siguiente figura 5.2 circuito cuenta con tres salidas (x1, x2 y x3) en lugar de una.

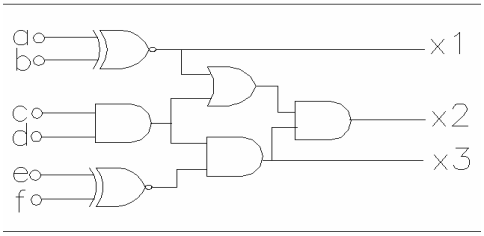


Figura 5.2 Circuito lógico realizado con compuertas[64]

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity logic is port (
4      a, b, c, d, e, f: int sat_logic;
5      x1, x2, x3: out std_logic);
6   end logic;
7   architecture booleanas of logic is
8   begin
9         x1 <= a xnor b;
10        x2 <= (((c and d) or (a xnor b)) nand
11                ((e xnor f) and (c and d)));
12        x3 <= (e xnor f) and (c and d);
13  end booleana;
```

[64] Véase en www.virtual.unal.edu.co/cursos/ingenieria/2000477/lecciones/040101.htm

3) Selección de una señal (**with-select-when**)

La declaración **with-select-when** se utiliza para asignar un valor a una señal con base en el valor de otra señal previamente seleccionada. El valor de salida C depende de las señales de entrada seleccionadas a(0) y a(1), de acuerdo con la tabla de verdad correspondiente.

a(0)	a(1)	C
0	0	1
0	1	0
1	0	1
1	1	0

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity circuito is port (
4       a: in std_logic_vector (1 downto 0);
5       c: out std_logic);
6   end circuito;
7   architecture arq_cir of circuito is
8   begin
9        with a select
10          c <= '1' when “00”,
11                '0' when “01”,
12                '1' when “10”,
14                '0' when others;
15  end arq_cir;
```

5.2 PROGRAMACIÓN DE ESTRUCTURAS BÁSICAS MEDIANTE DECLARACIONES SECUENCIAL

Las declaraciones secuenciales son aquellas en las que el orden que llevan puede tener un efecto significativo en la lógica descrita. A diferencia de una declaración concurrente, una secuencial debe ejecutarse en el orden en que aparece y formar parte de un proceso (**process**).

Declaración **if-then-else** (si-entonces-si no). Esta declaración sirve para seleccionar una condición o condiciones basadas en el resultado de evaluaciones lógicas (falso o verdadero). Por ejemplo la siguiente instrucción:

if la condición es cierta **then**
   realiza la operación 1;
else
   realiza la operación 2;
end if;

Si (**if**) condición se evalúa como verdadera, entonces (**then**) la instrucción indica que se ejecutará la operación 1. Por el contrario, si la condición se evalúa como falsa (**else**) correrá la operación 2. La instrucción que indica el fin de la declaración es **end if** (fin del si). Un ejemplo que ilustra este tipo de declaración es la siguiente figura 5.3.

Figura 5.3 Comparador de igualdad de dos bits[65]

[65] Véase en www.monografias.com/trabajos18/comparadores.html

El código correspondiente a esta figura es el siguiente:

```
1   - -Ejemplo de declaración de la entidad comparador
2   entity comp is
3   port (a, b: in bit_vector(1 dowto 0);
4             c: out bit);
5   end comp;
6   architecture functional of comp is
7   begin
8   compara: process (a,b)
9   begin
10      if a = b then
11         c <= '1';
12      else
13         c <= '0';
14      end if;
15  end process compara;
16  end funcional;
```

5.3 DISEÑO LÓGICO SECUENCIAL CON VHDL

Un sistema secuencial puede tener uno o más elementos combinacionales, la mayoría de los sistemas que se encuentran en la práctica incluyen elementos de memoria, los cuales requieren que el sistema se describa en términos de lógica secuencial.

Algunos de los circuitos secuenciales más utilizados son: flip-flops, contadores, registros, etc.

5.3.1 Diseño lógico secuencial

Un sistema secuencial está formado por un circuito combinacional y un elemento de memoria encargado de almacenar de forma temporal la historia del sistema.
La salida de un sistema secuencial no sólo depende del valor presente de las entradas, sino también de la historia del sistema, como se muestra en la figura 5.1.

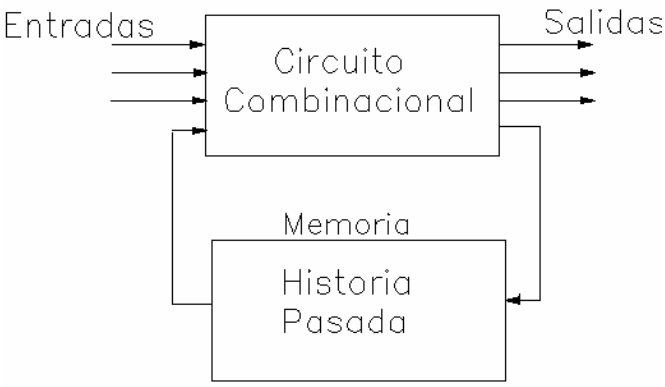


Figura 5.4 Estructura de un sistema secuencial[66]

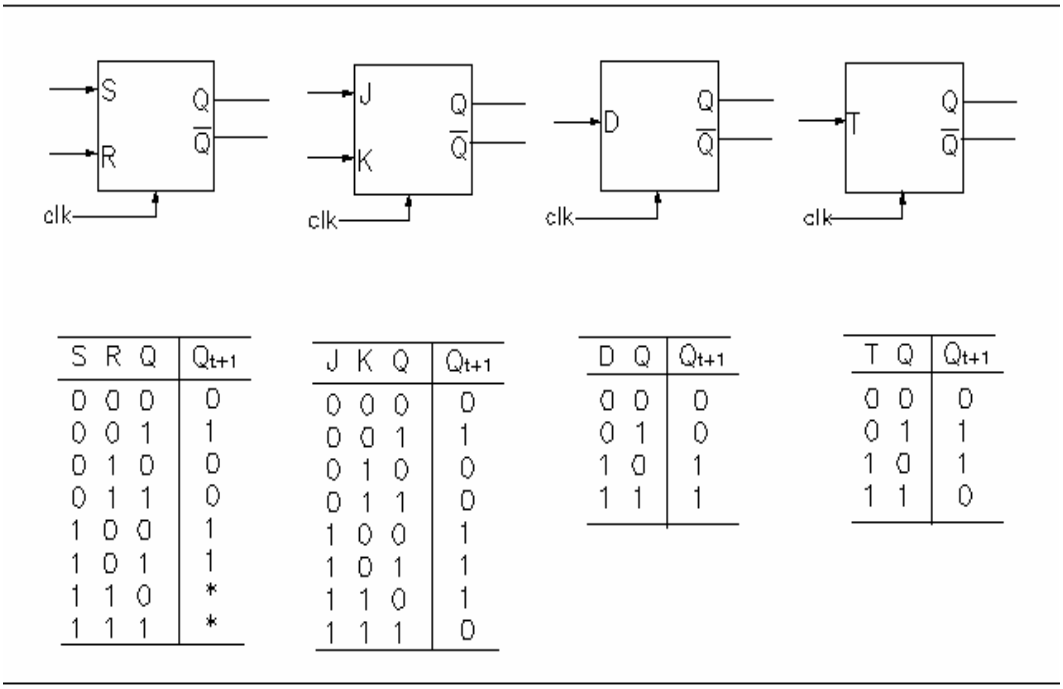
Básicamente hay dos tipos de sistemas secuenciales: síncronos y asíncronos; el comportamiento de los primeros se encuentra sincronizado mediante el pulso de reloj del sistema, mientras que el funcionamiento de los sistemas asíncronos depende del orden y momento en el cual se aplican sus señales de entrada, por lo que no requieren un pulso de reloj para sincronizar sus acciones.

[66] Véase en www.ilustrados.com/circuitos_secuenciales/secuenciales.pdf

5.3.2 Flip-Flops

El elemento de memoria utilizado indistintamente en el diseño de los sistemas síncronos o asíncronos se conoce como **flip-flop** o **celda binaria**.

La característica principal de un flip-flop es mantener o almacenar un bit de manera indefinida hasta que a través de un pulso o una señal cambie de estado. Los flip-flops más conocidos son los tipos SR, JK, T y D. En la figura 5.5 se presenta cada uno de estos elementos y la tabla de verdad que describa su comportamiento.



S	R	Q	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	*
1	1	1	*

J	K	Q	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

D	Q	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

T	Q	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figura 5.5 Flip-Flops y tablas de verdad [67]

Es importante recordar el significado de la notación Q y Q(t+1):

Q = estado presente o actual

Q(t+1) = estado futuro o siguiente

[67] Véase en www.ilustrados.com

Por ejemplo, la tabla de verdad que describe el funcionamiento del flip-flop tipo D se muestra en la siguiente figura 5.6.

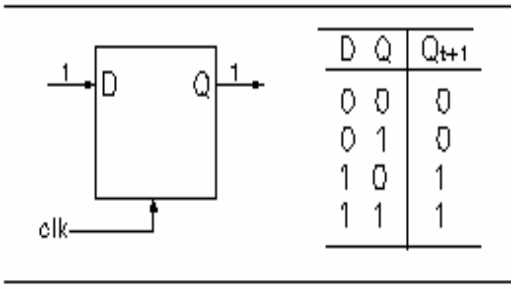


Figura 5.6 Diagrama y tabla de verdad[68]

Cuando el valor de la entrada D es igual a 1 figura 5.6 b), la salida Q(t+1) adopta el valor de 1: Q(t+1) = 1 siempre y cuando se genere un pulso de reloj. Es importante resaltar que el valor actual en la entrada D es transferido a la salida Q(t+1) sin importar cual sea el valor previo que hay tenido la salida Q en el estado presente.

La ejecución del proceso es sensible a los cambios en *clk* (pulso de reloj); esto es, cuando clk cambia de valor de una transición de 0 a 1 (clk = 1), el valor de D se asigna a Q y se conserva hasta que se genera un nuevo pulso. A la inversa, si clk no presenta dicha transición, el valor de Q se mantiene igual. Esto puede observarse con claridad en la simulación del circuito.

En el diseño secuencial con VHDL las declaraciones **if-then-else** son las mas utilizadas como se muestra en el siguiente listado.

[68]Véase en www.ilustrados.com

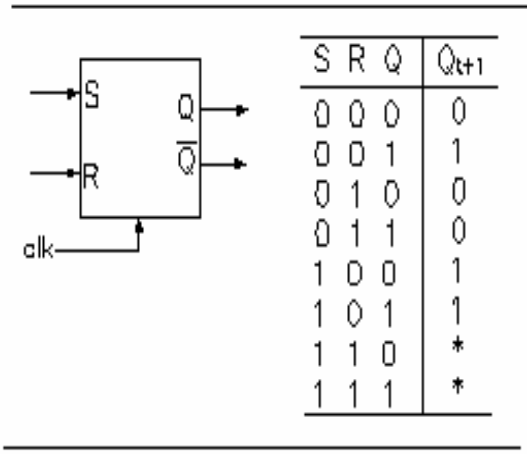
```
1   - - Descripción de un flip-flop disparado por flanco positivo
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity ffd is port ( D, clk: in std_logic;
5                        Q: out std_logic);
6   end ffd;
7   architecture arq_ffd of ffd is
8   begin
9       process (clk) begin
10              if (clk 'event and clk = '1') then
11                  Q <= D;
12              end if;
13      end process;
14  end arq_ffd;
```

Atributo 'event

En el lenguaje VHDL los atributos sirven para definir características que se pueden asociar con cualquier tipo de datos, objeto o entidades.

En el listado anterior la condición if clk 'event es cierta sólo cuando ocurre un cambio de valor; es decir un suceso (event) de la señal de clk. La declaración (if-then) no maneja la condición else, debido a que el compilador mantiene el valor de Q hasta que no exista un cambio de valor en la señal clk.

Programa que describe el funcionamiento de un flip-flop SR con base en la siguiente figura 5.7.



S	R	Q	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	*
1	1	1	*

Figura 5.7 Flip-Flop SR y tabla de verdad[69]

La tabla de verdad del flip-flop SR muestra que cuando la entrada S es igual a 1 y la entrada R es igual a 0, la salida Q(t+1) toma valores lógicos de 1. Por otro lado, cuando S=0 y R=1, la salida Q(t+1) = 0; en el caso de que S y R sean ambas igual a 1 lógico, la salida Q(t+1) queda indeterminada; es decir no es posible precisar su valor y este puede adoptar el 0 o 1 lógico.

Por último, cuando no existe cambio en las entradas S y R es decir, son igual a 0, el valor de Q(t+1) mantiene su estado actual Q.

El programa en VHDL puede realizarse utilizando instrucciones condicionales y un nuevo tipo de datos: *valor no importa* ('-'), los cuales permiten adoptar un valor de 0 o 1 lógico de manera indistinta.

[69] Véase en www.monografias.com

```
1   - -Código VHDL de un registro de 8 bits
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity ffsr is port (
5         S, R, clk: in std_logic;
6         Q, Qn: inout std_logic);
7   end ffsr;
8   architecture a_ffsr of ffsr is
9   begin
10  process (clk, S, R)
11  begin
12        if (clk 'event and clk = '1') then
13           if (S = '0' and R = '1' ) then
14                Q <= '0';
15                Qn <= '1';
16           elsif (S = '1' and R = '0') then
17                  Q <= '1';
18                  Qn <= '0';
19           elsif (S = '0' and R = '0') then
20                  Q <= Q;
21                  Qn <= Qn;
22           else
23                Q <= '-';
24                Qn <= '-';
25            end if;
26        end if;
27  end process;
26  end a_ffsr;
```

5.3.3 Registros

En la siguiente figura 5.8 se presenta la estructura de un registro de 8 bits con entrada y salida de datos en paralelo. El diseño es muy similar al flip-flop, la diferencia radica en la utilización de vectores de bits en lugar de un solo bit.

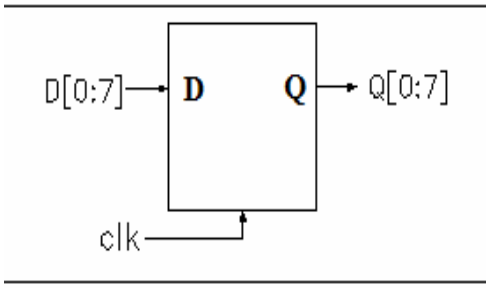


Figura 5.8 Registro paralelo de 8 bits[70]

```
1   - -Código VHDL de un registro de 8 bits
2   library ieee;
3   use ieee.std_logic_1164. all;
4   entity reg is port (D: in std_logic_vector(0 to 7);
6                       slk: in std_logic;
7                       Q: out std_logic_vector(0 to 7);
8   end reg;
9   architecture arqreg of reg is
10  begin
11      process (clk) begin
12      if (clk 'event and clk = '1') then
13          Q <= D;
14      end if;
15      end process;
16  end arqreg;
```

[70] Véase en www.ilustrados.com

5.3.4 Contadores

Los contadores son el grupo más sencillo de máquinas secuenciales y son circuitos digitales que permiten dar una secuencia de datos en forma cíclica. Un ejemplo de contador se puede ver con un sistema que entregue la secuencia de los dígitos decimales como: 0 1 2 3 4 5 6 7 8 9 0. Los contadores son entidades muy utilizadas en el diseño lógico. La forma usual para describirlos en VHDL es mediante operaciones de incremento, decremento de datos o ambas.

Como ejemplo la figura 5.9 que representa un contador ascendente de 4 bits.

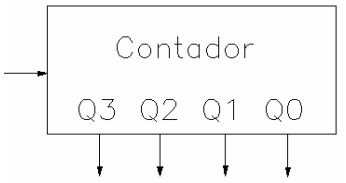


Figura 5.9 Contador binario de cuatro bits[71]

Ejemplo que describe un contador de 4 bits.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use work.std_arith.all;
4   entity cont4 is port (clk: in std_logic;
5           Q: inout std_logic_vector (3 downto 0));
6   end cont4;
7   architecture arqcont of cont4 is
8   begin
9       process (clk)
10          begin
```

[71]Véase en www.monografias.com/trabajos18/contadores.html

```
11              if (clk 'event and clk = '1') then Q <= Q + 1;
12              end if;
13      end process;
14  end arqcont;
```

La retroalimentación de una señal (Q <= Q + 1), ya sea dentro o fuera de la entidad, se utiliza el modo **inout.** El puerto correspondiente a Q se maneja como tal, debido a que la señal se retroalimenta en cada pulso de reloj. El uso del paquete **std_arith** permite usar el operador + con el tipo std_logic_vector.

El funcionamiento del contador se define básicamente en un proceso, en el cual se llevan a cabo los eventos que determinan el comportamiento del circuito. Al igual que en los otros programas, una transición de 0 a 1 efectuada por el pulso de reloj provoca que se ejecute el proceso, lo cual incrementa en 1 el valor asignado a la variable Q. Cuando esta salida tiene el valor de 15 (“1111”) y si el pulso de reloj se sigue aplicando, el programa empieza a contar nuevamente de 0.

5.3.5 Diseño de sistemas secuenciales síncronos

La estructura de los sistemas secuenciales síncronos basa su funcionamiento en los elementos de memoria conocidos como flip-flops. La palabra sincronía se refiere a que cada uno de estos elementos de memoria que interactúan en un sistema se encuentran conectados a la misma señal de reloj, de forma tal que solo se producirá un cambio de estado en el sistema cuando ocurra un flanco de disparo o un pulso en la señal de reloj.

Existe una división en el diseño de los sistemas secuenciales que se refiere al momento en que se producirá la salida del sistema.

-
-

- En la estructura de Mealy (fig. 5.10) las señales de salida dependen tanto del estado en que se encuentra el sistema, como de la entrada que se aplica en determinado momento.
- En la estructura de Moore la señal de salida solo depende del estado en que se encuentra.

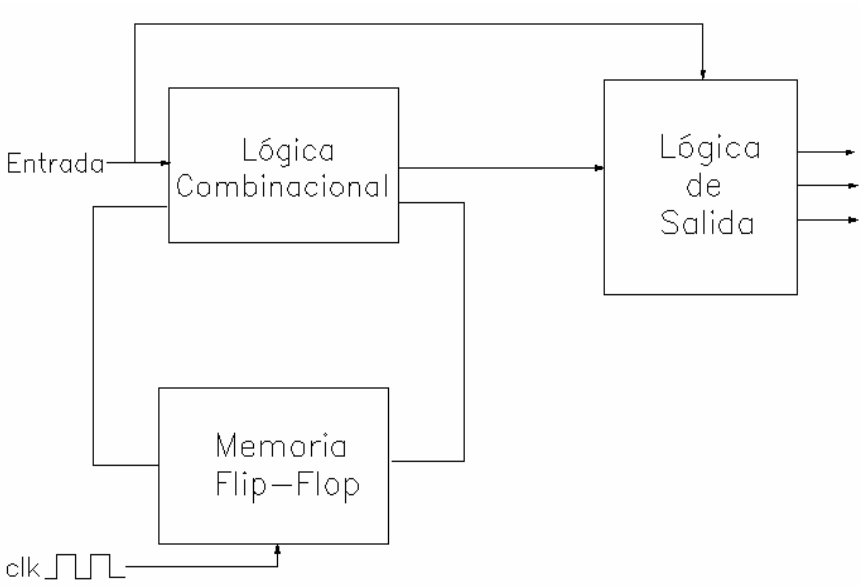


Figura 5.10 Arquitectura secuencial de Mealy[72]

5.4 UNIDAD LÓGICA ARITMÉTICA CON VHDL

ALU (Unidad lógica aritmética). Es el último componente de la CPU que entra en juego. La ALU es la parte inteligente del chip, y realiza las funciones de suma, resta, multiplicación o división. También sabe cómo leer comandos, tales como OR, AND o NOT. Los mensajes de la unidad de control le dicen a la ALU qué debe hacer. Es una unidad capaz de realizar diferentes operaciones aritméticas y lógicas sobre dos números binarios.

[72] Véase en www.monografias.com/unidad_logica.pdf

ESPECIFICACIONES DE LA ALU

Las características principales de la ALU que se pide son:

- Ancho de operandos de 4 bits.
- Posibilidad de realizar hasta 8 operaciones distintas.

- Entradas: dos operandos A[3:0] y B[3:0], y tres líneas de selección de operación, S[2:0].
- Salidas: resultado de la operación R[3:0] junto con una señal de acarreo, COUT, necesaria en algunas operaciones.
- Operaciones a realizar:

S2	S1	S0	Salida
0	0	0	0000 (***reset***)
0	0	1	**COMPARA**(A,B)
0	1	0	A **AND** B
0	1	1	B
1	0	0	A/2
1	0	1	Complemento a 2 de A
1	1	0	B - A
1	1	1	**ROR**(B)

La operación COMPARA(A,B) consiste en proporcionar en COUT un 1 si A >=B y 0 en caso contrario.

La operación ROR(B) consiste en desplazar los bits de B una posición a la derecha, es decir, R(2)<=B(3), R(1) <= B(2), R(0)<=B(1), R(3)<=B(0).

Como estrategia de diseño, puede realizar un circuito completamente combinacional, o utilizar multiplexores para canalizar 8 funciones diferentes hacia la salida.

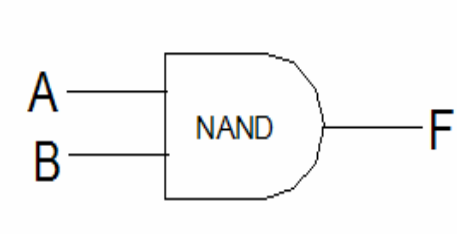
```
Ejemplo 1 de programación de la ALU
Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_unsigned.all;
Entity alu is
Port (
        op1, op2: in std_logic_vector(1 downto 0);
        operacion: in std_logic_vector(2 downto 0);
        resultado: out std_logic_vector(1 downto 0)
      );
end alu;
architecture functional of alu is
with operacion select
resultado <= op1 + op2 when “000”,
          <= op1 – op2 when “010”,
          <= connv_std_logic_vector(conv_integer
          (op1)*conv_integer(op2), 2) when “011”,
          <= op1 and op2 when “100”,
          <= op1 or op2 when “101”,
          <= (others => ‘0’) when others;
end functional;
```

Ejemplo 2 de una ALU

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity LC2_ALU is
  port(
        A: in std_logic_vector (15 downto 0);
        B: in std_logic_vector (15 downto 0);
        S: in std_logic_vector (1 downto 0);
        O: out std_logic_vector (15 downto 0)
      );
end LC2_ALU;
architecture bhv of LC2_ALU is
begin
  process(A, B, S)
begin
 case S is
    when "00" => O <= A+B;
    when "01" => O <= A and B;
    when "10" => O <= A;
    when "11" => O <= not A;
    when others => null;
  end case;
end process;
end bhv;
```

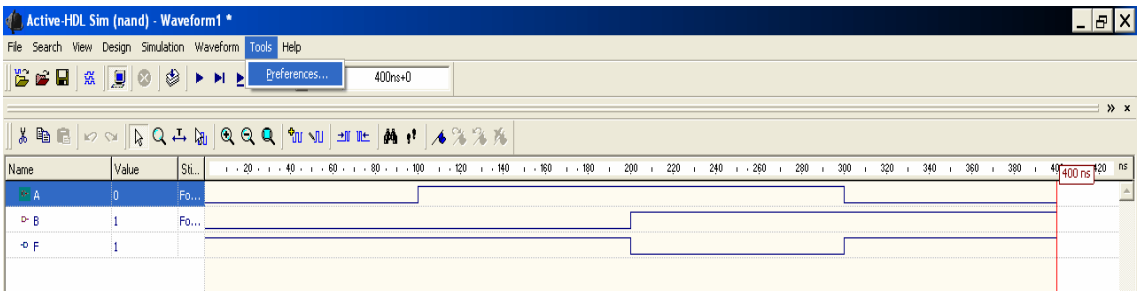
5.5 DESARROLLAR VARIOS PROGRAMAS

Ejercicio 1. Programación de una compuerta NAND

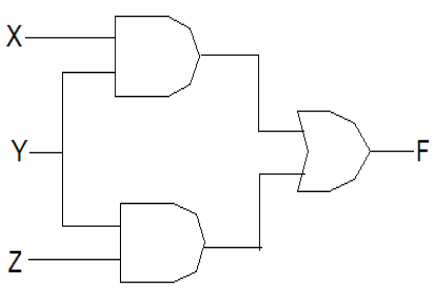



```
library ieee;
use ieee.std_logic_1164.all;
entity compuerta_nand is
   port(
            A, B: in std_logic; --Entradas simples
               F: out std_logic --Salida simple
        );
end compuerta_nand;
architecture simple of compuerta_nand is
    begin
            F <= A nand B; --Compuerta nand
    end simple;
```

Simulación de la compuerta NAND en Active-HDL Sim

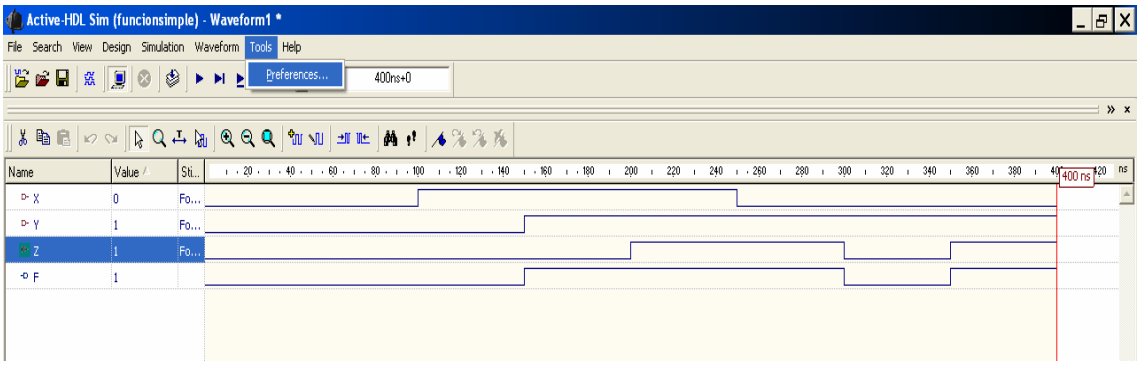


Ejercicio 2. Programación de una función lógica

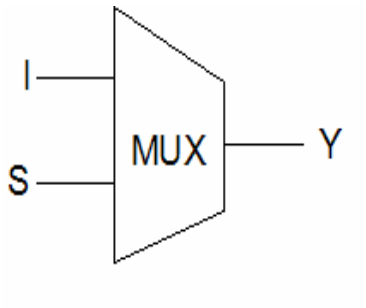



```
library ieee;
use ieee.std_logic_1164.all; --libreria estandar
entity funcion_simple is
    port(
        X, Y, Z: in std_logic; --3 entradas simples
            F: out std_logic --salida simple
        );
    end funcion_simple;
architecture basica of funcion_simple is
   begin
        F <= (X AND Y) OR (Y AND Z); --funcion and-or
end basica;
```

Simulación de la función lógica en Active-HDL Sim

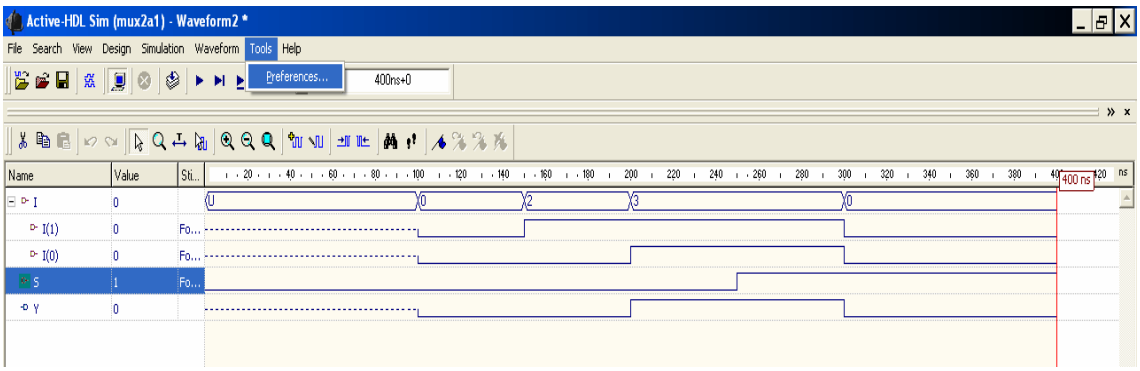


Ejercicio 3. Programación de un Multiplexor de 2 entradas y una salida




```
library ieee;
use ieee.std_logic_1164.all;
entity mux2_1 is
  port(
        I: in std_logic_vector(1 downto 0); --entradas
        S: in std_logic; --seleccion
        Y: out std_logic --salida
       );
 end mux2_1;
architecture simple of mux2_1 is begin
        Y <= I(0) when (S='0') else I(1); --asignacion condicional
end simple;
```

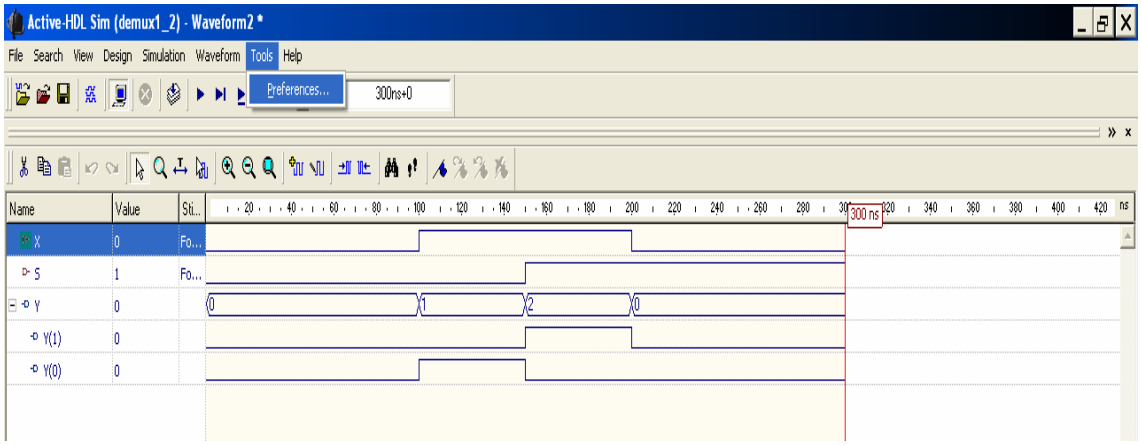
Simulación del multiplexor en Active-HDL Sim



Ejercicio 4. Programación de un demultiplexor de una entrada y 2 salidas

```
library ieee;
use ieee.std_logic_1164.all; --libreria estandar
entity demux1_2 is
   port(
            X: in std_logic; --entrada
            S: in std_logic; --seleccion
            Y: out std_logic_vector(1 downto 0) --salida  );
        end demux1_2;
architecture simple of demux1_2 is
   begin
          Y(0) <= X when (S='0') else '0'; --asignacion condicional
               Y(1) <= X when (s='1') else '0'; --asignacion condicional
        end simple;
```

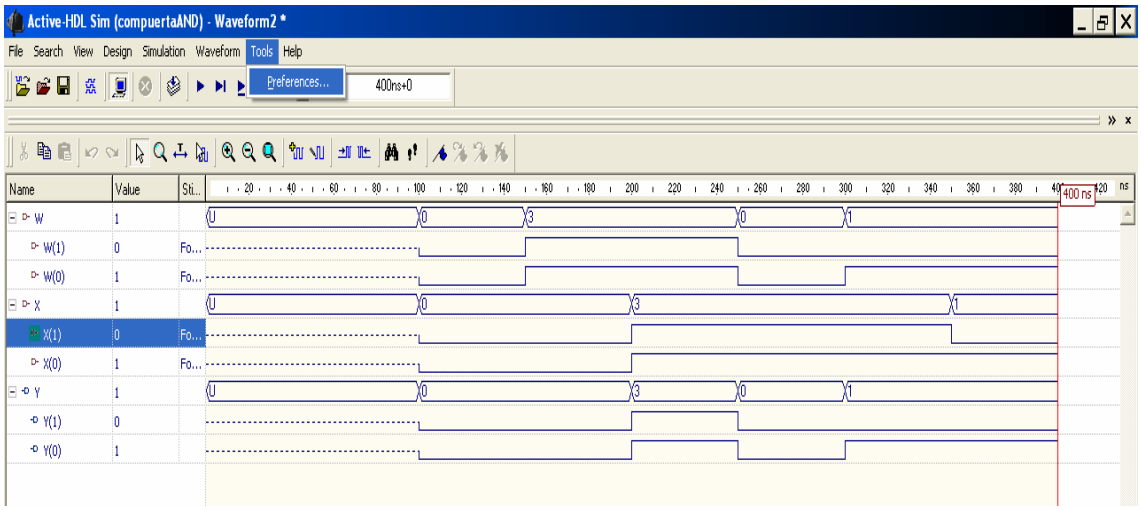
Simulación de un demultiplexor en Active-HDL Sim



Ejercicio 5. Programación de una compuerta AND

```
library ieee;
use ieee.std_logic.1164.all; --libreria estandar
entity operador_and is
   generic(
        n:integer:=8 );
port( X: in std_logic_vector(n-1 downto 0); --entrada X
         W: in std_logic_vector(n-1 downto 0); --entrada W
         Y: out std_logic_vector(n-1 downto 0) --salida
         );
end operador_and;
architecture simple of operador_and is
    begin
            Y<=X AND W;
         end simple;
```

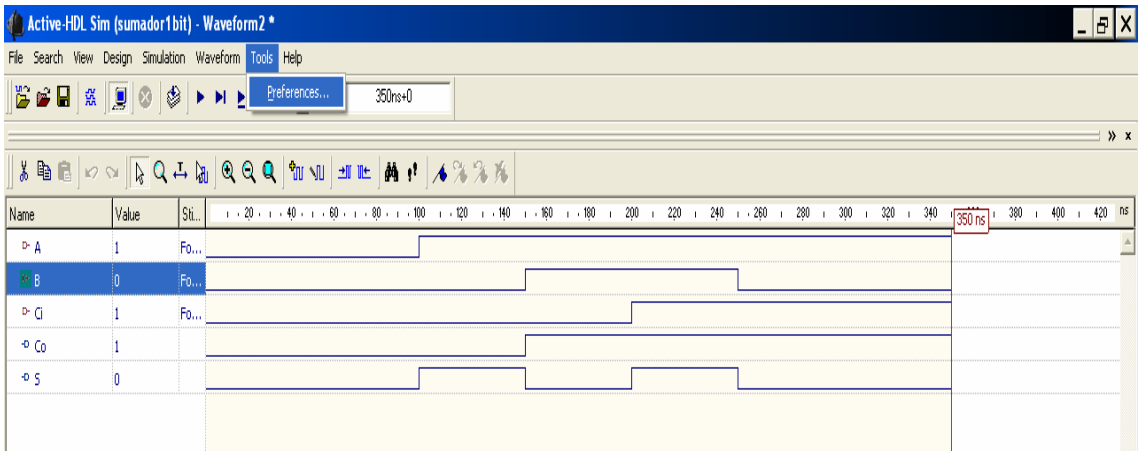
Simulación de una compuerta AND vectorial en Active-HDL Sim



Ejercicio 6. Programación de un sumador de 1 bit

```
library ieee;
use ieee.std_logic_1164.all;
entity suma_1 is
    port(
            A, B: in std_logic; --operandos
              Ci: in std_logic; --acarreo de entrada
             Co: out std_logic;--acarreo de salida
              S: out std_logic--suma
                );
        end suma_1;
architecture simple of suma_1 is
begin
   Co <= (A and B) or (A and Ci) or (B and Ci);
    S <= (A xor B) xor Ci;
end simple;
```

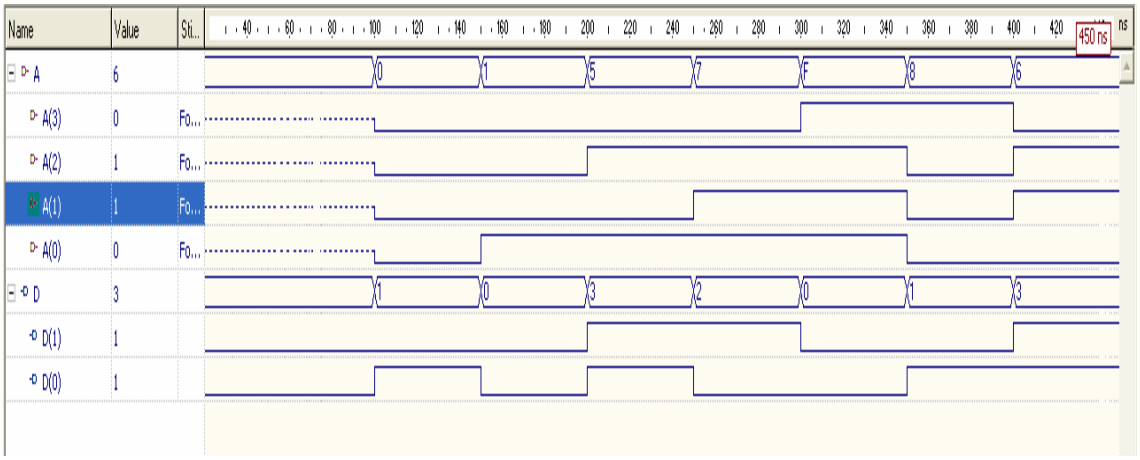
Simulación de un sumador en Active-HDL Sim



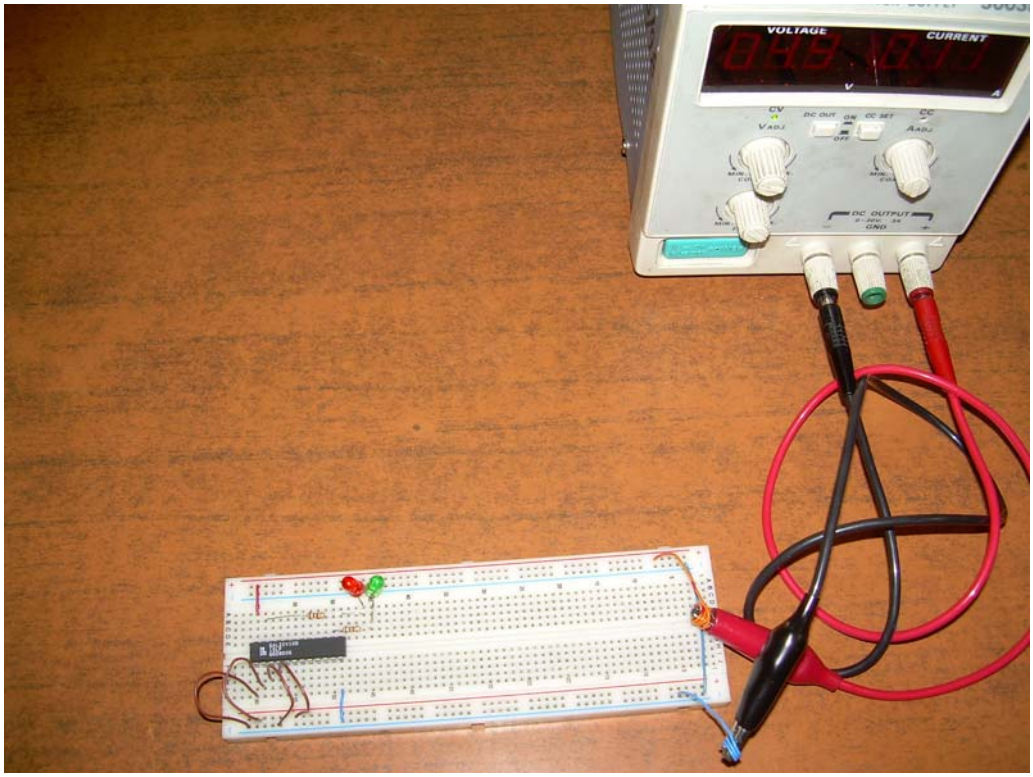
Ejercicio 7. Programación de una Rom de 4 entradas y 2 salidas

```
library ieee;
use ieee.std_logic_1164.all;
entity rom4x2 is
    port(A: in std_logic_vector(3 downto 0); --direccion de entrada
         D: out std_logic_vector(1 downto 0) ); --datos de salida
end rom4x2;
architecture tabla of rom4x2 is
begin
   process(A)
        begin
           case A is
                  when "0000" => D <= "01"; --datos en forma de tabla de verdad
                  when "0001" => D <= "00";
                  when "0010" => D <= "01";
                  when "0011" => D <= "01";
                  when "0100" => D <= "00";
                  when "0101" => D <= "11";
                  when "0110" => D <= "11";
                  when "0111" => D <= "10";
                  when "1000" => D <= "01";
                  when "1001" => D <= "01";
                  when "1010" => D <= "10";
                   when "1011" => D <= "11";
                  when "1100" => D <= "00";
                  when "1101" => D <= "00";
                  when "1110" => D <= "01"; when others => D <= "00";
           end case;
        end process;
end tabla;
```

Simulación de una memoria ROM en Active-HDL Sim



Fotografía del circuito de una memoria ROM

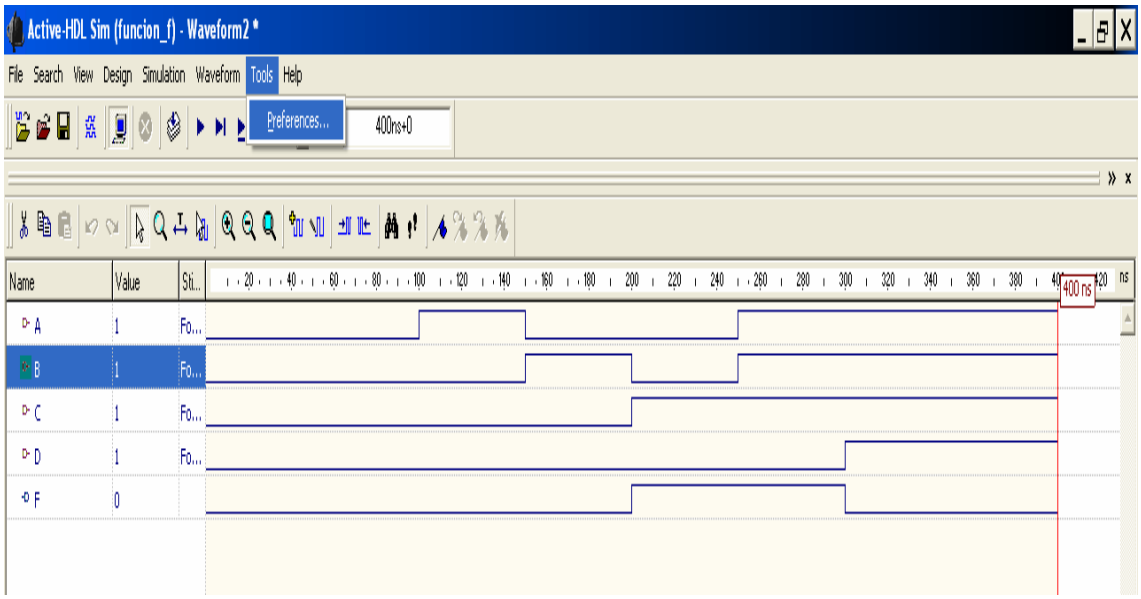


Ejercicio 8. Programación de una función F de cuatro variables A, B, C, D que representan un número binario, donde A es la variable menos significativa. La función F adopta el valor de uno si el número formado por las cuatro variables es inferior o igual a 7 y superior a 3. En caso contrario la función F es cero.

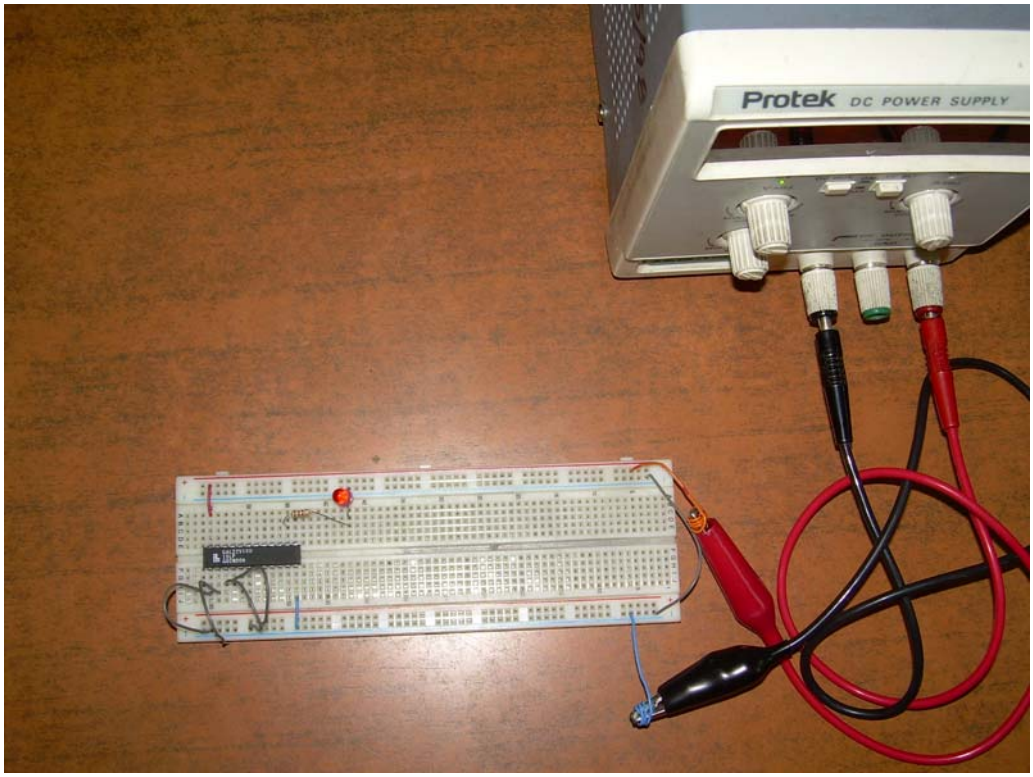
D	C	B	A	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

```
library ieee;
use ieee.std_logic_1164.all;
entity funcion is
port(
   D,C,B,A: in std_logic;
            F: out std_logic
    );
end funcion;
architecture a_func of funcion is
begin
   F <= '1'when (A='0' and B='0' and C='1' and D='0') else
          '1'when (A='1' and B='0' and C='1' and D='0') else
          '1'when (A='0' and B='1' and C='1' and D='0') else
           '1'when (A='1' and B='1' and C='1' and D='0') else
            '0';
end a_func;
```

Simulación de un sumador en Active-HDL Sim



Fotografía del circuito de un sumador



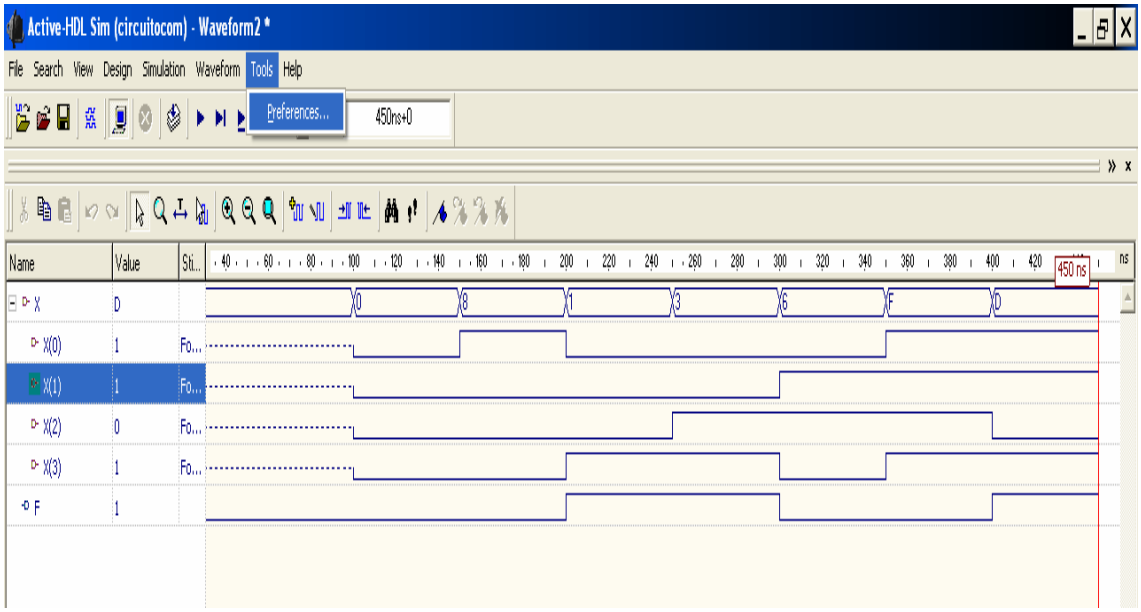
Ejercicio 9. Programación de un circuito combinacional que detecta números primos de 4 bits. La tabla de verdad que resuelve la función es la siguiente:

X0	X1	X2	X3	F
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
0	1	1	1	1

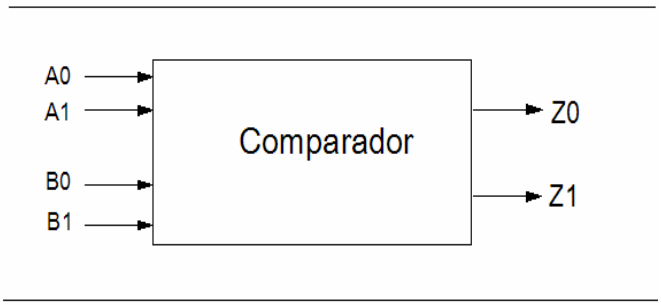
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

```
library ieee;
use ieee.std_logic_1164.all;
entity seleccion is
port(
   X: in std_logic_vector(0 to 3);
        F: out std_logic
   );
end seleccion;
architecture a_selec of seleccion is
begin
   with X select
        F <= '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 a_selec;
```

Simulación de un circuito combinacional en Active-HDL Sim



Ejercicio 10. Programación de un comparador de dos números A y B, cada número formado por dos bits (A1 A0) y (B1 y B0) la salida del comparador también es de dos bits y está representada por la variable Z (Z1 y Z0) de tal forma que si:



A = B entonces Z = 11

A < B entonces Z = 01

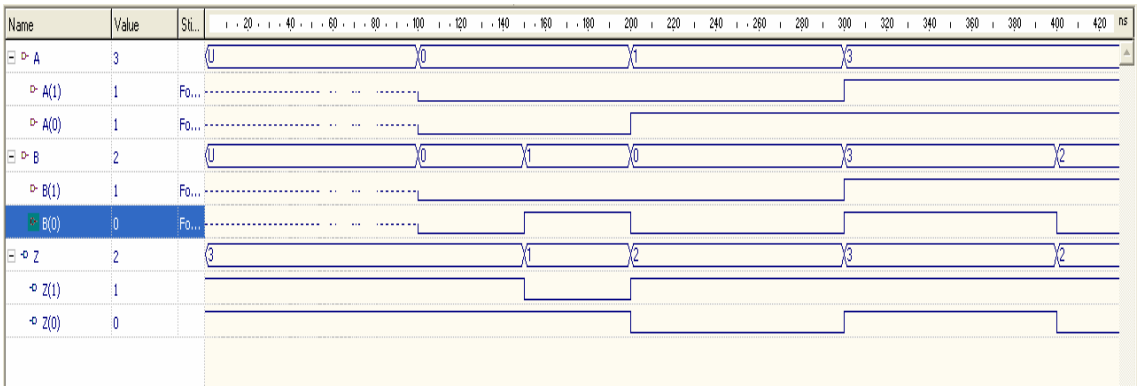
A > B entonces Z = 10

A1	A0	B1	B0	Z1	Z0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

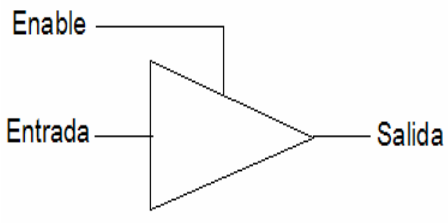
```
library ieee;
use ieee.std_logic_1164.all;
entity comparador is
port( A,B: in std_logic_vector(1 downto 0);
      Z: out std_logic_vector(1 downto 0));
end comparador;
architecture comp of comparador is
begin
   process (A,B)
        begin
```

```
            if A = B then
                    Z<= "11";
        elsif A < B then
                    Z <= "01";
         else
                    Z <= "10";
         end if;
    end process;
end comp;
```

Simulación de un sumador en Active-HDL Sim

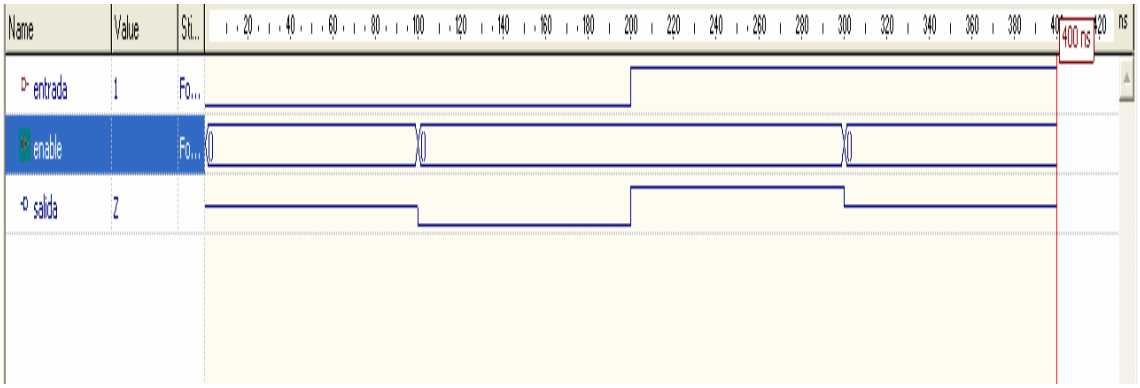


Ejercicio 11. Programación de un registro de tres estados. En este proceso se indica que cuando se confirma el habilitador del circuito (enable), el valor que se encuentra a la entrada del circuito se asigna a la salida; si por lo contrario no se confirma enable, la salida tomará un valor de alta impudencia (Z).



```
library ieee;
use ieee.std_logic_1164.all;
entity registro_3 is
port(
   enable, entrada: in std_logic;
                salida: out std_logic
          );
end registro_3;
architecture resg of registro_3 is
begin
   process(enable, entrada)
         begin
            if enable = '0' then
                     salida <= 'Z';
             else
                     salida <= entrada;
           end if;
    end process;
end resg;
```

Simulación de un registro de tres estados en Active-HDL Sim



Ejercicio 12. Programación de un medio sumador

Para describir el funcionamiento de los circuitos sumadores es importante recordar las reglas básicas de la adición.

0 + 0 = 0

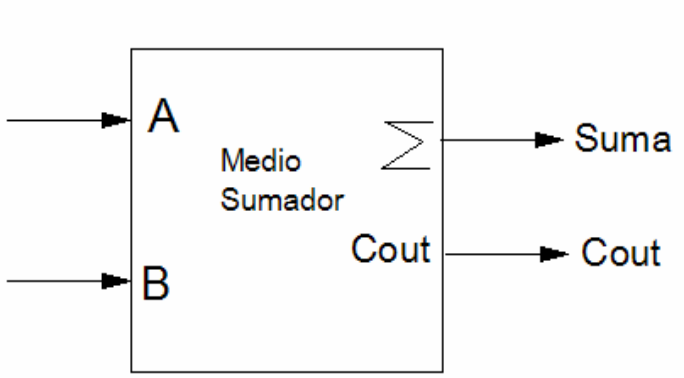
0 + 1= 1

1+ 0 = 0

1+ 1 = 10

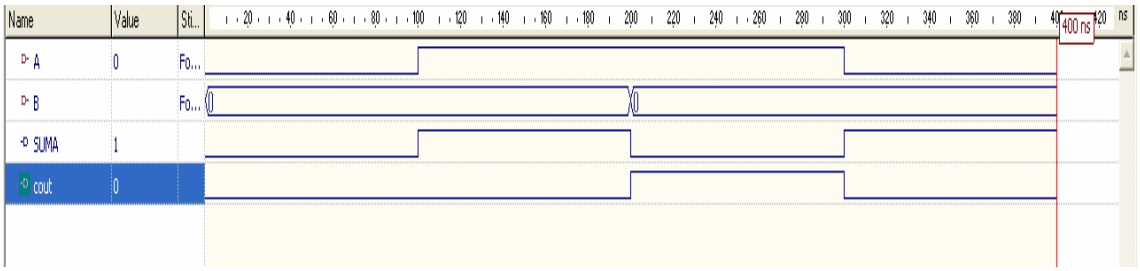
En el caso de la suma 1 + 1 = 10, el resultado “0” representa el valor de la suma, mientras que el “1” el valor del acarreo.

A	B	Suma	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1 (1+1=10)

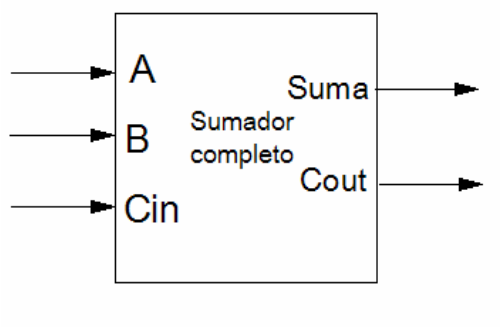


```
library ieee;
use ieee.std_logic_1164.all;
entity m_sumador is
port(A,B: in std_logic;
        SUMA, Cout: out std_logic);
end m_sumador;
architecture sumador of m_sumador is
begin
   SUMA <= A xor B;
      Cout <= A and B;
end sumador;
```

Simulación de un medio sumador en Active-HDL Sim



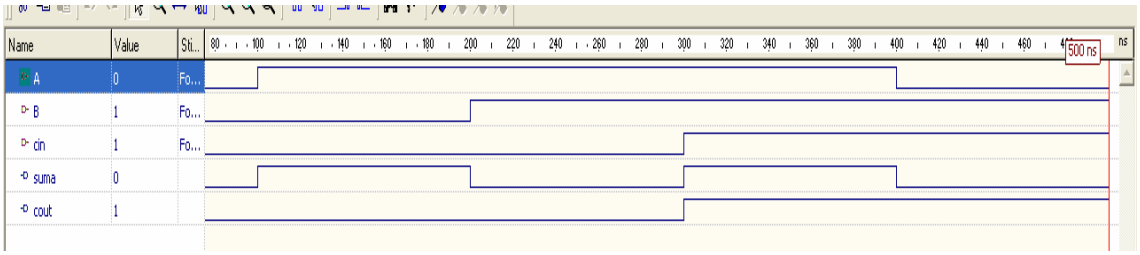
Ejercico 13. Programación de un sumador completo



A	B	Cin	Suma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
library ieee;
use ieee.std_logic_1164.all;
entity sum is
port( A,B,Cin: in std_logic;
        Suma,Cout: out std_logic
        );
end sum;
architecture sumc of sum is
begin
   Suma <= A xor B xor Cin;
   Cout <= ((A and B) or (A xor B)) and Cin;
end sumc;
```

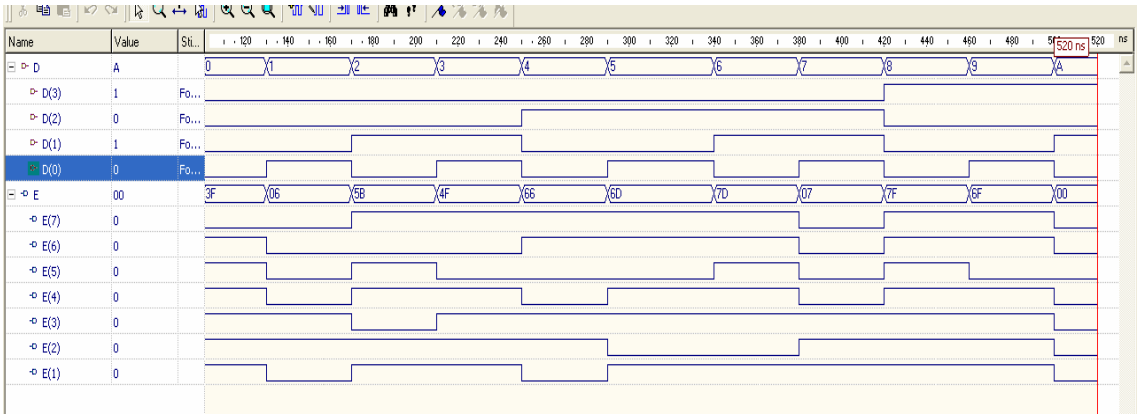
Simulación de un sumador completo en Active-HDL Sim



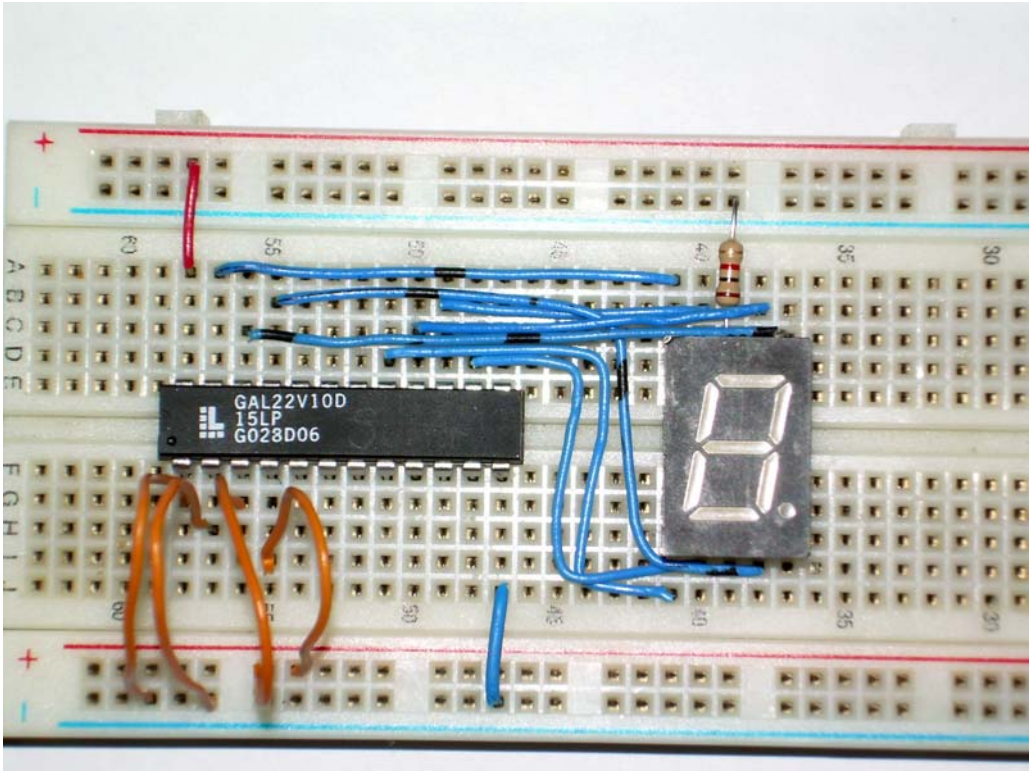
Ejercico 14. Decodificador de BCD a display de siete segmentos

```
library ieee;
use ieee.std_logic_1164.all;
entity BCD_7s is
port(
	D:in std_logic_vector(3 downto 0);
	E:out std_logic_vector(7 downto 1)
	);
end BCD_7s;
architecture simple of BCD_7s is
begin
	process(D)
		begin
			case D is
				when "0000" => E <= "0111111";
				when "0001" => E <= "0000110";
				when "0010" => E <= "1011011";
				when "0011" => E <= "1001111";
				when "0100" => E <= "1100110";
				when "0101" => E <= "1101101";
				when "0110" => E <= "1111101";
				when "0111" => E <= "0000111";
				when "1000" => E <= "1111111";
				when "1001" => E <= "1101111";
				when others => E <= "0000000";
			end case;
	end process;
end simple;
```

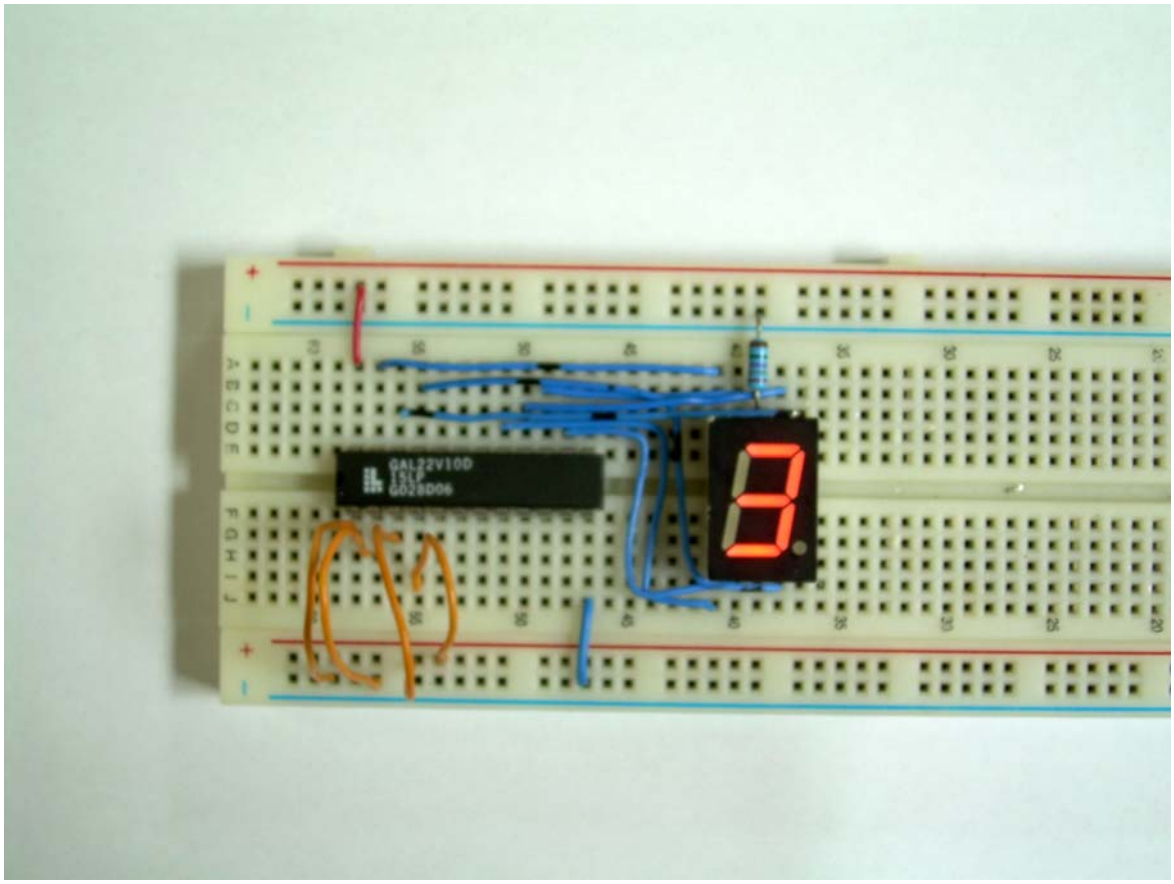
Simulación de un decodificador de BCD a display de siete segmentos en Active-HDL Sim



Fotografía del circuito decodificador de BCD a siete segmentos sin alimentación de voltaje



Fotografía del circuito decodificador de BCD a siete segmentos con un voltaje de 5volts

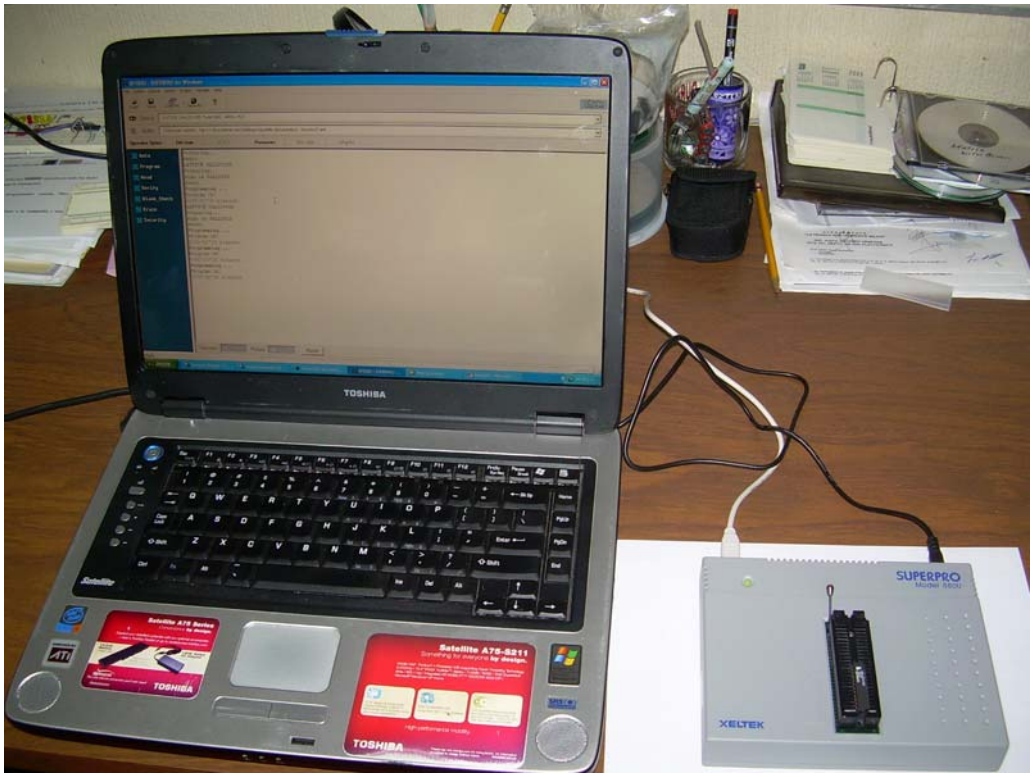


5.6 Programador

El programador permite interactuar entre el programa VHDL y el circuito integrado, este es el programador de dispositivos lógicos programables de la marca Xeltek que se utilizo para las prácticas.

Fotografía del programador Xeltek que se utilizo para las prácticas e

Fotografía de cómo se hace un vinculo con la pc y el programador



CONCLUSIÓN

El desarrollo de esta tesis me sirvió para adquirir un conocimiento bastante completo sobre las distintas familias de lógica programable y de los diferentes lenguajes de descripción de hardware especialmente VHDL. Se comprendió que son los PLD's y se conoció sus características, aplicaciones y la programación de estos, se comprobó el rendimiento tanto en capacidad como velocidad de los dispositivos lógicos programables, utilizando componentes individuales en lugar de un gran número de componentes, también se utilizó el lenguaje de programación VHDL y comprobamos la gran capacidad de configuración y amplia utilidad.

Al definir y analizar el concepto de PLD, se entendió que son, con el fin de comprobar su funcionamiento, arquitectura y sus características.
Los PLD's tienen un gran campo de aplicación, se utilizan en casi todos los nuevos equipos electrónicos de control, industriales, de consumo, de oficina, de comunicaciones, etc.

El campo de la electrónica se ha desarrollado grandemente en los últimos años, es así como nuevos procedimientos y tecnologías en el diseño de sistemas electrónicos se probaron para poder construir sistemas complejos con el menor riesgo de fallas.
La capacidad de los PLD's nos permitió implementar una gran variedad de diseños y aplicaciones de la lógica combinacional y secuencial utilizando el mismo dispositivo para diferentes aplicaciones ya que son reprogramables, de tal forma que siguiendo un proceso sencillo, y en poco tiempo, se puede obtener un dispositivo a la medida.

La combinación de los PLD's y los lenguajes descriptivo esto nos permite realizar la solución de ingeniería en una sola pastilla SOC, con las ventajas de utilizar menos espacios sobre la tarjeta, menores requerimientos de potencia, procesos de ensamble más rápidos y menos costo, mayor confiabilidad, dado que se tiene menos circuitos integrados, así como menos conexiones de circuito donde puedan ocurrir fallas,

procedimientos de detección de problemas más sencillos y pos ultimo la protección del desarrollo.

Se analizaron las ventajas y desventajas del uso de los PLD’s en la actualidad, esto con el fin de ver que dispositivo es más eficiente y tiene mejor rendimiento en el mercado.
Se logro investigar las aplicaciones de los PLD’s en la industria a nivel mundial, con la finalidad de saber cuales son los dispositivos lógicos programables que más se utilizan y por que se utilizan determinado tipo de dispositivo, también que compañías lanzan al mercado su mejor dispositivo, con esto comparamos cual es mejor y cuales se recomiendan para que la industria utilice.

El ahorro de tiempo en la elaboración de aplicaciones realizadas es bastante notorio, siempre y cuando se tenga un conocimiento previo del lenguaje de descripción de hardware que se utiliza y fundamento teóricos de dispositivos lógicos programables.

Como he comentado a través de la tesis los lenguajes descriptivos son técnicas de diseño digital que presentan grandes ventajas sobre las tradicionales, como: la rapidez de diseño, la facilidad de corregir errores, el de poder implementar en un solo circuito la aplicación además de la facilidad con que se aprende el lenguajes VHDL, al conocer la sintaxis y riqueza del lenguaje VHDL para la programación de los dispositivos nos dimos cuenta que nos facilita el trabajo de la programación.

Se logro utilizar los SPLD’s que son dispositivos mas pequeños que se aplican a circuitos lógicos simples y de bajo costo y además también los CPLD’s fueron utilizados en los diferentes ejercicios, por su parte implementan mas eficientemente diseños con parte combinacional abundante a un mediano costo.
En cambio, los FPGA’s son utilizadas para diseños que manejan mayor transferencia de datos y registros; pudiéndose implementar dentro de ellas memorias del tipo RAM, ROM, etc., a través de sus pequeños bloques de memoria.

La elección de la familia a utilizar dentro de un SPLD, CPLD o FPGA; dependió de la cantidad de lógica combinacional (compuertas), la cantidad de lógica secuencial (celdas lógicas y macroceldas) y en caso de FPGA’s, la cantidad de memoria a implementar.

Los dispositivos lógicos programables de última generación presentan un muy buen rendimiento tanto en capacidad como velocidad,

También un factor a tener en cuenta son los tiempos de propagación internos, como así también la cantidad de pines del dispositivo, con esto pudimos optimizar tiempos en todos los ejercicios.

BIBLIOGRAFÍA

MAXINEZ G., David y Jessica Alcalá Jara, ***VHDL, El arte de programar sistemas digitales,*** 2ª ed., Compañía Editorial Continental, México,2004, 350 pp.

NAGLE TROY H., ***Análisis y diseño de circuitos lógicos digitales,*** 4ª ed., Editorial Prentice Hall, 285 pp.

PARDO y Boluda, ***VHDL, El lenguaje para síntesis y modelado de circuitos,*** 2ª ed., Editorial Alfaomega, 120 pp.

T. L. Floyd, ***Fundamentos de sistemas digitales,*** 2ª ed., Editorial Prentice Hall, México 2005, 540 pp.

OTRAS FUENTES

http://www.ate.uniovi.es/fernando/Doc2003/EI/BAC_PLDs%20(3).pdf

http://www.ate.uniovi.es/fernando/Doc2003/EI/Introduccion%20(1).pdf

http://www.ate.uniovi.es/fernando/Doc2003/EI/Lenguaje%20VHDL.pdf

http://www.ate.uniovi.es/fernando/Doc2003/EI/Seleccion_PLDs%20(4).pdf

http://lapham25.tripod.com/archivos/plds.pdf

http://www.uag.mx/214/II_DISPOSITIVOS_LOGICOS_PROGRAMABLES.pdf

http://www.uag.mx/214/INDICE_ANTECEDENTES.pdf.pdf

http://www.gte.us.es/~leopoldo/Store/cdoctorado_t3.pdf

http://www.monografias.com/trabajos18/memorias-programables/memorias-programables.html

http://www.virtual.unal.edu.co/cursos/ingenieria/2000477/lecciones/040101.htm

http://www.virtual.unal.edu.co/PLD.htm

http://www.ilustrados.com/diagramas/ingenieria.pdf

http://www.lattice.com/memorias/ing.pdf

http://www.rincon delvago/dispositivoslogicosprogramables.htm

http://www.ilustrados.com/dispositivos programables/trabajo1.pdf

http:// www.latticesemi.com/gal/memorias.pdf

http:// www.latticesemi.com
http:// www.ilustrados.com
http:// www.logicaldevice.com
http:// www.monografias.com/robotica.shtml
http://www.monografias.com/semaforos.html
http://www.gte.us.es/creacióndedispositivos.pdf
http://www.ElectronicosOnline.com
http://www.lapham25.tripod.com/archivos/plds.pdf
http://www.cypress.com
http://www.monografias.com
http://www.monografias.com/sumador.html
http://www.ilustrados.com/compuertas/tutorial.pdf
http://www.ilustrados.com/circuitos_secuenciales/secuenciales.pdf
http://www.monografias.com/trabajos18/comparadores.html
http://www.monografias.com/trabajos18/contadores.html
http://www.monografias.com/unidad_logica.pdf
http://www.xicor.com