UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA
Y ELECTRICA
DIVISION ESTUDIOS DE POSTGRADO

DISEÑO DE SISTEMAS DIGITALES CON LOGICA PROGRAMABLE

POR

ING. ELIZABETH GUADALUPE LARA HERNANDEZ

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS
DE LA INGENIERIA ELECTRICA CON
ESPECIALIDAD EN CONTROL

SAN NICOLAS DE LOS GARZA, N. L.

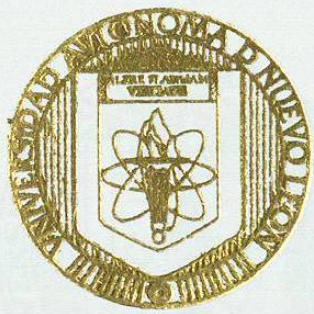
DICIEMBRE 2002

DISEÑO DE SISTEMAS DIGITALES CON LOGICA PROGRAMABLE

E. G. L. H.


1020149162

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA
Y ELECTRICA
DIVISION ESTUDIOS DE POSTGRADO



DISEÑO DE SISTEMAS DIGITALES CON LOGICA PROGRAMABLE

POR

ING. ELIZABETH GUADALUPE LARA HERNANDEZ

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS
DE LA INGENIERIA ELECTRICA CON
ESPECIALIDAD EN CONTROL

SAN NICOLAS DE LOS GARZA, N. L.

DICIEMBRE 2002

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

DIVISIÓN DE ESTUDIOS DE POST-GRADO

DISEÑO DE SISTEMAS DIGITALES CON LÓGICA PROGRAMABLE

POR

ING. ELIZABETH GUADALUPE LARA HERNÁNDEZ

TESIS

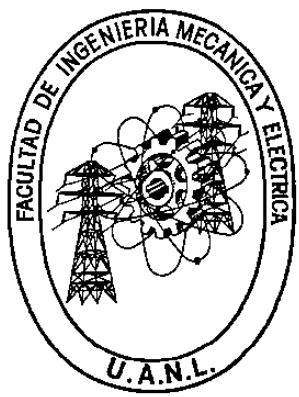
EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA CON ESPECIALIDAD EN CONTROL

SAN NICOLÁS DE LOS GARZA, N.L.
DICIEMBRE DE 2002

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

DIVISIÓN DE ESTUDIOS DE POST-GRADO



DISEÑO DE SISTEMAS DIGITALES CON LÓGICA PROGRAMABLE

POR

ING. ELIZABETH GUADALUPE LARA HERNÁNDEZ

TESIS

EN OPCIÓN AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA CON ESPECIALIDAD EN CONTROL

SAN NICOLÁS DE LOS GARZA, N.L.
DICIEMBRE DE 2002

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POST-GRADO

Los miembros del comité de tesis recomendamos que la tesis **"Diseño de Sistemas Digitales con Lógica Programable"** realizada por la Ing. Elizabeth Guadalupe Lara Hernández, matrícula 647306 sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Ingeniería Eléctrica con especialidad en Control.

El Comité de Tesis

M.C. Juan Ángel Garza Garza
Asesor

M.C. José Manuel Rocha Núñez
Coasesor

M.C. Guadalupe Ignacio Cantú Garza
Coasesor

Vo.Bo.
Dr. Guadalupe Alan Castillo Rodríguez
División de Estudios de Post-grado

San Nicolás de los Garza, N.L., Noviembre de 2002.

Dedico este trabajo de tesis

A Dios

Por darme vida y permitirme cumplir una meta más en mi carrera profesional.

A mis padres Absalón Lara Vargas y Flora Hernández de Lara

Por su apoyo incondicional y los ánimos que me brindaron para seguir adelante con esta tesis

A mis hermanas Eisenia, Nayeli y Eunice

¡Gracias por todo su amor!

AGRADECIMIENTOS

Al M.C. Juan Ángel Garza Garza por asesorarme en esta tesis, por todo su apoyo incondicional y ánimos. ¡Ya soy Liz!

Al M.C. José Manuel Rocha Núñez por no dejarme caer y apoyarme cuando mas lo necesitaba, por sus palabras de ánimo y el valioso tiempo que me brindó, ¡gracias por todo!.

Al M.C. Guadalupe I. Cantú Garza por sus consejos, ánimos y todas las facilidades que me ofreció para la realización de este trabajo.

Al M.C. César A. Leal Chapa gracias por su tiempo y orientación

Al M.C. Antonio Rodríguez García gracias por sus consejos y ánimos.

A mis compañeros de trabajo, gracias por sus amables consejos y su apoyo.

Expreso mis agradecimientos a todas las personas que me ayudaron de alguna u otra forma y que me tendieron la mano para culminar este trabajo de tesis

PRÓLOGO

La electrónica y computación en estos últimos años han evolucionado rápidamente, por ello es necesario actualizarse continuamente, de modo que con el uso y aplicación de estos desarrollos es posible realizar diseños en menor tiempo y utilizando una menor cantidad de circuitos integrados en su implementación.

Por medio de los lenguajes de descripción de hardware es posible simplificar el diseño de sistemas digitales, aunado a la gran capacidad de los Dispositivos Lógicos Programables PLD's que permiten implementar una gran variedad de diseños y aplicaciones de la lógica combinacional y secuencial, utilizando el mismo dispositivo para diferentes aplicaciones ya que es reprogramable, de tal forma que siguiendo un proceso sencillo, y en poco tiempo, se puede obtener un dispositivo a la medida (ASIC Application Specific Integrated Circuits) que antes solo era posible fabricarlo en la industria utilizando maquinaria especializada.

El procedimiento de diseño propuesto en esta tesis resulta de gran utilidad en la enseñanza de los sistemas digitales gracias a que los lenguajes de descripción de hardware HDL's como VHDL (Lenguaje de Descripción Hardware de Circuitos Integrados de muy alta velocidad) y ABEL (Lenguaje Avanzado de Expresiones Booleanas) son poderosos y fáciles de utilizar, los PLD's son económicos y fáciles de conseguir, además el software necesario para el desarrollo se pueden encontrar licencias gratuitas en Internet..

Estos aspectos y más son tratados mas a detalle en esta tesis.

ÍNDICE

8. CONCLUSIONES Y RECOMENDACIONES

SÍNTESIS

El presente trabajo tiene por objetivo ofrecer una alternativa de actualización del laboratorio de Electrónica Lógica II de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León cumpliendo con las expectativas de la reforma académica de las carreras de Ingeniero en Electrónica y Comunicaciones e Ingeniero en Electrónica y Automatización, con esta propuesta se pretende beneficiar al maestro al actualizarse con una de las tantas herramientas nuevas que existen en el mercado y a la vez la parte más importante de esta institución, "el alumno".

A consecuencia de la reforma académica se disminuyeron las horas de clases presénciales en un 40% y en laboratorios en un 50% con el fin de que el alumno además de adquirir conocimientos desarrolle habilidades, actitudes y valores, es decir, ampliar los espacios de aprendizaje en donde se espera que fuera del salón de clase o laboratorio el alumno siga aprendiendo, por tal razón esta tesis propone en dicho laboratorio el uso de nuevas herramientas de diseño digital como Lenguajes de Descripción de Hardware (HDLs) y Programas de Captura Esquemática implementados en Dispositivos Lógicos Programables.

Las habilidades que se esperan desarrollar en el alumno son por ejemplo el uso de Software de diseño digital, programadores de circuitos integrados, manejo de la computadora, consulta de manuales a través de Internet y el autoaprendizaje.

La metodología propuesta en esta tesis contempla el diseño por medio de bloques funcionales utilizando Dispositivos Lógicos Programables así como de circuitos integrados de función fija que actualmente se ven en el laboratorio, estos tipos de dispositivos no deben de excluirse ya en que en algunos casos es conveniente utilizarlos porque resultaría más económico que implementarlos con dispositivos lógicos programables.

Una observación importante para que esta metodología propuesta funcione es que hay que proporcionarles a los alumnos los medios para trabajar, como por ejemplo información (manuales, guía de practicas), espacios equipados con computadoras y programadores.

En esta tesis se hace una reseña del procedimiento con el que se trabaja actualmente en el laboratorio de Electrónica Lógica II y las prácticas que se desarrollan. En este laboratorio existe una carencia de equipo y material, algunas prácticas no se pueden realizar porque ya no se encuentra en el mercado los circuitos que se requieren para llevarlas a cabo.

En el presente trabajo encontrará una introducción al diseño digital, mostrando una comparación del diseño tradicional función fija contra el diseño con circuitos programables.

En esta comparación se presenta el procedimiento que se sigue en cada uno de estos diseños, mostrando sus ventajas y desventajas. Esto nos da un panorama de la importancia que tiene el diseño con circuitos programables y todos los beneficios que trae por consecuencia, pudiendo así concluir que el diseño programable es en la mayoría de los casos más beneficioso que el diseño tradicional de función fija, por las ventajas que ofrece las cuales son:

Cuando se trabaja con diseño tradicional función fija tenemos la desventaja que para llevar a cabo un proyecto en la mayoría de los casos es necesario interconectar varios circuitos integrados de función fija, la dificultad mayor que se presenta en el proyecto es en el alambrado y esto es porque se tiene que llegar necesariamente a la implementación física del proyecto para verificar si el funcionamiento de este cumple con las expectativas inicialmente planteadas.

En esta propuesta que se hace con diseño con lógica programable se tiene la ventaja de poco alambrado ya que se utiliza en la mayoría de los casos un solo circuito integrado y antes de llegar a la implementación física se puede efectuar una simulación en la computadora porque gracias a los lenguajes de descripción de hardware (HDLs) se puede realizar una simulación y en caso de que no se cumplan los requisitos se puede detectar la falla y corregirla fácilmente y comprobar así si se cumplen los requerimientos del diseño y de ser correcto se prosigue a implementarlo

En la presente tesis se hace una descripción de algunos dispositivos lógicos programables, presentando sus estructuras, características y diagramas lógicos de algunos de ellos.

Algunos de los dispositivos lógicos que se mencionan son, el PAL (Arreglo Lógico Programable), GALs (Arreglos Lógicos Genéricos) y FPGAs (Arreglos de Compuertas Programables en Campo), FPLA (Arreglo Lógico Programable en Campo), por mencionar algunos.

Encontrará también un apartado dedicado a captura esquemática. La captura esquemática se desarrolla a través de un programa en donde se realiza un diagrama de conexiones, con componentes y símbolos necesarios para realizar sus interconexiones y construir así su esquemático requerido.

Los componentes pueden ser compuertas, Flip Flops, memorias, bloques funcionales como comparadores, multiplexores y los símbolos como conectores, etiquetas, puertos de entrada/salida y algunas definiciones importantes que hay que conocer para poder desarrollar un esquemático.

Se describe el procedimiento para llevar a cabo la captura esquemática, concluyendo con el enlace y síntesis que se tiene que realizar previamente para llegar al archivo final que nos servirá para la programación del dispositivo programable a emplear en el proyecto.

Hay un capítulo destinado a Lenguaje de Descripción de Hardware (HDLs) donde se presentan las características importantes de él, resaltando los lenguajes mas usados actualmente como lo son el lenguaje ABEL (Lenguaje Avanzado de Expresiones Booleanas) y VHDL (Lenguaje de Descripción Hardware de Circuitos Integrados de muy alta velocidad).

En este apartado se muestran puntos importantes para desarrollar un programa en estos lenguajes, hasta llegar al archivo final que servirá para realizar la programación del dispositivo lógico programable a utilizar.

Con el planteamiento de estos temas Lenguaje de Descripción de Hardware y Captura Esquemática que se presentan en esta trabajo de tesis podemos hacer una observación importante que es la siguiente: No es conveniente utilizar Captura Esquemática en proyectos muy elaborados ya que resulta imposible trabajar con muchos componentes y conexiones, para este caso es recomendable utilizar los Lenguajes de Descripción de Hardware (HDLs).

1. INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA

En el laboratorio de Electrónica Lógica II de la Facultad de Ingeniería Mecánica y Eléctrica tiene prácticas que fueron diseñadas hace 25 años con circuitos integrados de función fija de mediana y gran escala de integración. En la actualidad existen nuevas herramientas para el desarrollo de sistemas digitales, los cuales utilizan dispositivos pequeños en tamaño, pero potentes y de fácil programación, estas herramientas no están disponibles en este laboratorio y por consecuencia los alumnos egresados no están siendo competitivos en el mercado laboral.

1.2 OBJETIVO DE LA TESIS

Proponer una herramienta con tecnología actual para el laboratorio de electrónica lógica II para que el alumno adquiera los conocimientos en el diseño de sistemas digitales y desarrolle habilidades en la implementación de estos circuitos.

1.3 HIPÓTESIS

Mi supuesto es que es posible encontrar los elementos para el desarrollo de estas prácticas que estén al alcance de las posibilidades físicas y económicas de la facultad y de los alumnos.

1.4 LÍMITES DEL ESTUDIO

Este trabajo está enfocado a los alumnos del Laboratorio de Electrónica Lógica II de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León en especial a los alumnos de las carreras de Ingeniero en Electrónica y Comunicaciones, Ingeniero en Electrónica y Automatización e Ingeniero Administrador de Sistemas.

1.5 JUSTIFICACIÓN DEL ESTUDIO

El manejo de herramientas para el diseño de sistemas digitales forma una parte importante del área de control.

El desarrollo de esta propuesta permitirá a maestros y estudiantes conocer y aplicar las nuevas herramientas para el desarrollo de sistemas digitales las cuales hará que los estudiantes tengan el nivel adecuado para enfrentar su vida profesional.

Actualizarme en nuevas herramientas para el diseño de sistemas digitales los cuales me permitirán hacer diseños más ambiciosos que utilizando las herramientas que actualmente se ven. Además al desarrollar esta tesis me permitirá compartir estos conocimientos tanto a docentes como para las nuevas generaciones futuras.

1.6 METODOLOGÍA

1. Investigación de los elementos en el mercado.
2. La selección de los posibles elementos que se adecuen a las características del proyecto.
3. Probar la factibilidad del diseño con los elementos seleccionados.
4. Elaborar una metodología para el diseño con éstos dispositivos.
5. Comprobar la metodología con diseños propuestos.
6. Evaluación de los diseños.
7. Propuesta definitiva.

1.7 REVISIÓN BIBLIOGRÁFICA

Diseño Digital, este es un libro muy completo, me sirvió de apoyo para familiarizarme con el tema de dispositivos lógicos programables.

Lógica Programable, este libro principalmente maneja ejercicios resueltos, aunque contiene las características de algunos de los dispositivos presentados en esta tesis, no presenta la programación de los dispositivos programables con lenguaje de descripción de hardware y captura esquemática.

VHDL Lenguaje para síntesis y modelado de circuitos, este libro se enfoca al lenguaje de descripción de hardware como su título bien lo dice, en esta tesis hay un capítulo destinado a los HDLs en donde se mencionan el VHDL y ABEL, este libro me sirvió porque tiene lo necesario para comprender el lenguaje VHDL.

Xilinx INC. Programmable Logic Data Book, éste es un manual de datos y presenta las características importantes de algunos dispositivos tratados en esta tesis, como la familia XC400E y XC400X de los FPGAs.

2. ANTECEDENTES

2.1 EVOLUCIÓN O HISTORIA DE LOS CIRCUITOS INTEGRADOS

La introducción de los tubos de vacío a comienzos del siglo XX propició el rápido crecimiento de la electrónica moderna. Con estos dispositivos se hizo posible la manipulación de señales, algo que no podía realizarse en los antiguos circuitos telegráficos y telefónicos, ni con los primeros transmisores que utilizaban chispas de alta tensión para generar ondas de radio. Por ejemplo, con los tubos de vacío se pudieron amplificar las señales de radio y de sonido débiles, y además podían superponerse señales de sonido a las ondas de radio. El desarrollo de una amplia variedad de tubos, diseñados para funciones especializadas, posibilitó el rápido avance de la tecnología de comunicación radial antes de la Segunda Guerra Mundial, y el desarrollo de las primeras computadoras, durante la guerra y poco después de ella.

Hoy día, el transistor, inventado en 1948, ha reemplazado casi completamente al tubo de vacío en la mayoría de sus aplicaciones. Al incorporar un conjunto de materiales

semiconductores y contactos eléctricos, el transistor permite las mismas funciones que el tubo de vacío, pero con un costo, peso y potencia más bajos, y una mayor fiabilidad. Los progresos subsiguientes en la tecnología de semiconductores, atribuible en parte a la intensidad de las investigaciones asociadas con la iniciativa de exploración del espacio, llevó al desarrollo, en la década de 1970, del circuito integrado.

El circuito integrado permitió una posterior reducción del precio, el tamaño y los porcentajes de error. El microprocesador se convirtió en una realidad a mediados de la década de 1970, con la introducción del circuito de integración a gran escala (LSI, acrónimo de gran escala de integración) y, más tarde, con el circuito de integración a mayor escala (VLSI, acrónimo de integración a muy grande escala), con varios miles de transistores interconectados soldados sobre un único sustrato de silicio.

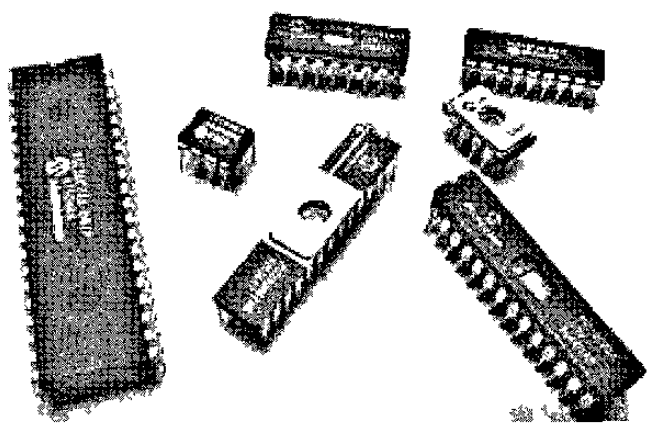


Figura 2.1 Circuitos Integrados

Los circuitos integrados han hecho posible la fabricación del microordenador o microcomputadora. Sin ellos, los circuitos individuales y sus componentes ocuparían demasiado espacio como para poder conseguir un diseño compacto. También llamado chip, un circuito integrado típico consta de varios elementos como reóstatos, condensadores y transistores integrados en una única pieza de silicio.

Para las realizaciones muy complejas que exigirían un número elevado de CI de función fija, se utilizan circuitos diseñados a medida que sólo sirven para una aplicación. Son los llamados CI específicos a una aplicación o ASICs (Application Specific

Integrated Circuit) en español circuito integrado de aplicación específica. Por regla general, los ASICs los producen los fabricantes de CI con las especificaciones proporcionadas por el usuario.

Los equipos realizados con ASICs ocupan menos espacio, son más fiables, consumen menos energía y en grandes series resultan más baratos que los equipos equivalentes realizados con CI de función fija. Por otro lado, estos circuitos son muy difíciles de copiar.

2.2 EVOLUCIÓN O HISTORIA DEL LABORATORIO DE ELECTRÓNICA LÓGICA II

Desde sus inicios el laboratorio de electrónica lógica II no ha sufrido muchos cambios, la última revisión que se hizo fue aproximadamente hace unos diez años y sólo se agregaron unas prácticas. Las prácticas que se están dando actualmente son con circuitos integrados de función fija de mediana y gran escala de integración y algunos de los dispositivos que se utilizan en ellas ya no se encuentran el mercado. La tecnología que se utilizaba para diseñar con estos circuitos integrados ya se encuentra obsoleta, es muy compleja en su diseño, e implementación de los sistemas ya que se utilizaban una gran cantidad de componentes, aumentando así su costo, además el mantenimiento no era tan sencillo por la cantidad de componentes utilizados.

Las prácticas que actualmente se ven en este laboratorio son las siguientes:

Práctica 1

Medición de los parámetros eléctricos

El objetivo de la práctica:

El objetivo de esta práctica es el de conocer las características eléctricas y tiempos de propagación de las familias lógicas T.T.L. (Transistor Transistor Logic) en español enlace de transistor a transistor lógico y CMOS (Complememt Metal Oxide Semiconductor) en español semiconductor de óxido metálico en inglés en particular de SN7404, SN7414 y 4049.

Práctica 2

Medición de los parámetros eléctricos

Objetivo de la práctica:

El objetivo de esta práctica es el de conocer y diferenciar los tipos de salida que existen en los circuitos digitales tales como, TOTEM-POLE, OPEN COLLECTOR Y 3 STATE. Utilizando para ello los siguientes componentes SN7404, SN7426 y SN74240.

Práctica 3

MSI Combinacional

Objetivo de la práctica:

El objetivo de esta práctica es el de verificar la operación de los bloques combinacionales de mediana escala de integración más comunes y desarrollar algunas aplicaciones típicas, en esta práctica se probará el funcionamiento del MULTIPLEXER. Para esta práctica se trabaja con el 74153 y TW's.

Práctica 4

MSI Combinacional

Objetivo de la práctica:

El objetivo de esta práctica es el de verificar la operación del DECODER-DEMULTIPLEXER, que es un circuito combinacional de mediana escala de integración (MSI), utilizando componentes como el 74156, 7404 y TW's con diodos.

Práctica 5

MSI Combinacional

Objetivo de la práctica:

El objetivo de esta práctica es el de verificar la operación de los bloques combinacionales de mediana escala de integración más comunes y desarrollar algunas aplicaciones típicas, se comprobará la operación de los decodificadores de prioridad y se realizarán algunas aplicaciones del decoder y multiplexer, utilizando los siguientes integrados 74147, 74148, 74153, 74156 y TW's.

Práctica 6

MSI Combinacional

Objetivo de la práctica:

La ALU (Arithmetic and Logic Unit) en español Unidad Lógica Aritmética es un bloque combinacional de MSI capaz de realizar operaciones aritméticas y lógicas tales como: suma, resta, comparación, uno y dos complemento, and, or, nand, nor, exor, etc.

Este bloque multifuncional es la parte más importante de la unidad central de proceso de un computador, actualmente la unidad aritmética y lógica esta contenida en un bloque de mayor complejidad.

Pensar en el diseño de un procesador partiendo de la Unidad Aritmética y Lógica, es ya innecesario, sin embargo la importancia de esta práctica radica en tener acceso directo al bloque ya que en un microprocesador el acceso sería vía programación. Para ello se utilizan los integrados 74181,7404 y TW's.

Práctica 7

MSI Combinacional

Objetivo de la práctica:

El objetivo de la práctica es el de verificar la operación de un contador síncrono programable y desarrollar algunas aplicaciones típicas, utilizando los siguientes integrados 74190,7404 y TW's.

Práctica 8

RAM – Memoria de lectura / escritura

Objetivo de la práctica:

El objetivo de esta práctica es el de familiarizarse con el uso de la memoria RAM, comprobar la operación de direccionamiento, lectura, escritura, memoria volátil, habilitación y deshabilitación del bloque así como efectuar una expansión y desarrollar una aplicación, utilizando los integrados SN7489 y TW's.

Práctica 9

Memoria de solo lectura EPROM

Objetivo de la práctica:

El objetivo de esta práctica es el de verificar la operación de una memoria no volátil y de solo lectura EPROM (Erasable Programmable Read Only Memory) en español Memoria de solo lectura programable y borrable, utilizando los integrados 2764, 7404 y TW's.

Práctica 10

Encoder de teclado

Objetivo de la práctica:

El objetivo de esta práctica es el de comprobar el funcionamiento de circuitos integrados de aplicación específica más complejos, tal como lo es el caso del 74C922, que es un circuito encodificador (lector) de teclado de 16 teclas, utilizando los siguientes integrados 74C922 y 7404

Práctica 11

LSI Contador multidígito

Objetivo de la práctica:

El objetivo de la práctica es el de probar un circuito integrado de Gran Escala de Integración (LSI), se recomienda utilizar el ICM7217A, que es un contador que tiene la capacidad de manejar 4 display multidígito, leer 4 TW's para programar el conteo y contar en ascendente – descendente.

3. INTRODUCCIÓN AL DISEÑO DIGITAL

3.1 DISEÑO TRADICIONAL FUNCIÓN FIJA

En el diseño tradicional función fija se trabaja como su nombre lo dice con dispositivos o circuitos integrados de función fija.

Para poder desarrollar un proyecto con este tipo de diseño tenemos que conocer el funcionamiento de algunos dispositivos y determinar cual o cuales cumplen los requerimientos para llevar a cabo el proyecto deseado, esto implica que en el diseño en la mayoría de los casos se utilicen mas de un componente con lo cual nos lleva a que el proyecto ocupe físicamente un mayor espacio que si utilizáramos un solo componente.

En este tipo de diseño tenemos que llegar necesariamente a la implementación para poder verificar el funcionamiento correcto del proyecto, si no se tiene éxito tenemos que regresar al diseño y revisar las conexiones necesarias o en su defecto volver a revisar paso

a paso el procedimiento hasta llegar de nuevo a la implementación. Este paso se repetiría cuantas veces sea necesario hasta que el proyecto efectúe el funcionamiento correcto.

De alguna manera en este tipo de diseño se tiene que tener muy en cuenta una cosa y es la forma de optimizar el proyecto, cuando se plantea el proyecto se aplica algún método para resolverlo y ya que se llegó al resultado se tiene que revisar de nuevo para ver si no hay variables innecesarias que hagan que se involucren dispositivos que pueden estar de mas en el proyecto.

3.2 DISEÑO CON CIRCUITOS PROGRAMABLES

En este tipo de diseño con circuitos programables se trabaja necesariamente con una computadora con el software necesario y un programador.

Con la ayuda del software contamos con una herramienta poderosa podríamos llamarlo así, ya que no es necesario llegar hasta la implementación del circuito para verificar el funcionamiento de este como lo sería en el caso de diseño tradicional, si la simulación no es correcta se tiene que revisar de nueva cuenta el procedimiento y solución del problema.

Con este diseño se plantea el problema y existen varias formas de cómo atacarlo con la ayuda del software y estas son tablas de verdad, diagramas de estados, esquemáticos, etc..

Se efectúa el diseño conveniente para posteriormente realizar una simulación del diseño para verificar su funcionamiento, con este diseño programable no necesitamos ver que dispositivos nos convienen mas para llevar a cabo el proyecto y otra cosa importante es que no tenemos que preocuparnos por optimizar el diseño.

El circuito integrado se programa ya que la simulación esté correcta.

Un mismo chip se puede reprogramar y utilizarlo en otros proyectos.

3.3 TRADICIONAL VS PROGRAMABLE

En seguida se presentan las ventajas y desventajas del método tradicional y programable.

Ventajas del diseño tradicional.

1. No se necesita computadora ni programador y esto implica menor capacitación a los diseñadores

Desventajas del diseño tradicional.

1. Se necesita optimizar el proyecto.
2. En la mayoría de los casos involucran mas de un integrado en el diseño y esto implica mas costo.
3. Se necesita llegar a la implementación del proyecto para verificar su funcionamiento.
4. Implica mayor tiempo ya que se requiere hacer una optimización del proyecto.
5. La implementación involucra mas tiempo ya que se requieren más componentes que en el diseño programable.

Ventajas del diseño Programable.

1. Se ocupan menos dispositivos que en el diseño tradicional esto implica menos costo.
2. Menos tiempo al implementar el circuito porque se ocupan menos componentes.
3. Se tiene la ventaja importante de que los diseños se pueden simular por la utilización del software.
4. En el diseño la optimización es realizada por la computadora en poco tiempo.
5. En rediseños futuros se puede utilizar el mismo componente ya que es reprogramable.

Desventajas del diseño Programable.

1. Se requiere una computadora con el software necesario y un programador esto implica mayor costo.
2. Capacitación a los diseñadores en el uso de computadoras y en el desarrollo de lenguajes de programación.

4. DISPOSITIVOS LÓGICOS PROGRAMABLES

4.1 CLASIFICACIÓN GENERAL

Existen en la actualidad unas 300 arquitecturas diferentes de dispositivos PLDs (Programmable Logic Device) en español dispositivos lógicos programables y su número se incrementa día a día.

Ya que generalmente los dispositivos PLDs disponen de muchas entradas y resultaría muy complicado mostrarlas en un dibujo, se utiliza una representación simplificada según la cual para las compuertas 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 compuerta AND, en realidad esta compuerta tiene tantas entradas como intersecciones de la línea producto. Si en una intersección hay una

X, significa que el fusible está intacto; si no hay una X, el fusible está fundido y no existe la conexión. En ocasiones, las compuertas OR también se dibujan con una sola entrada.

En el diagrama simplificado de la siguiente figura 4.1 aparece una matriz de compuertas AND de las seis entradas, cuyas salidas están conectadas a una compuerta OR. La intersección de las líneas producto con las líneas de entrada forman una matriz de compuertas AND programable de 6 x 3 fusibles. El circuito está programado para realizar la función OR –exclusiva entre las entradas A y B --. La compuerta 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 compuerta AND asociada es 1.

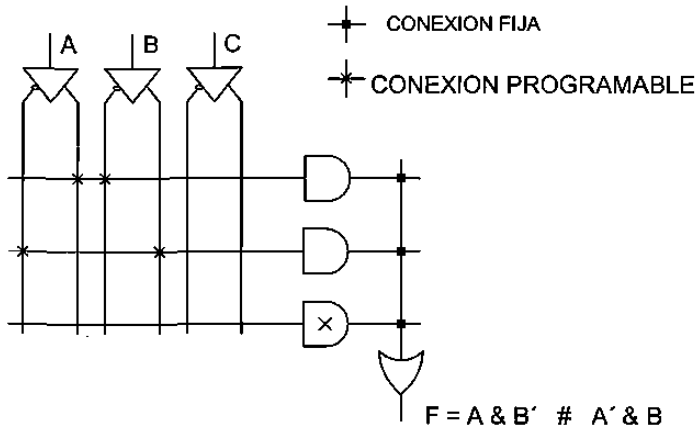


Figura 4.1 Representación simplificada de una función

4.1.1 PAL

PAL (Programmable Array Logic) en español Arreglo lógico programable

Es un PLD en el que se pueden programar las uniones en la matriz de compuertas AND, siendo fijas las uniones en la matriz de compuertas OR (véase la sig. Figura 4.2). Los dispositivos con arquitectura PAL son los más populares y los más utilizados.

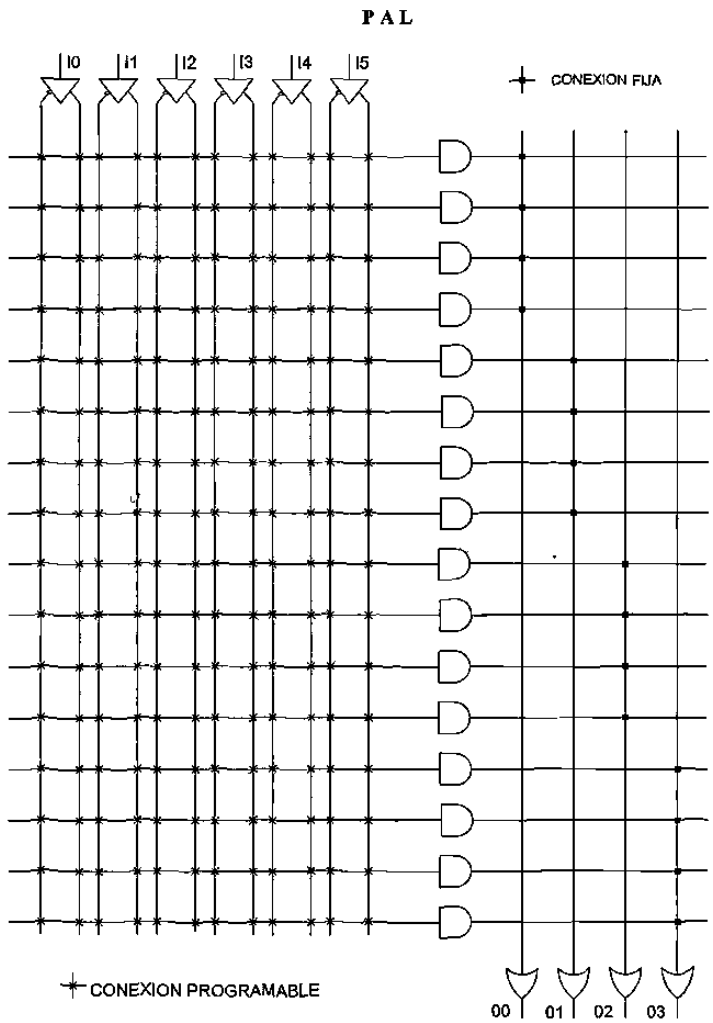


Figura 4.2 Estructura de una PAL

4.1.2 FPLA

FPLA (Field Programmable Logic Array) en español Arreglo lógico programable en campo.

Es un PLD en el que se pueden programar las uniones en ambas matrices (véase la siguiente figura 4.3) Son dispositivos más flexibles, pero resultan penalizados en tamaño y en velocidad debido a los transistores adicionales en la matriz de compuertas OR. Se utilizan fundamentalmente para construir máquinas de estados. Para otras aplicaciones, las PALs resultan más efectivas. Las PALs y las FPLA son sistemas combinacionales incompletos porque teniendo n entradas, disponen de menos de 2^n términos producto.

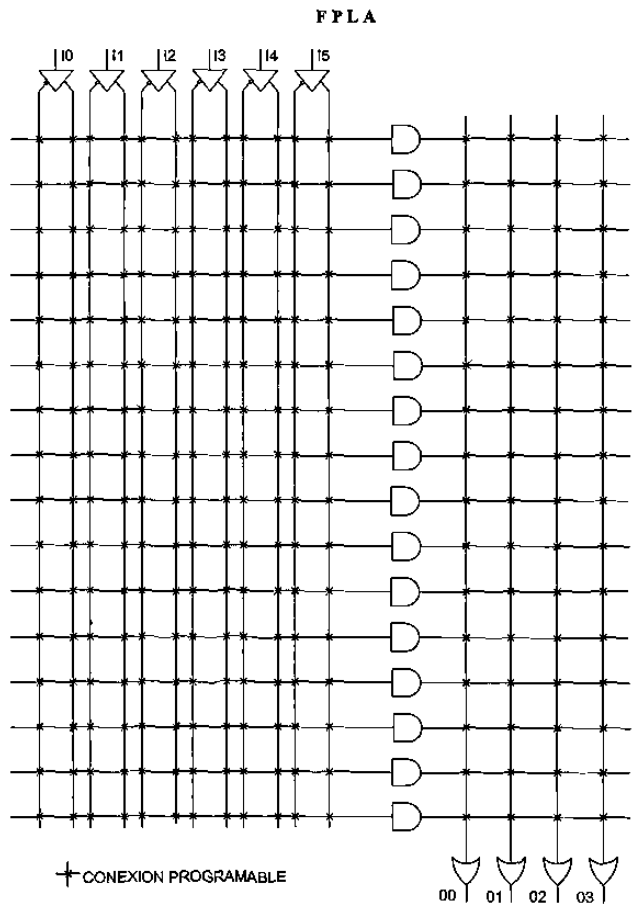


Figura 4.3 Estructura de una FPLA

4.1.3 PROM

PROM (Programmable Read Only Memory) en español memoria de solo lectura programable.

Es un dispositivo lógico programable (PLD) en el que las uniones en la matriz de compuertas AND es fija, siendo programables las uniones en la matriz de compuertas OR (véase la figura 4.4). Una PROM es un sistema combinacional completo que permite realizar cualquier función lógica con las n variables de entrada, ya que dispone de 2^n términos producto. Están muy bien adaptas para aplicaciones tales como: tablas, generadores de caracteres, convertidores de códigos, etc. Generalmente las PROM tienen menos entradas que las PAL y FPLA. Se pueden encontrar PROM con capacidades de potencia de 2, que van desde las 32 hasta las 8192 palabras de 4, 8 o 16 bits de ancho.

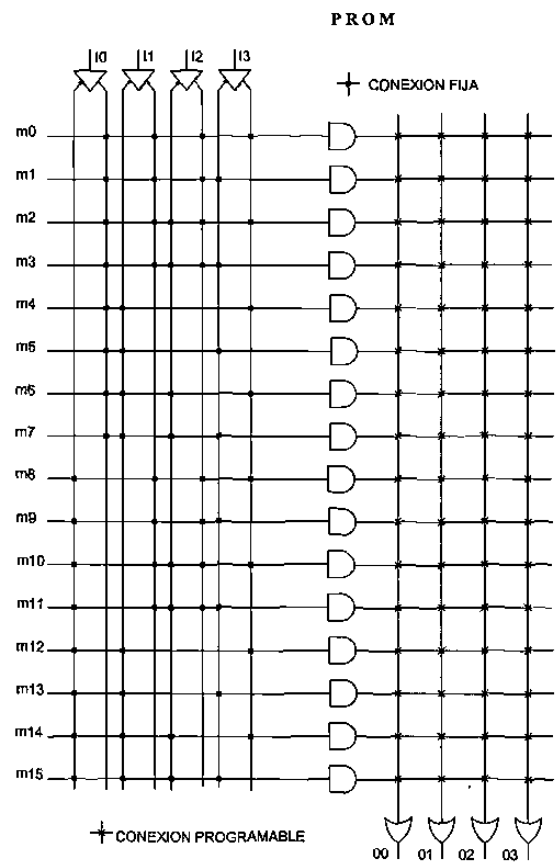


Figura 4.4 Estructura de una PROM

4.1.4 TIPOS DE PAL

4.1.4.1 SERIE GENERAL DE 20 TERMINALES

Es la serie más utilizada. Muchos de sus componentes fueron creados por MMI (Monolithic Memories Incorporated) en español Memoria Monolítica Incorporada. Actualmente estos dispositivos pueden adquirirse de varios fabricantes. Entre estos populares dispositivos lógicos programables se encuentran las PAL16L8, PAL16H8, PAL16P8, PAL16R4, PAL16R6 y PAL16R8.

En la figura 4.5 podemos ver la PAL16L8 provista de ocho salidas, correspondientes cada una de ellas a la salida de una compuerta OR de siete entradas. A cada compuerta OR, llegan las salidas de siete compuertas AND que realizan productos entre las terminales de entrada y las salidas provistas de realimentación. Asociado a cada salida hay una inversor de tres estados cuya habilitación depende de la salida de otra compuerta AND. Las PALs de tipo combinacional que no poseen el inversor en la salida llevan una H en su nombre y se utilizan con menor frecuencia que las PAL provistas de inversor; estas últimas llevan una L en su nombre. Véase el esquema de la PAL16H8 en la figura 4.6.

Se han diseñado PALs como la PAL16P8 que tienen la salida con polaridad programable mediante una compuerta XOR. Este tipo de PAL puede utilizarse como una PAL16L8 o una PAL16H8.

Con objeto de poder realizar circuitos secuenciales, se diseñaron las PAL16R4, PAL16R6 y PAL16R8 con flip-flops tipo D. Los registros de estas PALs se controlan con una señal de reloj común a todos ellos. La habilitación de las salidas de los registros se realizan con otra terminal común a todos los registros llamada (output enable) habilitador de salida. Las salidas provistas de registros no pueden utilizarse como salidas combinacionales. Dentro de esta serie hay dispositivos con registros y con salida de

polaridad programable que llevan las letras RP en su nombre. Así podemos encontrar los dispositivos PAL16RP4, PAL16RP6 y PAL16RP8.

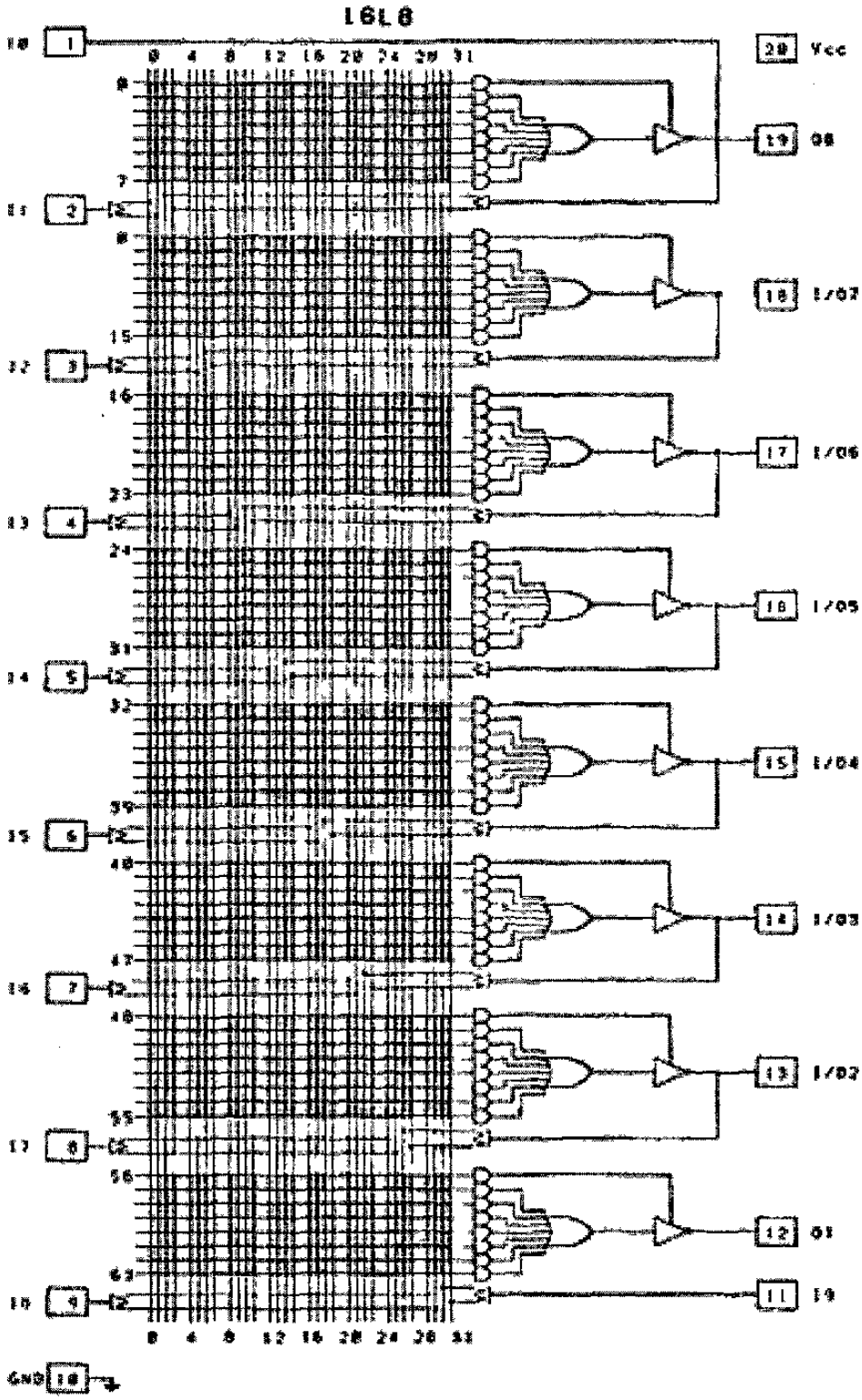


Figura 4.5 Diagrama lógico de la PAL16L8

4.1.4.2 SERIE ESPECIAL DE 20 TERMINALES.

Se agrupan en esta serie a las PALs de 20 terminales con baja densidad de fusibles. Suelen tener disponibles muy pocos términos producto por cada salida y además el número de términos producto varía de salida a salida. En muchos casos no disponen de realimentación. Por los motivos expuestos, estas pequeñas PALs están perdiendo popularidad con gran rapidez.

Algunos de los elementos pertenecientes a esta serie son: PAL10L8, PAL10H8, PAL10P8, PAL12L6, PAL12H6, PAL12P6, PAL14L4, PAL14H4, PAL14P4, PAL16C1, PAL16L2, PAL16H2 y PAL16P2.

4.1.4.3 SERIE GENERAL DE 24 TERMINALES.

Se trata de un conjunto de PALs que disponen habitualmente de ocho términos productos por cada salida. Hay PALs combinacionales y PALs con registros. Las salidas provistas de registros no pueden utilizarse como salidas combinaciones, al igual que sucedían con la serie general de 20 terminales.

Elementos pertenecientes a esta serie son: PAL20L8, PAL20H8, PAL20P8, PAL20R4, PAL20R6, PAL20R8, PAL20RP4, PAL20RP6, PAL20RP8 y las PALs provistas de 10 salidas como la PAL22P10 o la PAL20RP10.

También pertenecen a esta serie general de 24 terminales las PALR19L8, PALR19R4, PALR19R6, PALR19L8, PALT19R4, PALT19R6, PALT19R8. Las que empiezan por PALR19 tienen flip-flops tipo D en sus entradas, y las que empiezan por PALT19 tienen latches en sus entradas. No obstante, estos latches o flip-flops de las entradas pueden ser eliminados si no se desean.

4.1.4.4 SERIE ESPECIAL DE 24 TERMINALES.

Se agrupan en esta serie a las PALs dc 24 terminales con baja densidad de fusibles. Normalmente disponen de pocos términos producto por cada salida y muchas veces el número de productos varía de salida a salida. En muchos casos carecen de realimentación. Tal como ha sucedido con la serie especial de 20 terminales, estas PALs no han tenido mucha aceptación.

Algunos de los elementos pertenecientes a esta serie son: PAL12L10, PAL12H10, PAL12P10, PAL14L8, PAL14H8, PAL16L6, PAL16H6, PAL16P6, PAL18L4, PAL18H4, PAL18P4, PAL20Ci, PAL20L2, PAL20H2, PAL20L10, PAL20H10, PAL20P10, PAL6L16 y PAL8L14.

4.1.4.5 SERIE GENERAL DE 28 TERMINALES.

Se trata de una serie constituida por PALs que habitualmente poseen ocho términos producto por cada salida. La mayoría de las salidas tienen realimentación a la matriz de compuertas AND. Poseen una arquitectura similar a las series generales de 20 y 24 terminales, pero disponen de más líneas de entradas y salida.

A esta serie pertenecen las PAL24L10, PAL24R10, PAL24R8, PAL24R6 y PAL24R4.

4.1.4.6 PALS GENÉRICAS O UNIVERSALES.

Son dispositivos con flip-flops conectados a las terminales de salidas que pueden utilizarse o pueden ser omitidos. Si se evitan los registros, se obtienen salidas combinacionales. Estas PALs son más flexibles que las vistas hasta ahora y permiten obtener el equivalente a una PAL16R7, con siete salidas con registros y una salida combinacional, por poner un ejemplo.

La flexibilidad de las PALs genéricas reside en que sus salidas disponen de una circuitería especial conocida como macrocélula, generalmente, consiste en un flip-flop y varios multiplexores cuyo funcionamiento lo define un conjunto de fusibles de configuración distintos a los encontrados en la matriz de compuertas AND.

A este grupo pertenecen la PAL18U8, la PAL22V10 y la PAL26V12.

La PAL18U8 posee 20 terminales: ocho de entrada en su lado izquierdo, ocho de entrada-salida en su lado derecho, alimentación, tierra, reloj y habilitador. Las terminales reloj y habilitador pueden utilizarse como entradas normales si no se usan los registros, disponiéndose en este caso de un total de 10 entradas. Cada una de las ocho terminales de entrada-salida tiene nueve términos producto, uno de ellos dedicado al habilitador del tres-estado y los otros ocho preparados para la lógica. Cada terminal de entrada-salida puede programarse para ser activa a nivel alto o bajo y para que sea combinacional o provista de registro. Tiene dos nodos internos para forzar a uno (preset) en forma síncrona o forzar a cero (reset) en forma asíncrona de los registros.

Los PAL18U8 tiene 24 fusibles especiales para la configuración de sus salidas; ocho fusibles para polaridad activa a nivel bajo (0) o activa a nivel alto (1), ocho fusibles para salida con registro (0) o salida combinacional (1) y ocho fusibles para habilitar con compuerta AND dedicada (0) o por medio de la terminal 11(1).

La PAL22V10 fue diseñada por AMD (Advanced Micro Devices) fabricante de circuitos integrados y de PLD's para reemplazar a todas las PALs de 24 terminales. Disponible de las 10 salidas equipadas con macromoléculas como la mostrada en la figura 4.6. Estas 10 terminales pueden servir como entrada o como salida. Si actúan como salidas, se puede lograr que sean activas a nivel alto o bajo, que sean salidas

combinacionales o salidas con registro. Al igual que la 18U8 tiene dos nodos internos para reset asíncrono y preset síncrono de los registros. Cada salida dispone de una compuerta AND para habilitación del tercer-estado y de un número de compuertas AND, otras dos salidas 10 compuertas, otras dos salidas tienen 12 compuertas AND además de la compuerta AND destinada a la habilitación del tercer-estado. Debido al elevado número de compuertas AND de que dispone, la PAL22V10 permite la realización de diseños complejos.

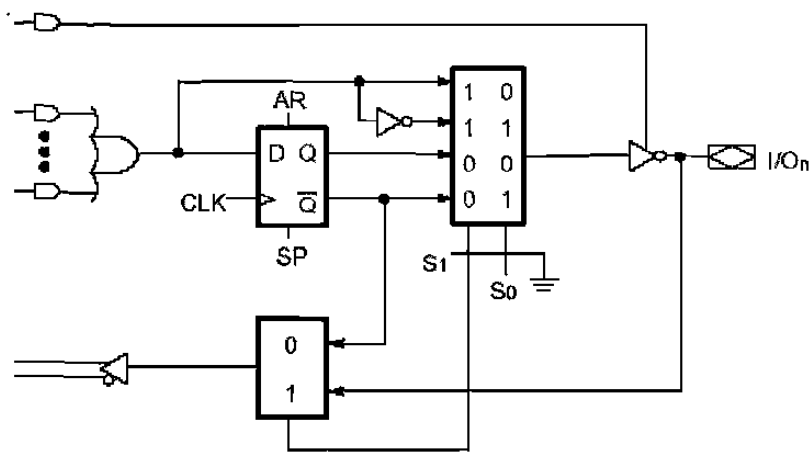


Figura 4.6 Macrocélula de la PAL22V10

Cada una de las 10 macromoléculas de la PAL22V10 tiene un flip-flop tipo D y dos multiplexores configurables mediante fusibles. El multiplexor de salida hace que la salida sea combinacional o provista de registro y que sea activa a nivel acto o bajo. El segundo multiplexor permite realimentar a la matriz de compuertas AND, la terminal o la salida invertida del flip-flop.

Si en la PAL22V10 se selecciona una salida combinacional, la realimentación se obtiene desde la misma terminal de salida. Si, por el contrario, se selecciona una salida provista de registro, la realimentación a la matriz de compuertas AND se realiza desde la salida invertida del flip-flop.

Algunos dispositivos lógicos programables (PLDs) ofrecen las dos formas de realimentación a la vez, es decir, que disponen de realimentación desde la terminal de salida y de realimentación desde la salida del registro. Si en estos dispositivos se

selecciona una salida provista de registro y al mismo puede actuar como entrada. De esta forma se puede disponer simultáneamente de un registro interior enterrado (buried) o sin acceso al exterior y de una entrada. Hay muchas aplicaciones (por ejemplo, registros de desplazamiento) que utilizan registros enterrados y también hay dispositivos lógicos programables (PLD's) que incluyen registros interiores sin posibilidad de acceder a una terminal, con objeto de atender a este tipo de aplicaciones.

La PAL26V12 es una versión de la PAL22V10 con 28 terminales que ofrecen más entradas y salidas que la segunda, aunque no ha tenido tanta aceptación.

4.1.4.7 GALs

GAL (Generic Array Logic) en español Arreglo lógico genérico.

Es una marca registrada de Lattice Semiconductor que designa a dispositivos borrables por medios eléctricos, que han sido diseñados con el objeto de sustituir a la mayoría de las PALs de la serie general manteniendo la compatibilidad de la configuración de terminales.

La arquitectura del GAL, es muy compleja como consecuencia de la diversidad de PALs a las que puede reemplazar. Este dispositivo está formado por arreglos o matrices que pueden ser fijos o programables. Cada terminal de entrada-salida dispone de macrocélulas. Los fusibles de las macrocélulas permiten configurar al GAL en tres modos diferentes: registro, complejo y simple.

Los GALs son reprogramables ya que usan la tecnología E^2 CMOS (CMOS borrable eléctricamente) y contiene configuraciones de salida programables.

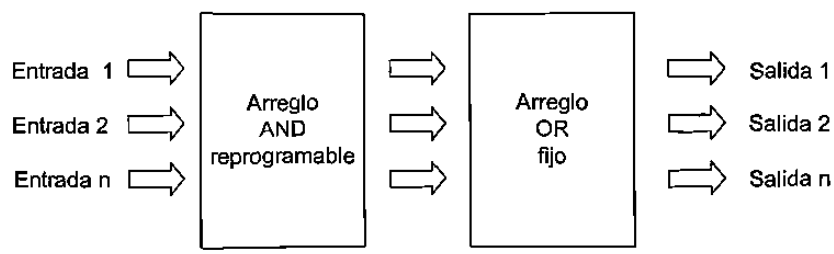


Figura 4.7 Diagrama de bloques del arreglo GAL

4.1.4.7.1 FUSIBLES DE SEGURIDAD

Todos los PLDs disponen de un fusible de seguridad, que una vez fundido impide la lectura del PLD programado proporcionando gran seguridad frente a la piratería. En los PLDs bipolares, la programación destruye ciertos fusibles y por ello, si se elimina la cubierta del PLD y se observa al microscopio, puede examinarse su contenido. Los PLD CMOS borrables eléctricamente o mediante luz ultravioleta son más seguros, y el fusible de seguridad ofrece suficientes garantías de protección contra el acceso al contenido.

4.1.4.7.2 FIRMAS ELECTRÓNICAS

Muchos dispositivos del tipo GAL soportan firmas electrónicas. Las firmas son un conjunto de fusibles que no están destinados a la realización de lógica, sino que están reservados para guardar información auxiliar como: la fecha en la que fue programada, su nivel de revisión, las iniciales del autor o cualquier otra cosa. Debido a que la firma electrónica puede leerse aunque se haya fundido el fusible de seguridad del GAL, se dispone de un método de identificación de los dispositivos programados.

4.1.4.8 PALs CON OR-EXCLUSIVA

Los dispositivos de este grupo tiene compuertas OR-exclusiva en su etapa final, justo antes de las células de salida. Se utilizan con frecuencia para la realización de

contadores. En ocasiones, se pueden realizar funciones lógicas que no caben en las PALs con organización de suma de productos.

4.1.4.9 PALs CON RELOJ INDEPENDIENTE PARA CADA REGISTRO

Los flip-flops con señales de reloj independientes resultan muy adecuados para la realización de máquinas de estados asíncronas. La PAL16RA8 de 20 terminales y la PAL20RA10 de 24 terminales poseen macrocélulas iguales. Cada macrocélula tiene un flip-flop tipo D con habilitación de tres estados. La entrada D del flip-flop está conectada a una suma de cuatro productos. Hay productos separados para el reloj, para la habilitación del tercer estado, para el reset asíncorono y para el set asíncrono. Sí no se desea utilizar los flip-flops, se pueden eliminar poniendo simultáneamente a nivel alto las entradas de reset y preset asíncronos.

4.2 CARACTERÍSTICAS DE LOS DISPOSITIVOS LÓGICOS PROGRAMABLES

Existe una amplia variedad de circuitos integrados (CIs) que pueden tener sus funciones lógicas "programadas" en su interior después de su fabricación. La mayor parte de estos dispositivos utilizan una tecnología que permite también su reprogramación, lo que significa que si usted encuentra un error en su diseño, puede ser capaz de arreglarlo sin reemplazar físicamente o volver a alambrar el dispositivo.

Históricamente, los arreglos de lógica programable (PLA) contenían compuertas AND y OR con una estructura de dos niveles, con conexiones programables por el usuario. Haciendo uso de esta estructura, un diseñador podía adaptar cualquier función lógica hasta un cierto nivel de complejidad utilizando la bien conocida teoría de síntesis lógica y minimización.

La estructura de los PLA sufrió mejoras y su costo disminuyó con la introducción de dispositivos PAL. Actualmente, tales dispositivos se conocen en forma genérica como dispositivos de lógica programable y son los "MSI" de la industria de la lógica programable.

La siempre creciente capacidad de los circuitos integrados creó una oportunidad para los fabricantes de CI para diseñar PLD más grandes para aplicaciones mayores de diseño digital. Sin embargo, por razones técnicas la estructura básica de dos niveles AND-OR de los PLD no podía ser escalada a mayores dimensiones. En su lugar, los fabricantes de CIs idearon arquitecturas complejas PLD (CPLD, complex PLD) para conseguir la escala requerida. Un CPLD es simplemente una colección de varios PLD y una estructura de interconexión del chip también es programable, lo que proporciona una amplia variedad de posibilidades de diseño. Los CPLD pueden ser escalados a tamaños más grandes si se aumenta la cantidad de PLD individuales y la complejidad de la estructura de interconexión en el chip CPLD.

4.3 FPGA

Los Arreglos de Compuertas Programables en Campo (FPGAs) han llegado hacer increíblemente populares por los prototipos y diseños de complejos sistemas Hardware. La estructura de un FPGA puede ser descrita como un "arreglo de bloques" conectados unos con otros, vía interconexiones programables. La principal ventaja de los FPGAs es la flexibilidad que ellos ofrecen. Un ingeniero puede refinar el diseño de un dispositivo

explotando la reprogramación de este. Xilinx introdujo por vez primera la palabra FPGA, en el XC2064, en 1985. Este dispositivo contenía aproximadamente 1000 compuertas lógicas. Desde entonces, la densidad de compuertas de los FPGAs ha incrementado a mas de 25 veces.

La aplicación de estos dispositivos ha sido abundante en muchos campos de la ciencia y pareciera que realmente no hay límite. Algunas de las aplicaciones de estos dispositivos son por ejemplo en la implementación de redes neuronales y en sistemas complejos de control donde se diseñan sistemas difusos o borrosos y claro esta que con estos "arreglos de bloques" se pueden diseñar sistemas de comunicaciones tan complejos y funcionales como nuestra capacidad e imaginación lo permita.

Una cosa curiosa e interesante en la que los FPGAs están involucrados es en la llamada tecnología evolutiva, dado que estos dispositivos se pueden reconfigurar de forma dinámica.

Los FPGAs por ser dispositivos programables, requieren de un software de apoyo en el cual se pueda describir el circuito que se quiere implementar, simular y finalmente programar el circuito físico.

Para entender el significado de VHDL podemos dividir esta palabra en dos acrónimos:

VHSIC- (Very high speed integrated circuit) en español circuitos integrados de muy alta velocidad

HDL – (Hardware Description Language) en español lenguaje de descripción de hardware

El VHDL ofrece las siguientes características:

Modelado. Descripción abstracta de circuitos integrados

Simulación. Vectores de prueba y captura de resultados

Síntesis. Traducción de instrucciones VHDL a circuitos digitales físicos

4.3.1 CONCEPTOS BÁSICOS DE FPGAs.

Los FPGAs han surgido como una solución para producir prototipos de bajo costo en un tiempo muy corto. Varios factores han contribuido al uso expandido de FPGAs:

- La velocidad y complejidad creciente del FPGAs
- La disminución rápida del costo por la puerta de FPGAs
- La disponibilidad de herramientas de bajo costo de síntesis para FPGAs
- La disponibilidad de chips para ser usados para la síntesis

Los dispositivos de Arreglos de Compuertas pueden dividirse, según su arquitectura, en dos categorías principales: los PLDs y los FPGAs. Los PLDs consiste de programable arreglos AND y fijar abanicos de entrada de compuertas OR que son seguidas por flip-flops. La utilización varía con las aplicaciones, pero es típicamente muy bajo a causa de la arquitectura rígida AND / OR (I/O).

Los FPGAs consisten de un conjunto fijo de células lógicas en que el diseño se combinará y encaminará (mapped and routed). Cada célula lógica se constituye de compuertas universales, los multiplexores, y memorias de acceso aleatorio (RAM), flip-flops, y buffers. Todas las células lógicas son idénticas y pueden configurarse por medio de métodos diferentes, por ejemplo: SRAM, EEPROM o Antifuse. Los contenidos lógicos de una de estas células lógicas se llama Bloque Lógico Configurado (CLB). Recientemente, el uso de FPGAs en el diseño de sistemas digitales ha crecido dramáticamente debido a varias ventajas. Entre las ventajas de usar FPGAs están:

- La densidad alta de compuertas
- El ciclo corto de diseño
- El costo bajo
- La reprogramabilidad (facilita cambios de diseño)
- El reemplazo de Chips MSI y SSI
- El prototipo rápido

Para perfeccionar el uso de dispositivos lógicos programables es necesario comprender la tecnología y la arquitectura de tales dispositivos.

4.3.2 FPGA XC4010XL DE LA FIRMA XILINX

Estos dispositivos se programan mediante un software que requiere una descripción lógica del sistema digital a implementar para generar un archivo que puede ser utilizado para programar el dispositivo.

Esta familia posee las siguientes características principales:

- Gran capacidad de circuitos lógicos:
- Memoria Select-RAM, con opción de escritura síncrona.
- Abundante cantidad de flip-flops.
- Generador de Funciones flexible.
- Buses internos capaces de permanecer en los 3 diferentes estados.
- Operación por encima de los 80 MHz.
- Arquitectura flexible.
- Configurable a través de archivo binario.
- Reprogramabilidad ilimitada.
- Características de realimentación.
- Verificación de la programación.
- Posibilidad de uso de nodos internos para seguir la operación del dispositivo.
- Compatibilidad con los demás dispositivos de la serie XC4000.
- La plataforma de desarrollo posee la capacidad de ejecutarse desde computadores comunes con ciertas características mínimas de memoria y velocidad.

  La serie XC4000Xl opera a 3.3 v.

Los FPGAs de esta serie provee los beneficios de la tecnología CMOS VLSI. Estos FPGAs combinan arquitecturas muy versátiles, junto con circuitería que puede operar a altas velocidades. Están conformados por los llamados CLBs (bloques lógicos configurables) caracterizados por su forma de conexión la cual es versátil y jerárquica.

En la tabla 4.1 se presentan las características de los FPGA's series XC4000E y XC400X

Dispositivo	Celdas lógicas	Máximo de Comp. Lógicas (No RAM)	Bits RAM (No Lógicas)	Total de compuertas	Matriz De CLBs	Total De CLBs	Número de Flip-Flops	Máximo De pines I/O
XC4002XL	152	1,600	2,048	1,000 - 3,000	8 x 8	64	256	64
XC4003E	238	3,000	3,200	2,000-5,000	10x10	100	360	80
XC4005E/XL	466	5,000	6,272	3,000-9,000	14x14	196	616	112
XC4006E	608	6,000	8,192	4,000-12,000	16x16	256	768	128
XC4008E	770	8,000	10,368	6,000-15,000	18x18	324	936	144
XC4010E/XL	950	10,000	12,800	7,000-20,000	20x20	400	1,120	160
XC4013E/XL	1368	13,000	18,432	10,000-30,000	24x24	576	1,536	192
XC4020E/XL	1862	20,000	25,088	13,000-40,000	28x28	784	2,016	224
XC4025E	2432	25,000	32,768	15,000-45,000	32x32	1,024	2,560	256
XC4028EX/XL	2432	28,000	32,768	18,000-50,000	32x32	1,024	2,560	256
XC4036EX/XL	3078	36,000	41,472	22,000-65,000	36x36	1,296	3,168	288
XC4044XL	3800	44,000	51,200	27,000-80,000	40x40	1,600	3,840	320
XC4052XL	4598	52,000	61,952	33,000-100,000	44x44	1,936	4,576	352
XC4062XL	5472	62,000	73,738	40,000-130,000	48x48	2,304	5,376	384
XC4085XL	7448	85,000	100,352	55,000-180,000	56x56	3,136	7,168	448

Tabla 4.1 Características de los FPGAs Series XC4000E y XC400X

Los FPGAs de XILINX incluyen básicamente dos elementos configurables por el usuario, estos son los bloques lógicos configurables (CLBs) y los bloques de entrada salida configurables (IOBs).

Los CLBs proveen los elementos funcionales para construir la lógica que el usuario requiera.

Los IOBs proveen la interfaz entre los pines externos del dispositivos y las líneas de señales internas.

Un CLB es un bloque lógico configurable en el cual se implementa la mayor parte de la lógica en un FPGA. Los principales elementos de un CLB son mostrados en la siguiente gráfica.

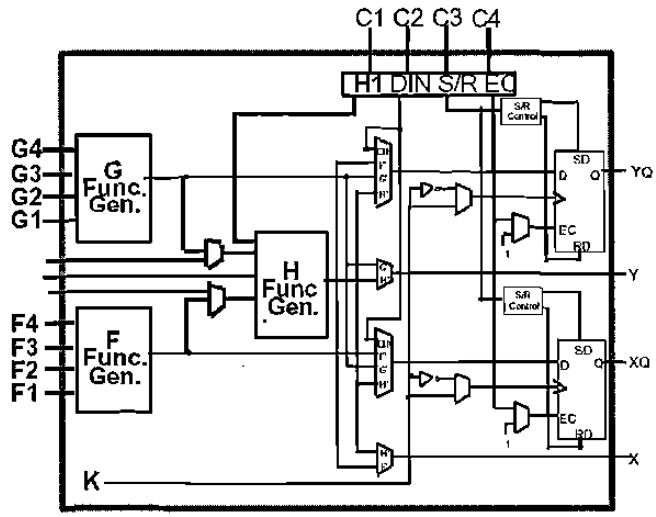


Figura 4.8 CLB de la familia XC4000

Los CLBs están compuestos básicamente por los siguientes elementos:

- Bloques generadores de funciones.
- Flip-flops.
- Latches (solo en la serie XC4000X).
- Entradas de reloj.
- Habilitadores de reloj.
- Sets y reset globales
- Señales de control.

Dos generadores de funciones de 4 entradas (F y G) ofrecen gran versatilidad al dispositivo. La mayoría de las funciones lógicas necesitan 4 o más pocas entradas, sin embargo un tercer generador de funciones es provisto (H). Esta función H posee tres entradas, además cada CLB puede implementar funciones por encima de 9 variables. Otra característica de estos CLBs de esta familia de FPGAs es que contienen elementos de almacenamiento que pueden ser usados para guardar las salidas de los generadores de

funciones, sin embargo los elementos de almacenamiento y los generadores de funciones pueden ser utilizados independientemente. Estos elementos de almacenamiento pueden ser usados como flip-flops o como latches. Los generadores de funciones son implementados como memorias LUT (Look up table).

Los CLBs pasan las salidas de los bloques generadores de funciones a las redes de interconexión de otros CLBs o pueden también almacenar los resultados combinacionales en flip-flops que hace parte de los CLBs y luego conectar sus salidas a la red de interconexión si es necesario. Cada CLBs posee dos flip-flops tipo de D los cuales tienen señales: reloj común (clock: K), habilitador de reloj (Clock enable EC), señales de set y reset.

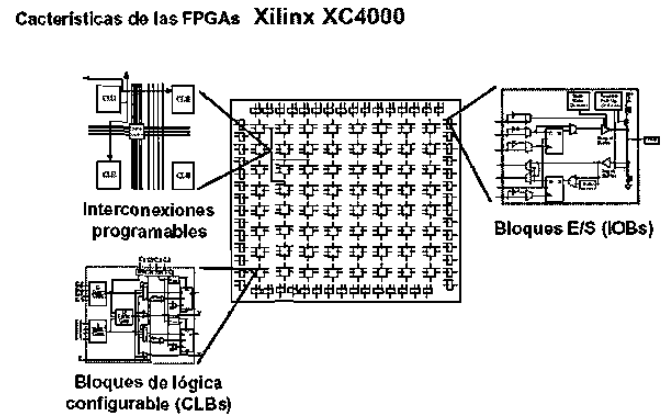


Figura 4.9. Bloques que conforman un FPGA

El dispositivo XC4010xl está compuesto por una gran cantidad de CLBs descritos, esta organización es una matriz de todos los CLBs. En particular este FPGAs esta compuesto por una matriz de 20x20 CLBs, es decir 400.

Además cada FPGA contiene una cantidad de IOBs (bloques entrada / salida) y rutas programables que interconecta los diferentes CLBs y los IOBs. La figura 4.20 ilustra esta distribución.

5. CAPTURA ESQUEMÁTICA

5.1 CARACTERÍSTICAS DE LA CAPTURA ESQUEMÁTICA

Por captura de esquemas se entiende el proceso de descripción, mediante un dibujo, de un circuito eléctrico. El dibujo del esquema puede incluir más que un simple diagrama de líneas. Puede incluir también información sobre tiempos, referencias, cables, conectores, notas y muchas otras propiedades importantes y valores necesarios para la interpretación del mismo por el resto de aplicaciones.

Un esquema viene especificado en la base de datos por dos partes fundamentales: las hojas y los símbolos. En principio, un esquema puede estar formado por varias hojas que es donde se dibujan los diversos componentes o símbolos que forman el circuito. En las hojas se especifican también las interconexiones, así como informaciones adicionales para el uso posterior del esquema en otras aplicaciones.

Los símbolos son cajas que se interconectan entre sí en la hoja de diseño. Un símbolo es un objeto que contiene un conjunto de modelos usados para describir los aspectos funcionales, gráficos, temporales y tecnológicos del diseño electrónico.

Hay dos tipos de símbolos. El primer tipo está formado por los símbolos que representan componentes básicos o primitivas. Estos componentes definen un elemento que se encuentra en el nivel más bajo de la jerarquía de diseño. Así, este tipo de componentes suelen ser resistencias, condensadores, transistores, compuertas lógicas, procesadores, chips de memoria, etc., que suelen estar ubicados en bibliotecas propias de la herramienta.

Un segundo tipo de símbolo es el formado por aquellos que especifican, no un elemento simple, sino otro circuito completo, compuesto a su vez por símbolos, etc., es decir, elementos que están por encima de los símbolos básicos dentro de la jerarquía. Normalmente este tipo de símbolos suelen tener asociados una hoja que es la que describe componentes, aunque, con la aparición de las descripciones mediante lenguaje, es posible encontrar que dentro del símbolo en un esquema se tiene una descripción mediante lenguaje en vez de una hoja que sería lo esperable. Las posibilidades de las herramientas de descripción actuales que permiten, sin demasiados problemas, unir en un mismo diseño descripciones mediante gráficos y descripciones mediante lenguaje.

El método clásico para la interconexión de los distintos símbolos de una hoja son los hilos o wires. Un hilo en el esquema tiene una correspondencia inmediata con el circuito real, se trata de un cable físico que conecta un pin de un chip con un pin de otro. Sin embargo, dado que un esquema puede representar un nivel de abstracción elevado dentro de una jerarquía, un cable puede representar una conexión con un sentido más amplio, como por ejemplo una línea telefónica, o un enlace de microondas a través de satélite.

Un cable en un esquema es un elemento que indica conexión y, en principio, puede tratarse tanto de un hilo de cobre como de una pista en un circuito impreso, o bien un conjunto de hilos, o un cable de una interfase serie, etc. Sin embargo, en los comienzos del diseño electrónico, donde los esquemas correspondían en la mayoría de los casos al nivel más bajo de una jerarquía, los cables eran siempre hilos conductores, y para representar un conjunto de hilos conductores se introdujo otro elemento adicional, el bus.

Un bus es una conexión que une dos componentes al igual que un cable, sin embargo se caracteriza por representar, no un único hilo, sino múltiples. La introducción de este elemento fue inmediata a partir del desarrollo de circuitos digitales, donde la conexión entre procesadores, memorias, etc., era fácilmente agrupable.

Actualmente, dada la gran complejidad de los diseños electrónicos, con miles de conexiones en una misma hoja, se hace necesario el uso de otras técnicas de interconexión de componentes. Una posibilidad que ofrecen la mayoría de las herramientas de CAD es la utilización de etiquetas. Es posible poner etiquetas a los pines o a los cables, de manera que dos pines o cables con la misma etiqueta, o nombre, están físicamente interconectados. Esto evita el tener que trazar múltiples conexiones entre componentes evitando así una aglomeración de hilos que harían ilegible cualquier esquema.

Otro elemento importante dentro de una hoja o esquema son los puertos. Los puertos son conexiones al exterior de la hoja, y realizan la labor de interfase del circuito con el mundo exterior. En general, un esquema se puede ver como una caja negra donde los puertos son la única información visible. Esta caja negra, junto con sus puertos, forman un componente que puede ser usado en otra hoja, que a su vez es un componente que puede formar parte de otra hoja y así sucesivamente. Los puertos pueden ser de entrada, de salida, o de entrada/salida, dependiendo de la dirección del flujo de la información.

5.2 PROCESO DE LA CAPTURA ESQUEMÁTICA

Para el proceso de la captura esquemática necesitaremos mencionar los cuatro componentes básicos que son: **símbolos, conectores, etiquetas y los puertos de entrada y/o salida.**

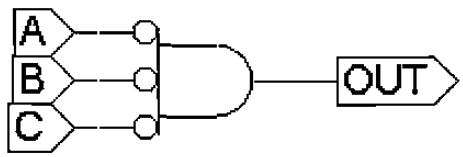


Figura 5.1 Componentes básicos

A continuación se explican brevemente cada uno de estos elementos:

Conectores (alambre)

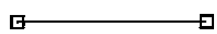


Figura 5.2 Conector

Estos se utilizan para la interconexión entre terminales de los símbolos o dispositivos de entrada/salida.

Símbolos

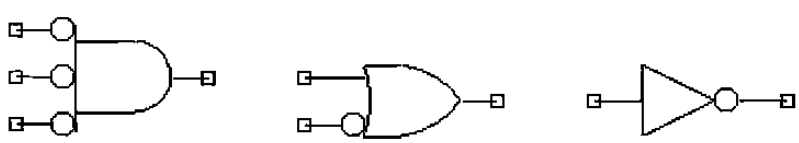


Figura 5.3 Símbolos

Son una representación gráfica de los componentes

Etiquetas (variables)

Figura 5.4 Etiquetas

Son nombres para identificar entradas y salidas de los símbolos

Puertos de entrada/salida

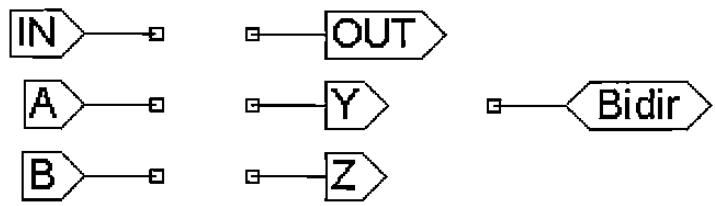


Figura 5.5 Puertos de entrada/salida

Los puertos son conexiones al exterior de la hoja, y realizan la labor de interfase del circuito con el mundo exterior estos pueden ser de entrada, salida o Puerto bidireccional.

En seguida se presentan algunas definiciones importantes cuando se trabaja con captura esquemática:

Editor Esquemátic:

El Editor Esquemático es la herramienta de captura esquemática de los SCS's (Sistemas de captura esquemática).Crea archivos esquemáticos (.sch) que pueden representar un diseño completo, o cualquier componente de un diseño jerárquico.

Editor de Símbolos

SCS viene con una biblioteca de símbolos normal para diseños con ispLSI. El Editor de Símbolos es usado para crear símbolos o elementos primitivos que representan un modelo esquemático independiente. También se pueden crear símbolos decorativos (como bloques del título).

Navegador Jerárquico

El Navegador Jerárquico combina todos los componentes de un diseño multi-nivel para visualizarlo y analizarlo. Usted puede cruzar el diseño completo y puede ver cada componente en su contexto jerárquico completo.

Visualizador de formas de onda

El visualizador de formas de onda es usado para observar los resultados de la simulación. Usted puede desplegar la forma de onda de cualquier red de su diseño. El visualizador es totalmente interactivo con el Navegador jerárquico; pulsando el botón sobre la red en el esquemático automáticamente despliega la forma de onda. El funcionamiento del visualizador de forma de onda se cubre en el Manual de Herramientas de forma de ondas.

5.2.1 REALIZACIÓN DE UN DIAGRAMA ESQUEMÁTICO

Para realizar un diagrama esquemático, recurriremos a los cuatro elementos básicos de la captura esquemática, ya que ellos nos permitirán formar el diseño necesario para resolver el problema planteado.

Del problema planteado se debe de obtener como solución un diagrama de conexiones, a partir de este diagrama se procede a la captura esquemática.

Para lograr la captura esquemática debemos cubrir lo siguiente.

Se procede a la captura esquemática recurriendo a la librerías de símbolos para buscar los símbolos necesarios (tales como compuertas lógicas y bloques funcionales) y llevarlos a la hoja donde se armará el diagrama, posteriormente habrá que hacer las conexiones necesarias entre símbolos, también tenemos que declarar las variables o etiquetas requeridas en el diseño (variables de entrada y variables de salida) para posteriormente identificarlas como tal (variables de entrada, salida y bidereccional).

Con el archivo obtenido en la captura esquemática podemos llegar a la simulación del problema y verificar que funciona correctamente.

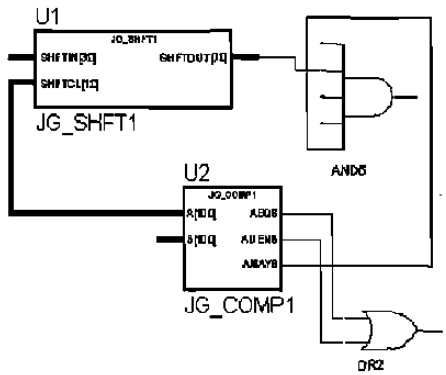


Figura 5.6 Diagrama esquemático

5.3 ENLACE (LINK) Y SÍNTESIS DE LA CAPTURA ESQUEMÁTICA CON EL DISPOSITIVO A PROGRAMAR.

El archivo que se obtuvo a partir de la captura esquemática se enlazará con las posibilidades del dispositivo seleccionado.

El proceso de enlace o link, verifica si los símbolos que se seleccionaron al hacer la captura esquemática existen en el dispositivo a programar.

El proceso de enlace y síntesis es semejante en varios software's por ejemplo para el software de Foundation Project Manager para efectuar la compilación o síntesis se haría lo siguiente (consideremos el ejemplo de la programación de un FPGA):

El proceso de síntesis del diseño para la programación del dispositivo se efectúa de manera automática. Este proceso involucra diferentes etapas, las cuales dependen si el dispositivo a utilizar. Para cada una de estas etapas se genera un reporte que muestra resultados intermedios en el proceso tales como: recursos utilizados del dispositivo, asignamiento de pines, retardos y máxima frecuencia de operación, etc., así como errores o posibles errores en alguna de las etapas.

El proceso de síntesis del dispositivo consta de las siguientes etapas:

a) *Translate*: convierte el archivo de diseño EDIF en un archivo NGD.

b) *Map*: agrupa los elementos básicos (flip–flops, compuertas, etc.) en bloques lógicos y genera el reporte de tiempo a nivel lógico.

c) *Place&Route*: localiza los bloques lógicos en el dispositivo y canaliza las señales entre ellos.

d) *Timing*: genera datos de simulación de tiempo y el reporte de tiempo de *post-layout.*

e) *Configure*: genera el archivo (.BIT o .SVF) adecuado para programar al dispositivo.

Con la ejecución de estas etapas el proceso de implementación termina. En caso de existir errores en alguna etapa, el proceso de implementación se detiene y se genera un reporte indicando la causa.

Al finalizar el proceso de síntesis la herramienta habrá generado el archivo con extensión .bit (.svf) para la programación del dispositivo.

6. LENGUAJE DE DESCRIPCIÓN DE HARDWARE

6.1 CARACTERÍSTICAS DE LOS HDLs

Los lenguajes de descripción de hardware (HDLs) son utilizados para describir la arquitectura y comportamiento de un sistema electrónico los cuales fueron desarrollados para trabajar con diseños complejos.

Comparando un HDL con los lenguajes para el desarrollo de software vemos que en un lenguaje de este tipo un programa que se encuentra en un lenguaje de alto nivel (VHDL) necesita ser ensamblado a código máquina (compuertas y conexiones) para poder ser interpretado por el procesador. De igual manera, el objetivo de un HDL es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere (ensamble) un circuito que pueda ser implementado físicamente.

La forma más común de describir un circuito es mediante la utilización de esquemas que son una representación gráfica de lo que se pretende realizar. Con la aparición de herramientas EDA (Electronic Design Automation, es el nombre que se le da a todas las herramientas de hardware y software en el diseño de sistemas electrónicos) cada vez más complejas, que integran en el mismo marco de trabajo las herramientas de descripción, síntesis, simulación y realización, apareció la necesidad de disponer de un método de descripción de información entre las diferentes herramientas que componen el ciclo de diseño. En principio se utilizó un lenguaje de descripción que permitía, mediante sentencias simples, describir completamente un circuito. A estos lenguajes se les llamó Netlist puesto que eran simplemente eso, un conjunto de instrucciones que indicaban las interconexiones entre los componentes de un diseño. A partir de estos lenguajes simples, que ya eran auténticos lenguajes de descripción de hardware, se descubrió el interés que podría tener el describir circuitos utilizando un lenguaje en vez de usar esquemas. Sin embargo, se siguieron utilizando esquemas puesto que desde el punto de vista del ser humano son mucho más sencillos de entender, aunque un lenguaje siempre permite una edición más rápida y sencilla.

Conforme las herramientas de diseño se volvieron más sofisticadas, y la posibilidad de desarrollar circuitos digitales mediante dispositivos programables era más viable, apareció la necesidad de poder describir los circuitos mediante un lenguaje de alto nivel de abstracción. No desde un punto de vista estructural, sino desde el punto de vista funcional. Este nivel de abstracción se había alcanzado ya con las herramientas de simulación, ya que para poder simular partes de un sistema era necesario disponer de modelos que describieran el funcionamiento de bloques del circuito o de cada componente si fuera necesario. Estos lenguajes estaban sobre todo orientados a la simulación, por lo que poco importaba que el nivel de abstracción fuera tan alto que no fuera sencillo una realización o síntesis a partir de dicho modelo. Con la aparición de técnicas para la síntesis de circuitos a partir de lenguajes de alto nivel de abstracción, se comenzaron a utilizar los lenguajes de simulación para sintetizar circuitos. Que si bien alcanzan un altísimo nivel de abstracción, su orientación era básicamente la de simular, por lo que los resultados de una síntesis a partir de descripciones con estos lenguajes no eran siempre las más optimas.

Además, los lenguajes de descripción de hardware al formar parte de las herramientas EDA permiten el trabajo en equipo. Así, al estructurar el desarrollo del proyecto, cada integrante del equipo de diseño puede trabajar en subproyectos antes de integrar todas las partes del sistema.

6.1.1 VENTAJAS DE LOS HDLs

Una metodología de diseño que utiliza un HDL posee varias ventajas sobre la metodología tradicional de diseño a nivel compuerta. Algunas de estas ventajas son listadas a continuación.

Es posible verificar el funcionamiento del sistema dentro del proceso de diseño sin necesidad de implementar el circuito.

Las simulaciones del diseño, antes de que éste sea implementado mediante compuertas, permiten probar la arquitectura del sistema para tomar decisiones en cuanto a cambios en el diseño.

Las herramientas de síntesis tienen la capacidad de convertir una descripción hecha en un HDL, VHDL por ejemplo, a compuertas lógicas y, además, optimizar dicha descripción de acuerdo a la tecnología utilizada.

Esta metodología elimina el antiguo método tedioso de diseño mediante compuertas, reduce el tiempo de diseño y la cantidad de errores producidos por el armado del circuito.

Las herramientas de síntesis pueden transformar automáticamente un circuito obtenido mediante la síntesis de un código en algún HDL, a un circuito pequeño y rápido. Además, es posible aplicar ciertas características al circuito dentro de la descripción para afinar detalles (retardos, simplificación de compuertas, etc.) en la arquitectura del circuito y que estas características se obtengan en la síntesis de la descripción.

Las descripciones en un HDL proporcionan documentación de la funcionalidad de un diseño independientemente de la tecnología utilizada.

Un circuito hecho mediante una descripción en un HDL puede ser utilizado en cualquier tipo de dispositivo programable capaz de soportar la densidad del diseño. Es decir, no es necesario adecuar el circuito a cada dispositivo porque las herramientas de síntesis se encargan de ello.

Una descripción realizada en un HDL es más fácil de leer y comprender que los netlist o circuitos esquemáticos.

Existen varios lenguajes de descripción de hardware (HDL), entre los que destacan:

- VHDL
- Verilog
- ABEL
- Hardware C
- Handel C

Así mismo el VHDL tiene varias plataformas las más conocidas son:

- Active- HDL
- Foundation
- Model-sim
- Leonardo Spectrum
- Warp
- Renoir

Los lenguajes de descripción de hardware actualmente usados son ABEL, VHL y Verilog.

6.2 ABEL

A Continuación se presenta una descripción de algunas características y sintaxis del lenguaje ABEL-HDL.

ABEL (Advanced Boolean Expression Language) Lenguaje Avanzado de Expresiones Booleanas.

Fue desarrollado por Data I/O Corporation para la implementación de funciones booleanas en dispositivos lógicos programables.

ABEL puede ser usado para describir el comportamiento de un sistema digital partiendo de:

- Ecuaciones Booleanas.
- Descripción del comportamiento usando instrucciones WHEN-THEN
- Tablas de Verdad.
- Tablas de Estado
- Diagramas de Transición

ABEL es un archivo de texto que contiene los siguientes elementos.

1	Documentación incluyendo nombre del programa y comentarios.
2	Declaraciones que identifican las entradas y salidas de las funciones lógicas que serán efectuadas.
3	Instrucciones que especifican las funciones lógicas que serán efectuadas.
4	Declaración del tipo de dispositivo en que las funciones lógicas especificadas se implementaran.
5	Vectores de prueba que especifican las salidas esperadas de las funciones lógicas para ciertas entradas

Tabla 6.1 Elementos de un archivo ABEL

ABEL necesita un procesador de lenguaje llamado *Compilador* cuyo trabajo es traducir el archivo de texto de ABEL en un Mapa de Fusibles (JEDEC) del dispositivo

físico seleccionado. Pasando por un proceso de *validación* de las instrucciones así como de *minimización* de las funciones para ajustar, si es posible, a la capacidad del dispositivo seleccionado.

6.2.1 SINTÁXIS BÁSICA DE ABEL-HDL

- **Identificadores:**

  Los identificadores son usados para definir variables y las reglas de uso son:

1	Los identificadores no pueden ser mayores de 31 caracteres. Ejemplo: Este_es_un_identificador_largo Esteesunidentificadorlargo
2	Deben de iniciar con un carácter alfabético o con un guión bajo. Ejemplos: HELLO Hello _K5input P_h
3	Los identificadores si son sensitivos a mayúsculas o minúsculas. Por ejemplo el identificador *output* es un identificador diferente a *Output* o *OUTPUT*.
4	los identificadores pueden ser separados por comas A, B,C
5	En las expresiones los identificadores o números pueden ser separados por operadores o donde los paréntesis proveen la separación.

Tabla 6.2 Reglas de uso para los identificadores

- **Identificadores no válidos**:

  7_ $4 Deben de comenzar con letra o guión bajo.

  Hel.lo No se deben de usar puntos.

  B6 kj No se debe de usar espacio y se interpreta como dos identificadores B6 y kj.

- **Palabras clave** (KEYWORDS)

Las palabras clave son identificadores reservados que se pueden escribir con minúsculas o mayúsculas o una combinación de estas, a continuación se enlistan las palabras clave más usadas:

Declarations	device	else	end	equations
Goto	If	istype	macro	module
Pin	State	state_diagram	state_register	test_vectors
Then	Title	truth_table	When	With

Tabla 6.3 Palabras claves

Las palabras claves (KEYWORDS) deben de estar separadas por lo menos por un espacio.

Las líneas escritas en un archivo ABEL deben de cumplir con los siguientes requisitos:

1. Una línea no puede exceder de 150 caracteres.
2. Los comentarios deben de empezar con comillas "
3. Las líneas o instrucciones deben de terminar con ;

Los caracteres ASCII soportados son:

a - z (alfabeto minúsculas).

A - Z (alfabeto mayúsculas).

0 - 9 (dígitos)

<space> <tab>

! @ # $? + & * ( ) - _ . = + [ ] { } ; : ' " ' \ | , < > . / ^ %.

Operadores Lógicos:

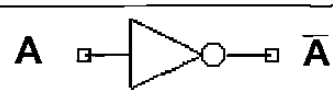
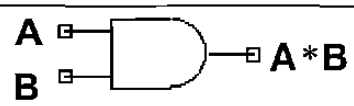
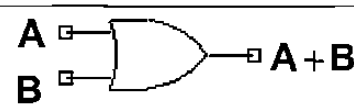
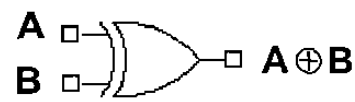
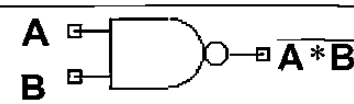
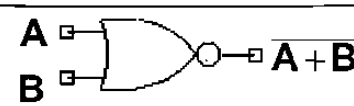
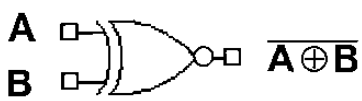
Operador	**Descripción**	Ecuación	Símbolo
!	NOT	!A	
&	AND	A&B	
#	OR	A#B	
$	EXOR	A$B	
!&	NAND	!(A&B)	
!#	NOR	!(A#B)	
!$	EXNOR	!(A$B)	

Tabla 6.4 Operadores lógicos

Ejemplos de ecuaciones con ABEL-HDL

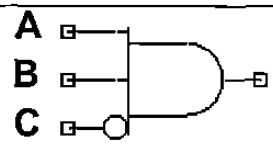
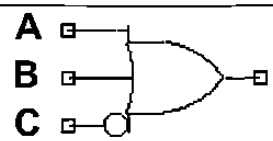
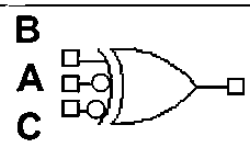
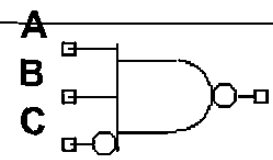
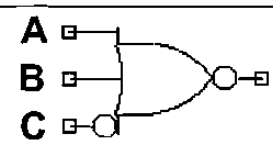
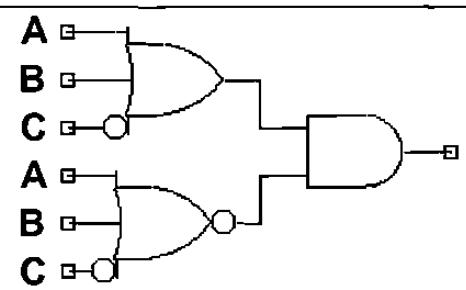
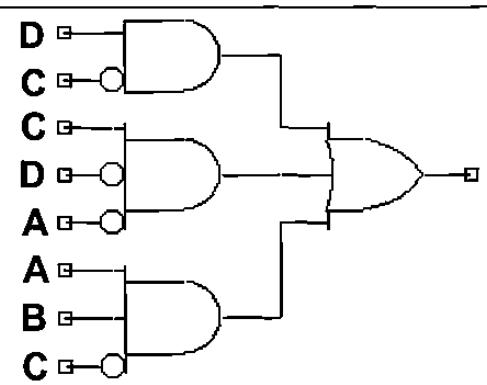
Circuito	Ecuación
	A & B & !C
	A # B # !C
	!A $ B $!C
	!(A & B & !C)
	!(A # B # !C)
	(A # B # !C)& !(A # B # !C)
	!C&D # !A&C&!D # A&B&!C

Tabla 6.5 Ejemplos de ecuaciones con ABEL

- **Números** *(Numbers)*

Los números pueden ser utilizados en cuatro diferentes bases: binario, octal, decimal y hexadecimal.

Si no se especifica una base, ABEL-HDL lo tomará como base decimal. Para indicar una base diferente del decimal es necesario utilizar el símbolo ^ y la inicial de la base.

Nombre	Base	Símbolo
Binario	**2**	^b
Octal	8	**^o**
Decimal	10	**^d** (default)
Hexadecimal	16	**^h**

Tabla 6.6 Ejemplos de indicación de bases de números

Ejemplos:

Base	Especificación en ABEL	Valor Decimal
Decimal	35	35
Hexadecimal	^h35	53
Binario	^b101	5
Octal	^o22	18

Tabla 6.7 Ejemplos de representación en ABEL

- **Declaraciones:**

  Es una colección de señales o constantes usadas como referencia de un grupo de expresiones simplificadas a un solo nombre.

  Ejemplos:

  Y = [D0, D1, D2, D4, D5]:

  X= [A, B, C, D];

  aset = [a2,a1,a0]; bset = [b2,b1,b0];

  COUNT = [Q9, Q8, Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0];

  En ABEL-HDL es posible cambiar las expresiones **.X.** por simplemente **X** o ***.C.*** por ***C*** usando una igualdad como se indica a continuación:

  X = .X.; **C = .C.** o las dos a la vez **C,X = .c.,.x.;**

- **Set:**

  Es una lista de constantes o variables que están separadas por comas o por dos puntos seguidos (..) que indican el rango del operador, en esta opción los paréntesis rectangulares son requeridos.

  Ejemplos:

[D0..D6]	Rango [D0,D1,D2,D4,D5,D6]
[b6..b0]	" Decremento el rango
[D7..D15]	Rango parcial
[b1,b2,a0..a3]	" Combinación de variables y rango
[!S7..!S0]	" Decremento el rango con nivel activo bajo

  No es permitido dentro del rango usar diferentes nombres de variables [**X0..D5**]

Partes de un programa en ABEL-HDL:

1	***Module,*** Inicio del programa
2	***TITLE*** Líneas de titulo y comentarios (opcional)
3	***Declaration*** Asignación de las terminales de entrada y salida del dispositivo
4	***Descripción lógica*** (ecuaciones, tablas de verdad, etc.)
5	***Test_vectors*** Vectores de prueba (opcional)
6	***End,*** Final del programa

Tabla 6.8 Partes de un programa ABEL-HDL

6.2.2 CONSTRUCCIÓN DEL ARCHIVO EN ABEL-HDL

1.- Todo programa debe contener al inicio la instrucción ***MODULE*** y al final ***END*** que indican el principio y el final del programa.

2.- Los cometarios y las líneas de título son opcionales pero es conveniente utilizarlos para describir el funcionamiento y partes del programa, estos deberán de empezar con comillas (“) , ejemplo “Entradas.

3.- **Declarations** por medio de este comando podemos declarar las entradas y salidas del sistema.

Al definir las variables de salida así como su asignación de terminales es necesario incluir el comando **ISTYPE ‘com’** para indicar que son salidas y combinacionales.

- **Descripciones Lógicas:** en esta sección se pueden usar los comandos:
  EQUATIONS permite expresar las ecuaciones.
  TRUTH_TABLE permite declarar una Tabla de Verdad o Tabla de Estados.
  WHEN y **THEN** permite referir el comportamiento en algunos casos.

STATE_TABLE permite describir el comportamiento del diagrama de transición.

- **Vectores de Prueba:** (Test_Vectors):

  Esta parte es opcional y podemos efectuar la comprobación o simulación del diseño sin necesidad de implementarlo.

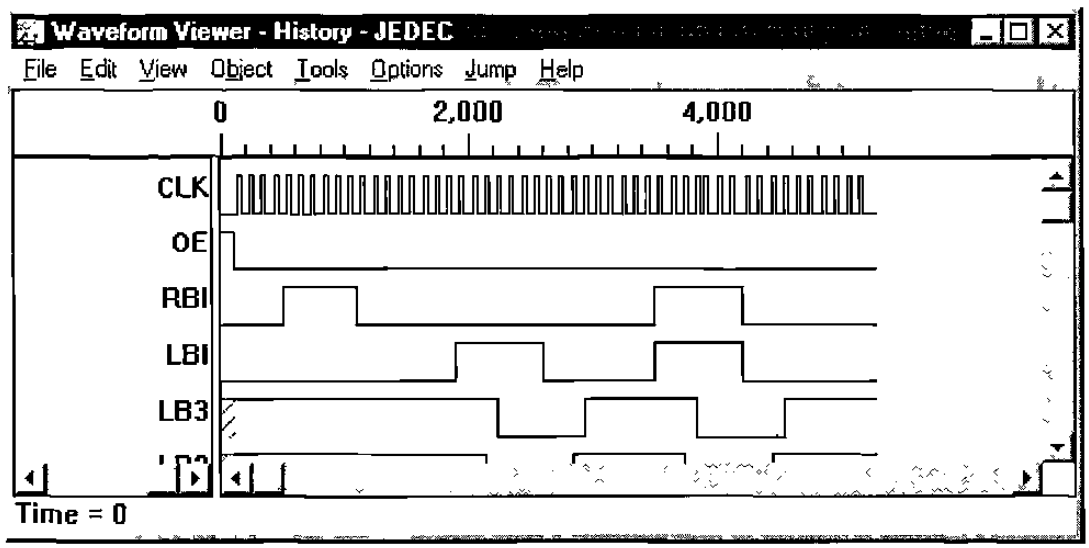


Figura 6.1 Ejemplo de simulación Vectores de Prueba (Test_Vectors)

Estructura del archivo en lenguaje ABEL-HDL

Encabezado	**MODULE EQ**
	Declarations
	"Entradas
Declaraciones	**A,B,C PIN 1,2,3;**
	"Salidas
	FX,FY PIN 19,18 ISTYPE 'COM';
Descripciones	**EQUATIONS**
Lógicas	**FX= A & ¡B & C # ¡B & C;**
	FY = (A # !B # C) & (A # !C);
Vectores de Prueba	**TEST_VECTORS**
	([A,B,C]->[FX,FY])
	[0,0,0]->[.X.,.X.];
	[0,0,1]->[.X.,.X.];
	[0,1,0]->[.X. ,.X.];
	[0,1,1]->[.X. ,.X.];
	[1,0,0]->[.X. ,.X.];
	[1,0,1]->[.X. ,.X.];
	[1,1,0]->[.X. ,.X.];
	[1,1,1]->[.X. ,.X.];
Final	**END**

6.3 VHDL

En principio se utilizó un lenguaje de descripción que permitía, mediante sentencias simples, describir completamente un circuito. A estos lenguajes se les llamó Netlist puesto que eran simplemente eso, un conjunto de instrucciones que indicaban el interconectado entre los componentes de un diseño, es decir, se trataba de una lista de conexiones.

A partir de estos lenguajes simples, que ya eran auténticos lenguajes de descripción hardware, se descubrió el interés que podría tener el describir los circuitos directamente utilizando un lenguaje en vez de usar esquemas. Sin embargo, se siguieron utilizando esquemas, puesto que desde el punto de vista del ser humano son mucho más sencillos de entender, aunque un lenguaje siempre permite una edición más sencilla y rápida.

Con una mayor sofisticación de las herramientas de diseño, y con la puesta al alcance de todos de la posibilidad de fabricación de circuitos integrados y de lógica programable, fue apareciendo la necesidad de poder describir los circuitos con un alto grado de abstracción, no desde el punto de vista estructural, sino desde el punto de vista funcional. Existía la necesidad de poder describir un circuito pero no desde el punto de vista de sus componentes, sino desde el punto de vista de su funcionamiento.

Este nivel de abstracción se había alcanzado ya con las herramientas de simulación. Para poder simular partes de un circuito era necesario disponer de un modelo que describiera el funcionamiento del circuito o sus componentes. Estos lenguajes estaban sobre todo orientados a la simulación, por lo que poco importaba que el nivel de abstracción fuera tan alto que no resultara sencilla una realización o síntesis a partir de dicho modelo.

Con la aparición de técnicas para la síntesis de circuitos a partir de un lenguaje de alto nivel, se utilizaron para la descripción precisamente estos lenguajes de simulación, que si bien alcanzan un altísimo nivel de abstracción, su orientación es básicamente la de simular. De esta manera, los resultados de una síntesis a partir de descripciones con estos

lenguajes no es siempre la más óptima. En estos momentos no parece que exista un lenguaje de alto nivel cuya orientación o finalidad sea la de la síntesis automática de circuitos, por lo que todavía se utilizan estos lenguajes orientados a la simulación también para la síntesis de circuitos.

6.3.1 EL LENGUAJE VHDL

El significado de las siglas VHDL es VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, es decir, lenguaje de descripción hardware de circuitos integrados de muy alta velocidad. VHDL es un lenguaje de descripción y modelado diseñado para describir, en una forma en que los humanos y las máquinas puedan leer y entender la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos y componentes:

VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automática de circuitos. El VHDL fue desarrollado de forma muy parecida al ADA debido a que el ADA fue también propuesto como un lenguaje que tuviera estructuras y elementos sintácticos que permitieran la programación de cualquier sistema hardware sin limitación de la arquitectura. El ADA (Electronic Design Automation) tenía una orientación hacia sistemas en tiempo real y al hardware en general, por lo que se le escogió como modelo para desarrollar el VHDL.

VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.

Uno de los objetivos del lenguaje VHDL es el modelado. Modelado es el desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la

simulación. Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis se parte de una especificación de entrada con un determinado nivel de abstracción y se llega a una implementación más detallada, menos abstracta. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño hacia el más bajo nivel de la jerarquía.

El VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño.

Algunas ventajas del uso de VHDL para la descripción hardware son:

- VHDL permite diseñar, modelar y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de compuertas;
- .Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por diversas herramientas de síntesis para crear e implementar circuitos.
- Los módulos creados en VHDL pueden utilizarse en diferentes diseños, lo que permite la reutilización del código. Además, la misma descripción puede utilizarse para diferentes tecnologías sin tener que rediseñar todo el circuito.
- Al estar basado en un estándar (IEEE Std 1076-1987, IEEE Std 1076-1993) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
- VHDL permite diseño Top-Down, esto es, describir (modelar) el comportamiento de los bloques de alto nivel, analizarlos (simularlos) y refinar la funcionalidad en alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.
- Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

6.3.1.1 VHDL DESCRIBE ESTRUCTURA Y COMPORTAMIENTO

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión, de esta manera se tiene especificado un circuito y se sabe cómo funciona. Ésta es la forma habitual en que se han venido describiendo circuitos, siendo las herramientas utilizadas para ello las de captura de esquemas y las de descripción netlist.

La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente le interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que es realmente un circuito, se pueden plantear algunos problemas a la hora de implementarlo a partir de la descripción de su comportamiento.

El VHDL va a ser interesante puesto que va a permitir los dos tipos de descripciones:

Estructura: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.

Comportamiento: VHDL también se puede utilizar para la descripción comportamental o funcional de un circuito. Esto es lo que lo distingue de un lenguaje de Netlist. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación, ya que permite simular un sistema sin conocer su estructura interna. Así, este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

Muchas veces la descripción comportamental se divide a su vez en dos, dependiendo del nivel de abstracción y del modo en que se ejecutan las instrucciones. Estas dos formas comportamentales de describir circuitos son la de flujo de datos y la algorítmica.

6.3.2 EJEMPLO BÁSICO Y ESTILOS DE DESCRIPCIÓN EN VHDL

VHDL presenta tres estilos de descripción de circuitos dependiendo del nivel de abstracción. El menos abstracto es una descripción puramente estructural. Los otros dos estilos representan una descripción comportamental o funcional, y la diferencia viene de la utilización y no de la ejecución serie.

Mediante un sencillo ejemplo, que además sirve para ilustrar la forma en que se escriben descripciones en VHDL, se van a mostrar estos tres estilos de descripción.
Ejemplo 1 Describir en VHDL un circuito que multiplexee dos líneas de entrada (a y b) de un bit, a una sola línea (salida) también de un bit; la señal selec sirve para seleccionar la línea a (selec= ' O') O b (selec= ' 1').

En la figura siguiente se muestra el circuito implementado con compuertas lógicas que realiza la función de multiplexeo.

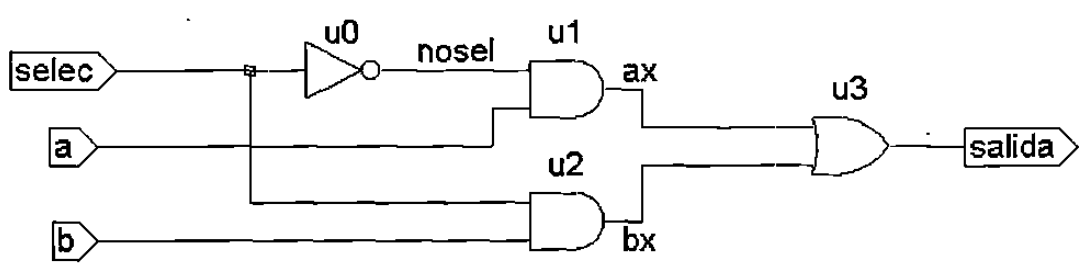


Figura 6.2 Esquema del ejemplo básico en VHDL

6.3.2.1 DESCRIPCIÓN ALGORÍTMICA

Lo que se va a realizar a continuación es la descripción comportamental algorítmica del circuito de la figura anterior, después se realizará la de transferencia entre registros, que sigue siendo comportamental, y por último se verá la descripción estructural mostrando así las diferencias.

La sintaxis del VHDL no es sensible a mayúsculas o minúsculas, por lo que se puede escribir como se prefiera. A lo largo de las explicaciones se pondrán siempre las palabras clave del lenguaje en mayúsculas para distinguirlas de las variables y otros elementos. Esto no significa que durante la descripción de diseños se tenga que hacer así, de hecho resulta más rápido escribir siempre en minúsculas. Se ha hecho así en todos los ejemplos para mayor claridad del código.

En primer lugar, sea el tipo de descripción que sea, hay que definir el símbolo o entidad del circuito. En efecto, lo primero es definir las entradas y salidas del circuito, es decir, la caja negra que lo define. Se le llama entidad porque en la sintaxis de VHDL esta parte se declara con la palabra clave ENTITY. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone a continuación:

Los comentarios empiezan con dos guiones

```
ENTITY mux IS
PORT   ( a:  IN bit;
         b:  IN bit;
         selec: IN bit;
         salida: OUT bit);
END mux;
```

Esta porción del lenguaje indica que la entidad mux (que es el nombre que se le ha dado al circuito) tiene tres entradas de tipo bit y una salida también del tipo bit. Los tipos de las entradas y salidas se verán más adelante. El tipo bit simplemente indica una línea que puede tomar los valores 'O' ó '1'.

La entidad de un circuito es única. Sin embargo, se mostró que un mismo símbolo, en este caso entidad, podía tener varias vistas, que en el caso de VHDL se llaman

arquitecturas. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra comportamental, ambas son descripciones diferentes, pero ambas corresponden al mismo Circuito, Símbolo, o entidad. Se muestra a continuación la descripción comportamental del multiplexor:

```
ARCHITECTURE comportamental OF mux IS
BEGIN
        PROCESS (a, b, selec)
        BEGIN
                IF (selec='O') THEN
                    salida<=a;
ELSE
                    salida<=b;
                END IF;
        END PROCESS;
END comportamental;
```

La lista sensible es una lista de señales que se suele poner junto a la palabra clave PROCESS, y en el caso del ejemplo es (*a,b, selec*) .Esta descripción comportamental es muy sencilla de entender, ya que sigue una estructura parecida a los lenguajes de programación convencionales. Es por lo que se dice que se trata .de una descripción comportamental algorítmica. Lo que se está indicando es simplemente que si la señal *selec* es cero, entonces la salida es la entrada a, y si *selec* es uno, entonces la salida es la entrada b. Esta forma tan sencilla de describir el circuito permite a ciertas herramientas sintetizar el diseño a partir de una descripción comportamental como la que se acaba de mostrar. La diferencia con un Netlist es directa: en una descripción comportamental no se están indicando ni los componentes ni sus interconexiones, sino simplemente lo que hace, es decir, su comportamiento o funcionamiento.

6.3.2.2 DESCRIPCIÓN FLUJO DE DATOS

La descripción anterior era puramente comportamental, de manera que con una secuencia sencilla de instrucciones se podría describir el circuito. Naturalmente, a veces resulta más interesante describir el circuito de forma que esté más cercano a una posible realización física del mismo. En ese sentido VHDL posee una forma de describir circuitos que además permite la paralelización de instrucciones y que se encuentra más cercana a una descripción estructural del mismo, siendo todavía una descripción funcional. A continuación se muestran dos ejemplos de una descripción concurrente, también llamada de flujo de datos o de transferencia entre registros:

```
ARCHITECTURE flujo1 OF mux IS
SIGNAL nosel,ax,bx: bit;
BEGIN
    nosel<=NOT selec;
    Ax<=a AND nosel;
    Bx<=b AND selec;
    salida<=ax OR bx;
END flujo1;
```

```
ARCH1TECTURE flujo2 OF mux IS
BEG1N
    salida<=a WHEN selec='O' ELSE
            b;
END flujo2;
```

En la descripción de la izquierda hay varias instrucciones todas ellas concurrentes, es decir, se ejecutan cada vez que cambia alguna de las señales que intervienen en la asignación. Este primer caso es casi una descripción estructural, ya que de alguna manera se están describiendo las señales (cables) y los componentes que la definen; aunque no es estructural, ya que en realidad se trata de asignaciones a señales y no una lista de componentes y conexiones. El segundo caso (derecha) es también una descripción de flujo de datos, aunque basta una única instrucción de asignación para definir el circuito.

6.3.2.3 DESCRIPCIÓN ESTRUCTURAL

Aunque no es la característica más interesante del VHDL, también permite ser usado como Netlist o lenguaje de descripción de estructura. En este caso esta estructura también estaría indicada dentro de un bloque de arquitectura, aunque la sintaxis interna es completamente diferente:

```
ARCHITECTURE estructura OF mux IS
SIGNAL ax, bx, nosel: bit;
BEGIN
        u0:    ENTITY inv PORT MAP (e=> selec, y => nosel);
        ul:    ENTITY and2 PORT MAP (el=>a, e2=>nosel, y =>ax);
        u2:    ENTITY and2 PORT MAP (b, selec, bx) ;
        u3:    ENTITY or2 PORT MAP (el=>ax, e2=>bx, y =>salida);
END estructura;
```

Se observa fácilmente que esta descripción es un poco más larga y mucho menos clara que las anteriores. En el cuerpo de la arquitectura se hace lo que en un netlist normal, es decir, se ponen los componentes y sus interconexiones. Para los componentes se utilizarán entidades que estarán definidas en alguna biblioteca, y para la conexiones se usarán señales que se declararán al principio de la arquitectura.

Al igual que ocurre en cualquier netlist, las señales o conexiones deben tener un nombre. En el esquema se le han puesto nombres a las líneas de conexión internas al circuito. Estas líneas hay que declararlas como SIGNAL en el cuerpo de la arquitectura y delante del BEGIN. Una vez declaradas las señales que intervienen se procede a conectar entre sí las señales y entidades que representan componentes. Para ello la sintaxis es muy simple. Lo primero es identificar y poner cada componente, que es lo que comúnmente se conoce como replicación, es decir, asignarle a cada componente concreto un símbolo. En este ejemplo se le ha llamado u a cada componente y se le ha añadido un número para distinguirlos, en principio el nombre puede ser cualquier identificador válido y la única condición es que no haya dos nombres iguales. A continuación del nombre viene la entidad correspondiente al componente, que en este caso puede ser una and2, una or2 o una compuerta inversora inv. Después se realizan las conexiones poniendo cada señal en

su lugar correspondiente con las palabras PORT MAP. Así, los dos primeros argumentos en el caso de la compuerta and2 son las entradas, y el último es la salida. De esta forma tan simple se va creando el netlist o definición de la estructura.

Aunque los dos primeros ejemplos se ajustan al VHDL'87 y al VHDL'93, el último sólo es válido para el VHDL'93. En la primera versión de VHDL no se permite la referencia directa a la entidad, por lo que se hace necesaria la definición de un componente y luego el enlace, en un bloque de configuración, del componente con la entidad que se quiera. Como es mucho más largo, y en este ejemplo sólo se pretendían mostrar las posibilidades del VHDL como lenguaje puramente estructural, se ha preferido utilizar esta última forma que además es más moderna y actual.

6.3.3 VHDL'87 Y VHDL'93

VHDL es efectivamente un lenguaje estándar. Sin embargo, y como ocurre en la mayoría de lenguajes estándar, existen varias versiones. .

Después de años de trabajo y definición apareció el primer VHDL estándar, fue en el año 1987 y de ahí su nombre, VHDL' 87. Esta primera especificación fue estandarizada por el IEEE y su número de registro fue el1076, así que la referencia correcta de esta versión es IEEE Std 1076-1987.

Esta primera versión pronto mostró algunas carencias, especialmente en lo relacionado con la síntesis de circuitos. Esto fue debido sobre todo a la rápida evolución de las herramientas que utilizaban el VHDL, y también a la diseminación del lenguaje en toda la comunidad científica. Durante varios años se recogieron las experiencias obtenidas del uso del primer estándar y se planteó un segundo que sigue vigente hoy día. Fue en 1993 y se trata del IEEE Std-1076-1993, que es conocido comúnmente como. VHDL'93.

Ambas especificaciones son bastante parecidas y sólo tienen pequeñas diferencias. Se puede decir que el VHDL'87 es un subconjunto del VHDL'93, de manera que cualquier descripción antigua va a ser entendida por herramientas nuevas que, en general, poseen la

capacidad de entender las nuevas construcciones del VHDL'93. Una excepción a esta regla general se encuentra en la declaración y uso de los archivos.

Se han ido comentando las diferencias entre el VHDL'87 y el VHDL'93, pero si bien no se resaltan todas, sí que se han escogido aquellas que van a ser más importantes. En general, se puede decir que es más sencillo utilizar la versión del 93 ya que impone menos restricciones, pero no todas las herramientas actuales de simulación o síntesis están todavía preparadas para entenderlo.

En la siguiente lista se enumeran la mayoría de las incorporaciones realizadas al VHDL'93 frente a la versión del 87:

- Se ha dado consistencia a la delimitación de varias estructuras, en concreto a las entidades, arquitecturas, configuraciones, paquetes, componentes, procedimientos, funciones, procesos, secuencias de generación, sentencias condicionales y de selección, bucles y registros.
- Referencia directa de entidades y configuración, lo que hace innecesario el uso de los componentes.
- Se añade la posibilidad de abrir y cerrar archivos, así como la de abrir para añadir.
- Es posible el uso de variables compartidas.
- Se definen nuevos atributos como 'image y 'path_name.
- Sentencia de informe report para escribir mensajes.
- Nuevos operadores que incluyen todos los de rotación y desplazamiento, además del XNOR.
- Generalización de las formas Octal y Hexadecimal, que sólo se podían usar con el tipo bit-vector, y que ahora también sirven para el tipo std_logic_vector.
- Incorporación de la posibilidad UNAFFECTED en las asignaciones condicionales.
- Los puertos en modo entrada (**IN**) se pueden conectar a constantes en el PORT MAP.
- Se añade la posibilidad REJECT para las asignaciones a señal con retraso inercial.
- Se admite la posibilidad de posponer la ejecución de procesos al final del ciclo de simulación, son los procesos POSTPONED.
- Las funciones se pueden declarar como puras o impuras.
- Se pueden realizar declaraciones dentro de las sentencias de referencia.

- A cualquier elemento sintáctico se le puede poner un alias.
- Se añaden los grupos para permitir el paso de información entre herramientas.
- Se permiten etiquetas para las sentencias en serie.
- Los identificadores pueden contener casi cualquier carácter imprimible.
- Los archivos se usan y declaran de forma diferente.

6.3.4 METODOLOGÍA DE DISEÑO UTILIZANDO VHDL

1. Definición de los requerimientos del sistema.

Antes de comenzar a realizar la descripción del diseño, es muy importante que se tenga una idea clara de los objetivos y requerimientos. Tales como: funciones del circuito, máxima frecuencia de operación, y los puntos críticos del sistema.

Esto servirá para poder definir a grandes rasgos cual será la arquitectura del circuito y así comenzar a realizar la descripción.

2. Descripción del circuito en VHDL.

Antes de comenzar a escribir el código es recomendable seleccionar alguna metodología de diseño como: Top-Down, Bottom-Up, o Flat. Los dos primeros involucran la creación de diseños jerárquicos que generalmente son grandes, y el último es utilizado normalmente en el diseño de circuitos pequeños.

La metodología Top-Down consiste en dividir el sistema en varios bloques de tal manera que se puedan resolver los problemas por separado, además, cada bloque a su vez se puede dividir en otros bloques si es necesario. El objetivo es que cada bloque tenga una función especifica representada mediante un componente que desempeñe dicha función. Bottom-Up es todo lo contrario, comenzamos por caracterizar los componentes básicos del circuito y con estos formamos bloques de mayor tamaño que representen un circuito más complejo que sus partes individuales. La metodología Flat es comúnmente utilizada

para diseños pequeños, donde los requerimientos son pocos y no muy complejos por lo que no nos distraen y no perdemos de vista la funcionalidad del circuito. Este método de diseño es el que utilizamos cotidianamente en el diseño de circuitos digitales, y se le llama Flat por que no es necesario seccionar el circuito para poder diseñarlo. Después de decidir cual será la metodología que debemos implementar entonces comenzamos a describir el circuito de acuerdo con lo que se había establecido.

Es recomendable utilizar algún tipo de diagrama a bloques con la descripción del funcionamiento de cada bloque, diagramas de estado, o usar alguna tabla de funcionamiento donde se resumen las funciones de cada bloque en particular. Obviamente existe la posibilidad de cometer errores en VHDL, pero generalmente estos son de son de sintaxis, como ";" al final de cada instrucción, o simplemente por no utilizar adecuadamente alguna instrucción. Algunas ocasiones se podrán tener problemas al tratar de sintetizar el código y esto se debe a que se comete el error de pensar en términos de programación en vez de enfocarnos en la descripción del circuito. Cuando se utiliza VHDL el objetivo principal es el diseño de hardware y para ello debemos de utilizar técnicas de síntesis apropiadas al lenguaje, ya que se suele cometer el error de comenzar a programar en vez de describir y esto provoca que nos olvidemos del objetivo que es el hardware.

La clave para describir y sintetizar fácilmente circuitos digitales con vhdl es pensar en términos de compuertas y registros y no en función de variables y subrutinas.

3. Simulación de la descripción en VHDL.

La simulación del código, o simulación funcional, nos permite detectar y corregir errores antes que se implemente en el dispositivo. La modularidad implementada facilita la evaluación del circuito, porque al describir el circuito por bloques podemos analizar cada uno por separado antes de unirlos. Esta simulación equivale a la depuración de programas en los lenguajes de computación.

4.- Síntesis.

Síntesis consiste en reducir una descripción realizada en un lenguaje de alto nivel de abstracción a un nivel de compuerta que pueda ser implementada en un circuito. Dicho de otra manera, síntesis es el proceso mediante el cual una descripción es convertida en un listado de conexiones (netlist) entre las compuertas, registros, multiplexores, etc. de un dispositivo lógico programable. Por ejemplo, una compuerta XOR puede ser sustituido por su equivalente: A XOR B = A'B+AB', o una instrucción IF puede ser en algunas ocasiones una compuerta AND, en otras

7. RECURSOS PARA LA IMPEMENTACIÓN

7.1 DISPOSITIVO

Descripción funcional:

El CY7C372i es un Dispositivo lógico programable complejo (CPLD), este circuito es eléctricamente borrable y reprogramable en sistema (ISR) y es parte de la familia FLASH370i de CPLD de alta funcionalidad y alta velocidad. La función ISR de Cypress, es implementada a través de 4 terminales de una interface serial. Los datos son desplazados y manejados (entrada y salida) a través de las terminales SDI y SDO respectivamente, usando la terminal del voltaje de programación (Vpp).

Las 64 macroceldas dentro del circuito CY7C372i están divididas en cuatro bloques lógicos. Cada bloque incluye 16 macroceldas, a 72 x 86 arreglos de productos de términos.

Los bloques lógicos dentro de la arquitectura FLASH370i, son conectados a través de un recurso extremadamente rápido llamado PIM (Matriz de interconexión Programable).

Como todos los miembros de la familia FLASH370i, el CY7C372i es rico en recursos. Cada dos macroceldas en el dispositivo cuentan con terminales de E/S, resultando un total de 32 terminales de E/S del circuito. Además de cuatro entradas dedicadas y dos entradas de reloj.

Finalmente, entre las características del CY7C372i se encuentra un sencillo modelo de tiempos. Como en otras arquitecturas de CPLD de alta funcionalidad, no presenta grandes retardos en la velocidad del circuito, así como efectos fanout, retardos de interconexión o retardos de expansión. Dependiendo del número de recursos utilizados o del tipo de aplicación, los parámetros de tiempo en el CY7C372i suelen ser los mismos.

Características del circuito:

64 macroceldas distribuidas en cuatro bloques lógicos

32 terminales de entrada / salida

6 entradas dedicadas, incluyendo 2 terminales de reloj

Reprogramable en sistema (ISR)

- Tecnología flash
- Interface JTAG

Alta velocidad

- F_{MAX}= 125 MHz
- T_{PD}= 10ns
- T_S= 5.5 ns
- T_{CO}= 6.5 ns

7.2 SOFTWARE

ISR Release 2.2 In-System Reprogrammable, Software for CPLDs CYPRESS.

Warp R6.3 creada por Cypress Semiconductor.

7.3 REQUISITOS MÍNIMOS PARA LA INSTALACIÓN DEL SOFTWARE

Requisitos de sistema para el software Warp

- PC IBM o equivalente (recomendable clase Pentium)
- 32 MB de RAM (recomendable 64 MB)
- Espacio en disco 110 MB
- CD-ROM
- Windows 98 o Windows NT 4.0
- Número de licencia

Requisitos para el software ISR

- PC IMB, Windows 98
- Un puerto paralelo libre
- Mínimo de 32 MB de RAM
- Aproximadamente 30 MB de espacio en disco duro

7.4 PROGRAMACIÓN DE UN CPLD CON EL USO DE WARP E ISR

7.4.1 DESCRIPCIÓN DEL SOFTWARE (WARP, ISR)

Warp R6.3 es una herramienta para el diseño con lógica programable creada por Cypress Semiconductor. En este software se encuentra la interfaz gráfica *Galaxy*, el simulador *Nova*, las notas técnicas *Release Notes* y la barra de herramientas *Warp Toolbar*.

La interfaz *Galaxy*, nos servirá para editar el programa que se quiera programar al CPLD, también en él se puede realizar la compilación y revisar el correcto funcionamiento antes de efectuar la implementación del chip

Este software es gratuito para las universidades, para solicitarlo puede visitar la página **http://www.cypress.com/**.

ISR Release 2.2 In-System Reprogrammable, Software for CPLDs CYPRESS. El ISR es el utilizado para programar el CPLD. Este convierte el archivo con extensión **.jed** generado por Warp en el archivo con extensión **.bit** que se grabará en el CPLD.

7.4.2 INSTALACIÓN DEL SOFTWARE

Para la instalación del software WARP se seguirán los siguientes pasos:

Abrir la carpeta de WARP

Seleccionar la carpeta con el nombre warp63_pc35_web.zip y dar dos clic's

Ahora ejecute el archivo warp63_pc35_web.exe

Al ejecutar este archivo mostrará una ventana como la siguiente en ella seleccionará la opción **YES**.

A continuación nos mostrará otra ventana en la cual seleccionaremos la opción **NEXT**.

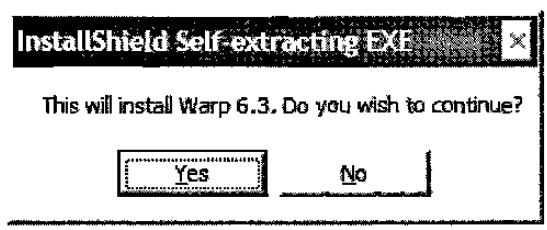


Figura 7.1 Instalación del WARP

En esta sección escribiremos el nombre, compañía y número de serie al terminar.

Nota: Si no se cuenta con el número de serie éste se puede solicitar por internet a la dirección **http://www.cypress.com/**.

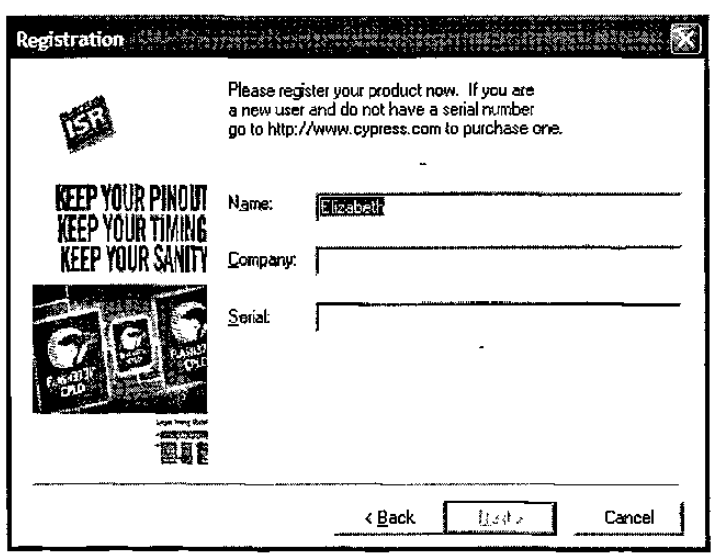


Figura 7.2 Registro

Al instalar el WARP por primera se seleccionará la opción instalar WARP completo.

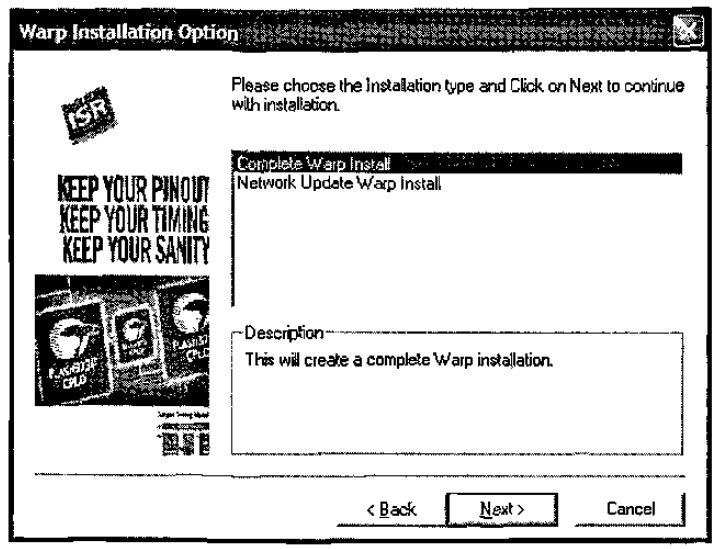


Figura 7.3 Tipo de instalación

Seleccionar la carpeta destino donde se instalará el WARP.

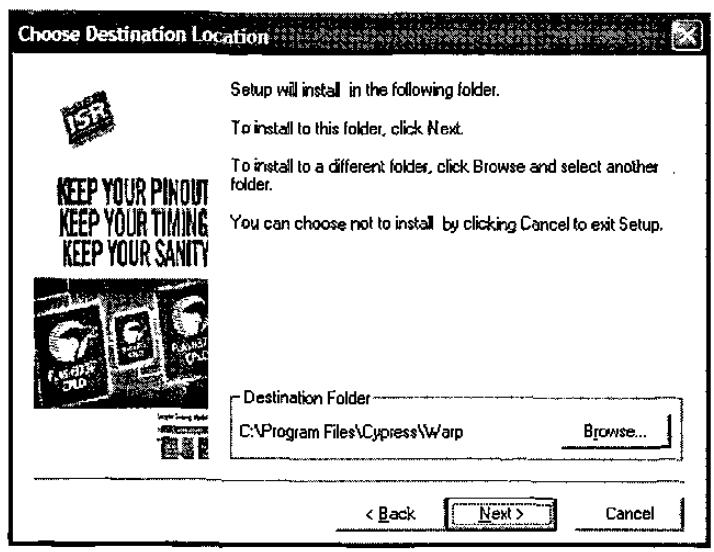


Figura 7.4 Destino del WARP

Si se desea instalar Active-HDL Sim y Active-HDL FSM seleccione la opción yes

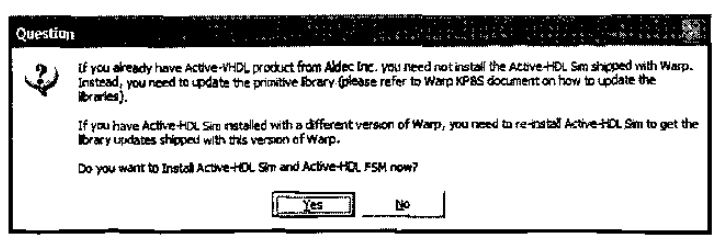


Figura 7.5 Instalar Active-HDL Sim

Empieza la instalación de archivos del WARP

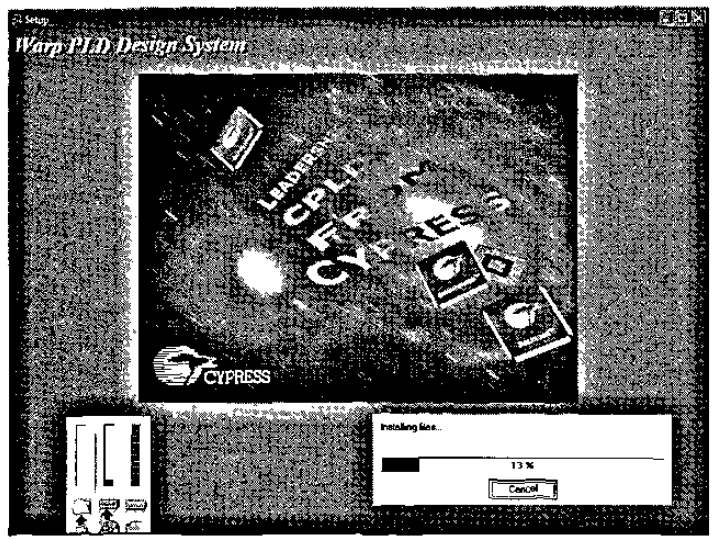


Figura 7.6 Instalación del WARP

En las ventanas siguientes se confirmará la instalación de Active-HDL Sim y la licencia.

Aquí Proporcionaremos la información personal nombre y compañía, al terminar de escribir esto se activará el botón next y daremos un clic en el.

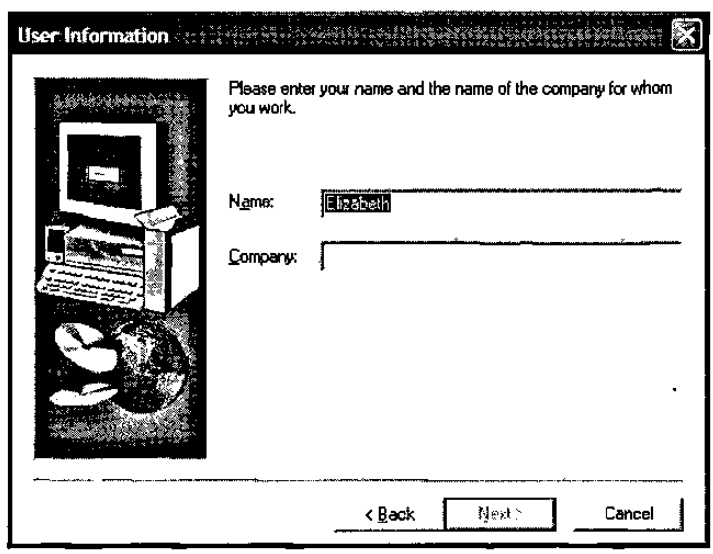


Figura 7.7 Registro personal

Proporcionar la localización del programa Active-HDL Sim.

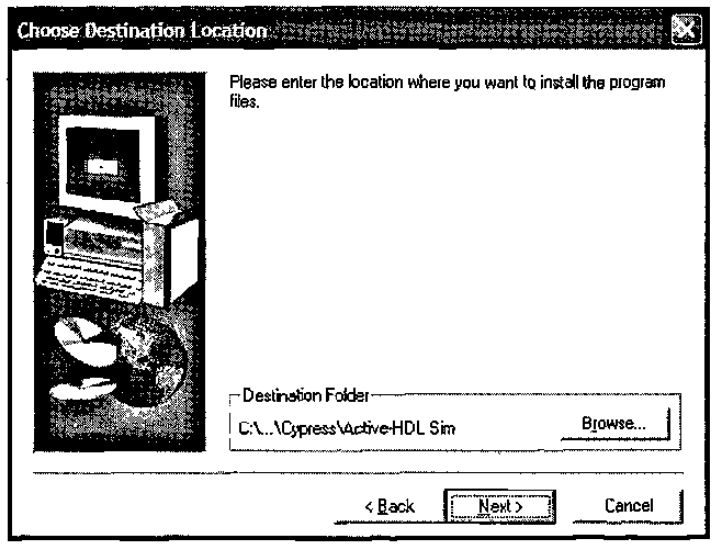


Figura 7.8 Destino de Active-HDL Sim

Seleccionar el tipo de instalación.

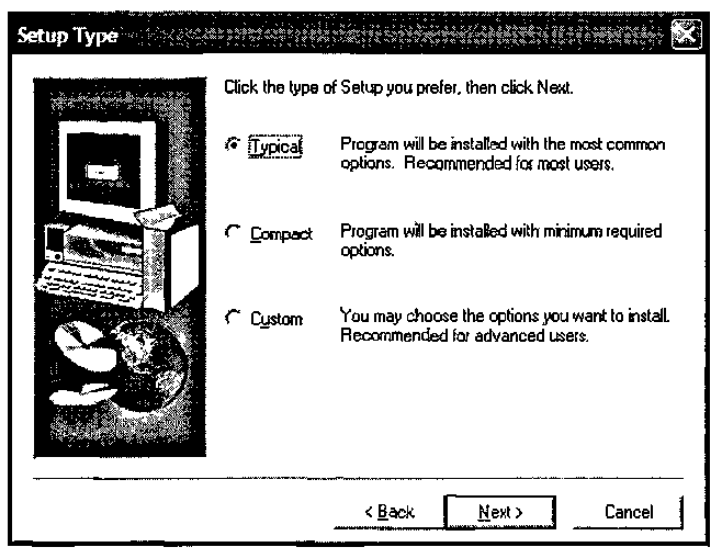


Figura 7.9 Tipo de Instalación

Ya que la información está completa se inicia la copia de archivos.

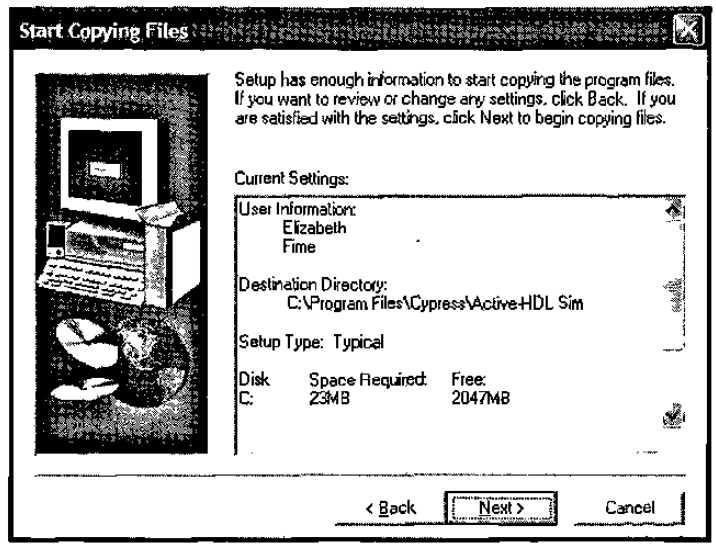


Figura 7.10 Inicio de copia de archivos

Fin de la instalación de Active-HDL Sim.

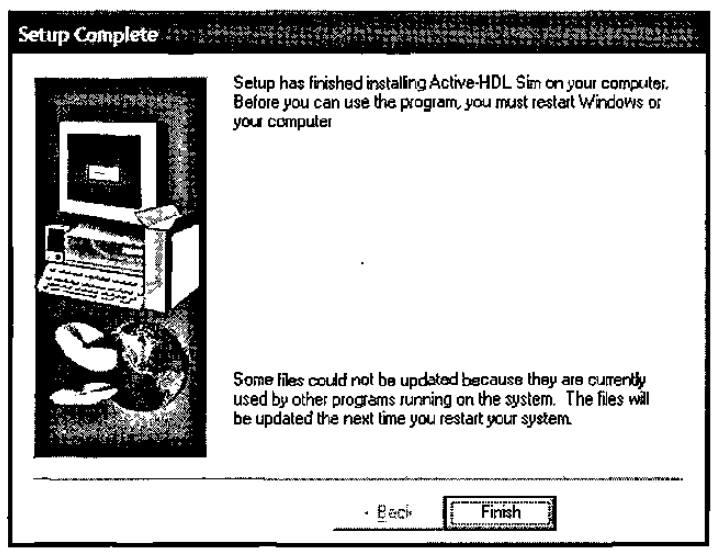


Figura 7.11 Fin de la instalación

Para la instalación del software ISR seguiremos los siguiente pasos:

Ejecutamos el archivo ISR_rel22

Seleccionaremos el archivo Setup.exe y lo ejecutamos

En seguida nos mostrará una ventana como la siguiente y seleccionaremos la opción NEXT

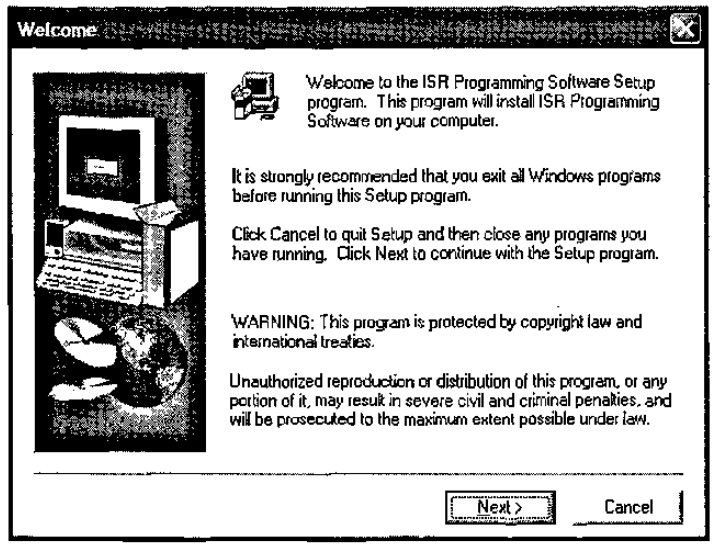


Figura 7.12 Instalación del ISR

Posteriormente nos mostrará la licencia y daremos un clic en la opción yes

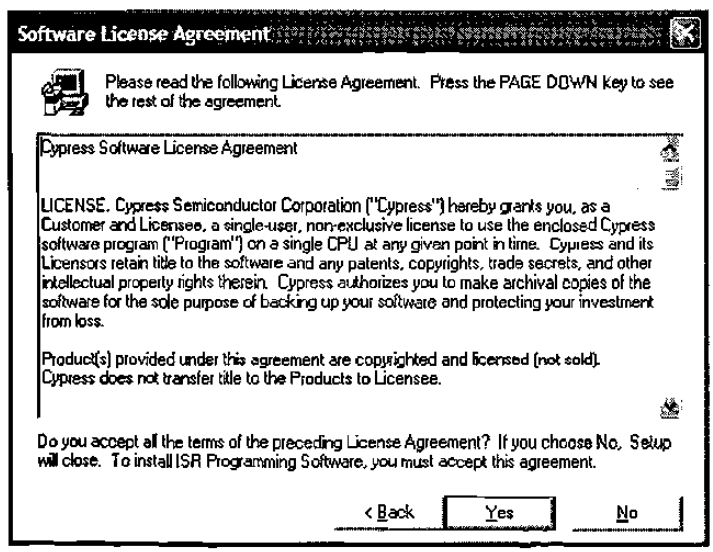


Figura 7.13 Aceptación de licencia del ISR

Ahora tendremos que escoger el destino o locación del software seleccionando el botón browse si no se desea establecer una locación en especial solo dé clic en next

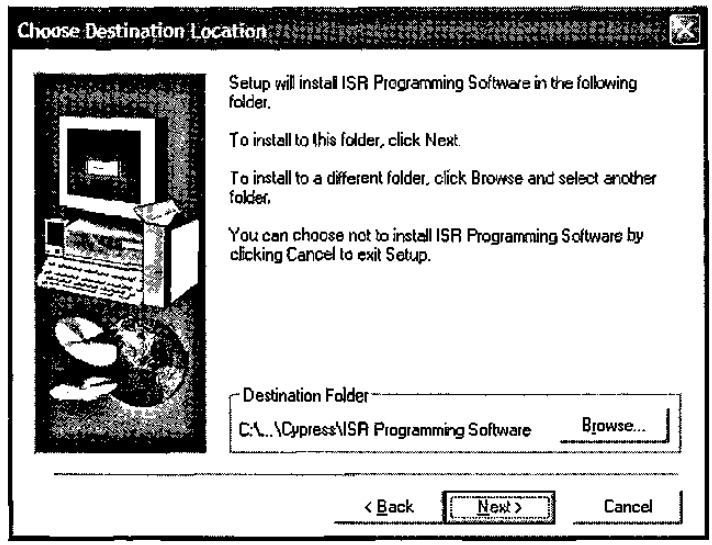


Figura 7.14 Destino de locación del ISR

Ahora seleccionaremos el Fólder del programa en este caso nos iremos directamente a la opción Next. En seguida nos mostrará una ventana en la cual nos indicará que se ha creado el icono del ISR en ella daremos un clic en OK al hacer esto nos mostrará una ventana en la cual nos preguntará si deseamos reiniciar nuestra computadora y daremos un clic en la opción Finish

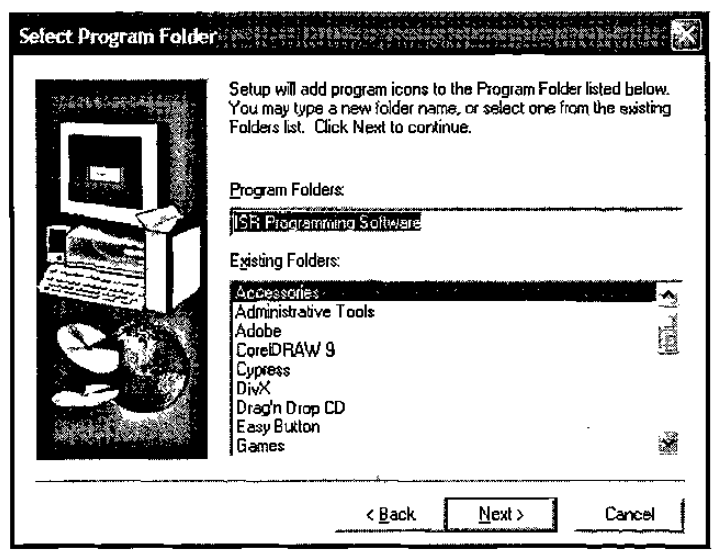


Figura 7.15 Selección del fólder del programa

7.4.3 USO DE LA INTERFAZ GALAXY

7.4.3.1 CREACIÓN DE UN ARCHIVO EN FORMATO VHDL (.VHD)

Pasos para escribir un programa utilizando la interfaz Galaxy.

Para empezar a crear un programa en la interfaz Galaxy buscamos en el menú inicio la opción **Cypress** y seleccionamos **Galaxy**.

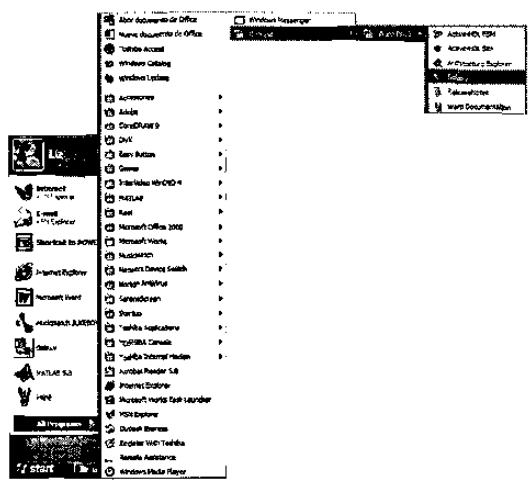


Figura 7.16 Ejecución de Galaxy

Esta es la pantalla de presentación de Galaxy.

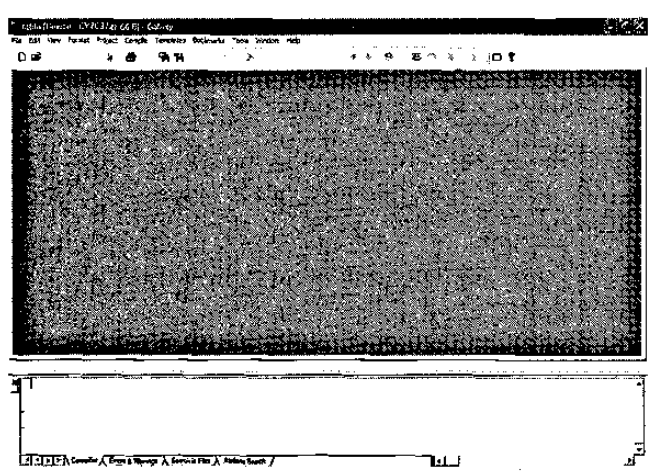


Figura 7.17 Presentación de Galaxy

Ahora seleccionaremos la opción **FILE, NEW**

Al ejecutar esta acción nos mostrará una pantalla como la siguiente, en ella seleccionaremos la opción **TEXT FILE.**

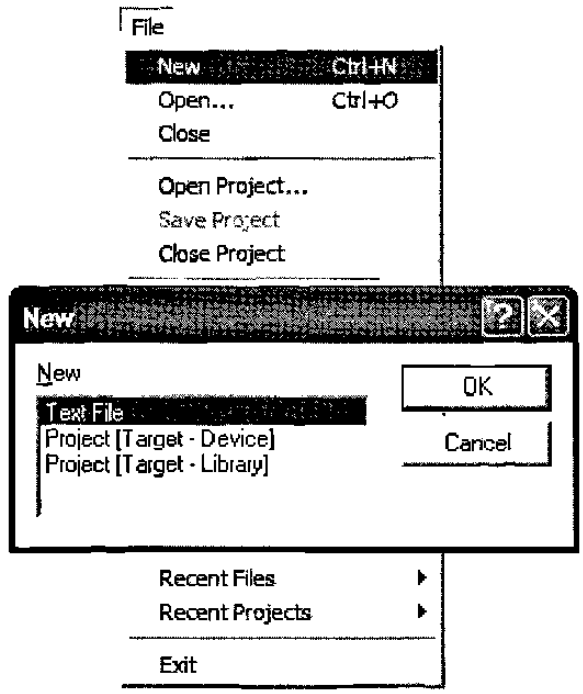


Figura 7.18 Crear un archivo de texto

Al seleccionar esta opción aparecerá una ventana como la mostrada aquí, en ella se escribirá el programa que se quiera crear y se escribirá en la parte superior de la ventana, en la parte inferior nos mostrará el proceso de compilación, los errores del programa al momento de la compilación, búsqueda en archivo y atributos de búsqueda.

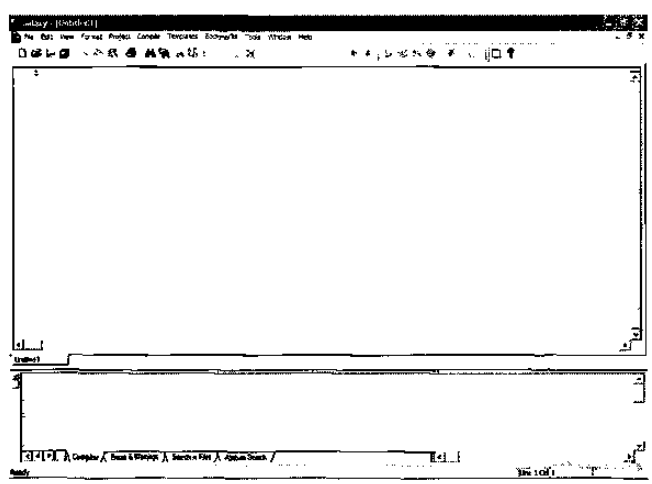


Figura 7.19 Editor de programa

La figura muestra la ventana en donde se puede observar el código del programa.

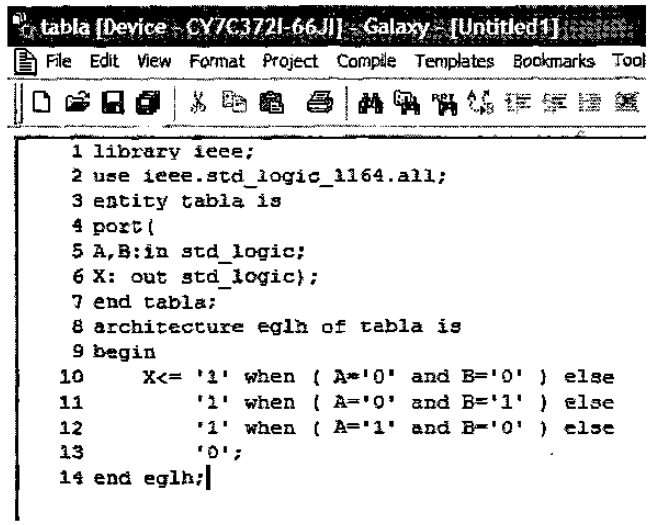


Figura 7.20 Código del programa

Al terminar de escribir el programa lo grabaremos, para ello iremos al menú **FILE** y seleccionaremos la opción **SAVE AS.**

En la ventana que nos muestra escribiremos el nombre al programa es importante que el archivo contenga la extensión **.VHD,** ejemplo **TABLA.VDH,** ya que si el archivo no lleva esta extensión no podremos compilarlo.

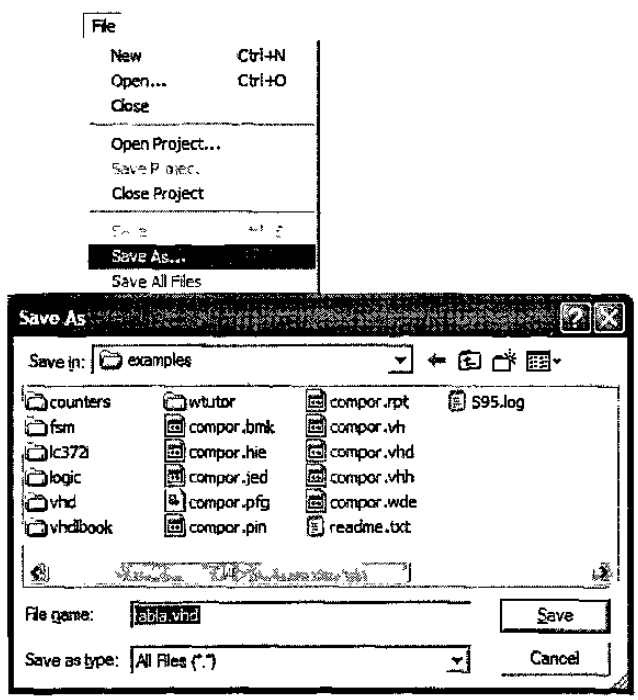


Figura 7.21 Guardar archivo de texto

Note que al guardar el programa la redacción de él muestra los comandos o directivas en letras azules y el archivo generado tiene extensión **.vhd**

Nota: Si antes de empezar a escribir el programa le asignamos el nombre al archivo al ir escribiendo el programa los comandos o directivas irán apareciendo en letras azules.

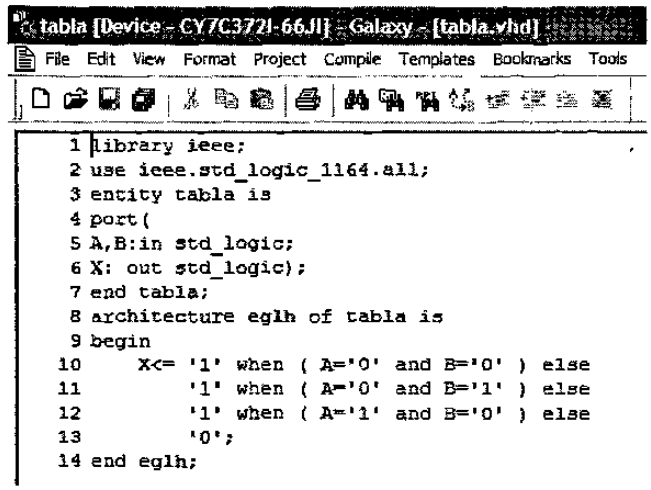


Figura 7.22 Programa editado en formato VHDL

El siguiente paso es crear un proyecto y para poder hacerlo haremos lo siguiente: En el menú **FILE** seleccionaremos la opción **NEW**, en ella seleccionaremos la opción **PROJECT [TARGET – DEVICE]**

NOTA: es aconsejable asignarle el mismo nombre del programa al proyecto.

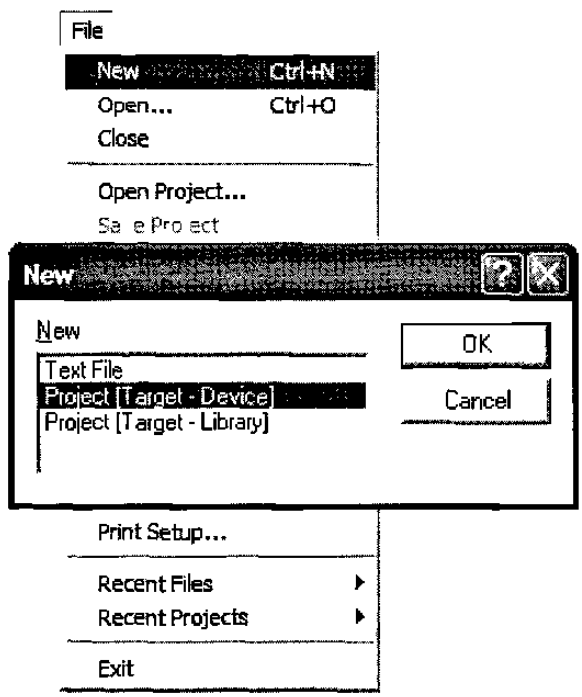


Figura 7.23 Crear un proyecto

Al dar clic en **OK** nos mostrará una ventana como la mostrada a continuación , en ella deberá estar seleccionado el tipo de proyecto VHDL, en la opción **PROYECT NAME** escribiremos el nombre del proyecto que en este caso será *tabla* el cual deberá ir sin extensión, después seleccionaremos la opción **NEXT.**

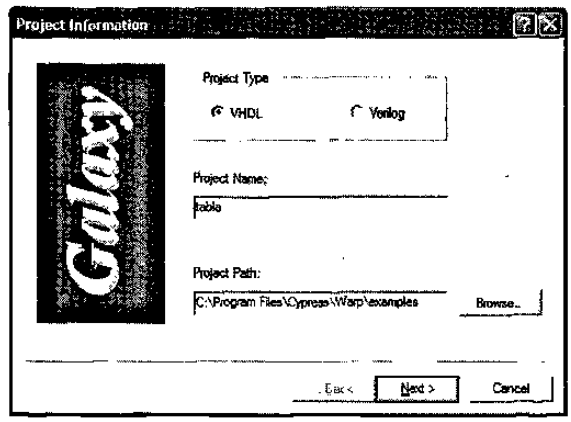


Figura 7.24 Asignar nombre al proyecto

En seguida nos mostrará la siguiente ventana donde seleccionaremos el archivo para añadirlo, este deberá aparecer en el cuadro *FILE IN THE PROJECT DIRECTORY*, daremos un clic sobre el archivo a añadir y luego daremos un clic en el botón *ADD.*

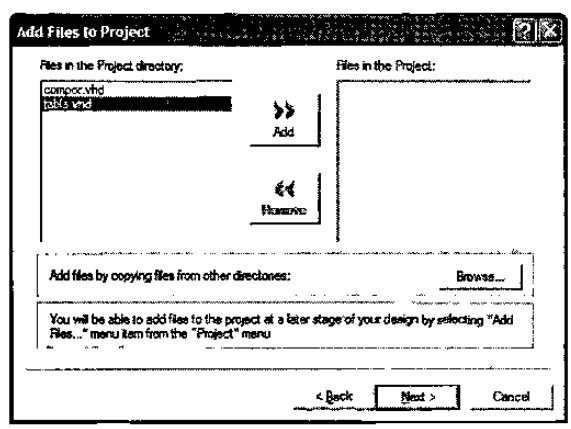


Figura 7.25 Seleccionar archivo a añadir

Al seleccionar el archivo con extensión **.VHD** aparecerá en el cuadro *FILES IN THE PROJECT* como se muestra a continuación, en seguida seleccionaremos la opción **NEXT.**

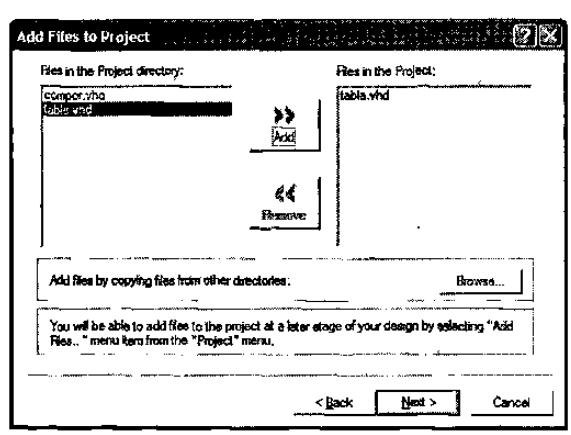


Figura 7.26 Archivo añadido

A continuación nos mostrará una ventana como la siguiente donde seleccionaremos el dispositivo con el cual trabajaremos.

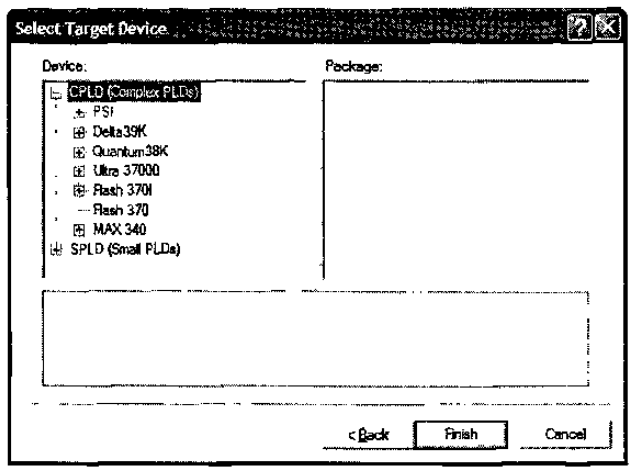


Figura 7.27 Seleccionar dispositivo

En nuestro caso seleccionaremos la opción **FLASH 370I**, para después dar clic en **c372I** y utilizaremos el dispositivo **CY7C372I-66JI**.

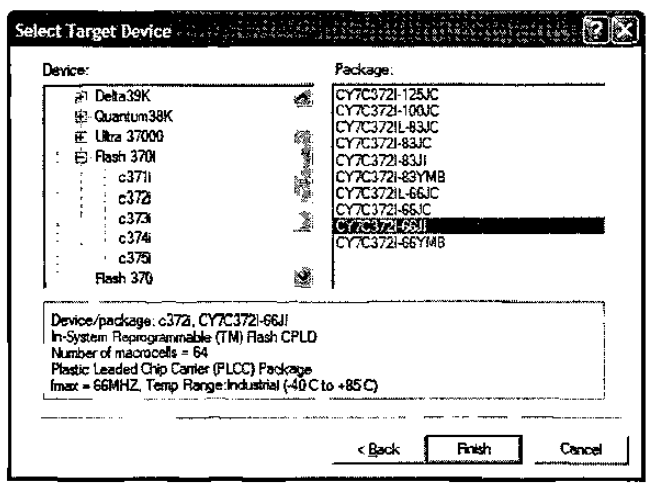


Figura 7.28 Dispositivo seleccionado

Al seleccionar el botón **FINISH** de la ventana anterior aparecerá otra donde nos dice si deseamos guardar el nuevo proyecto en ella seleccionamos la opción **YES**.

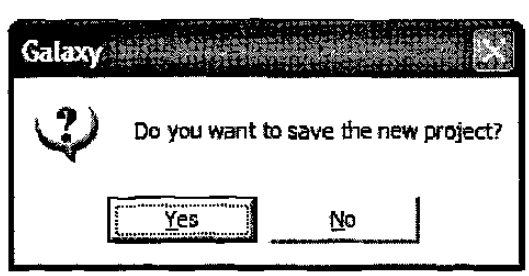


Figura 7.29 Salvar el nuevo proyecto

Ahora si observamos en la parte de arriba de la ventana aparecerá el nombre del proyecto y el nombre del programa.

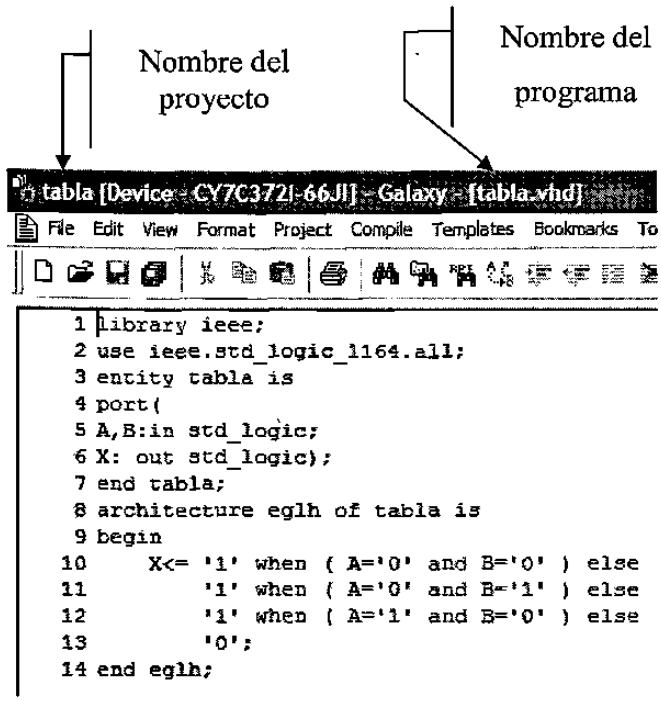


Figura 7.30 Proyecto terminado

7.4.3.2 COMPILACIÓN

Después de todos estos pasos estamos listos para compilar el archivo!, para esto nos iremos al menú principal a la opción **COMPILE** y seleccionaremos la opción **PROJECT**, como se muestra a continuación.

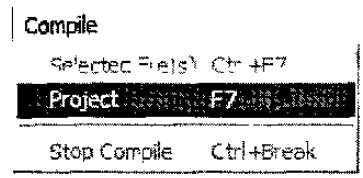


Figura 7.31 Compilación del proyecto

Al ejecutar esta opción en la parte inferior de la pantalla aparecerán los errores de la compilación (si es que los hay).

NOTA: Con la compilación del programa se generan varios archivos con el mismo nombre pero diferente extensión como por ejemplo **.pfg, .jed, .pin, .wde, .vh, .rpt**.

El archivo que en este caso será **tabla.jed** nos servirá para la programación del chip; el software ISR lo convertirá a un archivo con extensión **.bit** y con este archivo programaremos el CPLD

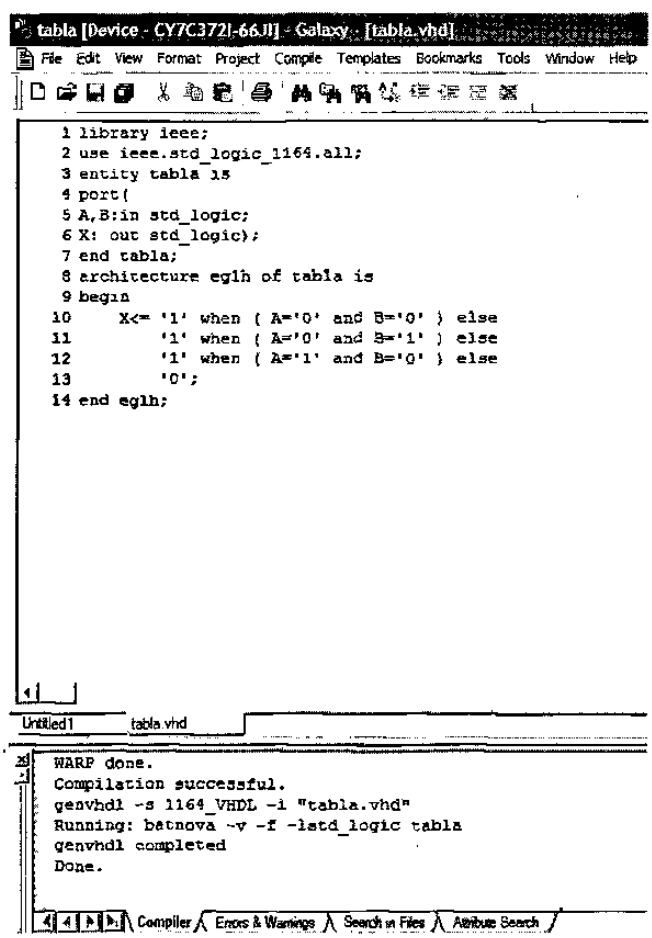


Figura 7.32 Proyecto compilado

Otro archivo importante de la lista de archivos que se generan al efectuar la compilación es el archivo con extensión **.rpt** en este archivo se encontrará un reporte con una serie de información importante como por ejemplo el número de macroceldas que se utilizaron en el chip, la configuración de las terminales para la implementación de éste. Para poder ver este archivo seleccionaremos del menú **VIEW** la opción **REPORT FILE.**

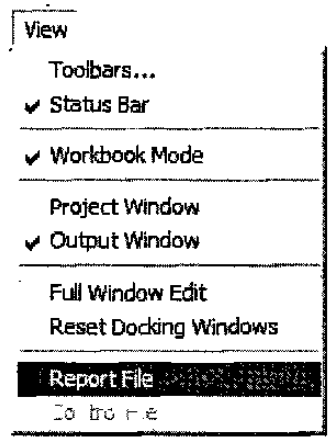


Figura 7.33 Abrir archivo reporte

A continuación se presenta una breve descripción de los archivos generados al momento de la compilación:

Archivo con extensión **.pin**: contiene información de las terminales usadas del dispositivo incluyendo la variable asignada y si es de entrada, salida o bidireccional además la fecha en que fue creado el archivo, los tipos de entrada y salida que se usaron en el programa y el dispositivo que se seleccionó para la programación.

Archivo con extensión **.hie**: contiene la información del número de línea en donde se encuentra la entidad y arquitectura del programa y los nombres asignados a cada una de ellas.

Archivo con extensión **.vh**: contiene la información de las variables ocupadas en el programa y indica el tipo de asignación entradas o salidas.

Archivo con extensión **.vhh**: contiene información del nombre de la entidad y las variables que se ocuparon y los tipos de datos que se le asignaron a cada una de ellas.

Archivo con extensión **.wde** solo contiene la trayectoria del archivo stdlogic.vif.

Aquí se presentan dos secciones del archivo reporte (archivo con extensión **.rpt)** donde se exponen las ecuaciones obtenidas en la compilación del programa y la configuración de las terminales del chip (esta última parte lo encontrará casi en la parte final del archivo reporte).

En el archivo reporte **(.rpt) se** encontrará una bitácora del proceso de compilación.

Al terminar con estos pasos es conveniente que en el menú **FILE** seleccionemos la opción **SAVE ALL FILES,** así quedarán grabados todos los archivos creados hasta este momento.

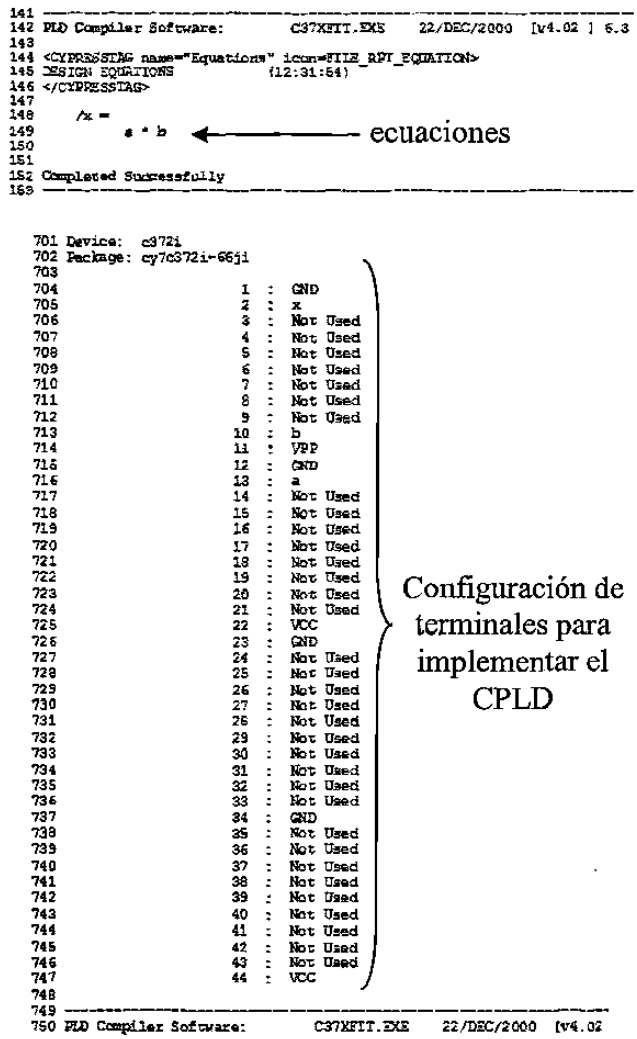


Figura 7.34 Ecuaciones y configuración de terminales

NOTA: En la implementación del circuito se deberá de energizar todas las VCC's y conectar todas las GND's que aparecen en el archivo reporte.

7.4.3.3 SIMULACIÓN

Para efectuar la simulación del programa se seguirán los siguientes pasos:

Dentro de la interfaz Galaxy en el menú **TOOLS** seleccionamos la opción **ACTIVE-HDL-SIM** como se muestra a continuación.

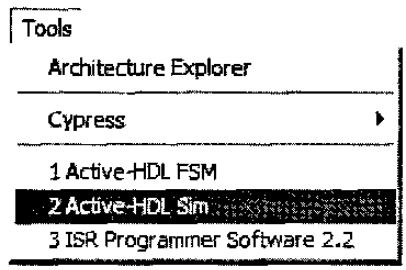


Figura 7.35 Abrir pantalla de simulación

Al seleccionar esta opción nos mostrará la siguiente ventana.

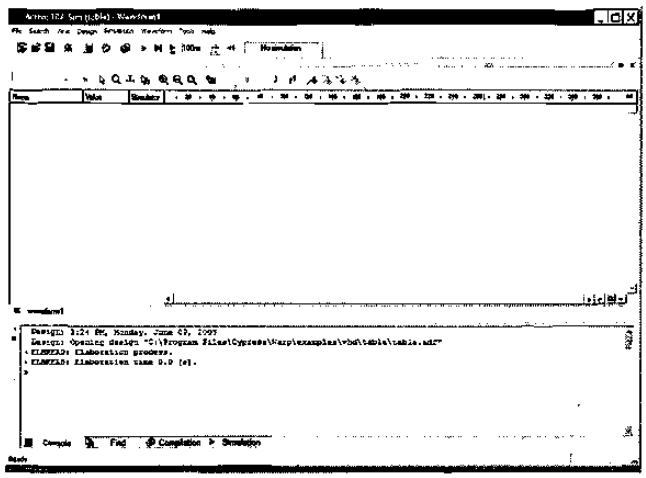


Figura 7.36 Presentación de Active-HDL

Ahora seleccionaremos del menú **FILE** la opción **OPEN VHDL**, esto es para cargar el programa que se editó en la interfaz Galaxy.

Ahora seleccionaremos el programa para cargarlo al simulador (para ejemplo que estamos trabajando seleccionaremos tabla.vhd).

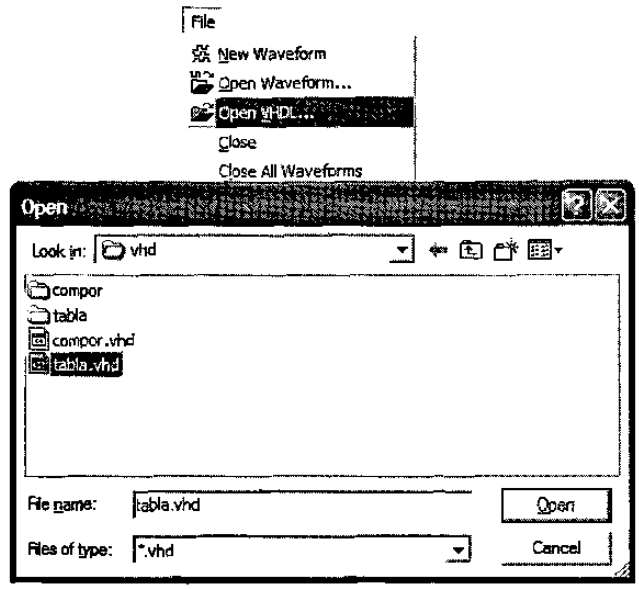


Figura 7.37 Abrir programa en VHD

Ahora nos mostrará en la parte superior de la ventana el nombre del archivo que deseamos simular y en la parte inferior nos muestra que el programa ha sido cargado y los errores que se obtuvieron al momento de la compilación.

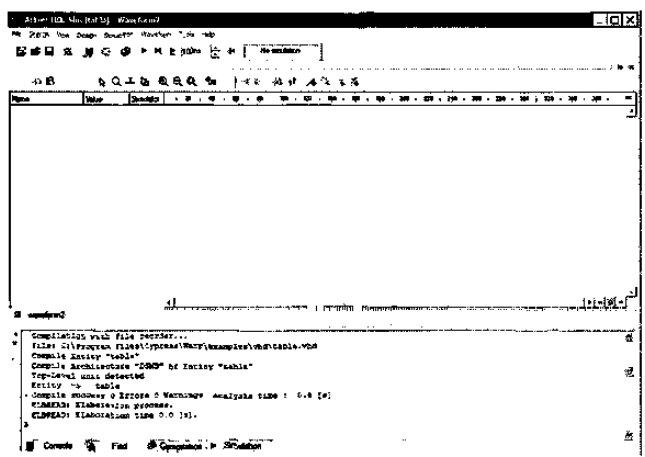


Figura 7.38 Archivo VHD cargado

El siguiente paso sería cargar las señales que se deseen simular para poder realizar esta acción seleccionaremos del menú **WAVEFORM** la opción **ADD SIGNALS**, como se muestra a continuación o si se desea seleccione el icono  de la barra de herramientas **WAVERFORM EDITOR.**

Al seleccionar esta opción nos mostrará una ventana como la mostrada a continuación donde se seleccionarán las variables a simular.

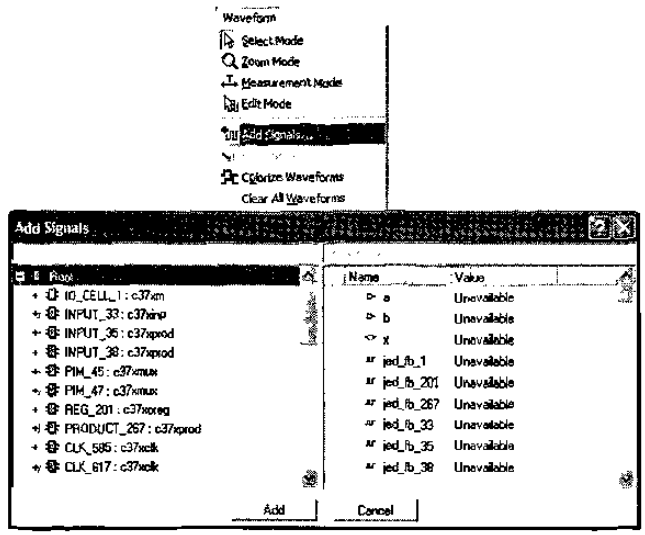


Figura 7.39 Añadir señales

Al ir seleccionando las variables éstas aparecerán en la ventana principal del simulador.

NOTA: Puede seleccionar todas las variables a la vez dando un clic sobre la variable que se desee simular y presionando la tecla SHIFT.

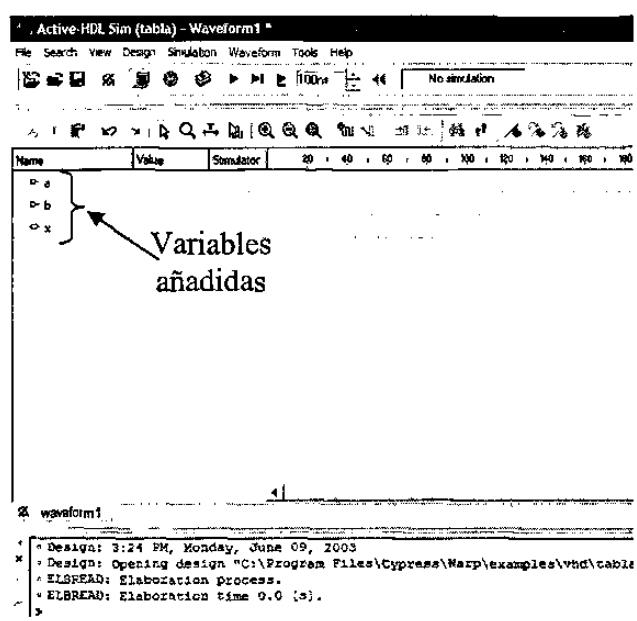


Figura 7.40 Variables añadidas

Para generar el tipo de señal de entrada se posiciona el mouse en la casilla de **VALUE** o **SIMULATOR** y se da un clic al botón del lado derecho del mouse y se selecciona la opción **SIMULATORS**.

Al seleccionar esta opción nos mostrará una ventana como la siguiente, en ella se podrá seleccionar el tipo de entrada que se desee.

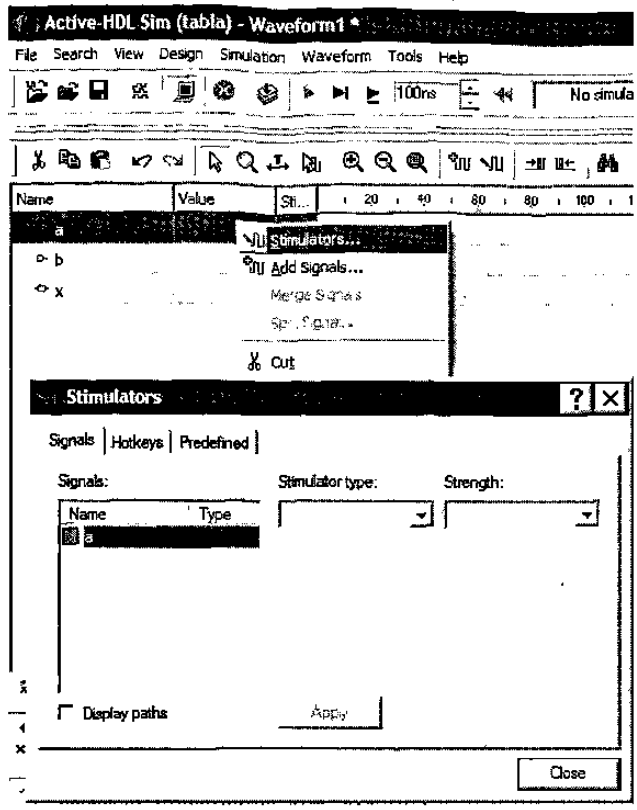


Figura 7.41 Asignar valor a las variables

Para seleccionar el tipo de simulación para la variable "a" nos ubicamos en **SIMULATOR TYPE**, el cual nos permite seleccionar varios tipos de entrada como se muestra a continuación.

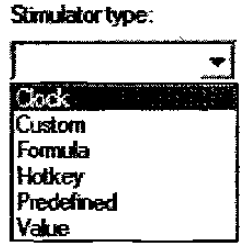


Figura 7.42 Tipo de simulador

En este ejemplo se seleccionaron para las dos entradas "a" y "b" un simulador tipo reloj.

Ejemplo de la selección del tipo de simulación para la entrada "a".

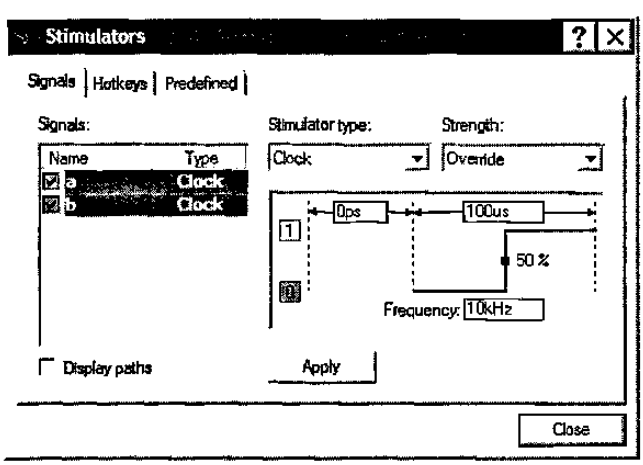


Figura 7.43 Asignación de tipo reloj

A continuación se muestra como quedarán en la ventana donde se efectuará la simulación.

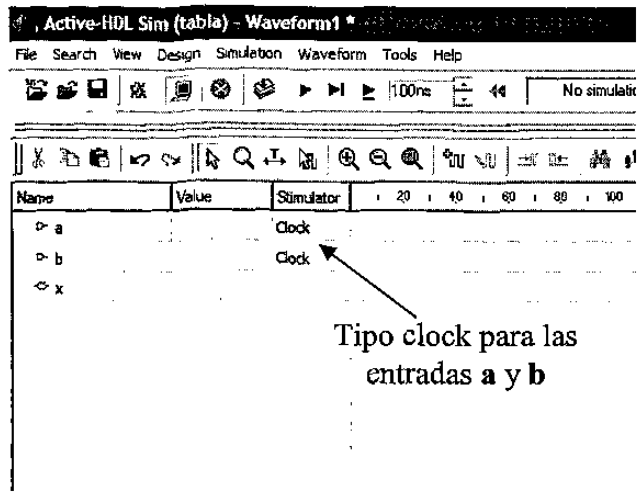


Figura 7.44 Fin de asignación de simulador

En el menú **SIMULATION** seleccionaremos la opción **RUN** para simular el programa.

Nota: También se puede seleccionar el icono ▶ .

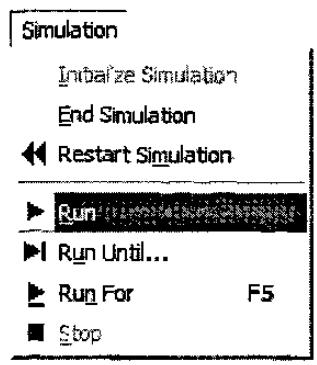


Figura 7.45 Correr simulación

Así se mostrará la simulación.

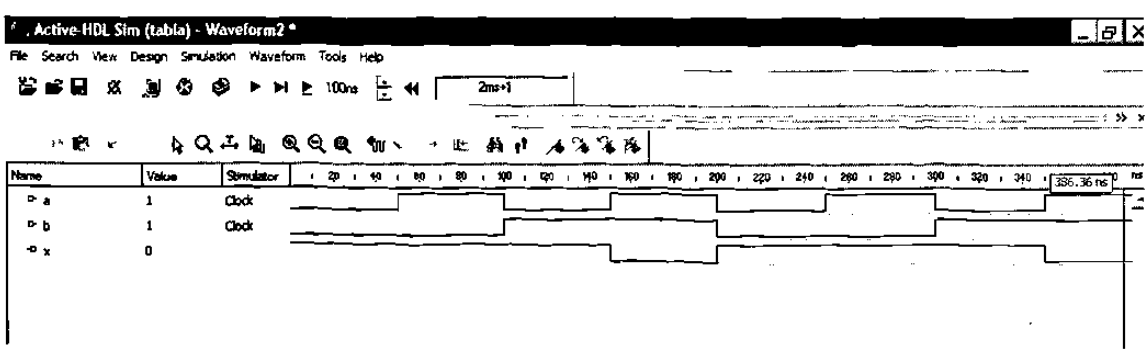


Figura 7.46 Simulación terminada

7.4.4 PROGRAMACIÓN DEL CPLD

Para la programación del CPLD utilizaremos el software ISR

Ahora se tiene un archivo con extensión **.JED** este archivo nos servirá para grabar el CPLD, para esto necesitaremos del uso del programador ISR *(In System Reprogrammable)*, el archivo que utilizaremos se llama **isr.exe**, este convierte el archivo con extensión **.JED** que se generó del WARP a uno con extensión **.BIT**, este archivo es el que se grabará en el CPLD, para lograr esto se tendrán que seguir los siguientes pasos.

Antes de ejecutar el programa es necesario conectar el programador al puerto paralelo de la computadora y asegurarse que el dispositivo CPLD esté insertado correctamente en la base además de proporcionar una alimentación al programador de 5 volts de CD.

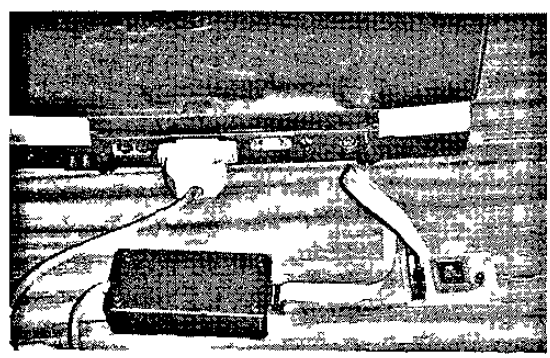


Figura 7.47 Programador conectado al puerto paralelo.

El dispositivo CPLD deberá quedar insertado en la base del programador como se ilustra en la figura 7.48.

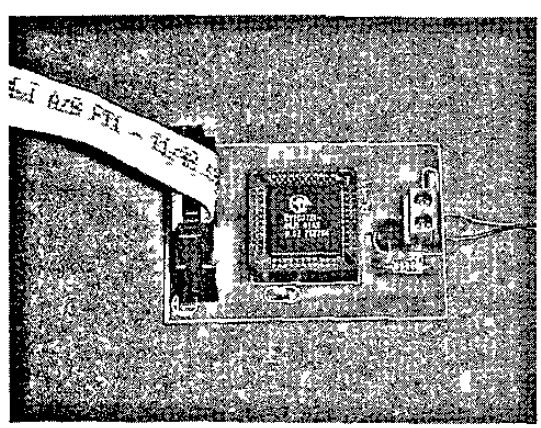


Figura 7.48 CPLD en el programador

Enseguida ejecutaremos el programa *ISR Programming Software 2.2* seleccionando el menú de inicio (start) y luego la opción *ISR Release 2.2*.

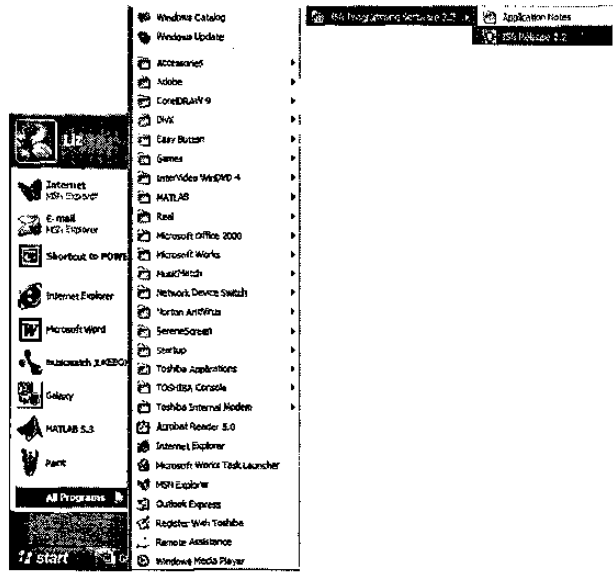


Figura 7.49 Ejecución del ISR

Una vez efectuado el procedimiento anterior nos mostrará la siguiente ventana de presentación:

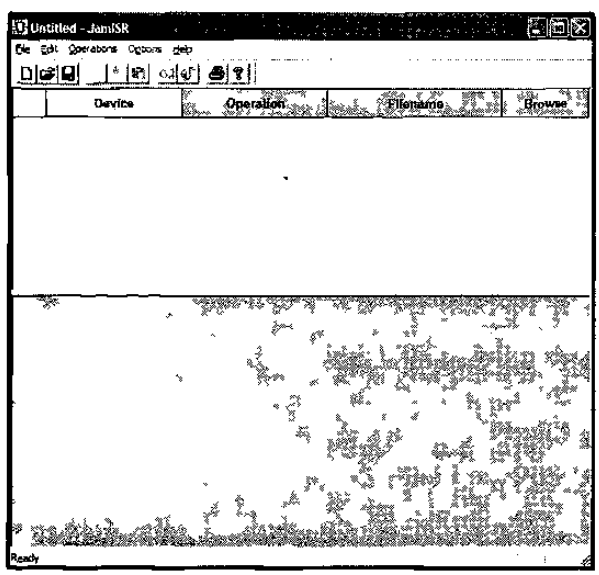


Figura 7.50 Presentación del ISR

Ahora el siguiente paso será crear un archivo con extensión **.ISR**, para esto seleccionamos el menú **FILE** en seguida seleccionamos la opción **NEW**, al hacer esto nos mostrará una ventana como la mostrada a continuación:

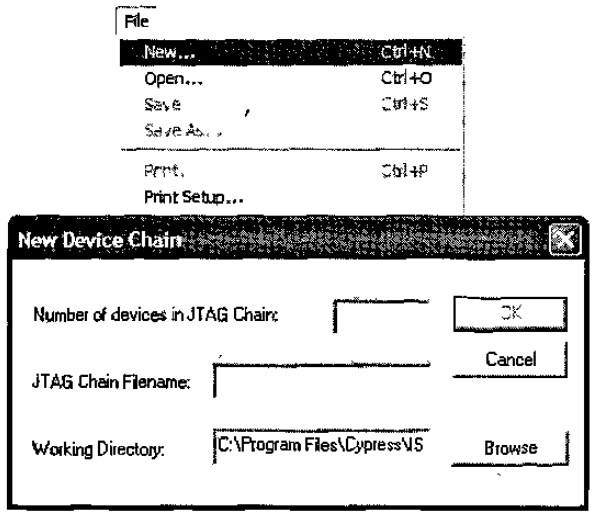


Figura 7.51 Creación de archivo .ISR

En la primera celda escribimos el número de dispositivos a programar, en la segunda el nombre del archivo que se va a cargar en el CPLD y en la última la ubicación del directorio en donde se encuentra el archivo de extensión JED. Para finalizar daremos un clic en OK

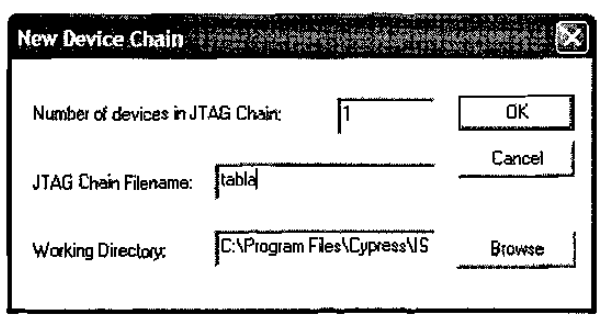


Figura 7.52 Asignación de datos

Ahora la ventana de presentación es la mostrada a continuación en ella daremos un clic en **BROWSE** y nos mostrará una ventana como se muestra a continuación donde seleccionaremos nuestro archivo con extensión **.JED** en seguida daremos clic en el botón **OPEN.**

Figura 7.53 Cargar archivo .JED

Al hacer esto ahora la ventana quedará como se muestra a continuación, note que en **FILENAME**, aparece la ubicación donde se encuentra el archivo **.JED**

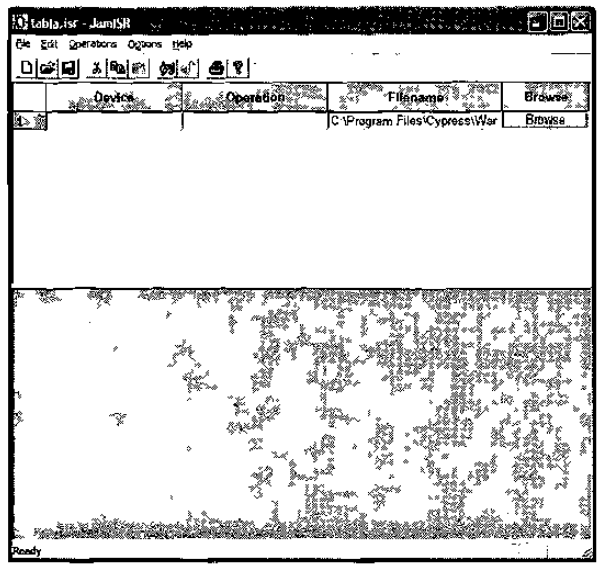


Figura 7.54 Archivo .JED cargado

Para seleccionar el dispositivo daremos un clic en **DEVICE** donde nos mostrará una lista, en ella seleccionaremos el dispositivo de trabajo que en este caso es el CPLD **CY7C372I**

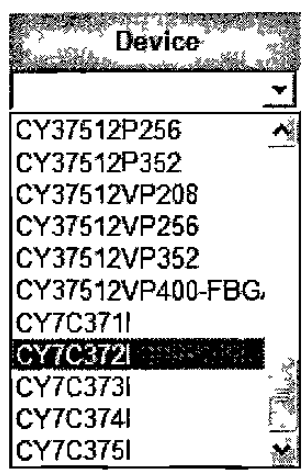


Figura 7.55 Selección de dispositivo a programar

El paso siguiente será ir al menú **OPERATION** y dar un clic y nos mostrará una lista de opciones por ejemplo, borrar el dispositivo, programar y verificar, leer dispositivo, etc.

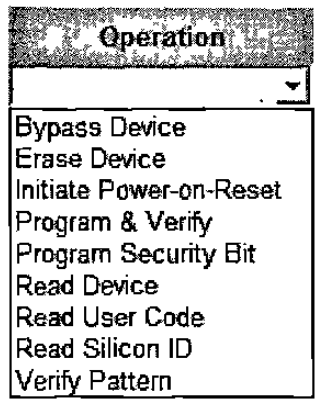


Figura 7.56 Opciones de operación

Para grabar el CPLD seleccionaremos primero la opción **ERASE DEVICE**, con esta opción borraremos el chip y prepararlo para grabar el programa

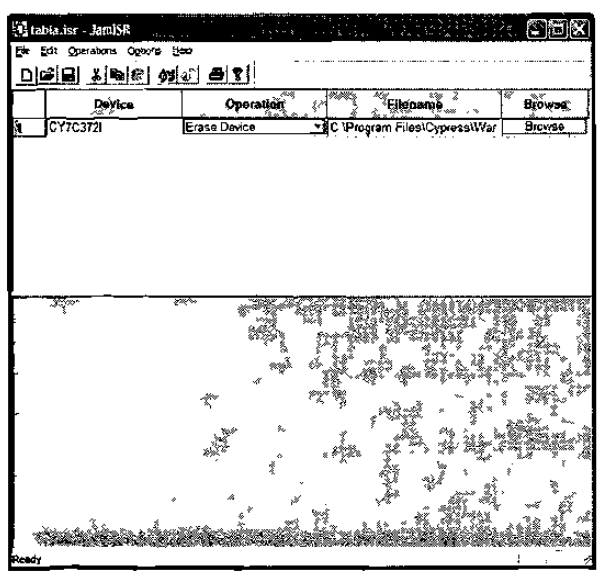


Figura 7.57 Selección de Erase Device

Ahora seleccionaremos del menú **OPERATIONS** la opción **COMPOSE JAM FILES**, o bien daremos un clic en el icono 

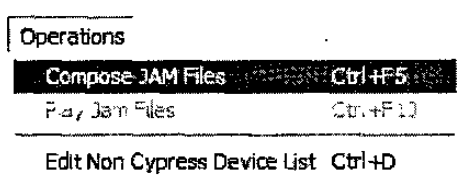


Figura 7.58 Ejecución Compose JAM Files

Si observamos en la parte inferior de la ventana muestra los resultados, nos indica el número de dispositivos programados y los errores.

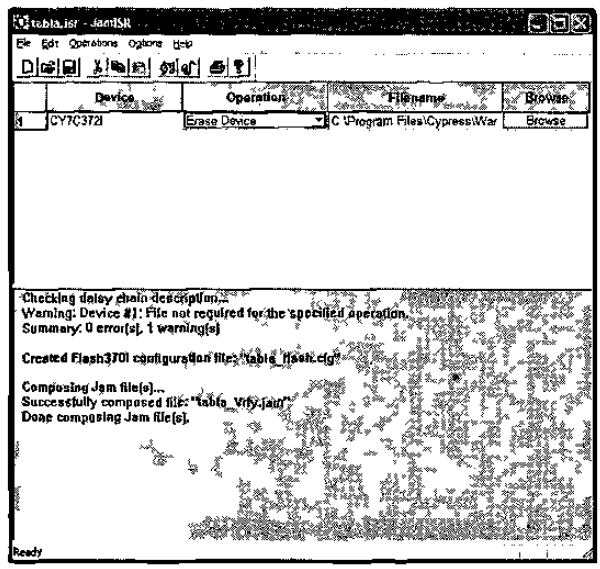


Figura 7.59 Resultados de Compose JAM Files

Después de ejecutar esta acción seleccionaremos del menú **OPERATIONS** la opción **PLAY JAM FILES** o bien daremos un clic en el siguiente icono 

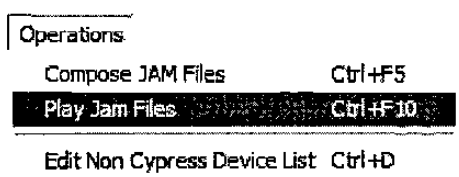


Figura 7.60 Ejecución de Play Jam Files

Al ejecutar esta acción en la parte inferior de la ventana nos muestra los resultados.

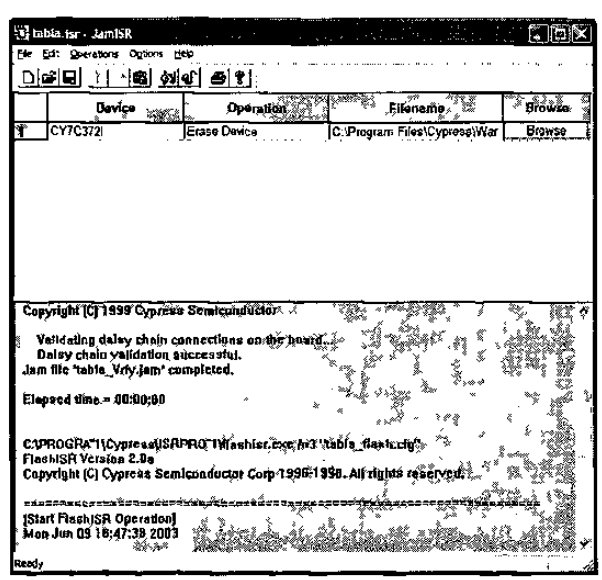


Figura 7.61 Resultados de Play Jam Files

El CPLD está listo para ser grabado, para poder hacerlo seleccionaremos en el menú **OPERATION** la opción **PROGRAM & VERIFY** y se seguirá la misma secuencia que se realizó para borrar el dispositivo (seleccionar *COMPOSE JAM FILES* y en seguida la opción *PLAY JAM FILES*)

NOTA: Esta versión no acepta mas de ocho caracteres en los nombres de los directorios (formato MS-DOS) asegúrese que no excedan este requisitito ya que nos marcaría un error en el procedimiento.

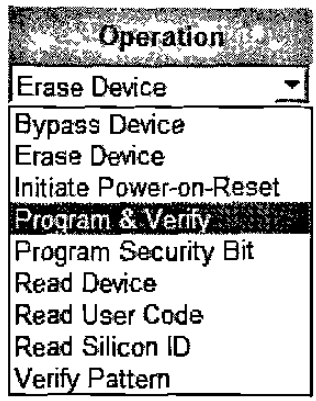


Figura 7.62 Selección de Program & Verify

Aquí se presenta la ventana sin el formato MS-DOS

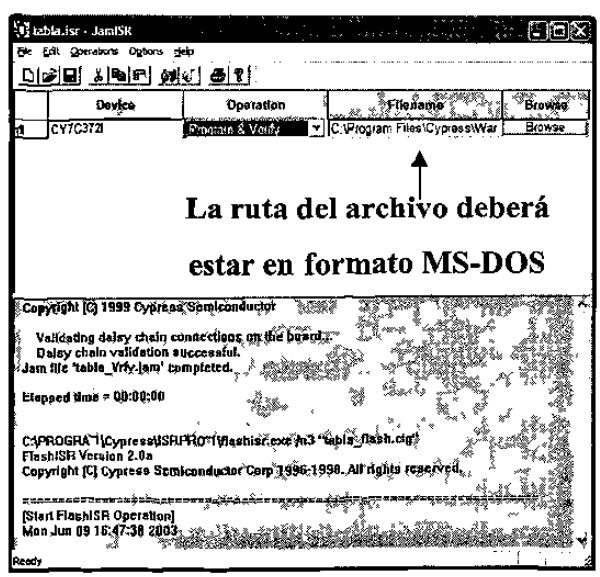


Figura 7.63 Ruta sin formato MS-DOS

Esta es la ventana que nos mostrará al ejecutar *COMPOSE JAM FILES*

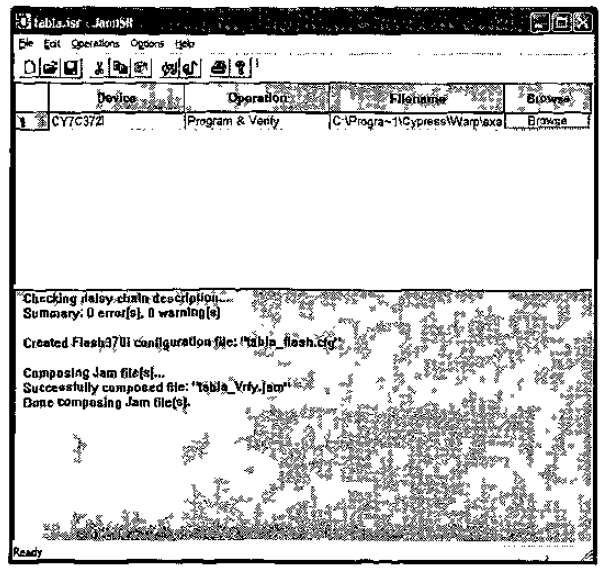


Figura 7.64 Ruta con formato MS-DOS

Esta ventana es la que nos mostrará al ejecutar la opción *PLAY JAM FILES*

El chip ya está programado si observamos en el resumen de la operación el checksum indicará un número diferente de **0x0** esto indica que el dispositivo tiene información programada.

Solo falta implementar el circuito, consultando en el archivo reporte la configuración de las terminales del chip.

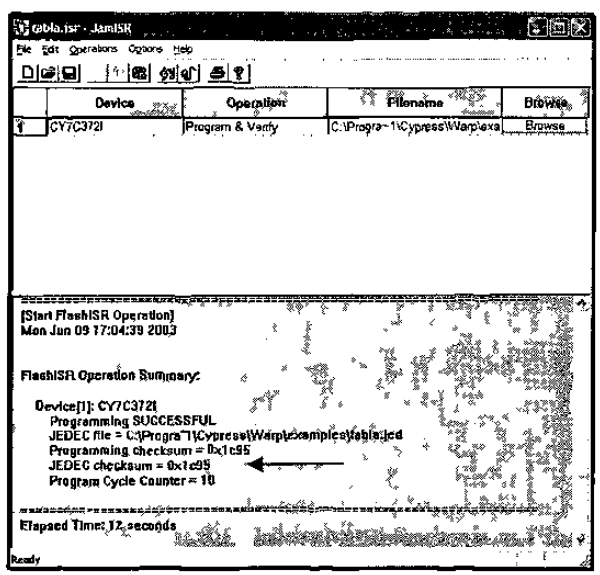


Figura 7.65 Fin de la programación del CPLD

Ya que esté programado el CPLD se quita del programador y se introduce en la tablilla para posteriormente implementar el circuito.

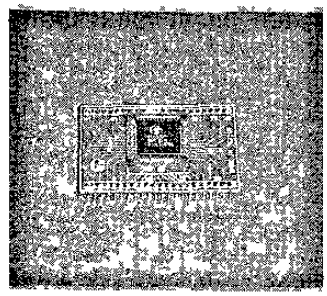


Figura 7.66 Tablilla con el CPLD

Circuito implementado (esta implementación corresponde al ejemplo del archivo tabla.vhd)

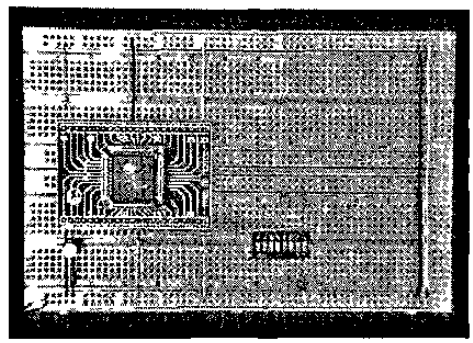


Figura 7.67 Circuito implementado

Ejemplo de un contador binario de 4 bits ascendente con capacidad de carga y clear.

Descripción:

La entrada CLK es activada con la transición positiva de un pulso y por cada transición positiva el valor de las salidas Q(3)...(0) se incrementa en una unidad, donde Q(3) es el bit más significativo.

La entrada LD_L (entrada activa a nivel bajo) en donde las salidas Q(3)...(0) se cargan con el valor que haya en las entradas D(3), D(2), D(1), D(0) cuando esta es activada sin necesidad del pulso de CLK

La entrada CLR_L, tiene el propósito de que las salidas Q(3)...Q(0) tomen el valor de cero (reset síncrono) cuando esta es activada por medio de un nivel bajo sin necesidad del pulso de CLK.

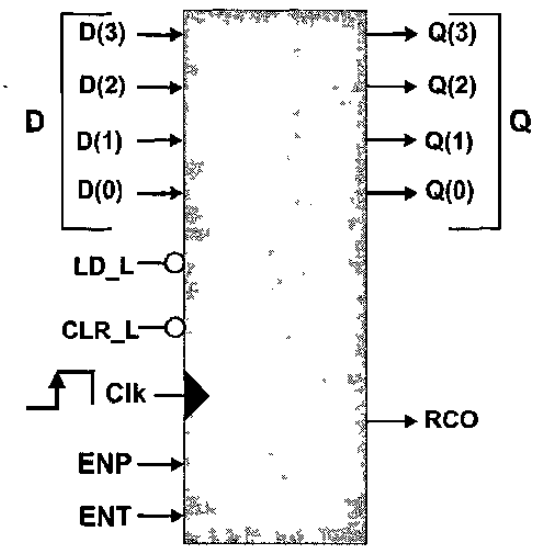


Figura 7.68 Diagrama de bloques del Contador

La salida RCO (ripple carry out) genera una transición cuando el conteo pasa de 15 a cero. La cuenta sólo se realiza cuando hay un uno en las entradas ENP y ENT (son entradas de enable).

Siguiendo el procedimiento para crear un archivo en VHDL con la ayuda de la interfaz Galaxy visto anteriormente, aquí se presenta la programación de un contador antes mencionado que es similar al contador 74163.

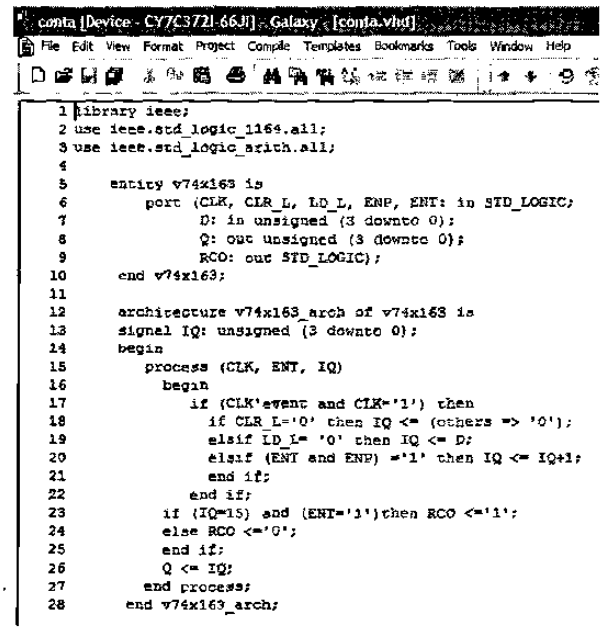


Figura 7.69 Programa Contador 74163

En el simulador Active-HDL Sim haremos la simulación de nuestro programa, añadimos a las entradas los valores que se requieran para la simulación, en este caso se utilizó el tipo de simulador **Custom** y un valor determinado para la entrada de datos.

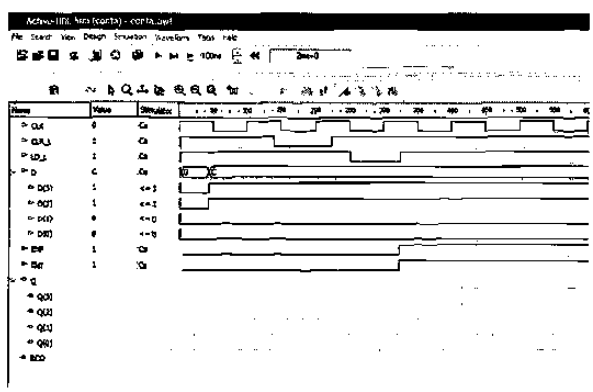


Figura 7.70 Señales de entrada

Ahora iniciamos la simulación y vemos como se comportan las salidas

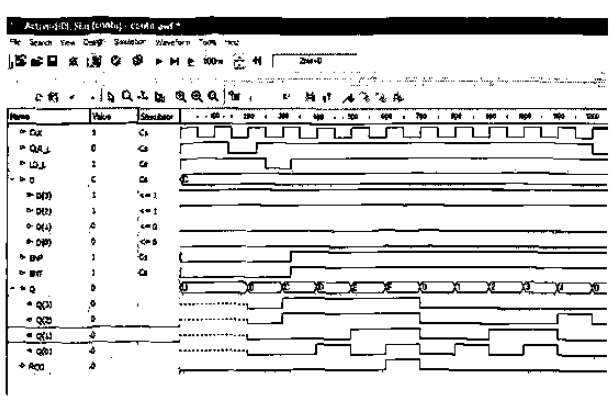


Figura 7.71 Simulación completa

8. CONCLUSIONES Y RECOMENDACIONES

8.1 CONCLUSIONES

El avance tecnológico nos ha permitido contar con herramientas computacionales capaces de programar los dispositivos lógicos programables.

Como se ha mostrado en esta tesis el diseño programable presenta una serie de ventajas sobre el método de diseño tradicional; por mencionar algunas de estas, tenemos diseños más rápidos, prueba del diseño antes de la implementación del proyecto a lo que se le llama simulación, la reprogramación del dispositivo en caso de fallas en el diseño o cambios en el mismo, reducción de costos y principalmente, estar a la vanguardia en cuestión tecnológica.

Con respecto al software, tenemos los HDL's (Lenguajes de Descripción de Hardware) tales como ABEL y VHDL (por mencionar algunos), que ofrecen una enorme ayuda ya que gracias a ellos los diseños se pueden llevar a cabo a través de una programación.

El trabajar con captura esquemática se hace más fácil el diseño, ya que se trabaja en forma gráfica y no con ecuaciones, pero también representa una desventaja en diseños grandes ya que es difícil y se podría decir imposible de comprenderlos debido a que el diseño contiene demasiados componentes y conexiones. Aquí resaltaríamos la factibilidad de manejar estos diseños utilizando HDL (lenguaje de Descripción de Hardware).

8.2 RECOMENDACIONES

Con las ventajas antes mencionadas en el apartado de conclusiones, se observa el gran beneficio que se puede obtener al incluir esta metodología en el laboratorio de Electrónica Lógica II que actualmente emplea circuitos de función fija y algunas memorias.

Se tendrán los siguientes beneficios si se implementa el diseño programable en este laboratorio:

1. El alumno no tendrá que trabajar con muchos circuitos integrados, por consecuencia será menos costoso.
2. Un solo dispositivo le será útil para hacer todas las prácticas
3. Tanto el maestro como el alumno adquirirá los conocimientos del funcionamiento de una de las tantas herramientas actuales en el mercado.

4. Se verá una renovación considerable de este laboratorio ya que desde hace muchos años no se le ha hecho una revisión a las prácticas ni al material que se ocupa actualmente en él.
5. Existirá una uniformidad de las prácticas de laboratorio, ya que los maestros difieren en las prácticas expuestas durante el semestre.
6. Las practicas podrían ser individuales o en equipos pequeños desarrollando en cada alumno diferentes habilidades.

En resumen creo que sería muy benéfico tanto para alumnos como para maestros conocer y manejar una herramienta actual.

Aunque es difícil adaptarse a un cambio inmediato, tengo la convicción de que es posible lograrlo empleando una metodología adecuada y estableciendo un tiempo para llevarlo a cabo.

Las posibles opciones para llevar a cabo el cambio serían capacitar a maestros, ofrecer cursos de actualización, generar un manual para las prácticas de laboratorio, instruir a becarios, entre otras acciones.

BIBLIOGRAFÍA

Barrón Ruíz Mariano, Lógica Programable: ejercicios resueltos con OrCAD/PLD, McGraw Hill, 1994

Pardo Fernando, VHDL Lenguaje para síntesis y modelado de circuitos, Alfaomega,1999.

Wakerly John F., Diseño digital principios y prácticas, tercera edición, Prentice Hall, 2001

Xilinx Inc. Programmable Logic Data Book, 1994

Pardo Fernando, Boluda José A., VHDL Lenguaje para síntesis y modelado de circuitos, Alfaomega, 2000

Maxinez David G., Alcalá Jessica, VHDL el arte de programar sistemas digitales, CECSA, 2002

LISTADO DE TABLAS

LISTADO DE FIGURAS

GLOSARIO

ABEL: (Advance Bolean Expresión Language): Es un lenguaje de descripción de Hardware universal para el diseño con PLD.

Antifusible: (Antifuse) Es lo contrario a fusible, es un circuito abierto que se puede programar para ser una baja impedancia.

Arquitectura: (architecture): La estructura organizacional de un sistema. Una arquitectura puede ser descompuesta recursivamente en: partes que interactúan entre sí por medio de interfaces, relaciones que conectan las partes, y restricciones para ensamblar las partes.

Arreglos de Compuertas: Es un grupo de transistores que se configuran por el usuario en los niveles de conexión metálicos, formando funciones lógicas.

ASICS: (Application Specific Intefrated Circuits) Circuitos integrados de aplicación específica.

Atributo: (attribute) Una propiedad de un tipo, identificada mediante un nombre.

Bit: (Binary Digit) Contracción de Dígito Binario.

Buffer: Una compuerta lógica que realiza la función lógica identidad, es decir, que deja pasar la entrada sin cambios. Se utiliza para aislar varias partes de un sistema o para proporcionar amplificación de tensión o de corriente.

Byte: Grupo de ocho bits.

Chip: Un trozo de material semiconductor que contiene un circuito integrado. Si no está encapsulado se le llama dado (die).

Circuito integrado: (CI) Un tipo de circuito en el que todos sus componentes se encuentran integrados en un único chip semiconductor de muy pequeño tamaño.

CMOS: (Complementary Metal Oxide Semiconductor) MOS complementarios. Un tipo de circuito integrado realizado con transistores de canal-N y de canal-P. Los circuitos CMOS apenas consumen energía cuando no cambian de estado.

Compilar: (Compiling) Rutina que transforma un programa escrito en un seudocódigo o en un lenguaje de programación automática en una serie de instrucciones en lenguaje básico de máquina.

Comportamiento: Los efectos visibles de una operación o evento, incluyendo sus resultados.

CPLD: (Complex Programmable Logic Device) Es un integrado donde se tienen varios PLDs con una red de rutas que permite interconectarlos y realizar funciones lógica más complejas.

Dispositivo: Se refiere a un circuito integrado CI.

EDA: (Electronic Design Automation): Es el nombre que se les da a todas las herramientas (tanto hardware como software) para la ayuda al diseño de sistema electrónicos.

EPROM: (Erasable Programmable Read-Only Memory) Memoria de solo lectura programable y borrable. Un tipo de memoria semiconductora.

Flip Flop: Un circuito digital biestable que puede permanecer en cualquiera de los dos estados binarios. Normalmente se aplica a los dispositivos que cambian de estados binarios. Normalmente se aplica a los dispositivos que cambien de estado cuando se les aplica un pulso en su entrada de reloj.

FPGA: (Field Programmable Gate Array) Consiste de un arreglo de bloques lógicos, rodeado de bloques de entrada/salida programables y conectados a través de interconexiones programables.

FPLA: (Field Programmable Logic Array) Dispositivo lógico programmable formado por una matriz de compuertas AND programmable, seguida de otra matriz de compuertas OR también programable.

Fusible: Es un elemento de baja resistencia que puede ser modificado en un circuito abierto. La programación del fusible se denomina "quemar" el fusible y suele ser térmicamente mediante corrientes elevadas para este.

HDL: (Hardware Description Lenguaje) Es un lenguaje que permite describir un diseño lógico usando ecuaciones Booleanas, tablas de verdad y de estados asi como la descripción lógica.

Implementación: La definición de cómo está construido o compuesto algo.

JEDEC: (Joint Electron Device Engineering Concil) Los archivos JEDEC contienen el mapa de fusibles del PLD listo a ser programado.

Link: (enlazar) Parte de un subprograma que lo vincula con el programa principal. Medio de correlación entre dos o más partes.

LSI: (Large Scale Integration). Una medida de complejidad de un circuito integrado.

Macrocélula: Es la salida típica de un PLD. Generalmente contienen un flip-flop y varios multiplexores que le proporcionan gran flexibilidad.

MMI: (Monolithic Memories, Incorporated). Empresa que creó la arquitectura PAL; actualmente es parte de AMD.

MSI: (Medium Scale Integration). Una medida de complejidad de un circuito integrado.

PAL: (Programmable Array Logic) Es una arquitectura que simplifica la de los PLAs. En esta los arreglos de OR son fijos y los de AND son programables.

PLA: (Programmable Logic Array) Es una arquitectura que utiliza un arreglo de AND programable, en serie con un arreglo de OR programable.

PLD: (Programmable Logic Device) Es un circuito que puede ser configurado por el usuario para que realice una función lógica. Estos suelen estar constituidos por un arreglo

de compuertas ANDs seguidos por un arreglo de compuertas Ors. Normalmente se utiliza para pequeños PLDs como PALS y FPLAs.

Programable: Que se puede configurar de una manera deseada.

Registro de entrada: Es un flip flop o un Latch en algunos CPLDs que mantiene las señales de entrada, utilizado cuando se multiplexa el bus.

Tabla de verdad: (Truth Table) Es una forma de representación tabular de una función en la que se indica el valor de la salida o salidas para cada una de las posibles combinaciones que las variables de entrada pueden tomar.

Triestado: Es un tipo de salida de un dispositivo lógico que puede tomar el valor de uno, cero y alta impedancia.

TTL: (Transistor Transistor Logic) Familia de dispositivos lógicos bipolares más usada.

Verilog: Lenguaje de diseño donde se introduce la descripción de hardware de alto nivel.

VHDL: (VHSIC Hardware Description Languaje) Es uno de los lenguajes de programación de dispositivos lógicos más utilizados. Creado por el Departamento de Defensa de Estados Unidos.

VHSIC: (Very High Speed Integrated Circuit). Es un programa de investigación de dispositivos VLSI de alta velocidad, promovido por el Departamento de Defensa de los Estados Unidos.

AUTOBIOGRAFÍA

Mi nombre es Elizabeth Guadalupe Lara Hernández. Nací el 11 de Septiembre de 1974 en la ciudad de Monterrey, Nuevo León. Mis padres son el Biól. Absalón Lara Vargas y la Sra. Flora Hernández Segura.

El grado que deseo obtener es el de Maestro en Ciencias de la Ingeniería Eléctrica con especialidad en Control. El título de esta tesis es, Diseño de Sistemas Digitales con Lógica Programable.

Soy Ingeniero en Control y Computación, egresada de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León. Hice mis estudios de preparatoria en la Preparatoria No. 1 de la misma Universidad.

Mi experiencia profesional ha sido en el ramo de la docencia, actualmente tengo 4 años trabajando en la Facultad de Ingeniería Mecánica y Eléctrica, específicamente en la coordinación de Eléctrica, Electrónica y Automatización, donde he impartido e imparto los siguientes cursos, Teoría de Control I, Instrumentación Analógica, Control de Procesos, Electrónica Industrial, Álgebra Lineal, Ingeniería de Control, Electrónica Lógica I y diversos laboratorios.