UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

SIMULADOR DE PAL ASISTIDO POR COMPUTADORA BAJO AMBIENTE WINDOWS

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

MARIA GUADALUPE SERNA LOPEZ

ALEJANDRO RIOS MARTINEZ

Director de Tesis: Ing. Sergio Ambriz Maguey

MEXICO, D. F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

" Que mis ideales y experiencia positivas los reciban la familia y la sociedad. "

Esma López María Guadalupe

Dedico en forma incondicional esta tesis a mis padres Jorge Fausto e Isabel por brindarme dia a dia cariño, apoyo y mucho amor. Mamita gracias por darme cada mañana tu bendición haciéndola descender desde el cielo como rocío sobre mí para seguir mi camino cuesta arriba. Papá gracias por enseñarme que el estudio es perseverancia, dedicación y la mejor de las herencias que puedes dar a tus hijos, gracias por ser constante en tu difícil función de padre por conducirnos por el camino de la vida.

a mis hermanos Bety, Jorge, Laura, Lourdes, Alberto, Lidia y Alejandro; porque juntos formamos una verdadera familia, donde cada uno es el apoyo de los demás.

a la familia Ríos Martínez agradezco su amistad y apoyo para continuar con mis logros presentes y futuros.

a mis colegas, amigos y asesores Sergio Ambríz M y Martín Pérez M. :

Sergio Ambríz M. gracias por compartir tus conocimientos para superar los obstáculos y conducirme al logro de mis propósitos.

Martin Pérez M. esta tesis es el reflejo de la culminación de tus esfuerzos, gracias por ayudarnos a Alejandro y a mí a ensanchar nuestros horizontes y elevar el espíritu hasta alcanzar los valores que se requieren para ser competentes el día de mañana en nuestra profesión de Ingenieros.

a todos los compañeros del **Laboratorio de CAD**, donde en equipo hemos trabajado para crear los cimientos que nos conducen a la superación personal. La motivación y cooperación de ustedes fué vital para la realización de este trabajo, gracias: Rosario, Juan Carlos, Germán, Alberto y Jorge.

Y todo mi empeño y dedicación para alguien muy especial.....

Ale:
Si a ti que siempre me enriqueces
cuando buscas la verdad
e intentas con sencillez y modestia,
de completar mi pensamiento con el tuyo,
para corregir mis errores,
y ahundar mi visión.

Si a ti con quien camino paso a paso
sin temor a tropezar,
porque simpre estas conmigo
porque siempre cuento contigo.....

<< sólo un minuto de tu tiempo >>

a mi Universidad " Por mi raza hablará el espiritú "

DEDICO ESTA TESIS:

A mis padres **Salvador y María de La Luz**, por haberme apoyado en todo momento hacia la culminación de esta meta que hoy concluye, quiero dar las gracias por estar siempre ahí, donde se que los voy a encontrar, en cualquier instante...

A mis hermanos, **Salvador, Alicia, María Elena y José**, porque sólo ellos conocen el sentimiento de ser un hermano, y que con sus enseñanzas hoy me conducen hacia un camino llano ...

A la familia Serna López, por el cariño y apoyo incondicional durante estos últimos seis años, quien con su humildad y sabiduría de la escuela de la vida me han dado temple para no desistir y si en cambio, perseverar. Este triunfo también es suyo. A **Ma. Isabel, Jorge Fausto, Bety, Laura, Jorge, Lulú, Tico, Alex y Lidia**, gracias por el breve espacio ...

A **Dios**, por haberme concedido el inigualable orgullo de saberme respaldado siempre, el cualquier situación, no importa dónde, yo sé que estarás conmigo...

A **Martín Pérez y Sergio Ambríz**, por su incomparable deseo de superación, por la confianza que depositaron en mí para que hoy me sienta más seguro en mi desempeño personal y profesional, gracias ...

A los **compañeros del Laboratorio de CAD**, por haberme aportado sus conocimientos y su amistad.

A tí, tu sabes quién, por estar aquí, por tu motivación, tus principios, tu cariño, por ser mi compañera de tesis (mi respeto y admiración), tu paciencia y sobre todo por que siempre me has brindado <<*sólo un minuto de tu tiempo*>>. Hoy nada de esto sería posible sin tu respaldo.

A la **Universidad Nacional Autónoma de México**, por darme abrigo seguro y brindarme las posibilidades del éxito.

Alejandro Ríos Martínez.

INDICE

INTRODUCCION

En el proceso de diseño de sistemas digitales se utilizan diferentes técnicas como son: el empleo de Cartas **ASM** (*Algoritmo de máquinas de Estado*), lógica combinacional, así como la computación gráfica. Estas herramientas facilitan el diseño de sistemas tan complejos como se requieran, pero, a medida que aumenta su complejidad, la tarea de optimizar su implementación crece, teniendo como consecuencia el aumento de costos y tiempo en las diferentes etapas de diseño (*diseño lógico, implantación del diseño, verificación del diseño y la lógica, edición modificación, simulación, etc.*) Con la aparición de la computadora, se ha facilitado cualquier proceso de diseño, considerando su construcción básica a base de circuitos digitales.

El gran desarrollo tecnológico que la humanidad está viviendo implica que los nuevos sistemas electrónicos son mucho más complicados con la optimización y fabricación de los mismos. El uso de herramientas computacionales con interfase gráfica, hace que la interacción hombre-máquina sea más amigable, logrando que los sistemas incrementen la productividad al disminuir costos y tiempo de producción. La elaboración de esta tesis persigue el objetivo de ser parte integral de sus herramientas de trabajo.

capítulo I

Estudiaremos la evolución histórica de la lógica programable como una herramienta de diseño digital, también nos introduciremos en los conceptos básicos para el manejo de los dispositivos programables.

capítulo II

Analizaremos como se realiza la programación de los **PLD**, las técnicas de proceso, así como las arquitecturas internas que los componen.

capítulo III

Veremos un método de diseño digital del diagrama **ASM** y reducción de mintérminos. También conoceremos como se genera el código **JEDEC.**

capítulo IV

Panorama de las técnicas de Simulación base para el desarrollo del sistema. También se hace investigación para proponer un algoritmo de simulación en base a las características del PAL.

capítulo V

Daremos un seguimiento al desarrollo del sistema en ambiente de programación Windows utilizando la plataforma de Borland C++.

capítulo VI

Visualizamos el entorno del sistema desarrollado como tesis llamado "SimPAL", elaborando prueba para mostrar su funcionamiento.

CAPITULO I

LA LOGICA PROGRAMABLE

Con el sistema **SIMULADOR DE PAL BAJO AMBIENTE WINDOWS** se pretende dar una herramienta de diseño útil y fácil de usar. Para lograr nuestro objetivo principal de la realización de esta investigación, en un principio es importante conocer como se introdujeron los **PLD** como respuesta a las necesidades del diseño de circuitos.

En el presente capítulo estudiaremos el arreglo, estructura y performance de los **PLD** para compararlos con otras dispositivos alternativos y conocer su lógica programable con el propósito de manejar, verificar y simular su estructura.

La lógica programable ha evolucionando en la última década, convirtiéndose en una herramienta que permite el diseño digital con un mínimo de componentes. La clave para los Dispositivos Lógicos Programables (*Programmable Logic Design* : **PLD**) es el uso de celdas programables integradas (típicamente uniones de fusibles que pueden ser fundidos), que permiten configurar los componentes lógicos en un diseño específico, haciendo posible agilizar el proceso de implementación del diseño sin cambios en el Circuito Impreso (*Printed Circuit Board* : **PCB**).

Aún cuando los **PLD** no ofrecen la densidad de integración de los circuitos **VLSI** (*Very Large Scale Integrated* : Muy Alta Escala de Integración), son más flexibles que éstos y más rentables, debido a que son baratos y ocupan poco espacio; también, gracias a su capacidad de soportar funciones complejas han tenido gran extensión en diversas aplicaciones sobre todo en circuitos secuenciales.

I.1 HISTORIA DE LOS PLD

En la era de la lógica preprogramable o tradicional, el diseño lógico consistía de muchos circuitos integrados **TTL** (*Transistor-Transistor-Logic* : Lógica Transistor-Transistor), **SSI** (*Small Scale Integrated* : Pequeña escala de integración) y **MSI** (*Medium Scale Integrated* : Mediana Escala de Integración) combinados con las funciones booleanas.En aquél entonces, los diseñadores se basaban en libros como *"The TTL Databook"* de Texas Instruments y otros similares, para encontrar las especificaciones de los dispositivos y determinar cuáles estaban disponibles para sus diseños, sin embargo, las conexiones se volvían complejas durante el alambrado de los circuitos y se presentaban corto circuitos.

En 1975, Signetics Corporation, introdujo el primer Dispositivo Lógico Programable sin memoria - el 82S100 (ahora PLS100)- **FPLA** o Arreglo Lógico de Campo Programable. Napoleone Cavlan "el padre de la lógica programable" dirigió las aplicaciones en Signetics del **PLA** (*Programmable Logic Array* : Arreglo Lógico Programable), teniendo como meta mejorar y modificar los sistemas digitales, mientras la **PROM** (*Programmable Read Only Memory* : Memoria únicamente de lectura)- introducida anteriormente por Harris- mostraba lo promisorio que sería utilizar estos dispositivos para algunas aplicaciones, otros reducían potencialmente el número de elementos para un circuito impreso (**PCB**) y minimizaban los esfuerzos para hacer los cambios en un diseño digital; de esta manera se podían evitar los corto circuitos y conexiones "puenteadas" para los diseñadores, teniendo la opción de cambiar hacia los **PLD**.

National Semiconductor produjo el **PLA**, que consiste de arreglos de compuertas **AND** programable seguido de un arreglo programable **OR**, almacenando Suma-de-productos (*Sum Of Products* : **SOP**) en la implementación. Mediante la combinación de celdas programables basadas en la tecnología de las **PROM** y el concepto del **PLA**, Cavlan concibió el primer dispositivo de campo programable, el **FPLA** (*Field Programmable Logic Array* : Arreglo Lógico de Campo Programable).

El pionero del **FPLA** fué Signetics al introducir la familia de Fusibles Lógicos Integrados (*Integrated Fuse Logic* : **IFL**) que consta de 28 pines de 0.6 pulgadas de ancho del **DIP** (*Dual In-line Package* : Encapsulado de doble línea) y su arquitectura consiste de arreglos **AND** programables seguidos de un arreglo **OR** programable, permitiendo la implementación de la lógica **SOP**.

El intento inicial de Birkner fué convencer a la dirección de llevar el concepto de **PAL**, el cual se recibió con cierto escepticismo. Sin embargo, el efecto en Signetics de los **FPLA** había generado un mercado interesante para los nuevos circuitos integrados de 20 pines (Figura 1.2), lo que representaba una ventaja sustancial, por eso Birkner presentó la idea a la industria mediante conferencias, pero sus ideas sólo las retomaron Data General y Digital Equipment.

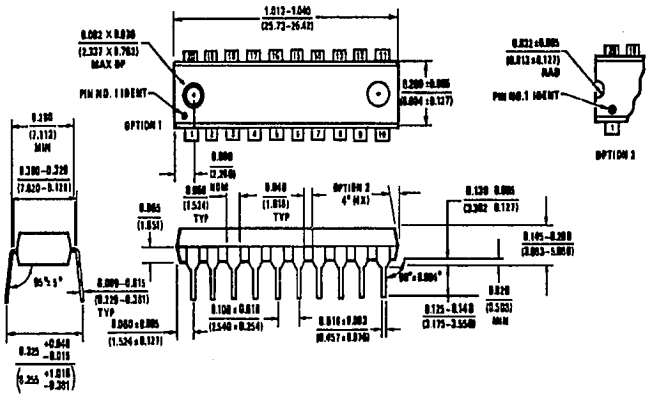


Figura 1.2 Descripción de un DIP de 20 pines. National Semiconductor

Los primeros **PAL** fueron los **PAL16L8, PAL16R4, PAL16R6** y **PAL16R8** (Figura 1.3); teniendo 35 ns de entrada-a-salida en retardos de propagación, además, estos dispositivos tienen 8 salidas y 16 entradas. El encapsulado tipo "L" tiene ocho combinaciones de salida que funcionan por medio de lógica negativa conocida como "Low Active", mientras que los encapsulados de tipo "R" tienen 4, 6 y 8 registros de salida respectivamente (Figura 1.4).

Las compañías de minicomputadoras encontraron que los **PAL** pueden reducir el número de circuitos que necesitan ser implementados en los diseños y permiten cambios en la lógica sin "corto circuitos ni puentes con cableado", pero la producción de los dispositivos **PAL** resultó insuficiente para la demanda, los problemas de producción hicieron que alcanzara un costo de $ 5.00 por unidad como precio inicial.

La lógica programable ha evolucionando en la última década, convirtiéndose en una herramienta que permite el diseño digital con un mínimo de componentes. La clave para los Dispositivos Lógicos Programables (*Programmable Logic Design* : **PLD**) es el uso de celdas programables integradas (típicamente uniones de fusibles que pueden ser fundidos), que permiten configurar los componentes lógicos en un diseño específico, haciendo posible agilizar el proceso de implementación del diseño sin cambios en el Circuito Impreso (*Printed Circuit Board* : **PCB**).

Aún cuando los **PLD** no ofrecen la densidad de integración de los circuitos **VLSI** (*Very Large Scale Integrated*: Muy Alta Escala de Integración), son más flexibles que éstos y más rentables, debido a que son baratos y ocupan poco espacio; también, gracias a su capacidad de soportar funciones complejas han tenido gran extensión en diversas aplicaciones sobre todo en circuitos secuenciales.

I.1 HISTORIA DE LOS PLD

En la era de la lógica preprogramable o tradicional, el diseño lógico consistía de muchos circuitos integrados **TTL** (*Transistor-Transistor-Logic* : Lógica Transistor-Transistor), **SSI** (*Small Scale Integrated* : Pequeña escala de integración) y **MSI** (*Medium Scale Integrated* : Mediana Escala de Integración) combinados con las funciones booleanas.En aquél entonces, los diseñadores se basaban en libros como *"The TTL Databook"* de Texas Instruments y otros similares, para encontrar las especificaciones de los dispositivos y determinar cuáles estaban disponibles para sus diseños, sin embargo, las conexiones se volvían complejas durante el alambrado de los circuitos y se presentaban corto circuitos.

En 1975, Signetics Corporation, introdujo el primer Dispositivo Lógico Programable sin memoria - el 82S100 (ahora PLS100)- **FPLA** o Arreglo Lógico de Campo Programable. Napoleone Cavlan "el padre de la lógica programable" dirigió las aplicaciones en Signetics del **PLA** (*Programmable Logic Array* : Arreglo Lógico Programable), teniendo como meta mejorar y modificar los sistemas digitales, mientras la **PROM** (*Programmable Read Only Memory* : Memoria únicamente de lectura)- introducida anteriormente por Harris- mostraba lo promisorio que sería utilizar estos dispositivos para algunas aplicaciones, otros reducían potencialmente el número de elementos para un circuito impreso (**PCB**) y minimizaban los esfuerzos para hacer los cambios en un diseño digital; de esta manera se podían evitar los corto circuitos y conexiones "puenteadas" para los diseñadores, teniendo la opción de cambiar hacia los **PLD**.

National Semiconductor produjo el **PLA**, que consiste de arreglos de compuertas **AND** programable seguido de un arreglo programable **OR**, almacenando Suma-de-productos (*Sum Of Products* : **SOP**) en la implementación. Mediante la combinación de celdas programables basadas en la tecnología de las **PROM** y el concepto del **PLA**, Cavlan concibió el primer dispositivo de campo programable, el **FPLA** (*Field Programmable Logic Array*: Arreglo Lógico de Campo Programable).

El pionero del **FPLA** fué Signetics al introducir la familia de Fusibles Lógicos Integrados (*Integrated Fuse Logic* : **IFL**) que consta de 28 pines de 0.6 pulgadas de ancho del **DIP** (*Dual In-line Package* : Encapsulado de doble línea) y su arquitectura consiste de arreglos **AND** programables seguidos de un arreglo **OR** programable, permitiendo la implementación de la lógica **SOP**.

Los nuevos **FPLA** recibieron poca aceptación por parte de los diseñadores de sistemas digitales, porque el dispositivo con 28 pines no fué muy popular y prefirieron completar sus diseños con dispositivos **LS, HS y HC** porque los precios de estos dispositivos tenían un costo aproximado a $30 U.S.D. por 100 piezas. En los **PCB** se hicieron indispensables los "puentes" y conexiones entre módulos de tarjetas, para la operación sin corto circuitos, mismos que vinieron a producir cambios en los procedimientos de documentación, ingeniería y manufactura. Convencido de que era mucho mejor tomar el camino del diseño de los circuitos digitales, John Martin Birkner cambió a Silicon Valley en 1975, donde trabajó con Monolitic Memories Inc. (**MMI**) en la manufactura de **PROM** y dispositivos lógicos estándar, ahí concibe la idea de una lógica programable y después de estudiar en Signetics la realización de los **FPLA** así como los beneficios del concepto, realiza nuevas investigaciones con **PLD** posteriores a los **FPLA**.

Una vez que Birkner concibe el concepto de Arreglo Lógico Programable (**PAL**) como un dispositivo de arquitectura similar pero más simple que el **FPLA**, el **PAL** se constituye por el arreglo **AND** programable seguido de un arreglo **OR** fijo no programable, esto es, cada compuerta **OR** puede tener un número fijo de productos **AND**. Con este avance se reduce substancialmente el tamaño y permite una señal más rápida de propagación, mientras que la implementación de la lógica **SOP** puede ser almacenada de manera más eficiente, de manera tal que el **PAL** tiene un ahorro de espacio al contener solamente 20 pines en el encapsulado; como consecuencia, el nuevo diseño gustó más a los programadores pues ofrece más ventajas que las **PROM**.

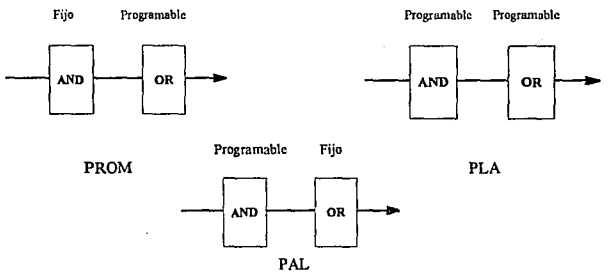


Figura 1.1 Arquitecturas de los Dispositivos Lógicos Programables

Birkner precisó cual podía ser el número razonable de términos de productos para cada salida (suma de productos) y después de trabajar empíricamente con muchos circuitos que le dieron algunos problemas durante el diseño de una minicomputadora en Computer Automation, determinó que la mayoría de las funciones boolenas utilizaban ocho o menos términos de **SOP**.

El intento inicial de Birkner fué convencer a la dirección de llevar el concepto de **PAL**, el cual se recibió con cierto escepticismo. Sin embargo, el efecto en Signetics de los **FPLA** había generado un mercado interesante para los nuevos circuitos integrados de 20 pines (Figura 1.2), lo que representaba una ventaja sustancial, por eso Birkner presentó la idea a la industria mediante conferencias, pero sus ideas sólo las retomaron Data General y Digital Equipment.

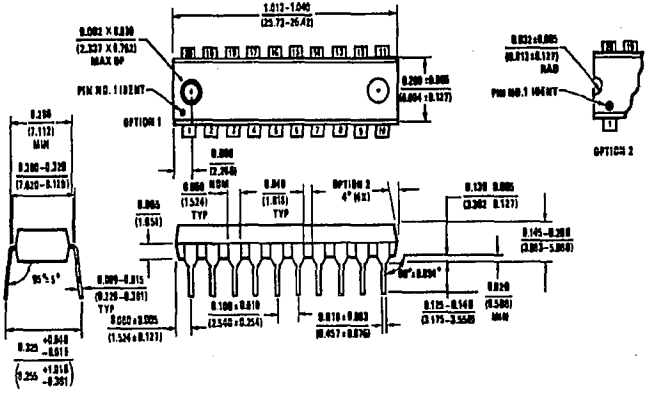


Figura 1.2 Descripción de un DIP de 20 pines. National Semiconductor

Los primeros **PAL** fueron los **PAL16L8, PAL16R4, PAL16R6** y **PAL16R8** (Figura 1.3); teniendo 35 ns de entrada-a-salida en retardos de propagación, además, estos dispositivos tienen 8 salidas y 16 entradas. El encapsulado tipo "L" tiene ocho combinaciones de salida que funcionan por medio de lógica negativa conocida como "Low Active", mientras que los encapsulados de tipo "R" tienen 4, 6 y 8 registros de salida respectivamente (Figura 1.4).

Las compañías de minicomputadoras encontraron que los **PAL** pueden reducir el número de circuitos que necesitan ser implementados en los diseños y permiten cambios en la lógica sin "corto circuitos ni puentes con cableado", pero la producción de los dispositivos **PAL** resultó insuficiente para la demanda, los problemas de producción hicieron que alcanzara un costo de $ 5.00 por unidad como precio inicial.

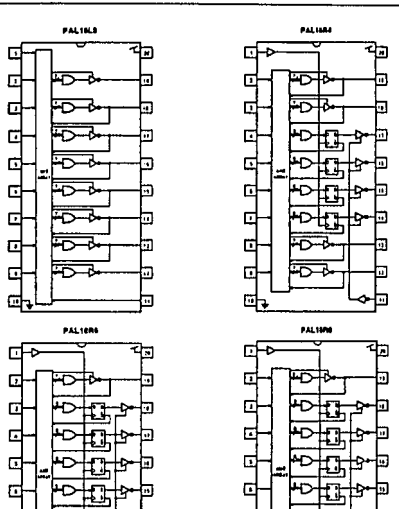


Figura 1.3 Arquitecturas de la familia PAL de 20 pines

Una vez que Data General, Digital Equipment y MMI tomaron la iniciativa, comenzaron con la programación de liga de fusibles con máscara programable, se marcó la introducción de la familia de dispositivos HAL (*Hard Array Logic* : Arreglos Lógicos Duros), motivo suficiente para que MMI eventualmente improvisara el proceso de manufactura, produciendo un alto volúmen de PAL; para 1978, MMI publicó el primer manual de PAL y se abrió el mercado para los diseñadores. En el manual explicaban los conceptos y las ayudas para beneficiar la realización de sus labores, incitándolos a perder el miedo en la utilización de estos circuitos integrados, además, en el manual se incluían los programas fuente del PALASM (ensamblador de PAL) en lenguaje Fortran y se podían incluir ecuaciones boolenas. El PALASM puede compilar las definiciones lógicas dentro del formato que hubiese sido dado de alta, algunas veces es capaz de simular la operación de un PAL en conjunción con el diseño específico de las pruebas de los vectores, verificando las operaciones lógicas antes de programar realmente las partes.

TIPOS DE DISPOSITIVOS

Tipo de dispositivo	Entradas Dedicadas	Registros de salida (con retroalimentación)	Combinatoria	
			I/O	Salida
PAL16	10	-	6	-
PAL16	8	4	4	-
PAL16	8	6	2	-
PAL16	8	8	-	2

VERSIONES : VELOCIDAD / POTENCIA

SERIE	EJEMPLO	COMERCIAL		MILITAR	
		T_{PD}	I_{CC}	T_{PD}	I_{CC}
Estándar	PAL16R8	35 ns	180 mA	45 ns	180 mA
A	PAL16R8A	25 ns	180 mA	30 ns	180 mA
A2	PAL16R8A2	35 ns	90 mA	50 ns	90 mA
B	PAL16R8B	15 ns	180 mA	20 ns	180 mA
B2	PAL16R8B2	25 ns	90 mA	30 ns	90 mA
D	PAL16R8D	10 ns	180 mA		
-7	PAL16R8-7	7 ns	180 mA		

Figura 1.4 Tablas comparativas de las versiones de los dispositivos PAL

Como herramienta para los diseñadores de **PLD**, **MMI-Signetics** introduce el AMAZE (Autómata de Mapeo de Ecuaciones) compilador de **PLD** realizado en 1984 para soportar estos dispositivos, de igual manera otros fabricantes ofrecen software que soporta a otros dispositivos.

Tanto Signetics como **MMI** continuaron con nuevos dispositivos **PLD** y dieron a conocer una gran variedad de diseños, incrementando mejoras, asi que en 1982 una gran variedad de dispositivos lógicos estaban disponibles para los diseñadores y algunas compañías manufactureras se interesan en la fabricación, como es el caso de National Semiconductor y Advanced Micro Devices (**AMD**). La tecnologia de la lógica programable implementa lo que podriamos llamar segunda generación de dispositivos, y para 1983 se da a conocer el **PAL22V10**, mismo que introduce avances no disponibles anteriormente. El más notable es la inclusión de macroceldas flexibles que en cada circuito integrado tiene 10 pines de salida disponibles, donde cada macrocelda es capaz de ser combinada como registro de salida (*programándola como salida polarizada*), o como un pin de salida. La salida del buffer tres-estados puede controlar por separado cada producto, almacenando operaciones bidireccionales, asi, todos los registros pueden automáticamente re-iniciarse durante el encendido, y los registros pueden ser pre-cargados con un valor conocido.

En 1983 Assisted Technology liberó la versión 1.01 del **CUPL** (Compilador Universal para Dispositivos Programables); compilador que soporta 29 dispositivos estándar pero, la industria de **EPROM** (*Erasable Programmable Read Only Memory* : Memoria de sólo lectura programable -

borrable con luz UltraVioleta) tenía mayor popularidad sobre el mercado del **PAL**. En ese sentido las **EPROM** tenían ventaja en el almacenamiento de programas y de datos, gracias a que la programación de las **EPROM** tienen un archivo objeto que contienen un código binario para los patrones de bits que serán creados y transferidos al programador de **EPROM**. Numerosos formatos para dicho propósito se realizaron sin llegar a formar un estándar formal. Intel, Motorola, Tektronix Tek-hex han desarrollado otros formatos pero requieren soporte adicional, lo cual incrementa los costos finales.

La industria de los **PLD** ha madurado, teniendo en claro que es menester el desarrollo de un estándar, que finalmente fué confinado a la Juntura de Dispositivos Electrónicos de Ingeniería (**JEDEC**) para producir un juego de compiladores de salida para **PLD** con tendencia a ser un estándar. Esto requería duplicar esfuerzos para tener un programador del **PLD** que soportara un sólo formato, sin embargo, en octubre de 1983 el **JEDEC** de Productos de Ingeniería Solid State liberó la versión 3.0 para Transferencia de formatos a través de la preparación de datos como un sistema para dispositivos lógicos programables. El estándar fué revisado y aprobado en mayo de 1986 con la versión 3-A.

El siguiente avance se presentó en julio de 1984, con la introducción del primer producto de Altera-Corp, el EP300. Lo más significativo fué la implementación utilizando tecnología **CMOS EPROM** (*Complemented Metal-Oxide Semiconductor Erasable Program Read Only Memory* : Memoria de sólo lectura programable / borrable de tipo Semiconductor de metal-oxido complementario), la cual ofrece baja potencia de operación y permite el borrado con luz ultravioleta (**UV**).

Altera llama a estos dispositivos como Dispositivos Lógicos Programables y Borrables (**EPLD'S**). Así, en 1985 la familia de los **PLD** se incrementa con Lattice Semiconductor Corp., al introducir el Arreglo Lógico Genérico (*Generic Array Logic* : **GAL**).

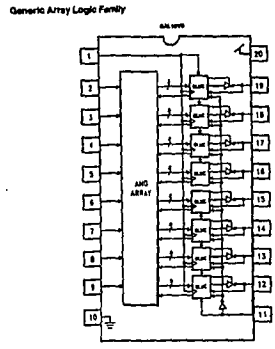


Figura 1.5 Arreglo Lógico Genérico GAL16V8

El **GAL16V8** (Figura 1.5) ofrece baja potencia de operación, como la tecnología **CMOS**, posee la propiedad de ser borrable, con la diferencia de que el **GAL** es eléctricamente borrable, con unos cuantos segundos de aplicación de voltaje, comparado con los 20 minutos y la exposición ultravioleta que requerían los **EPLD**, de esta manera, los **GAL** también son llamados **PLD** eléctricamente borrables **EEPLD**.

Los primeros cambios radicales en los **PLD** vienen en 1985 de la compañía Xilinix. En febrero de 1984 esta compañía se apunta un triunfo al desarrollar la familia **CMOS** utilizando arreglos de compuertas programables asociados con sistemas de desarrollo, en esos Arreglos de Celdas Lógicas (**LCA**) no existen productos de términos y suma de términos. Es la tercera generación de las consideradas arquitecturas de **PLD**, que consiste de una matriz de Bloques Lógicos Configurables (*Control Logic Block* : **CLB**) mediante bloques de entrada/salida (*Input Output Block* : **IOB**), con interconexión de redes de bloques, los **LCA** son los primeros **PLD** que utilizan celdas de **RAM** estática (*Static Random Access Memory* : **SRAM**) para configuraciones de funcionalidad lógica; el uso de las celdas **SRAM** tienen ventajas y desventajas: del lado positivo, el dispositivo será completamente manufacturado para pruebas, y también será dinámico en sistemas reconfigurables: del otro lado, las celdas **LCA** naturalmente son volátiles y requieren de memoria externa de almacenamiento en esta configuración cuando necesitan ser cargados.

La tercera generación fué anunciada en 1985, aunque apareció a principios de 1986 en Westcon, cuando Signetics introduce la Macro Celda Lógica programable (**PML**) con arquitectura de un arreglo **NAND** de compuertas, y salidas de tipo **NAND** dentro de un arreglo **NAND**. Estos múltiples niveles pueden ser implementados debido a la eficiencia del uso de los dispositivos de silicón, una vez que las funciones han sido internamente definidas, se puede rutear en algún pin de salida, vía interconexiones con el circuito integrado.

En 1986 aparece otra familia similar a la anterior que utilizan la implementación "**foldback**", introducida por Exel Microelectronics aunque la producción no comenzó hasta 1987 con el **ERASIC** (*Erasable Aplication Specific Integrated Circuit* : Circuito integrado borrable de aplicaciones específicas, familia basada en tecnología **CMOS EEPROM** que utiliza compuertas **NOR** de un mejor desempeño o performance. El XL78C800 fué liberado en el popular empaquetado de 0.3 pulgadas por 24 pines, soportado por el software **ABEL** (Lenguaje avanzado de expresiones booleanas).

Una nueva tendencia aparece en 1986 con el primer dispositivo de función-aplicación específica (*Aplication Specific PLD* : **ASPLD**), sin embargo, tiene una arquitectura de propósito general que puede admitir funciones específicas siendo más eficiente utilizar los recursos de un circuito integrado con funciones más avanzadas en aplicaciones especiales.

En 1988 se anuncia otro cambio cuando dos nuevos contratos abren el mercado de los **PLD**. El primer anuncio viene de Actel Corp, quien introduce una nueva serie de dispositivos de arreglos de compuertas programables utilizando **RAM** basada en las celdas **LCA** de Xilinix, las partes de Actel son en tiempo programable (*On Time Programmable* : **OTP**) eliminando la necesidad de configurar dispositivos cada vez que se encienden. El segundo anuncio importante es revelado por Gazelle Microcircuits, quien anuncia el primer **PLD** basado en GaAs (galio arsénico) como tecnología; tiene el distintivo de improvisar velocidad-potencia en el producto sobre la base de la tecnología del silicón, siendo cada vez más rápido y más moderado el consumo de energía.

El segundo punto importante de desarrollo es un aceptable método de programación de las partes, esto es, la compañía desarrolla un programador basado en láser seleccionando los fusibles que se utilizan para la aplicación, pero desafortunadamente, desde la unidad de programación los fusibles seleccionados utilizan directamente el haz láser y esto es más laborioso y costoso.

Algunos de los dispositivos lógicos programables no pueden cumplir con el rango completo de arreglo de compuertas para un buen desempeño, al incrementar rápidamente la densidad en un dispositivo, aunque la velocidad en años recientes ha mejorado con los desarrollos en el área del **CAD** (*Computer Aided Design*: Diseño Asistido por Computadora) y sus herramientas. Las ventas por estos dispositivos fueron alrededor de $2.12 billones durante 1990 (Figura 1.6), comparadas con los $20 millones en 1984.

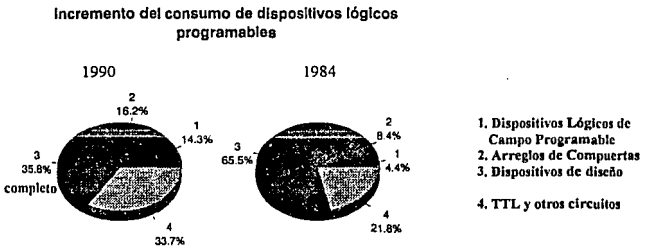


Figura 1.6 Incremento en el consumo de PLD

I.2 CONCEPTOS BASICOS PARA EL MANEJO DE PLD

FAMILIAS LOGICAS

El circuito digital se construye invariablemente con Circuitos Integrados (CI). Las compuertas digitales se clasifican no solamente por su operación lógica, sino por la familia a la cual pertenecen, Cada familia tiene un circuito electrónico básico propio, desarrollando funciones y circuitos digitales más complejos. El circuito básico en cada familia es una compuerta **NOR** o una compuerta **NAND**, por esta razón muchas familias de CI han sido introducidas comercialmente, pero aquéllas que han alcanzado mayor popularidad son las siguientes:

TTL	Lógica Transistor-Transistor
ECL	Lógica de Emisor Acoplado
MOS	Semiconductor de Oxido-Metal
CMOS	Semiconductor de Oxido-Metal Complementario
I^2L	Lógica de Inyección Integrada

La familia **TTL** tiene una lista extensa de componentes que realizan funciones digitales y es comúnmente la familia lógica más popular. La **ECL** se usa en sistemas que requieren operaciones de alta velocidad y la **MOS** e **I^2L** son frecuentemente utilizadas en circuitos de bajo consumo de voltaje.

• Lógica Positiva y Negativa

La señal binaria a la entrada o salida de cada compuerta puede tener uno de dos valores, excepto en la transición, el valor lógico de una señal puede ser 0 ó 1. Considerando los dos valores de la señal binaria, un valor debe ser mayor que el otro puesto que tienen que ser diferentes para poder distinguirlos.

Escoger el nivel alto **H** para representar la lógica "1", mediante el cual se define el sistema de lógica positiva o escoger el nivel **L** para representar lógica "0", para definir el sistema de lógica negativa.

Los términos positivo y negativo no son adecuados ya que ambas señales pueden ser positivas o negativas, en este sentido, no es la polaridad de las señales lo que determina el tipo de lógica, sino la asignación de los valores lógicos de acuerdo a las amplitudes relativas de las señales.

Las hojas de especificaciones técnicas de datos de los circuitos integrados definen funciones digitales no en términos de lógica 1 o lógica 0 sino en términos de niveles **H** y **L**, en cada familia hay un rango de valores de voltaje que el circuito puede reconocer como nivel alto o nivel bajo (Figura 1.7).

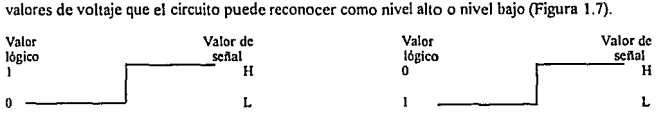


Figura 1.7 Asignación de amplitud de señal y tipo de lógica

La conversión de lógica positiva a lógica negativa, es esencialmente una operación que cambia unos por ceros y ceros a unos en las entradas y salidas, por esta razón, debido a que dicha operación produce una función dual, el cambio de todas las terminales de una polaridad a otra dará el mismo resultado que tomar el dual de la función, el resultado de esta conversión es que todas las operaciones **AND** se conviertan a operaciones **OR** y viceversa.

• Caracteristicas especiales

Las características de **CI** lógicos digitales se comparan analizando el circuito de la compuerta básica de cada familia. Las parámetros más importantes que son evaluados y comparados son:

FAN-OUT. Especifica el número de cargas normales que puede accionar la salida de la compuerta sin que su operación normal se vea afectada, en otras palabras, el **FAN-OUT** es el número máximo de entradas que pueden conectarse a la salida de una compuerta y se expresa con un número.

DISIPACION DE POTENCIA. Es la potencia necesaria para operar la compuerta y representa la potencia real que se expresa en milivolts (mV)

RETARDO DE PROPAGACION. Es el tiempo promedio de demora en la transición de programación de una señal de la entrada a la salida, cuando las señales binarias cambian de valor. Este intervalo de tiempo se define como el retardo de propagación de la compuerta y por lo general se expresa en nanosegundos (ns).

MARGEN DE RUIDO. Es el máximo voltaje de ruido agregado a la señal de entrada de un circuito digital para que no cause un cambio indeseable a la salida del circuito. Existen dos tipos de ruido, el ruido **DC** (*Corriente Directa*) causado por la desviación en los niveles de voltaje de la señal y el ruido **AC** (*Corriente Alterna*) que es el pulso aleatorio que puede ser generado por otras señales conmutadas. El márgen de ruido se expresa en Volts y representa la máxima señal de ruido que puede ser tolerada por una compuerta sin afectar su funcionamiento.

• Circuitos

El circuito básico de la familia lógica digital **RTL** (Lógica Resistencia-Transistor) es la compuerta **NOR**, mientras que en la familia **DTL** (Lógica Diodo-Transistor) es la compuerta **NAND**, en esta familia cada entrada esta asociada con un diodo y una resistencia de 5 K ohms que forman una compuerta **NAND**.

Lógica HTL. Existen ocasiones en que los circuitos digitales deben operar en un ambiente que produce señales de ruido muy altas y para operar en tales circunstancias, exite una clase de compuerta tipo **DTL**, que posee un umbral alto de inmunidad al ruido, razón suficiente para llamarse compuerta lógica de umbral alto (*High Threshold* : HTL).

Lógica de Inyección Integrada (I^2L). La lógica de inyección integrada es la familia lógica digital más reciente, utilizándose principalmente para funciones de alta escala de integración (**LSI**), gracias a esta característica, su principal ventaja es la densidad de empaquetamiento de compuertas que puede lograrse en cada circuito integrado. La compuerta básica I^2L cuando se conecta a otras compuertas realiza la función lógica **NOR**.

Lógica Transistor-Transistor (TTL). La compuerta básica **TTL** fué una mejora de la compuerta **DTL**, y a medida que la tecnología progresó se agregaron mejoras hasta convertirse en la familia más socorrida para el diseño de sistemas digitales.

La compuerta **TTL** fué construida con diferentes valores de resistencias para producir compuertas con baja disipación y alta velocidad, debido a que el retardo de propagación de una familia lógica depende de dos factores: tiempo de almacenamiento y constante de tiempo **RC** (Resistencia-Capacitor) y conociendo que la velocidad de las compuertas es inversamente proporcional al retraso de propagación. Todas las versiones **TTL** están disponibles en paquetes **SSI** y en funciones más complejas se encuentran en versiones **MSI** y **LSI**. Las diferencias en las versiones **TTL** se concentran en los valores de las resistencias y el tipo de transistores utilizados en su compuerta básica, en cualquier caso, las compuertas **TTL** en todas las versiones vienen en tres tipos diferentes de configuración de salida:

- colector abierto **(OPEN COLLECTOR)**
- tipo totem **(TOTEM-POLL)**
- tres estados **(TRI-STATE)**

Lógica de Emisor Acoplado (ECL). La lógica de emisor acoplado es una familia lógico-digital no saturada, es decir, los transistores no se saturan porque es posible lograr un retardo de propagación de 2 ns o menos, es decir, esta familia cuenta con el menor retraso de propagación y su utilidad principal es en los sistemas que requieren alta velocidad de operación, inmunidad al ruido y baja disipación de potencia. Las salidas proporcionan funciones **OR** y **NOR**, asimismo, las salidas de dos o más compuertas **ECL** pueden ser conectadas conjuntamente para conformar una lógica alambrada y cumplir una función específica.

Semiconductor de Oxido-Metal (MOS). El transistor de efecto de campo **(FET)** es un transistor unipolar, ya que su operación depende del flujo de un solo tipo de portador, existen dos tipos de **FET**: el transistor de juntura de efecto de campo **(JFET)** y el semiconductor de óxido de metal **(MOS)**; el primero se ocupa en circuitos analógicos y el último en circuitos digitales. En comparación con las familias lógicas bipolares, la familia **MOS** es más lenta en cuanto a velocidad de operación, requieren menor potencia, tienen menor márgen de ruido y requieren mayor intervalo para suministro de voltaje; la ventaja sustancial es que requieren menos espacio en el área del **CI** para la implementación de compuertas.

Semiconductor de Oxido-Metal Complementario (CMOS). La familia **MOS** complementaria utiliza los **MOSFET** de canales P y N en el mismo circuito, la familia **CMOS** es más rápida y consume menos potencia que las otras. Estas ventajas son opacadas por la elevada complejidad del proceso de fabricación y menor escala de integración. El operar los **CMOS** con valores altos de voltaje producen mayor disipación de potencia, sin embargo, el tiempo de retardo de propagación disminuye y el margen de ruido mejora. Las ventajas de los **CMOS** pueden enunciarse del modo siguiente:

- Baja disipación de potencia
- Excelente inmunidad al ruido
- Amplio rango de voltajes de suministro

Familia Lógica IC	FAN-OUT	Disipación de Potencia (mW)	Retardo de propagación (ns)	Margen de Ruido (V)
Estándar TTL	10	10	10	0.4
Schottky TTL	10	22	3	0.4
TTL baja potencia	20	2	10	0.4
ECL	25	25	2	0.2
CMOS	50	0.1	25	3

Figura 1.8 Tabla comparativa de las diferentes familias de circuitos integrados.

Qué es un dispositivo lógico programable (PLD)?.

Es un circuito integrado digital capaz de ser programado con gran variedad de funciones lógicas diferentes, porque su programación se realiza por medio de la fusión de celdas dentro del **CI**, entre los dispositivos programables encontramos **EPROM, EEPROM** o **RAM** (Memoria de acceso aleatorio), pero hay que hacer hincapié en que nos referimos con acceso aleatorio al hecho de que tardan el mismo tiempo en accesar cualquier localidad de memoria.

Existen **PROM** de alta velocidad, sin embargo, los más lentos son los dispositivos **MOS/ CMOS** (Semiconductores de óxido de metal/Semiconductores de óxido de metal complementario), **EPROM** y **EEPROM**; los cuáles tradicionalmente no son considerados como **PLD**. Algunos dispositivos **MOS** de alta velocidad **EPROM** están disponibles, pero generalmente los dispositivos estándar tienen accesos relativamente lentos -típicamente 200 ns (nanosegundos) o más- esto los hace poco prácticos.

La tabla 1.9 muestra algunos parámetros de comparación entre los dispositivos **ROM** y las compuertas estándar, que pueden ser considerados para la implementación de un diseño:

	ROM	COMPUERTA
DISEÑO	SENCILLO	COMPLEJO
FABRICACION	FACIL	DIFICULTOSO
MANTENIMIENTO	FACIL	DIFICIL
COSTO	ALTO	REDUCIDO
VELOCIDAD	BAJA	MUY RAPIDO

Tabla 1.9 Criterios comparativos para la implementación con ROM vs COMPUERTAS ESTANDAR

Por qué utilizar PLD?

En la actualidad el diseño lógico digital tiene la oportunidad de abarcar seis categorías primarias:

*Dispositivos estándar **SSI/MSI** (Pequeña escala de integración/Mediana escala de integración).*

Incluyen algunos dispositivos llamados de la serie 400, así como dispositivos **CMOS, TTL** y algunos otros comunes, que son relativamente inexpansibles o para bloques de construcción lógica. La prioridad de la lógica programable disponible en arreglos de compuertas y celdas estándar, hacen que estos dispositivos sean de alta duración para muchos sistemas digitales. Cada día son más recurridos los sistemas digitales que provienen de buffers, registros de corrimiento, contadores, flip-flops, compuertas y otras funciones lógicas.

Estos dispositivos se dice que no son caros desde el punto de vista del costo por **CI** (típicamente abajo de $0.20 U.S.D. en compuertas simples), pero sí lo son, desde el punto de vista del costo por compuerta. El bajo costo unitario es resultado de los grandes volúmenes de producción, ayudado por las tecnologías ya maduras de los semiconductores de pequeño tamaño.

*Dispositivos estándar **LSI/VLSI** (Gran escala de integración/Muy grande escala de integración).*

La segunda categoría es comúnmente conocida como dispositivos estándar **LSI / VLSI** (grande / muy grande escala de integración) donde se incluyen algunos dispositivos como Microprocesadores, Controlador de interrupciones, Controladores de acceso directo a memoria **(DMA)** y algunos otros periféricos. Con una producción de volúmenes razonables, estos dispositivos de alto nivel, funcionalmente rápidos y comparativamente de bajo costo tienen alta revolución digital. Los encontramos disponibles en un rango de precios entre $2.00 U.S.D y por debajo de los $20.00 U.S.D.

Dispositivos de esta categoría se consideran con el mejor costo por compuerta porque son dispositivos de diseño completo, reflejando un costo menor porque la manufactura es de altos volúmenes, y como consecuencia un bajo costo por unidad de fabricación.

Dispositivos con Arreglo de compuertas.

Se dice que estos dispositivos contienen "un mar de compuertas" porque se incrementa tanto el precio como el número de compuertas para su uso, generalmente son prefabricados con silicón y dos capas finales de tres diferentes metales. Los arreglos de compuertas están disponibles desde 1,000 compuertas hasta 50,000 o más, aunque los populares se encuentran en tamaños de 10,000 compuertas.

Las arquitecturas de estos dispositivos no permiten un uso eficiente de estos chips dada la eficiencia de utilización por compuerta que está alrededor del 40% y en casos de eficiencia máxima al 75% para tareas sencillas. Las interconexiones entre compuertas pueden estar definidas como arreglos de compuertas para algunos diseños que envuelven cargas de Ingeniería no Recurrente **(NRE)**, es decir ingeniería que aún no ha sido del todo automatizada.

Dispositivos estándar con celdas.

Son arreglos de compuertas de diseño semicompleto que envuelven **NRE** para la generación del diseño. Utilizan dispositivos que contienen librerías de celdas como bloques funcionales utilizando tecnologías **SSI, MSI** y **LSI**, algunas de las ventajas que se ofrecen al utilizar celdas estándar, es la construcción de bloques, donde cada celda es una librería de celda por separado, un circuito distinto, eficientemente diseñado para una implementación particular y una función de celda única.

Los costos pueden ser menores que los arreglos de compuertas dependiendo del volúmen de aplicación y las celdas que son utilizadas para uso común.

Dispositivos de diseño completo.

Son desarrollos que envuelven costosos sistemas **NRE**, así como largos tiempos para proporcionar alternativas de grandes volúmenes de aplicación. Aplicaciones típicas ocurren en empresas automotrices así como fabricantes de calculadoras y relojes digitales, por lo que resultan eficientes pero su desarrollo es extremadamente lento (ver cuadro comparativo de la figura1.10).

Dispositivos lógicos programables.

Combina muchos dispositivos estándar con arreglos de compuertas, resultando una herramienta muy versátil, de costo reducido, similar a los arreglos de compuertas y que pueden ser de funciones extremadamente diferentes dependiendo de su programación, convirtiéndose en dispositivos **ASIC** (circuitos integrados de aplicación específica).

La atracción particular de los dispositivos programables es la siguiente:

- Sistemas de diseño que no tienen buen desempeño bajo niveles de compuertas lógicas;
- Descripción funcional del control y el camino de datos que pueden ser cumplidos directamen te con las descripciones de los **PLD**.
- Subsistemas de complejidad arriba de 1000-2000 compuertas de dos entradas que pueden ser implementadas en un simple CI.
- Con los **PLD** reprogramables, los errores pueden ser corregidos en el prototipo sin una posible inversión costosa.
- Las demandas del diseño asistido por computadora son aún muy grandes en los **PLD**.
- Los modernos **PLD** tienen velocidades y consumos de potencia comparables con los diseños de alta velocidad de la lógica **CMOS**.
- El capital invertido es bajo.

- El tiempo de diseño es más corto que con los dispositivos **ASIC**.

- El diseño del circuito impreso puede llegar a simplificarse porque tenemos posibles espacios libres para la colocación de los pines y se pueda dar buen seguimiento a la señal.

LOGICA ALTERNATIVA	COSTOS DE NRE (típicos)	TIEMPOS DE DESARROLLO	INTERACCION CON EL DISEÑO	COSTOS RELATIVOS POR COMPUERTA	VOLUMEN DE PRODUCCION
DISEÑO COMPLETO	muy altos	> 1 año	NRE + meses	muy bajos	> 100,000
CELDAS ESTANDAR	> $ 50,000	meses	NRE + tiempo del dispositivo	bajos	> 20,000
ARREGLO DE COMPUERTAS	$ 10,000 - $ 50,000	semanas-meses	NRE + tiempo del dispositivo	mediano a bajo	> 5,000
PLD	$ 0	horas-semanas	horas	medio	> 10,000
LSI/VLSI	$ 0	nada	-	muy bajo	algunos
SSI/MSI	$ 0	nada	-	muy alto	algunos

Figura 1.10 Comparación entre las diversas alternativas para implementaciones lógicas.

Análisis de costos de producción.

Algunas consideraciones que deben tomarse en cuenta en el área de costos (Figura 1.11) al utilizar los **PLD** son los siguientes:

- Dispositivos extra para compensar los dispositivos defectuosos
- Diseñar un dispositivo disponible con poco tiempo de retardo
- Envuelve prueba/inspección
- Inventario
- Programación
- Pruebas de post-programación
- Devolución de dispositivos defectuosos
- Cambios de diseño de rendimiento
- Fallas en el sistema del **PLD**
- Fallas en los campos del **PLD**

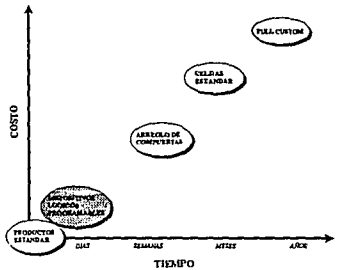


Figura 1.11 Comparación de costos vs tiempo de desarrollo de las diferentes alternativas de dispositivos lógicos

Tendencias futuras de los PLD

La búsqueda de un **PLD** ideal marca la tendencia de los cambios en la tecnología, el compás en el empaquetamiento, consumo de potencia, confiabilidad, respuesta, facilidad de uso y otros factores, los compromisos están marcados para cada una de las áreas.

Otra tendencia importante es el diseño de arquitecturas de propósito general moderadamente rápidas, hacia la búsqueda de un dispositivo estándar; la demanda es enfocada hacia una herramienta flexible, de alta densidad, muchas salidas, veloz, bajo consumo de potencia, bajo costo, reprogramable y con el mejor soporte tanto en hardware como en software.

El conocimiento de aplicaciones especiales permite diseñar dispositivos de aplicación específica algunas veces llamados **PLD** de función específica, por ejemplo, decodificador de direcciones, máquinas de estados, buses de microprocesadores y desarrollos especializados parecen ser los "blancos" fundamentales de la aplicación de estos dispositivos; de la misma manera, microprocesadores, controladores de video, controladores de comunicaciones y otras aplicaciones populares de **LSI** serán incorporadas en la optimización por los **PLD**; procesadores digitales de señales y territorios comerciales de compuertas serán inundados por los bajos costos **"high-end"** de los Dispositivos Lógicos Programables.

La tecnología **CMOS** implementada en estos circuitos permitirá retardos de propagación por debajo de los 12 ns., mientras que los dispositivos de galio-arsénico permitirán cerca de 6 ns. de retardo. Esto permitira la aparición de un nuevo empaquetamiento de dispositivos llamados **PLCC** (empaquetamiento de un chip plástico) de alta densidad.

1.3 BASES PARA EL DISEÑO DE LOGICA PROGRAMABLE

El empleo de la lógica programable en diseños digitales principia con la matriz de diodos con fusiones de aluminio en los puntos de intersección, esto es, a principios de los años sesenta. Después éstos evolucionaron a lo que son las **PROM**, a través de la adición de un decodificador a las entradas, el resultado fue una memoria direccionable que también puede ser vista como un dispositivo lógico universal con una matriz de compuertas **AND** fija (decodificador) alimentando a una matriz de compuertas **OR** programable (arreglo de diodos). Un circuito **PROM** representativo es el que se muestra en la figura 1.12.

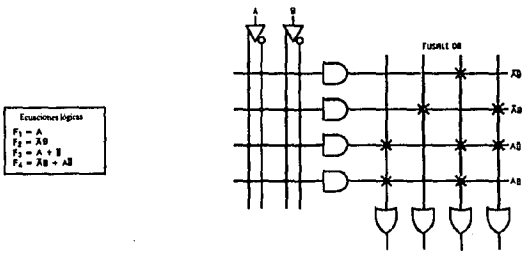


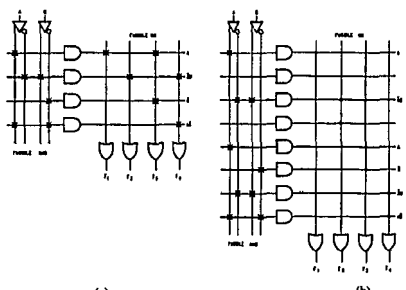
Figura 1.12 Funciones Lógicas y Arquitectura de una PROM con un arreglo AND Fijo y un arreglo OR Programable.

La desventaja del uso de las **PROM** como dispositivo lógico deriva de su universalidad, es decir, el número de términos de productos disponibles es 2^n, donde n es el número de variables de entrada, cada variable adicional (pines de entrada) dobla el tamaño de la matriz y como resultado, las **PROM** comerciales ofrecen entradas limitadas (p. ej. los pines de direcciones). Este método rápidamente degradó el desempeño del dispositivo, debido a la lógica de decodificación y a las dimensiones del arreglo, así como el incremento del costo por uso ineficiente del circuito integrado. Muchas aplicaciones lógicas requieren de más entradas, pero no la flexibilidad de un decodificador completo.

Este dilema fue resuelto introduciendo una segunda matriz de fusión en lugar del arreglo de decodificadores, permitiendo únicamente la selección de aquellos términos de producto requeridos por el diseño, haciendo más eficiente el uso de la matriz programable, al igual que las **PROM**, la matriz fue elaborada a base de fusibles que podían ser configurados en el campo siendo llamados Arreglos Lógicos Programables (**PLA**). La arquitectura básica de un **PLA** se muestra en la figura 1.13(a), al contrario de las **PROM**, los **PLA** pueden manejar funciones lógicas requiriendo más variables de entrada con menos de 2^n términos productos.

Sin embargo, los fusibles adicionales de la segunda matriz en el **PLA** requieren de circuitos adicionales de selección y programación que los hace más caros que una **PROM**, por otra parte, un arreglo programable implica un retardo mayor que un decodificador alambrado, siendo el factor dominante en el retardado de un señal a través de un **PLD**. Por lo tanto es difícil para un **PLD** con arreglos programables **AND** y **OR** competir con la velocidad de un **PROM** de tamaño similar.

Costos eficientes muy similares con los de los dispositivos **PROM** fueron alcanzados únicamente al reducir los circuitos redundantes. Una solución fue alambrar el arreglo **OR** y permitir al usuario programar el arreglo **AND.** Este arreglo es conocido como arquitectura de Arreglo Programable Lógico (**PAL**), que se muestra en la figura 1.13(b). fué la llave que abrió el potencial de la lógica programable a los diseñadore;. el dispositivo **PAL** es más efectivo en costo que un **PLA** y puede sustituir la flexibilidad del arreglo **OR** al ofrecerse en una variedad de configuraciones básicas. Asimismo, como se redujo el número de arreglos programables a través de los cuales debe pasar una señal lógica, se incremento considerablemente el desempeño del dispositivo.



(a) (b)

Figura 1.13. Arquitectura PLA con arreglos AND y OR programables (lado izquierdo). Arquitectura PAL con arreglo AND programable y arreglo OR fijo (lado derecho)

El desarrollo del concepto del **PAL** condujo a familias de productos de diversas tecnologías, ofreciendo un rango de bloques de diseño, requerimientos de energía y desempeño. Los avances en el desarrollo de la tecnología de fusión de celdas han conducido a las celdas programables de fusibles (verticales) ofreciendo mayores alcances de programación y una propagación de señales más rápida., las ventajas de los dispositivos uniprogramables descritos arriba permiten configurar los circuitos integrados en el campo.

Una vez fundidas las celdas, no pueden ser configuradas. Existirían mayores ahorros si los **PLD** fueran reconfigurados, con lo cual se permitiría reemplear el dispositivo y hacer pruebas exhaustivas de fábrica para alcanzar una mayor programabilidad y mejorar su confiabilidad.

Desarrollos recientes en la tecnología de semiconductores han hecho accesibles las celdas borrables eléctricamente para dispositivos lógicos y memorias. Dichas celdas reconfigurables han sido empleadas para crear los Arreglos Lógicos Genéricos (**GAL**), dado que ofrecen todas las configuraciones lógicas que pudieran requerirse así permitir la modificación de prototipos y también la reconfiguración o mejoramiento de sistemas en el sitio en que se encuentran instalados.

VENTAJAS DE DISEÑO

Los diseñadores de sistemas lógicos normalmente trabajan bajo ciertas restricciones, la reducción del costo y tamaño del sistema requieren diseños compactos y eficientes, la confiabilidad del sistema fuerza a los diseñadores a decidir entre nuevas soluciones y métodos existentes probados. Revisiones futuras demandan un grado de flexibilidad que deberá ser anticipado, al incrementarse la complejidad de los sistemas, la sofisticación de los componentes requiere herramientas aún más poderosas y la concepción del diseño óptimo para tantos parámetros requiere un rango de habilidades que deberán ser constantemente desarrolladas.

Los **PLD** no resuelven todos estos problemas pero proporcionan una forma eficiente para manejar los principales puntos de una metodología uniforme.

Las principales ventajas desponibles a los diseñadores a través del uso de **PLD** se describen a continuación:

• DISEÑO SIMPLIFICADO DEL SISTEMA

Los **PLD** permiten a los usuarios especificar exactamente las funciones que serán implementadas en la lógica. Esto evita el problema de la interconexión de varios componentes **SSI** para obtener los mismos resultados. Al mismo tiempo, los **PLD** ofrecen ventajas de velocidad debido a la reducción de interconexiones. La metodología consiste en escribier las ecuaciones que definan la función deseada con la ayuda de un programa de computadara (software) y utilizando las ecuaciones para configurar al dispositivo. Esta metodología acelera tanto la concepción como la implementación del diseño. Un beneficio más que nos proporcionan los **PLD** es que en la mayoría de los cambios que se necesitan hacer o incorporar al diseño después de la implementación pueden ser reacomodados alterando la configuración interna del **PLD**.

• INCREMENTO DE LA DENSIDAD FUNCIONAL

A pesar de que el desarrollo de los dispositivos **LSI** y **VLSI** integran una gran cantidad de lógica en un solo circuito, los diseñadores de sistemas han tenido problemas con la potencia, el espacio y manejo asociado con una gran cantidad de paquetes **SSI/MSI** empleados para diseños específicos que son requeridos, mismos que no se encuentran disponibles, lo anterior decrementará la densidad funcional.

Los **PLD**, sin embargo, ofrecen una solución compacta con una alta funcionalidad y menos desperdicio en líneas de entrada-salida e interconectadas, de esta manera, la densidad funcional puede igualar la densidad de circuitos diseñados específicamente para el usuario sin los cambios de ingeniería asociados. Por otro lado, la combinación de varias funciones en un solo circuito integrado reduce la potencia, así como el espacio y tiene además la ventaja de mejorar el desempeño del sistema al reducir las interconexiones.

Por los anteriormente expuesto, podemos clasificar las familias con cierta prioridad, la figura 1.14 nos muestra las categorías básicas en el entorno del diseño digital.

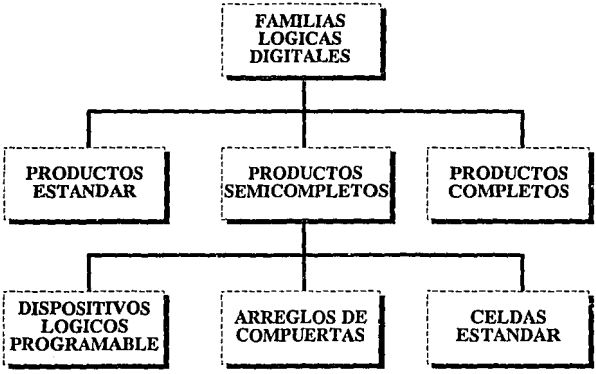
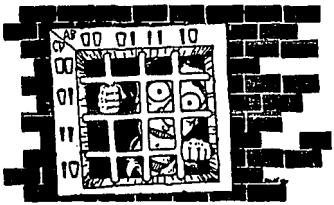


Figura 1.14 Categorías básicas de la Lógica Digital



CAPITULO II

TECNICAS DE PROGRAMACION DE LOS PLD

Una vez que sabemos el performance de los **PAL**, veremos las técnicas actuales de su programación, así como el software y hardware que se utiliza para este fin.

Para saber físicamente como se lleva acabo esta tarea, conoceremos las arquitecturas internas de los sistemas digitales, y en particular del dispositivo de interés, posteriormente nos enfocamos al ambiente gráfico el cual mencionaremos a partir de los otros capítulos.

II.1 TECNICAS ACTUALES DE PROGRAMACION DE PLD

Tecnologías bipolares

El primer desarrollo de **PLD** fué adoptado para la tecnología existente en memorias **PROM**, los dispositivos basados en tecnologías bipolares almacenaban operaciones en altas velocidades de manera que la programación se realizaba mediante ligaduras de fusibles (Figura 2.1). El mayor enemigo de estos dispositivos bipolares era el calor, por esta razón se debe tener cuidado durante la programación para no rebasar el límite de voltaje y duración estimada del pulso de programación.

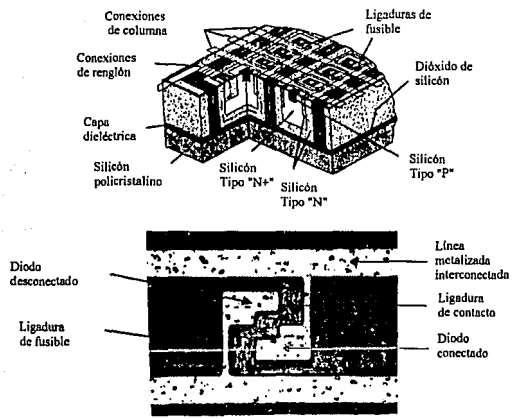


Figura 2.1 Estructura monolítica interna de un PAL

Los requerimientos de los **PLD** para programar los fusibles son:

- Algunos requieren un voltaje alto (H) en el pin **Vcc** al momento de la programación
- Otros utilizan una combinación de varios niveles de voltaje en diversos pines de entrada.
- Cada fusible tiene una dirección específica que es utilizada en una secuencia de programación para ser fundido.

Los dispositivos **PAL** bipolares se basan en dos tecnologías conocidas como:

1. **IMOX** (*Ion-Implanted-Oxide-Isolated Structure* : Estructura de implantación de iones de óxido). **IMOX** proporciona dispositivos de alto rendimiento con campos programables, generalmente estos dispositivos son diseñados básicamente para la familia **AM2900**.
2. **Fusibles platino-silicio**. Estos dispositivos se estabilizan más rápidamente y tienen buen desempeño en campos programables.

Técnicas de fusión

Los circuitos **PAL** son diseñados para utilizar un algoritmo de programación que pueda minimizar minimizar los requerimientos en el programador universal para una quema de fusibles rápida y eficiente, la secuencia para programar un **PAL** es la siguiente (Figura 2.2):

1. Aplicar un voltaje **Vcc** al chip
2. Seleccionar la dirección del fusible a programar mediante niveles de voltaje para tecnologías **TTL/SL** apropiados para los pines de dirección
3. Las salidas se deshabilitan
4. El voltaje de programación es aplicado a una salida
5. El fusible habilitado es acompañado por un pulso de entrada y un nivel de operación de voltaje normal para **TTL**. Esta acción en las compuertas hace pasar un flujo de corriente por los fusibles apropiados, resultando una apertura de fusible en pocos microsegundos
6. El voltaje de programación de la salida es bajo (**L**) y posteriormente anulado
7. El dispositivo es habilitado, se requirie un pulso de reloj para el estado de salida, indicando que la programación se ha llevado a cabo. Si la programación no ha ocurrido una secuencia prolongada de pulsos es aplicada hasta que se efectúe la programación
8. La secuencia de los pasos 2 al 7 es repetida para cada bit que necesite ser programado

Esta técnica tiene sus ventajas puesto que es relativamente sencilla y puede ser utilizada por otros fabricantes. Existe una técnica alternativa que consiste en colocar dos fuentes de voltaje controladas por corriente para aplicar un nivel de voltaje a la salida, sin necesitar tiempos críticos de programación. Con este procedimiento se obtienen mayores ventajas puesto que el fusible no permanece abierto durante el curso de programación, en cambio, con el fusible de platino y silicio los pulsos prolongados no dañan al dispositivo; este algoritmo es diseñado para minimizar el tiempo de programación mediante el manejo de un pulso rápido seguido por uno prolongado.

La Migración Inducida en Avalancha (**AIM**) es una nueva tecnología de programación basada en un transistor NPN abierto por la base, el transistor es localizado por cada juntura programable en dos caminos diferentes para una misma señal, que al coincidir hace funcionar al colector y al emisor fundiendo la juntura y haciendo que la corriente se "caiga" de tal manera que funde el fusible; existen también metodologías para **PLD** borrables con luz ultravioleta (**EPLD**) y **PLD** eléctricamente borrables (**EEPLD**).

Durante la programación es conveniente tener en cuenta las siguientes precauciones:

El número de fallas posibles con la tecnología utilizada puede ser debido a la adquisición de un equipo de mala calidad o que no ha sido calibrado apropiadamente, o bien, debido a que los voltajes críticos necesarios para los pines no son los correctos.

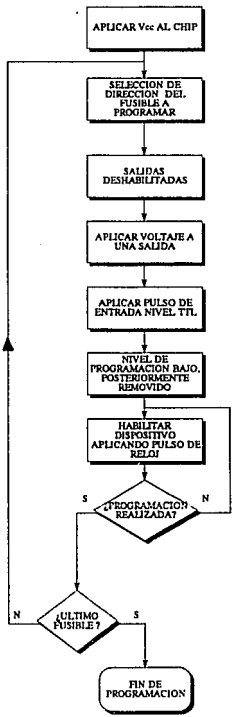


Figura 2.2 Algoritmo de programación de fusibles

Características del fusible

Cuando un pulso rápido (menor que 500 nanosegundos de tiempo de levantamiento) es aplicado a un fusible, este toma un valor, determinado por el cuadrado de la temperatura de las resistencias que es aproximadamente de 2 Volts, este valor es pequeño independientemente de la corriente aplicada. Durante el período de tiempo en que el fusible intenta separarse, el platino y el silicio se dividen en dos secciones distintas, indicando una separación limpia del fusible (Figura 2.3); entonces se dice que la estructura ha sido fundida debido a la concentración de una densidad de energía en el centro del aluminio que interconecta las líneas, cuando esto pasa se dice que el centro del fusible es una superficie de tensión.

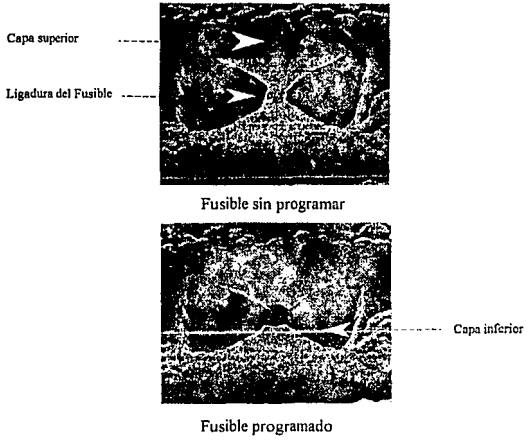


Figura 2.3 Aspecto del fusible antes y después de la programación

II.2 SOFTWARE Y HARDWARE ORIENTADO A LA PROGRAMACION DE PLD

Cuando los **PLD** fueron introducidos en 1981, el mapa de fusibles se realizaba de forma manual, ahora, es conveniente tener un software de desarrollo tipo ensamblador para que la elaboración del mapa sea agilizada, el software proporciona la facilidad de convertir los diseños en un formato entendible mediante dispositivos programadores utilizados en plataformas diversas, por ejemplo, PC (Computadora Personal) y compatibles.

Los paquetes de software más comunes pueden manejar todos los tipos de **PLD** estándar de propósito general y se clasifican en dos grandes categorías: ensambladores y compiladores.

- Los ensambladores generalmente están enfocados al uso de símbolos y señales que pueden mandar niveles a un fusible, esto favorece a las ecuaciones Booleanas.
- En contraste con los ensambladores de bajo nivel de tipo Booleano, encontramos los compiladores que permiten describir en algún lenguaje de alto nivel un diseño. Un compilador universal (Figura 2.4) ofrece una sintáxis general y de "**fácil**" descripción para programar dispositivos lógicos sin importar el fabricante.

Cuando hablamos de un software de Diseño Lógico Programable en términos de un compilador (universal) se refiere a la habilidad de éste para soportar la mayor cantidad dispositivos lógicos programables, así como una gran variedad de diseños en los formatos de entrada entre los cuales encontramos: máquinas de estado, ecuaciones Booleanas de alto nivel, tablas de verdad o diseños lógicos esquemáticos; en algunas literaturas estos programas son conocidos con el nombre de **BEE+** (*Boolean Equation Entry* : Ecuaciones Booleanas de entrada).

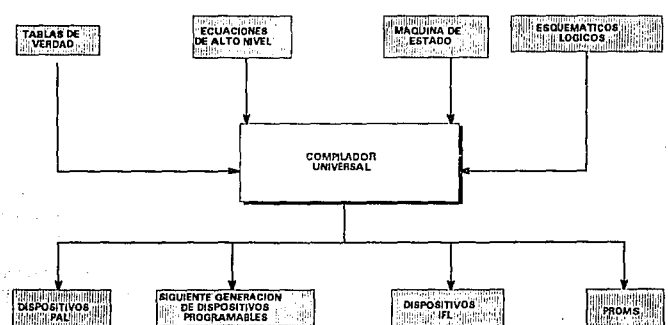


Figura 2.4 El Compilador Universal.

A continuación se mencionan algunos de los fabricantes de software de **PLD** asistido por computadora que pueden estar disponibles para los programadores.

ABEL Sistema para soportar lenguajes de entrada que almacenan diseños donde se puedan describir de manera familiar el diseño de ecuaciones de alto nivel, tablas de verdad y diagramas de estados. Proporciona suma de productos Booleanos que son convertidos a mapas de fusibles en el estándar **JEDEC, ABEL** esta disponible para computadoras personales y para sistemas de estaciones de trabajo **SUN** y **VAX**.

AMAZE (*Automated Map And Zap Ecuation*) Signetics. Incluye un ensamblador que puede manejar las arquitecturas de los dispositivos **PLA** de Signetics, el lenguaje que maneja es llamado programa estándar Signetics; acepta ecuaciones Booleanas, tabla de Estados, esquemáticos en formato FutureNet u Orcad. Incluye un simulador de dispositivos y es capaz de generar automáticamente vectores de prueba, convirtiendo los archivos a formato **JEDEC**.

APEEL Desarrollado por Internacional **CMOS** Tecnology. Es únicamente para **PLD** eléctricamente borrable (**EEPLD**) con salidas estándar **JEDEC** para programadores universales, soporta sumas de productos Booleanas y contiene un editor de texto para procesar lenguaje y un simulador de **PLD**.

A+ PLUS Altera Corporation. Ofrece soporte de múltiple diseño con formatos de entrada que incluyen ecuaciones Booleanas, descripción de estados y esquemáticos. Soporta programación jerárquica que puede ser modular para ser incorporada a diseños grandes. Acepta descripción de diseños en forma variada mediante el lenguaje de diseño Altera (**ADL**).

CUPL de Logical Devices Incorporated. Esta provisto de una variedad de entrada de diseños que incluye ecuaciones Booleanas, tablas de verdad y diagramas de estados, básicamente es similar al paquete **ABEL**

PLDS II Intel Corporation. Es un sistema similar al diseñado por Altera con mucha potencia en los diseños que pueden ser automatizados, optimizando la utilización de dispositivos complejos con un simulador lógico y analizador de tiempos.

ORCAD/PLD Producto basado en captura de esquemáticos que pueden admitir ecuaciones Booleanas, tablas de verdad, descripción de estados y esquemáticos para **PLD**. El software de **OrCad** es mucho menos poderoso que **ABEL** o **CUPL**, sin embargo, incluye módulos de minimización lógica y de interfase para el simulador **Orcad/VST**. La interfase es leída através del formato **JEDEC** en **Orcad/MOD** para crear los tiempos de circuito de los **PLD**.

PALASM - Advanced Micro-Devices. Primer ensamblador de **PLD** con respuesta rápida para los **PAL**, utiliza las conversiones en formato **JEDEC** y fue concebido únicamente para aceptar entradas de ecuaciones booleanas de la forma suma de productos.

PLAN National Semiconductor. Software desarrollado y distribuido por la compañía para soportar sus dispositivos. Tiene una sintáxis basada en el **PALASM** pero en ocasiones es mucho más flexible. Soporta dispositivos independientes del método de diseño, incluye selección automática de dispositivos y ecuaciones Booleanas de la forma suma de productos, tabla de estados y esquemáticos. El **PLAN** esta disponible para computadoras personales de manera gratuita con cargo a la Cia. National.

TANGO-PLD Accel Tecnologies Incorporated. Es otra manera de utilizar herramientas de esquemáticos para diseñar con **PLD**. Es un producto sofisticado considerando su bajo precio. Contiene un lenguaje de diseño llamado **TDL**, es similar a la programación en lenguaje C, utilizando el algoritmo de minimización lógica Expresso, la simulación y los vectores son almacenados en archivos y la transferencia se efectúa en formato **JEDEC**. (Figura 2.5)

HARDWARE

La herramienta de hardware utilizada por los **PLD** es un programador que físicamente configura los dispositivos, generalmente, los programadores vienen sólo con una conexión para la plataforma de computadora mediante una conexión al puerto del tipo RS232C, el cual puede hacer llamadas hacia archivos con formato estándar **JEDEC**.

PAQUETE DE SOFTWARE	*DEPENDIENTE/ INDEPEND.*	*SOPORTE DE DISPOSITIVOS*	*ECUACIONES BOOLEANAS*	*TABLAS DE VERDAD*	*ESQUEMATICO*	*NETLIST (CONEXIONES)*	*MAQUINA DE ESTADOS*	*FORMA DE ONDA*	*REDUCCION LOGICA*	*COSTO APROXIMADO*
PALASM2 MAII (AMD)	D	Propios de la compañía	X				X			0
AMAZE Signetics	D	Propios de la compañía	X		X		X			0
PLAN National	D	Propios de la compañía	X							200
HELP Harris	D	Propios de la compañía	X						Opcional	0
PLPL AMD	D	Propios de la compañía	X				X		X	0
ABEL ICT	D	Propios de la compañía (PEEL)	X							0
A+PLUS Altera Plus	D	Propios de la compañía	X	X	X	X	X		X	3000
IPLDS II Intel	D	Propios de la compañía	X	X	X	X	X		X	3500
ERASE Dev Exel	D	Propios de la compañía	X	X	X	X	X		X	-
XACT Xilinx	D	Propios de la compañía (LCA)	X	X	X				X	3000
ACTION LOGIC Actel	D	Propios de la compañía	X		X				X	-
CUPL Logical Devices	I	Universal	X	X	X	X	X		X	1000
ABEL Data I/O	I	Universal	X	X	X	X	X		X	1500
PLDesigner Mrc	I	Universal	X	X	X	X	X	X	X	4500
PLD Design Sys HP	I	Universal	X	X	X	X	X	X	X	8000
LOGAC Elan, Cetus	I	Universal	X	X	X		X		X	2000
ELDS Pinball	I	Universal (CMOS PLD's)	X							800

Figura 2.5 Resumen del software más difundido para el diseño de PLD

Generalmente, los programadores (Figura 2.6) pueden encontrarse en dos categorías: dispositivos generales y dispositivos específicos:

- . Los programadores de dispositivos generales están disponibles para programar una tercera parte de los dispositivos estándar ya fabricados.
- Los programadores de dispositivo específico son normalmente producidos por un fabricante para PLD no estándar o herramientas disponibles que no han sido publicadas en detalle.

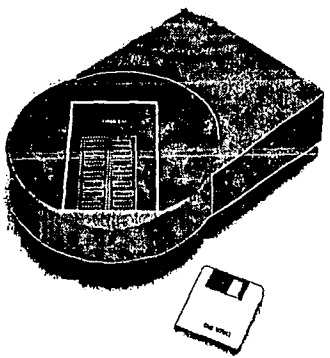


Figura 2.6 Moderno programador universal de PLD

Regularmente todos los programadores están conformados para el estándar **JEDEC** de mapa de fusibles. El programador de dispositivo general ha sido desarrollado como una herramienta inicialmente utilizada para configurar **PROM**, pero muchos de los programadores de primera generación muestran los requerimientos considerables para la intervención de un operador que complete la función de programar el dispositivo, es por esta razón que los últimos desarrollos han producido herramientas más sofisticadas para la culminación de sistemas de datos de entrada y salida.

Casi todos los programadores consisten de una cabina para el control electrónico donde se conecta el dispositivo, así como una plataforma de software mediante comunicación para poder traer archivos con la configuración del mapa de fusibles en formato **JEDEC**, el panel de control por lo general es el teclado y los datos de entrada pueden ser accesados en forma manual por un conector universal que adapta la circuitería con el algoritmo de software y/o sockets para el dispositivo específico. Los programadores universales recientes contienen sockets para manejar pines y software configurables al dispositivo.

Virtualmente todos los programadores solicitan un puerto de comunicación para conexión a la plataforma provista de un control "inteligente" para la transferencia del archivo con el mapa de fusibles. Algunas tarjetas son instaladas en plataforma tipo **PC** con conexión directa al programador (Figura 2.7).

Figura 2.7 Conexiones del programador a una plataforma tipo PC

Alguno criterios para la aprobación de un programador son los siguientes:

Todos los detalles de parámetros requeridos por el programador universal, formas de onda y esquemas de dirección son proporcionados por la hoja de especificaciones de cada dispositivo. Sin embargo se enuncian algunos de los requerimientos mínimos para un programador.

1. Soporte de todos los productos **PLD** estándar
2. Soporte al menos para el 98 % de los productos programables de una compañía en particular
3. Aceptar al estándar **JEDEC** vía un puerto de entrada RS232C
4. Poder generar el checksum del estándar **JEDEC**
5. Verificar voltajes extremos altos y bajos después de la programación
6. Programación y verificación del fusible de seguridad
7. Soporte para prueba de vectores del estándar **JEDEC**
8. Rápida programación y verificación

II.3 ARQUITECTURAS INTERNAS

La mayoría de los sistemas digitales están constituidos por compuertas lógicas combinadas con dispositivos de memoria, a porción combinada acepta señales lógicas de entradas externas y de las salidas de los elementos de la memoria, así el circuito combinatorio opera sobre estas entradas a fin de producir diversas salidas, algunas se emplean para determinar los valores binarios que se almacenan en los elementos de la memoria. Las salidas de algunos elementos, a su vez, se dirigen hacia las entradas de compuertas lógicas en los circuitos combinatorios, Este proceso indica que las salidas externas de un sistema digital son función de sus entradas externas y de la información almacenada en los elementos de la memoria (Figura 2.8).

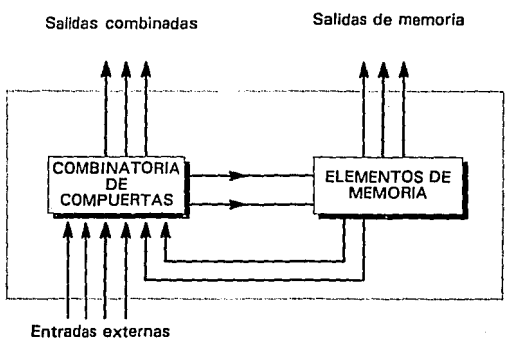


Figura 2.8 Diagrama general de un sistema digital

El elemento de memoria más importante es el **flip-flop** (abreviado FF), que está formado por un ensamble de compuertas lógicas (aunque una compuerta lógica por sí misma, no tiene la capacidad de almacenamiento, pueden conectarse varias de ellas de manera que permiten almacenar información).

El símbolo de un FF tiene dos salidas, marcadas como Q y $\overline{Q}$, que son inversas entre sí, a salida Q recibe el nombre de salida *normal* del FF, mientras que $\overline{Q}$ es la salida *negada o invertida* del FF.

La entrada de un FF sólo tiene que recibir un pulso momentáneo para cambiar el estado de su salida y ésta permanecerá en el nuevo estado aún después de la aparición del pulso de entrada. Esta es la característica de *memoria* del FF.

FLIP-FLOP D

El flip-flop D disparado por flanco de subida, utiliza un circuito detector de flanco para asegurar que la salida tome el valor que se encuentra en la entrada D *sólo* cuando ocurra la transición activa del reloj, si no se usa este detector de flanco, el circuito resultante operará en forma diferente. Este circuito se denomina registro básico D y tiene la configuración mostrada en la Figura 2.9.

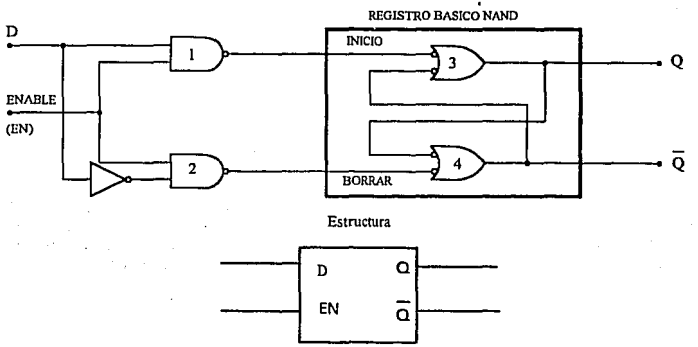


Figura 2.9 Registro básico D

El circuito contiene el registro básico NAND y las compuertas NAND 1 y 2 de conducción se denomina entrada de *habilitación* (*enable*, y se abrevia como EN) en lugar de entrada de reloj, porque su efecto sobre las salidas Q y $\overline{Q}$ no está restringido sólo a sus propias transiciones.

Operación del registro básico D

La operación es la siguiente:

1. Cuando EN se encuentra en ALTO, la entrada D producirá un estado BAJO en cualquiera de las entradas, INICIO o BORRAR, del registro básico NAND para causar que Q tome el mismo nivel de D. Si D cambia mientras EN se encuentra en ALTO, Q seguirá dicho cambio de manera exácta. En otras palabras, mientras EN = 1, la salida Q será una réplica exácta de D; en este estado se dice que el registro básico D es "transparente".
2. Cuando EN va hacia el estado BAJO, la entrada D se inhibe y deja de afectar el registro básico NAND, porque las salidas de las dos compuertas de conducción se mantienen en ALTO. De este modo, las salidasQ y $\overline{Q}$ permanecen en el nivel que tenían justamente antes de que EN cambiara hacia el estado BAJO, en otras palabras, las salidas son "ancladas" en sus niveles actuales de corriente mientras EN permanezca en BAJO, sin importar si D cambia (Figura 2.10).

Entradas		Salida
EN	D	Q
0	X	Q_0 (no cambia)
1	0	0
1	1	1

"X" indica "no importa" "Q_0" es el estado de Q antes de que EN cambie hacia el estado BAJO

Figura 2.10 Tabla de verdad del flip flop D

FLIP-FLOP D SINCRONIZADO POR RELOJ

El FF D sólo tiene una entrada síncrona de control, D, letra que proviene de *dato*. La operación del FF D es muy sencilla: Q va hacia el mismo estado en que se encuentra la entrada D cuando ocurre una *transición con pendiente positiva* (TPP: el reloj cambia de 0 a 1) en CLK. En otras palabras, el nivel presente en D será almacenado en el flip-flop en el momento en que se presente una TPP.

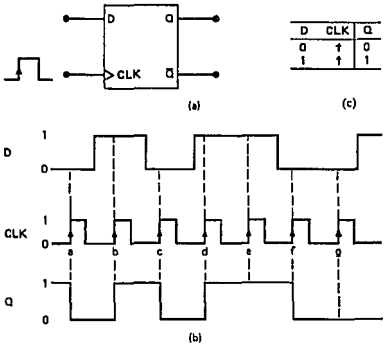


Figura 2.11 (a) Flip-flop D que dispara con transiciones de pendiente positiva; (b) Formas de onda

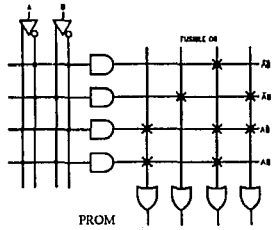
Estructura básica

Los dispositivos que emplean un Arreglo Lógico Programable (**PAL**) poseen varias características en común con las memorias (**PROM**) y los Arreglos Lógicos Programables (**PLA**), ya que estos dispositivos comparten una estructura básica interna a base de compuertas **AND/OR** pero varían en sus características lógicas y en el modo de programación.

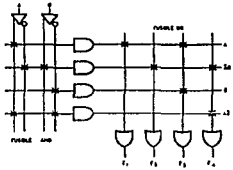
La estructura básica consiste de dos niveles; el primer nivel es un arreglo de compuertas **AND** que acepta las entradas para la creación de funciones Booleanas (*minitérminos o suma de productos*) y el segundo nivel lo constituye el arreglo **OR** que presenta las salidas que forman los *productos de sumas* (Figura 2.12). Así, el arreglo **OR** puede combinar varias funciones **AND** de acuerdo al tipo de dispositivo programable utilizado, es decir, se cuenta con diferentes dispositivos que presentan características específicas tales como:

- número de entradas
- número de salidas
- registros

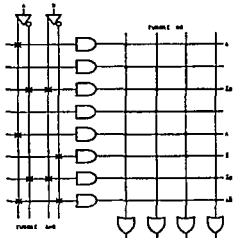
Funciones Lógicas de Diseño



PROM



PLA



PAL

Figura 2.12 Diferencias entre las arquitecturas PROM, PLA y PAL.

La estructura básica hace que los dispositivos programables sean ideales para la implementación de

lógica Booleana, la cual es generada por alguna técnica de diseño lógico para la obtención de funciones Booleanas.

Arquitectura de un PAL

La arquitectura de un **PAL** es inversa a la arquitectura de una **PROM** es decir, en un PAL se programa el campo **AND** y el arreglo **OR** es fijo, mientras que en la **PROM** el campo fijo es el arreglo de compuertas **AND** y el programable es el campo **OR**, además, las entradas no son decodificadas puesto que no se cumple que el número de compuertas **AND** sea 2^n (*donde n es el número de entradas*), razón por la cual los dispositivos **PAL** eliminan la ineficiencia de las **PROM**. Podemos afirmar que la utilización de dispositivos **PAL** permite tener un número de entradas más acorde a nuestras necesidades; en otras palabras, el incremento del número de entradas de los **PAL** no incrementa dramáticamente la cantidad de fusibles requeridos para el empleo del arreglo de compuertas.

Al hablar de fusibles se hace referencia a la conexión física que existe en el arreglo **AND**, así, la programación se lleva a cabo quemando los fusibles por medio de un dispositivo especial que elimina la conexión presentada originalmente. Esta alteración a la configuración interna del dispositivo puede ser temporal o definitiva de acuerdo al dispositivo programador utilizado.

El arreglo de compuertas **OR** de los dispositivos **PAL** es dedicado. En una compuerta **OR** de salida se pueden tener varias conexiones de compuertas **AND** en particular.

Arquitectura del plano AND

Este plano proporciona la interconexión de las entradas (*verdadera o complemento*) que requieren compuertas **AND** de manera lógica para crear de esta forma los términos producto lógico y producto control, de donde el término producto lógico es utilizado para las funciones lógicas y el término producto control es requerido para aplicaciones en las funciones de control como: habilitación de salidas, inicialización, registro precargado y prueba.

El número total de entradas y el término producto es determinado por el tamaño del plano **AND** que ofrece la arquitectura del dispositivo **PAL**, por tal motivo se debe seleccionar el tipo de **PAL** que cumpla con los requerimientos de la función lógica deseada.

Arquitectura del plano OR

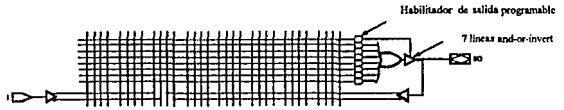
Este plano determina la conectividad de las compuertas **AND** con las salidas y en la cual además se definen tres características principales:

- El número de compuertas **OR**
- El número de términos producto por salida
- La distribución de términos producto

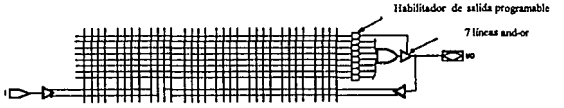
En una arquitectura típica de un dispositivo **PAL**, las salidas de las compuertas **AND** son conectadas a compuertas **OR** mediante el arreglo del plano **OR**. Existe una limitación en los **PAL** a ser considerada,

la cual está presente en el plano **AND-OR** donde hay un número finito de entradas a compuertas **AND** y un número de compuertas **OR** dedicadas como salidas que deben ser tomadas en cuenta al momento de diseñar.

Un término de productos es lógicamente verdadero mientras todas las líneas de entrada conectadas a él (vía celdas programables) sean altas. Si ninguna de las entradas al arreglo, ya sea verdadero o su complemento están conectadas a la línea de productos, entonces esta entrada al arreglo representa un valor "no importa" con respecto a este término de productos (Figura 2.13).



2.13 a) Salida bidireccional activada BAJO



2.13 b) Salida bidireccional activada ALTO

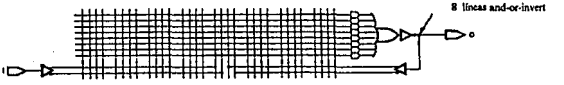


Figura 2.13 c) Salida dedicada activada BAJO

Arquitectura de las macroceldas de salida

Las especificaciones mínimas para las macroceldas de salida se enuncian a continuación:

- Macrocelda con salida normal
- Macrocelda con uso de flip-flop para almacenamiento
- Macrocelda con salidas organizadas
- Macrocelda con flexibilidad para realimentación

Por otro lado, la(s) salida(s) se puede(n) configurar para ser utilizadas de acuerdo a las siguientes propiedades lógicas requeridas por la aplicación del diseño: Salida secuencial y combinacional.

Salida secuencial

Los circuitos secuenciales se clasifican en dos grandes grupos: asíncronos y síncronos

En los sistemas secuenciales asíncronos, los cambios de estado se producen en cuanto están presentes las entradas adecuadas, con los retrasos inherentes a las velocidades finitas de conmutación de los dispositivos físicos utilizados.

En un sistema secuencial síncrono, por el contrario; los cambios de estado se producen únicamente cuando, además de estar presentes las entradas adecuadas, se produce la transición de una cierta señal, compartida por todos los flip-flops del sistema y por lo tanto sincronizar su funcionamiento. Esta señal se denomina reloj del sistema, y los cambios de estado se producen en sus transiciones de cero a uno o de uno a cero, dependiendo de la tecnología propia de los circuitos electrónicos utilizados (generalmente transiciones negativas en tecnología **TTL** y positiva en tecnología **CMOS**).

En realidad un circuito lógico de cualquier tipo puede considerarse formado por compuertas, elementos intrínsecamente asíncronos, puesto que proporcionan salidas instantáneas en cuanto están presentes las entradas adecuadas. La combinación de estas compuertas entre sí es lo que da a un circuito su característica funcional de combinacional o secuencial y dentro de estos últimos síncronos o asíncronos.

El uso de flip-flops en la macrocelda de salida del dispositivo PAL se puede clasificar en alguno de los siguientes tipos:

A) flip-flop D (disparado por flanco de subida)
B) flip-flop J-K
C) flip-flop R-S
D) flip-flop T o Latches

Las salidas con flip-flops tiene su mayor aplicación dentro del área de diseño de sistemas síncronos, mientras que las salidas con latches son requeridas para aplicaciones lógicas asíncronas.

Tradicionalmente, la velocidad de respuesta, la simplicidad de arquitectura y el uso de flip-flop D en la salida, son los criterios más importantes para la selección de los dispositivos **PAL** para la realización de aplicaciones especiales.

Salida combinacional

La estructura de las macroceldas de salida, determinan la flexibilidad para la realimentación; esta característica es uno de los requerimientos más importantes para tener salidas combinacionales, salidas con registros, configuración de pines para entrada/salida. Por otra parte, la realimentación puede utilizar líneas múltiples o simples, con el fin de incrementar la flexibilidad del dispositivo para aplicaciones más especializadas.

En muchos dispositivos, todos los términos de productos disponibles para cada salida son unidos en una compuerta **OR** para producir la función lógica de salida. Los dispositivos originales **PAL**, que son conocidos como "*Small PAL Family*": Familia **PAL** pequeña" están basados en esta simple arquitectura. Estos dispositivos son comúnmente empleados como decodificadores de direcciones, multiplexores y aplicaciones lógicas de

control aleatorio. En otros dispositivos, una o más líneas de productos son empleados para controlar otros atributos de las señales de salida correspondientes, como habilitar el estado de alta impedancia (TRI-STATE). Los dispositivos de esta familia tienen salidas TRI-STATE con retroalimentación (Figura 2.14 a) de un pin del dispositivo al arreglo lógico. Esto hace que el pin funcione de manera bidireccional para entrada/salida o para actuar como una entrada dedicada adicional.

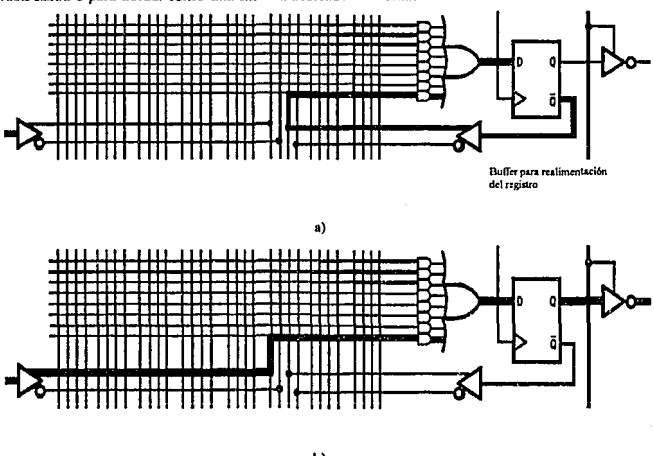


Figura 2.14 Registro lógico a) con retroalimentación b) sin retroalimentación

La función de transferencia fundamental de un dispositivo **PAL** es **INVERT-AND-OR** y quizá sea especificado directamente por una ecuación Booleana de suma de productos.

Arquitectura de pines entrada/salida

La programación de los pines para ser empleados como entrada/salida, es uno de los más importantes recursos que presentan los dispositivos **PAL**, con lo cual se pueden llevar a cabo implementaciones de las más complejas y variadas funciones lógicas.

Además se pueden determinar los pines que funcionarán como entradas dedicadas, pines para salida o control dinámico de pines para entrada/salida de acuerdo a los requerimientos de diseño.

Flexibilidad para controlar la habilitación de salidas y entrada/salida bidireccional

El diagrama lógico de una estructura bidireccional de un **PAL** se muestra en la figura 2.15

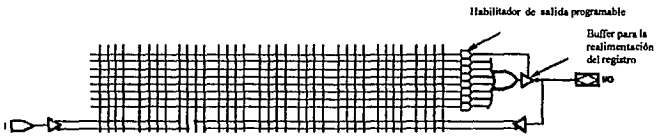


Figura 2.15 Diagrama lógico de una estructura bidireccional

La función más importante de los dispositivos **PAL** con estructura bidireccional, es la flexibilidad que proporciona en el control para habilitación de salidas, así la salida habilitada puede ser dedicada, es decir, controlada por un pin, programada o controlada por un término producto de un arreglo de compuertas **AND**.

Por otro lado, el buffer asociado con el pin de salida puede ser programado en alguno de los siguientes modos de acuerdo a los requerimientos:

- como salida dedicada
- como entrada dedicada
- control dinámico para entrada/salida

Cuando se opta por la programación como salida dedicada, el buffer a la salida se encuentra habilitado, por lo tanto la función lógica es realimentada hacia el arreglo **AND**. Con esta realimentación se permite la implementación de funciones lógicas más complejas haciendo uso de dos o más niveles de compuertas **AND-OR**.

De otra manera, cuando se programa como entrada dedicada, el término **AND-OR** asociado con este pin no es utilizado. Por lo tanto, el diseñador no se encuentra limitado a un número fijo de pines para ser empleados como entrada/salida, teniendo así un rango de programación de acuerdo a las necesidades de diseño.

Finalmente, cuando se programa para un control dinámico de entrada/salida, habilitar o deshabilitar por lógica combinacional una o más entradas, este pin puede ser utilizado como entrada con el fin de retener la capacidad lógica de las compuertas **AND-OR**. Esta peculiaridad se usa especialmente en aplicaciones de control, transferencia de datos, etc. Una entrada/salida serial es un ejemplo del concepto antes

mencionado; cuando se utiliza el pin para corrimientos hacia la izquierda es una entrada serial, y si existe corrimiento hacia la derecha funciona como salida serial.

Con el modo de programación control dinámico se proporciona la máxima utilización de los recursos que permite la arquitectura de un **PAL**. También es importante mencionar la utilización de la salida en activo bajo, con realimentación hacia el arreglo **AND**, la cual es útil para la implementación de niveles lógicos múltiples, las compuertas extras hacen de estas salidas excelentes para el control de generación de señales, codificación y

decodificación de las mismas, así podemos decir que se pueden utilizar los dispositivos **PAL** en las áreas de comunicaciones, sistemas de adquisición y procesamiento de señales donde se requiera velocidad de respuesta, fiabilidad y precisión, así como en el diseño de estaciones de trabajo.

Características de la Arquitectura PAL

Los dispositivos **PAL** poseen una gran cantidad de atributos que los hacen perfectos para la implementación de funciones lógicas de los cuales a continuación se mencionan sus características más distintivas de dichos dispositivos a manera de recapitulación (Figura 2.16):

Pines programables para entrada/salida
Flexibilidad para el control de habilitación de salidas y entradas/salidas bidireccionales
Estructura de salida dedicada
Polaridad programable
Flexibilidad de esquema de frecuencia
Características adicionales:
- Accesibilidad
- Control
- Observación

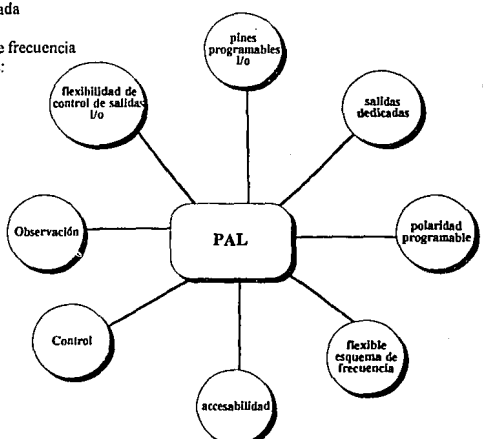


Figura 2.16 Características distintivas de los dispositivos PAL

Caracterísiticas generales de los PAL

Finalmente para hacer una descripción general de los dispositivos **PAL**, a manera de justificar su elección se comparan otros dispositivos lógicos programables **PLD**, de tal manera que a continuación se mencionan en forma concisa y general las características de los dispositivos **PAL**.

Tiempo requerido para el proceso de diseño

Los **PLD** son la vía más óptima para la reducción del tiempo involucrado dentro de la fase de diseño. La flexibilidad de la arquitectura que presentan los **PAL** permite realizar la interfase entre diferentes bloques lógicos, por lo tanto, cada bloque puede ser diseñado con un mínimo de interacciones para de esta manera simplificar la elaboración de: prototipos de algún sistema, tarjetas para computadoras o simplemente ofrecer más funciones por tarjeta, lo cual se traduce a una mayor integración física.

Otra ventaja que ofrecen los **PLD** es la reducción gradual de los costos así como el tiempo asociado que implica los cambios de errores lógicos en el diseño de sistemas que se encuentran en procesos finales, así como modificaciones a productos ya existentes; dichas modificaciones se pueden llevar a cabo reprogramando los dispositivos, creando de esta manera un nuevo mapa de fusibles para el arreglo de compuertas **AND** alterando así la configuración anterior.

Velocidad de ejecución

La velocidad en el tiempo de ejecución que requieren los sistemas, se puede incrementar de manera notable a través del uso de lógica programable, dado que el diseñador tiene la libertad de optimizar la arquitectura del diseño empleando un **PLD** para la implementación de una aplicación más precisa. Así, el diseñador puede elaborar su sistema de la manera más eficiente e incrementar la velocidad de ejecución de dicho desarrollo.

Reducción en los tiempos de retardo de las señales

Cuando una función lógica es implementada utilizando elementos estándar **TTL SSI/MSI** (lógica combinacional) el retardo total en la respuesta del sistema incluye el tiempo de respuesta requerido por algunos buffer contenidos en los circuitos integrados, así como el tiempo de respuesta requerido por las compuertas utilizadas; no ocurre lo mismo cuando se utiliza un **PLD**, donde el tiempo de propagación de la señal se reduce considerablemente debido a su arquitectura.

Reducción de costos

La disponibilidad de los dispositivos **PAL** puede proporcionar el desarrollo de una lógica completa equivalente a 300 compuertas, con lo cual se puede reemplazar más del 90 % de los elementos estándar **TTL**. La implementación de un diseño utilizando lógica programable puede significar, de manera considerable, la reducción de espacio en diseño de circuitos impresos o el número de tarjetas necesarias para implementar una función específica.

Reducción de componentes

Comparando contra los elementos estándar **TTL SSI/MSI**, la lógica programable nos proporciona una reducción en el número de circuitos integrados necesarios para llevar a cabo la implementación de una función lógica, logrando de esta manera una mayor integración física del diseño.

Reducción de interconexiones

La parte con menor confiabilidad dentro de un sistema digital son las conexiones que existen entre los diferentes circuitos integrados, la reducción del número de circuitos integrados requeridos para un diseño, implica un menor número de conexiones y por consiguiente, se incrementa su confiabilidad.

Seguridad del diseño

Para poder utilizar los dispositivos **PAL** se requiere de una programación especial, de modo que se evita la duplicación de dichos dispositivos, lo cual es ideal para cualquier aplicación especializada donde la seguridad del diseño es lo esencial.

Bloques LSI

Además de los dispositivos **PROM** y **PLA**, existen actualmente algunos otros tipos de **PLD** disponibles para el diseño en el área de electrónica. Los dispositivos **PAL** son un caso especial de los **PLA** y poseen la misma estructura básica de compuertas **AND-OR**, pero en los **PAL** solamente el arreglo **AND** es programable y cada compuerta **OR** es dedicada a un número fijo de compuertas **AND**.

Como consecuencia, las funciones lógicas se pueden realizar en un dispositivo **PAL** utilizando términos que pueden ser comunes a más de una función. Además, los **PAL** son menos costosos en comparación con otros dispositivos programables y de fácil programación.

Existen otras variantes de dispositivos que se construyen empleando la estructura básica **AND-OR**, dentro de los cuales se puede mencionar a los dispositivos lógicos reprogramables (**EPLD**) desarrollados por la corporación Altera. Dichos dispositivos consisten de una serie de macroceldas, conteniendo típicamente un arreglo lógico programable, así como flip-flop a la salida.

Una de las ventajas de los arreglos programables, es la concerniente a la "*rigidez*" de la arquitectura, impuesta por las interconexiones fijas que se encuentran involucradas. Esta rigidez es resultado de un pequeño porcentaje (normalmente menor al 20 %) de compuertas que son utilizadas en una aplicación típica.

Empero, se elimina esta ineficiencia, ya que dada la estructura del arreglo de compuertas nos proporcionan un vasto arreglo de dos entradas a compuertas **AND**. Las funciones más complejas pueden ser implementadas en un arreglo utilizando interconexiones apropiadas (usando la facilidad de realimentar el arreglo de compuertas **AND**, mediante el uso de flip-flops, etc.). Por otra parte, la utilización de compuertas dentro del arreglo puede ser mayor del 90 % en algunas aplicaciones especializadas.

Hasta hace poco los **PAL** no eran muy utilizados dada su baja velocidad y alto costo. Con los recientes avances tecnológicos permiten la construcción de dispositivos **PAL** con tiempos de respuesta de pocos nanosegundos y un precio altamente competitivo. Existen **PAL** que pueden ser programados por el usuario después de haber sido construído el circuito integrado.

Un **PAL** tiene n entradas I_0 a I_{n-1} de las cuales se dispone internamente tanto en su forma directa como en la complementada. Las compuertas **AND**, 1 a k poseen cada una *n* entradas y son capaces por lo tanto de realizar cada una de ellas un producto de todas o algunas de las entradas, tanto en su forma directa como complementada. Se pueden formar por lo tanto, k productos distintos. El producto concreto que hará cada compuerta **AND** depende de las conexiones indicadas por un punto localizados en el plano **AND**. Asi, la compuerta uno realiza el producto $k_1 = I_0/I_1 ...$ y la compuerta tres $k_2 = I_2 ...$

Las m salidas 0_1 a 0_m del dispositivo **PAL** proceden de las compuertas **OR** de k entradas, cada una puede realizar la suma de todos o algunos de los productos k_1 a k_k según las conexiones que se realicen en el plano **OR**.

El número de productos k que se pueden realizar con un **PAL** es un número pequeño y muy inferior a las 2^n combinaciones posibles de las entradas. Un dispositivo **PAL** típico tiene n=14 entradas, m = 8 salidas y sólo puede realizar k = 96 productos en vez de los 16384 posibles con 14 variables.

Se deduce que los dispositivos **PAL** están especificamente diseñados para realizar funciones lógicas que tengan un pequeño número de términos de un gran número de variables y que hay que utilizar la función ya simplificada. Para aplicaciones de este tipo el empleo de **PAL** conduce a una solución más económica y confiable que la utilización de **PROM**. Para algunas aplicaciones el número de productos que se puede realizar con las variables de entrada es demasiado reducido. Esto se puede solucionar conectando en paralelo las entradas de varios dispositivos **PAL** y realizar la función **OR** con las salidas equivalentes de los distintos **PAL**.

Cuando el número de salidas sea insuficiente, se pueden utilizar varios **PAL** con sus entradas equivalentes conectadas en paralelo. De esta forma se obtiene un **PAL** que tiene el mismo número de entradas y términos que cualquiera de los **PAL** siendo sin embargo el número de salidas igual al producto de número de **PAL** utilizados por el número de salidas.

La primera generación de dispositivos **PAL** es conocida como la familia de 20 pines que incluye dispositivos como el 16L8 o 16R8 que tiene 64 términos de productos y 8 productos de control, con un total de 2 K fusibles. ***La segunda generación*** de dispositivos **PAL** es conocida como la familia 22V10 que contiene un arreglo de 5808 fusibles con un total de 132 términos de producto. De esos 132, 120 son productos lógicos y 12 son productos de control. Cada término de producto consiste de un número fijo de entradas que pueden ser dedicadas o retroalimentadas, o pueden consistir de pines bidireccionales de entrada y salida.

Familia de 20 pines

Contiene 4 arquitecturas **PAL** de las más populares en la industria. Procesadas con tecnología **TTL Schottky** a base de titanio-tugsteno para las ligaduras de fusibles. Se encuentran las series A, serie A-2,

serie B, serie B-2, así como dispositivos isoplanares de tecnología **FAST-Z TTL** con fusibles verticales pertenecientes a la serie D y serie -7. La familia **PAL** de 20 pines esta provista de alta velocidad celdas programables con lógica **SSI/MSI** convencional. La lógica más conveniente son la suma de productos de funciones Booleanas que pueden ser utilizadas rápidamente para la programación del dispositivo. La figura 2.17 nos muestra la familia de 20 pines de dispositivos PAL.

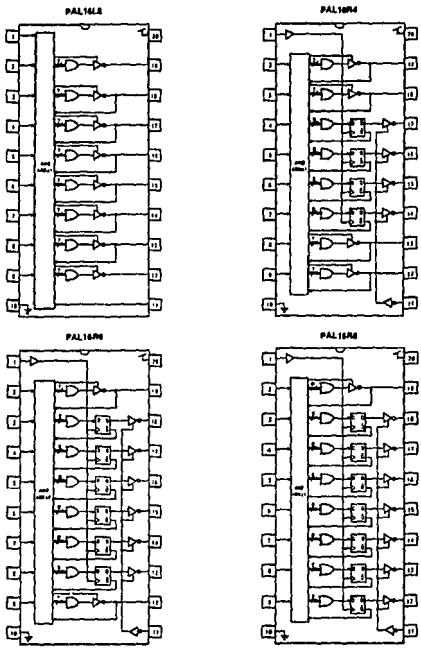


Figura 2.17 Familia de 20 pines de dispositivos PAL

De la figura se deduce que las caracteristicas de los DIP son diferentes en cuanto a sus atributos para los propósitos que sean destinados. Es decir, alguno de estos dispositivos será mejor para una aplicacioón que otro de los tres.

Para conocer mejor la familia de 20 pines y las versiones de los circuitos integrados existentes en el mercado se enlistan a continuación algunas de las caracteristicas que es importante considerar (Figura 2.18).

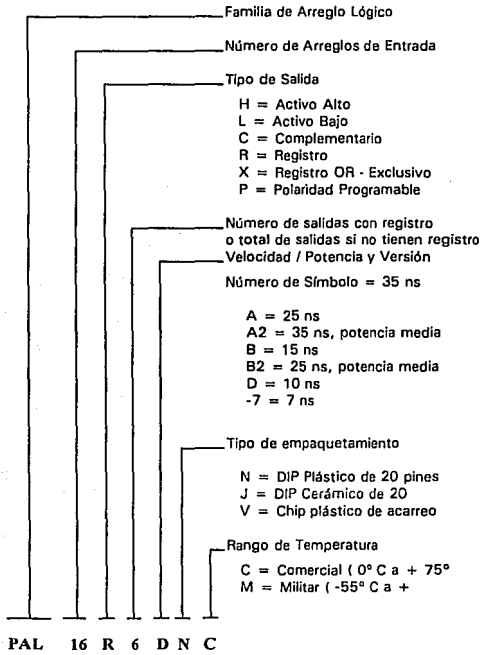


Figura 2.18 Orden de información para las versiones de un circuito integrado

PANORAMA DE LA ARQUITECTURA

Salidas con registros

Además de las funciones entrada/salida TRI-STATE, los dispositivos **PAL** también proveen registros sincronizables (*edge-triggered*) en algunas salidas de funciones lógicas. La salida de cada registro es también retroalimentada dentro del arreglo lógico para facilitar el diseño de circuitos secuenciales como máquinas de estado, contadores, corrimientos, etc.

Compuertas OR-Exclusivas

Algunas arquitecturas de **PAL** agregan una compuerta **XOR**, combinado dos pequeñas funciones **AND-OR** en cada salida. Las compuertas **XOR** combinadas con registros a la salida facilitan el diseño de contadores, máquinas de estado y pequeñas funciones aritméticas.

Polaridad programable en la salida

Algunas de las más recientes familias de **PLD** tienen polaridad programable de salida. Algunas celdas de programación adicional están incluidas en la ruta de las salidas lógicas para opcionalmente invertir las señales de salida individualmente. Esto permite una mejor interfase del sistema, al eliminar el uso de inversores externos. La inversión de las salidas junto con las entradas complementarias del arreglo también permiten ecuaciones lógicas de productos de sumas para ser directamente implementados empleando el Teorema de De Morgan. Esto podria dar resultado a una minimización lógica compacta para varias aplicaciones.

La polaridad programable la tienen todos los dispositivos de las siguientes familias: **TTL PAL, Registro Asíncrono (RA), PAL, GAL y ECL PAL.**

Características de Productos

Las características mencionadas anteriormente aparecen en la arquitectura de los PLD más nuevos, y están clasificadas en 2 categorías:

1) empleo diversificado de términos lógicos para funciones de control en el circuito integrado
2) más celdas de programación fuera del arreglo utilizado para invocar rutas lógicas periféricas opcionales.

Los dispositivos *Medium*-**PAL** emplean un término de producto por cada salida para controlar la habilitación del TRI-STATE, los dispositivos de Registro Asíncrono, PAL16R8A, utilizan tres términos de productos adicionales "especializados" para controlar el reloj, reset y preset de cada salida del registro (Figura 2.19). Este tipo de **PLD** es empleado principalmente en aplicaciones lógicas de control aleatorio, los cuales normalmente son implementados con compuertas **SSI/MSI** y flip-flops, como el **CI** 74LS74.

Los dispositivos **GAL** utilizan celdas de programación adicional para redireccionar sus rutas de salida lógica para emular una amplia variedad de arquitecturas **TTL PAL**, además de otras configuraciones originales. Estas "celdas de arquitectura" seleccionan entre salidas registradas o combinacionales,

señales de control TRI-STATE, rutas de retroalimentación de entrada/salida y polaridad de salida. El término "Macrocelda Lógica de Salida" (**OLMC**) es comúnmente utilizada para describir periféricos lógicos sofisticados y que pueden ser configurados por el usuario por medio de switches para programar la arquitectura. La figura 2.20 nos muestra el esquema lógico de un **GAL OLMC**.

Afortunadamente, las rutas lógicas y los switches de arquitectura son configurados automáticamente a través del software de desarrollo de acuerdo a las ecuaciones lógicas del diseñador.

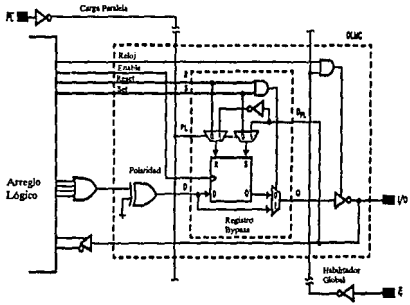


Figura 2.19 Macrocelda de Registro Asíncrono (RA)

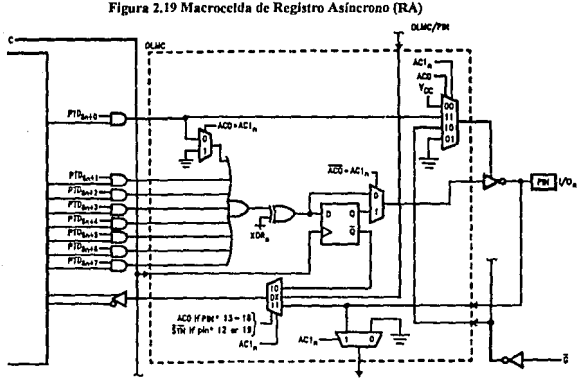


Figura 2.20 Estructura de la Macrocelda lógica de salida (OLMC)

II.4 ESTANDARES DE PROGRAMACION DE PLD

Cuando los **PLD** fueron introducidos por primera vez, el diseñador lógico tenía que desarrollar un mapa de conexiones (Figura 2.21) que mostraba los fusibles que debían quemarse y lo enviaba al fabricante de la **PROM, PAL** o **FPLA**. Actualmente, la disponibilidad de equipo de programación de bajo costo ha facilitado que los usuarios programen sus propios **PLD**. En el mercado existen programadores universales que pueden programar las **PROM, PAL** y **FPLA** más comunes. El dispositivo que va a programarse se coloca en la base del programador; éste programa y prueba el dispositivo de acuerdo con los datos que fueron proporcionados por el usuario.

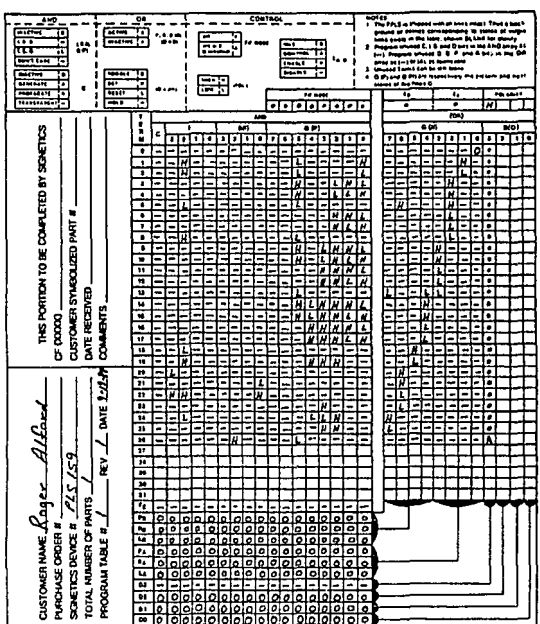


Figura 2.21 Ejemplo de una carta de especificación lógica del diseñador de PLD

La programación junto con los datos de prueba se desarrollan utilizando para ello software de programación disponible en el mercado y que se pueda ejecutar en computadoras personales. Al utilizar esta programación, el usuario introduce en la computadora los datos que describen las funciones lógicas que desea que queden programadas en el **PLD** además de la información sobre cómo probar el dispositivo. Entonces el programa genera el mapa de fusibles y los datos de prueba en una forma que pueda enviarse sobre un cable hacia la memoria del programador de **PLD**. Una vez que el programador tiene los datos, procede a programar y probar el dispositivo. Cuando termina de hacerlo, el programador indica si el dispositivo ha pasado o no el procedimiento de prueba. Si lo pasa, el dispositivo se quita de la base del programador y se coloca en el prototipo del circuito para realizar más pruebas con él.

La programación con el estándar JEDEC

La programación soportada por los **PLD** fué improvisada mediante la introducción del estándar **JEDEC** 3 (*Joint Electronic Device Engineering Council* : Dispositivos electrónicos de juntura para ingeniería). Este estándar fué provisto por los usuarios de **PLD** para el formato de intercambio de datos en dispositivos programadores. El formato estándar fué propuesto en 1980 y definido en un archivo de formato para computadora que hacía posible la transferencia del diseño y prueba a través de los dispositivos y programadores de los diferentes fabricantes. La idea de un formato común no era nueva; Intel y Motorola habían creado un estándar para los dispositivos **PROM**, sin embargo, el estándar **JEDEC** fue más importante porque satisfacía necesidades específicas de direccionamiento de los **PLD**, incluyendo los requerimientos de prueba de datos.

Con el formato **JEDEC** es posible realizar un diseño con un lenguaje y procesar automáticamente la información de un **PLD** de manera que sea entendible para un programador universal compatible con el código **JEDEC**.

En las primeras líneas del archivo los campos de comentarios son terminados por un asterisco (*). La siguiente línea es compuesta de cuatro campos opcionales que especifican el número de entradas en el archivo, el total de número de fusibles en el dispositivo a programar, el número de vectores probados en el archivo y el fusible con un valor de default sin especificar fusibles.

El fusible de datos contiene en las líneas unos o ceros que son prefijos del caracter L y del fusible de offset. El valor del fusible cuando es uno indica que el fusible específico esta desconectado (*"BLOWN"*) mientras que un valor cero indica que el fusible esta permanentemente intacto o conectado.

La programación de fusibles contiene varias pruebas de vectores. Cada vector de prueba es prefijado por el caracter "V" y el número del vector. En este caso hay un total de veinte caracteres en cada prueba del vector. Cada caracter corresponde a un pin del dispositivo con el caracter cargado hacia la izquierda marcamos el pin número uno. Los siguientes caracteres son válidos en el estándar **JEDEC** para vectores de prueba.

0 controlador de entrada baja
1 controlador de entrada alta
2-9 controlador de entrada de supervoltaje
B registro de precargado "buried"
C controlador de entrada baja, alta, reloj de pulso bajo

F entrada o salida flotante
H salida de prueba para estado alto
K controlador de entrada alta, baja, reloj invertido alto
L salida de prueba para estado bajo
N pin no utilizado (incluye potencia y tierra)
P registro de precargado
X salida sin prueba
Z salida de prueba para alta impedancia

El campo final del archivo es conocido como fusible de **Checksum**. Este fusible es utilizado para detectar errores de transmisión cuando el archivo es transferido através de una computadora hacia el dispositivo periférico programador. El **Checksum** (aparece en el archivo con un caracter "**C**" seguido por un valor hexadecimal) es calculado por los valores de los fusibles (incluyendo los archivos no especificados directamente en el archivo) y una suma de 16-bit (módulo 65535) de palabras de 8-bit consecutivos en la programación de datos (Figura 2.22).

```
PAL16R4;
CLK C1 C2 C3 SW1 SW2 NC NC NC GND
/OAB S0 S1 NC /C /B /A S2 NC VCC

/a := c*b*a*sw1 + c*b*/a*/c1 + /c*b*a*c3 + c*/b*a*c2*sw2
/b := c*/b*a*/c2 + c*b*/a*c1 + c*/b*a*c2*sw2
/c := /c*b*a + c*/b*/a

/s0 = c*b*a
/s1 = c*b*/a + /c*b*a
/s2 = c*/b*a + /c*b*a
PLAN v3.14  09-11-1992  13:26
Source filename: MOTOR.BEQ   Device: PAL16R4 *
QP20* QF2048* F0*
L0256
11111111111111111111111111111111
11111111110111101101111111111111
11111111110111011110111111111111*
L0512
11111111110101011101111111111111
10111111111011011011111111111111
11111111101011011101111111111111
11110111110111100101111111111111*
L0768
11111011110111101101111111111111
01111111111011011011111111111111
11110111110111100101111111111111*
L1024
11111111110111011101111111111111
11111111110111011011111111111111*
L1536
11111111111111111111111111111111
11111111110101011101111111111111
11111111110111011011111111111111*
L1792
11111111111111111111111111111111
11111111110111011101111111111111*
C3E16* 0000
```

Figura 2.22 Ejemplo de archivo estándar JEDEC

El estándar EDIF

En 1983, las necesidades del intercambio de datos en un estándar en el área de **CAE** (Electrónica Asistida por Computadora) propicia una reunión de 6 fabricantes de herramientas **CAE** - Daisy System, Motorola, National Semiconductor, Mentor Graphics, Tektronix y Texas Instruments - los cuales forman un estándar para intercambio de datos en diseño. Esto fue el nacimiento del formato **EDIF** (*Electronic Disegn Interchange Format* : Formato de Intercambio de información de Diseño Electrónico) que fué liberado en 1987 y adaptado para implementación por algunas compañías que promueve la Asociación de Industrias Electronicas (**EIA**) para usuarios y vendedores de **CAE**.

El formato **EDIF** es soportado en algunas herramientas de diseño para **PLD**. El **EDIF** fue desarrollado en 1983 y 1984 en un esfuerzo conjunto de un consorcio de estaciones de trabajo en **CAE** y vendedores de semiconductores. La versión 2.00 del **EDIF** fue adoptada como un estándar por **ANSI** (Instituto Nacional de
Estándares Americanos) y la Asociación de Industrias Electrónicas.

El intento del estándar **EDIF** es proveer un formato común de lista de conexiones, datos esquemáticos y otros datos de diseño eléctrico para detallar información.

II.5 ESTANDARES DE PROGRAMACION PARA AMBIENTES GRAFICOS

Estándares en computación: Quién, Qué, Cuándo y !Cuidado!

Quién fija los estándares ?

Tres grupos son los involucrados en el proceso de estandarización, grupos intitucionales establecidos por comités definen, implementan y dirigen los estándares, las tres instituciones son: Instituto Nacional de Estándares Americanos (**ANSI**), Instituto de Ingenieros Eléctricos y Electrónicos (**IEEE**) y la Organización de Estándares Internacionales (**ISO**) son los responsables de dar a conocer los estándares en computación. En fechas recientes han surgido otras organizaciones persiguiendo el mismo fin, por ejemplo, Fundación de Sistemas de Software Abierto (**OSF**) y Unix Internacional (**UI**).

Qué clase de tecnologías son estándarizadas ?

Cada aspecto de la tecnología en computación es sujeto a una estandarización, como por ejemplo, protocolos, interfases, redes y medidas para el microprocesador que pueden ser objetos de estudio de estas organizaciones; en general las investigaciones que permitan una medida de comparación a usuarios terminales son analizadas por las instituciones de estandarización.

Cuándo es aplicable un estándar ?

Un producto es estandarizado cuando el usuario tiene requerimientos, aún cuando son mínimos, de igual manera son necesarios para un buen funcionamiento de interconectividad e intercambiabilidad. Una aplicación reconocida como un estándar tiene muchos beneficios debido a que puede ser portado a diferentes plataformas que soporten el mismo estándar.

Cuidado! los estándares tienen limitaciones

No todos los estándares proporcionan seguridad total, debido a que son versiones incompletas que presentan ambiguedades; estas ambiguedades surgen al momento de la aplicación y es por esto que se hace necesario una revisión cuando los cambios se han acumulado, razón por la cual las versiones se van modificando en números progresivos.

Estándar gráfico GKS

GKS es un sistema conocido como Sistema Gráfico de Kérnel (*Graphics Kernel System*), que en escencia son subrutinas o librerías utilizadas para desarrollos y creación de gráficos en 2 dimensiones. Esta compuesto por elementos conocidos como **primitivas gráficas**, entre los que encontramos: Polilínea, polimarco, círculo, rellenado de áreas, texto escalable y transformaciones geométricas, cada subrutina puede encontrarse disponibles para lenguajes de programación como C, Fortran y Pascal (**ANSI**).

GKS fué adoptado como estándar en 1985, siendo la primera librería gráfica definida para lenguajes de propósito general (Figura 2.23).

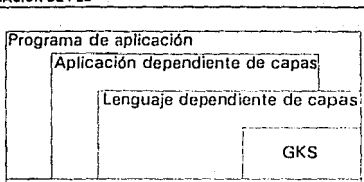


Figura 2.23 Modelo de capas de la estructura del GKS

GKS fué adoptado como un sistema de 2D de propósito general por **ISO** en 1984 y por **ANSI** un año después. El estándar enfatiza las gráficas interactivas para funciones gráficas del tipo **RASTER** (Figura 2.24).

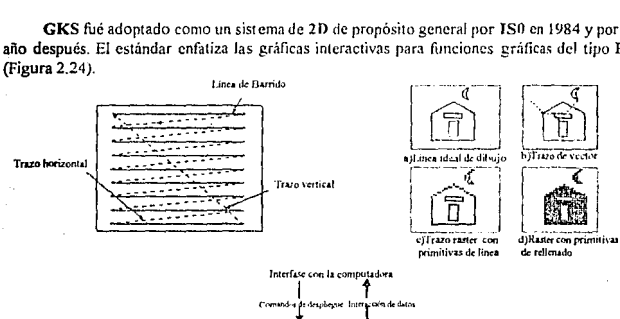


Figura 2.24 Arquitectura para despliegue tipo RASTER

Estándar gráfico PHIGS

El estándar gráfico **PHIGS** (*Programmer's Hierarchical Interactive Graphics Standard*: Estándar de programación jerárquica de gráficas interactivas) constituye una interfase de **ANSI** para el desarrollo de aplicaciones gráficas interactivas en 2D y 3D. El conjunto de funciones de **PHIGS**, proporciona facilidades para la creación, despliegue y manipulación de elementos gráficos, así como la programación,

configuración y manejo de los dispositivos periféricos asociados a una estación de trabajo y PC, facilitando el desarrollo de aplicaciones en el campo del Diseño y Manufactura asistidos por computadora (**CAD/CAM**).

La utilización de estándares gráficos permite una gran independencia entre los programas de aplicación y los dispositivos físicos, brindando portabilidad del software hacia los diferentes tipos de hardware, conservando sus características y evitando la necesidad de efectuar cambios drásticos.

PHIGS incorpora las facilidades de los estándares como el **GKS** para el manejo de entidades gráficas, además de permitir el manejo independiente de ellas por medio de unidades o segmentos de programa llamadas estructuras, con lo cual se tiene acceso a nuevas capacidades. Un estándar gráfico permite tener una interfase entre la aplicación desarrollada por el usuario y los dispositivos físicos de graficación con que se cuenta (teclado, monitor, ratón, digitalizador, etc.), el estándar se encarga de habilitar dichos dispositivos de control.

En un principio se definieron 6 tipos de primitivas gráficas para **2D** y **3D** como línea, polilínea, movimiento, marco, polimarcos, texto. Posteriormente se definieron las funciones para los dispositivos de entrada como **LOCATOR, STROKE, VALUATOR, CHOICE, STRING** y **PICK** (Figura 2.25).

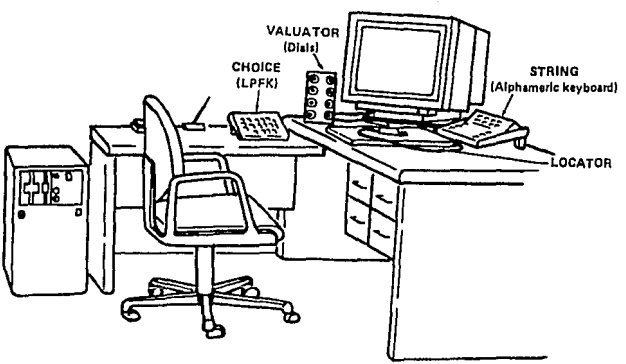


Figura 2.25 Dispositivos de entrada para el ambiente en PHIGS

PHIGS es una evolución lógica de **GKS** donde son incluidas funciones gráficas en **3D** más poderosas, manipulando estructuras de datos complejos que representan objetos. El desarrollo en **PHIGS** se basa en listas y tablas de estructuras que puedan ser editadas y enlazadas interactivamente por el usuario,

manteniendo una transformación geométrica aplicada en todas las primitivas gráficas subsecuentes. También puede hacer el despliegue de las listas del sistema que incluyen el **CSS** (almacenamiento central de estructuras) como parte integral de un estándar.

INTERFASES TIPO CGI

El propósito del estándar **CGI** (*Computer Graphic Interface*: Interfase gráfica para computadora) originalmente conocido como **VDI** (*Virtual Device Interfase* : Interfase de Dispositivo Virtual) consite en especificar precisamente que códigos son asentados para una terminal con hardware gráfico y definir cada función del dispositivo.

El propósito de **CGM** (*Computer Graphic Metafile* : Archivo gráfico de Computadora) estándar conocido originalmente como **VDM** (*Virtual Device Metafile* : Archivo de dispositivo virutal) es definir el camino en que las imágenes gráficas serán almacenadas en un dispositivo independiente para ser transferido a otros sistemas.

IGES (Sistema Inicial para Intercambio de Gráficos) como **CGM** se enfocan a la codificación y almacenamiento de información gráfica de tal forma que puedan ser transferidos de una instalación de una computadora a otra para una reconstrucción posterior. **IGES** fué diseñado inicialmente para sistemas **CAD/CAM**, superficies, primitivas gráficas y superficies tipo splines, así como para los diferentes tipos de elementos finitos. Es decir, fué diseñado para captura no únicamente gráfica, sino dibujos de ingeniería, información mecánica, partes y ensambles.

INAPLPS (Protocolo Norte Americano de Sintáxis para Niveles de Presentación) establece un estándar para la transición de texto y gráficos de un gran número de terminales de video, texto y teletexto, que esperan ser instaladas a escuelas, casas y oficinas alrededor del mundo.

PHIGS PLUS (*Programmer's Hierachical Interactive Graphics System Plus Lumière and Surface*) ofrece mayores ventajas al programador para explotar los gráficos tipo raster; es una forma extendida de **PHIGS** que incluye capacidades de rendimiento mayores, fuentes de luz, modelos de reflexión y soporte de primitivas avanzadas, como por ejemplo dibujo a detalle (**Rendering**).

El propósito de **PHIGS PLUS** es hacer una extensión del **PHIGS** con una mayor calidad en gráficos de 3D y geometrías más sofisticadas. **PHIGS PLUS** retiene todas las ventajas de **PHIGS** con rendimientos mayores debido a la adición de nuevos componentes para **3D** como: luces, sombras, triángulos, etc.
Las primitivas de **PHIGS** vs **PHIGS PLUS**

- Polilínea
- Polígonos
- Polimarca
- GDP (circulo, elipse)
- Arreglo de celdas
- Texto (Geométrico y anotación)

Básicamente **PHIGS PLUS** presenta los mismos modelos de programación que PHIGS. El almacenamiento de elementos en estructuras y la manipulación jerárquica de datos es llamada CSS. Estas estructuras pueden ser agrupadas en otras que están organizadas cíclicamente para gráficas. La salida gráfica es producida en una "workstation" mediante los CSS transversales, interpretando estructuras.

La transformación en pipeline es lo mismo para **PHIGS**. Los efectos son controlados por primitivas de atributos, orientación y posición en conjunción con fuentes de luz que pueden estar activadas.

Aparte de los CSS existen otras estructuras de datos no menos importantes: la descripción de tablas que proveen información acerca de las capacidades de **PHIGS** así como los valores asignados por default, los tamaños de implementación y los recursos del sistema.

- Las listas transversales de estados y los estados actuales de los procesos.
- La tabla de descripción de estaciones de trabajo que proporcionan información acerca de las capacidades de cada tipo de estación.
- Las listas de estado de la workstation que describe los estados actuales dependiendo de los atributos y otros datos como tabla de fuentes de luces que puede variar en el tiempo de corrida.

PHIGS PLUS hace aplicaciones específicas de color directo así como indexados. Manejando diferentes modelos de color como: RGB (Rojo, Verde, Azul), CMY (Cyan, Magenta, Amarillo) HSV (Tinte, Saturación , Color) y YIQ (sistema de dosificación de intensidad de luz). Cada workstation mantiene una tabla referente a las fuentes de luces que afecta la apariencia de un área de definición influenciada por un cálculo de reflectancia y presentación en pipeline. Todos los tipos de color tienen fuentes de ambiente direccionales y posicionales o bien a cambio tienen fuentes tipo spot con posición, atenuación, ángulos y primitivas con área de influencia (conos).

Después de que las primitivas han sido generadas se tienen algunos patrones y decoraciones que pueden ser aplicadas a los diseños, mediante colores asociados con unos colores intrínsecos. Los colores intrínsecos definen primitivas que son modificadas durante la aplicación de luces y sombras, la interacción a través de las superficies primitivas y la iluminación determinan las características de reflexión; tales fuentes pueden ser aplicadas por el frente, por detrás o por alguna de las caras de la superficie que pueden tener diferentes características de reflexión y por lo tanto diferentes métodos de sombreado.

PHIGS PLUS introduce una versión aumentada para las polilíneas de **PHIGS**, sumando habilidades para asociar valores de color a los vértices, cada polilínea es coloreada independiente de la otras, aunque pueden aplicarse métodos de sombreado, también es introducido las primitivas de curvas no uniformes conocidas como B-spline, delimitadas mediante un parámetro que especifica los límites. La aproximación de estas curvas es especificada en espacios paramétricos, coordenadas del mundo, coordenadas polares y coordenadas cartesianas.

Las principales aplicaciones de **PHIGS PLUS** están orientadas hacia:
- Diseño Mecánico por Computadora **MCAD**
- Defensa del gobierno, militar y mapeo
- Investigaciones científicas, visualización, modelado molecular
- Simulación en arquitectura

PEX (PHIGS PLUS EXTENSION TO X-Window)

Es una extensión del sitema **X-Window** para soportar eficientemente **PHIGS**. **PEX** almacena cada ventana en una pantalla de despliegue como un dispositivo independiente para gráficas tridimencionales, se desenvuelve como una red en el ambiente de **X-Window** así como ramificación para las decisiones de diseño en el ambiente **X** y **PHIGS**

Tiene aplicaciones en :

gráficas 3-D ditribuidas
Protocolos de redes, cliente/servidor
Interoperable
Consorcio MIT

Z-PHIGS

El sistema **Z-PHIGS** fué desarrollado en 1993 de acuerdo con el estándar **ISO 9592, ANSI** y **PHIGS** para **3D**. Cuenta con más de 300 funciones diferentes para construcción y despliegue de aplicaciones en tercera dimensión, con todas las ventajas de **PHIGS**. Las librerías gráficas de **Z-PHIGS** están disponibles en Pascal, C y C++.

Primitivas geométricas, atributos de dibujo, transformaciones en **3D**, superficies ocultas y vistas en ventana son algunas de las propiedades potentes de este sistema para gráficos en PC. **Z-PHIGS** tiene un mejor desempeño bajo **MS-WINDOWS** versión 3.1 y un procesador 386 o superior. Permite diseños en el área de **CAD**, Diseño gráfico, visualización, animación y aplicaciones tipo multimedia. No requiere **Windows SDK** para la elaboración de diseños sofisticados. Esta provisto de funciones de dispositivo independiente, esto es, sus aplicaciones pueden tener muchas aplicaciones independientes del dispositivo

OSF/MOTIF

La Fundación de Sistemas de Software Abierto **(OSF)** fue fundada para desarrollar aplicaciones para los ambientes que pueden encerrar portabilidad, escalabilidad e interconectabilidad para aplicaciones de software. La **OSF** fue fundada por siete compañías: Apollo, DEC, Hewlett Packard, Groupe Bull, Siemens AG, Nixdorf e IBM.

La Fundación de sistemas de software abierto (**OSF**: *Open Software Foundation*) desarrolló una interfase de usuario que fuese portable a las demás plataformas de computadora conocida como **MOTIF**. Esta interfase opera con **UNIX** y el sistema **X-Window** versión 11 release 5.

Este software tiene un estilo de programación basado en definición de clases orientado a objetos, mediante una jerarquía. Es un software enfocado a desarrollo de aplicaciones, interfases y administradores de sistemas de ventanas.

La finalidad es hacer un software portable a las plataformas que cuentan con ambientes de ventanas y convenciones tipográficas estándar.

Sistema X-WINDOW

X-WINDOW es un sistema que soporta despliegues de gráficas tipo bit-map (mapa de bits). Para **X-Window** un despliegue (display) se define como una estación que cuenta con: un teclado, un dispositivo apuntador (mouse) y una o más pantallas (Figura 2.26).

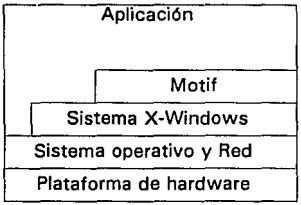
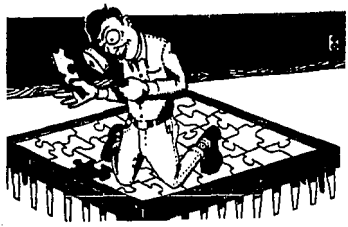


Figura 2.26 Modelo de interfase para desarrollo con X-Window/MOTIF

Las pantallas múltiples pueden trabajar juntas o separadas, siempre y cuando sean controladas por el mismo teclado o ratón. **X-Window** es un sistema de ventanas orientado hacia el uso de redes de computadoras. Permite que una aplicación pueda correr en un sistema, y despliegue su salida en un sistema diferente. Para esto, **X-Window** sigue el modelo cliente servidor. El servidor realiza las siguientes tareas: permite el acceso al despliegue a varios clientes, realiza dibujos en dos dimensiones y maneja las estructuras de datos que implementan a las ventanas. El cliente es un programa de aplicación que envía peticiones al servidor para ejecutar funciones del sistema de ventanas. El cliente y el servidor se comunican haciendo uso del protocolo **X**.

Un usuario de **X** interactúa directamente con el servidor para correr aplicaciones localmente o en sistemas remotos. La salida de dichas aplicaciones puede desplegarse de manera local o remota. El proceso distribuido ayuda a resolver el problema de cargas desbalanceadas entre varios sistemas. Cuando una máquina se encuentra sobrecargada, o no cuenta con los recursos suficientes para ejecutar alguna aplicación, los usuarios pueden utilizar otras máquinas para correr sus programas.

En el ambiente **X**, los clientes no controlan el aspecto de las ventanas en un despliegue. Esta apariencia es tarea de una aplicación llamada administrador de ventana (Window manager). Un administrador de ventanas es una aplicación como cualquier otra, excepto por el control que mantiene sobre las ventanas de despliegue. El administrador de ventanas permite al usuario mover las ventanas, cambiar el tamaño de alguna ventana, correr nuevas aplicaciones, controlar qué ventanas son visibles, etc.



CAPITULO III

ANALISIS DE HERRAMIENTAS PARA LA SIMULACION

Cualquier sistema digital es susceptible de tener circuitos combinacionales (*las salidas en cualquier momento dependen por completo de las entradas presentes en ese tiempo*), sin embargo, la mayoría de los sistemas que se encuentran en la práctica también incluyen elementos de memoria que son dispositivos capaces de almacenar dentro de ellos información binaria. La información binaria almacenada en los elementos de memoria en cualquier momento dado define el estado del circuito secuencial.

La secuencia de control y las tareas para el procesamiento de datos de un sistema digital se especifican mediante un algoritmo en el hardware denominado diagrama del algoritmo de máquina de estados (**ASM**).

En este capítulo presentamos un método de diseño digital del diagrama **ASM**, los bloques que lo componen y las relaciones de temporizado entre los bloques. A partir de ello analizaremos como las entradas externas en el circuito secuencial son funciones no sólo de las entradas sino también del estado presente de los elementos de memoria.

III.1 TECNICA DE LAS CARTAS ASM

La información binaria almacenada en un sistema digital puede clasificarse ya sea como datos o como control de la información, esta última proporciona señales de mando que supervisan las diversas operaciones de la sección de datos con objeto de llevar a cabo las tareas deseadas de procesamiento de datos.

El diseño lógico de un sistema digital puede dividirse en dos partes distintas; una parte se ocupa del diseño de los circuitos digitales que llevan a cabo las operaciones de procesamiento de datos, la otra se ocupa del diseño del circuito de control que supervisa las operaciones y sus secuencias.

Las relaciones entre el control lógico y el procesador de datos en un sistema digital se muestra en la figura 3.1 donde el subsistema de procesador de datos manipula los datos en los registros de acuerdo con los requisitos del sistema; a su vez el control lógico inicia los mandos en secuencia apropiada al procesador de datos. El control lógico usa las condiciones de estado del procesador de datos para servir como variables de decisión para determinar la secuencia de las señales de control.

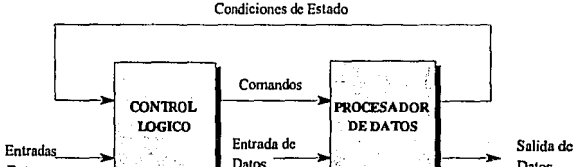


Figura 3.1 Interacción del control lógico y el procesador de datos

El control lógico que generan las señales para dar la secuencia de las operaciones en el procesador de datos es un circuito secuencial cuyos estados internos dictan los comandos de control para el sistema. En cualquier momento el estado del control secuencial inicia un conjunto predeterminado de mandos, dependiendo de las condiciones de estado y otras entradas externas, el circuito secuencial pasa al estado siguiente para iniciar otras operaciones. Los circuitos digitales que actúan como control lógico proporcionan una secuencia de tiempo de señales para iniciar las operaciones en el procesador de datos y determinar el siguiente estado del mismo subsistema de control.

La secuencia de control y las tareas de procesamiento de datos de un sistema digital se especifican mediante un algoritmo en el hardware que consta de un número finito de pasos de procedimiento que señalan como obtener una solución a un problema donde la parte del diseño digital más creativa es la formulación de algoritmos de hardware para lograr los objetivos requeridos.

Una forma conveniente de especificar la secuencia de los pasos del proceso y las trayectorias de decisión para un algoritmo es un diagrama de flujo. El diagrama de flujo para un algoritmo de hardware

traduce las palabras a un diagrama de información que enumera la secuencia de las operaciones junto con las condiciones necesarias para su ejecución. Un diagrama especial de flujo que ha sido desarrollado específicamente para definir algoritmos de hardware digitales se denomina diagrama de Máquina de Estado Algoritmico (ASM) que es la estructura básica de un sistema digital.

El diagrama ASM se asemeja a un diagrama convencional de flujo pero se interpreta en forma algo diferente. Un *diagrama convencional* de flujo describe la secuencia de los pasos de procedimiento y las trayectorias de decisión para un algoritmo sin ocuparse de sus relaciones en el tiempo. El *diagrama ASM* describe la secuencia de eventos lo mismo que las relaciones de temporizado entre los estados de un controlador secuencial y los eventos que ocurren cuando pasa de un estado al siguiente, es decir, está adaptado para especificar con precisión la secuencia de control y las operaciones de procesamiento de datos en un sistema digital, tomando en consideración las restricciones del hardware digital.

Máquina de Estados

Todos los módulos de un sistema lógico se pueden representar por un modelo general de Máquina de Estados, el cuál contiene los elementos necesarios para la descripción de un módulo en función de sus entradas, salidas y tiempos. En la figura 3.2 se muestran las tres diferentes fases que forman el modelo general de donde se puede identificar :

- Función Estado-Siguiente
- Función Estado-Presente
- Función de Salida

Figura 3.2 Modelo general de la Máquina de Estados

El Estado de Máquina, es la memoria del estado presente para poder determinar el estado siguiente. El término Estado de Máquina, proporciona la información necesaria y suficiente para determinar las salidas y el estado siguiente (*determinado por la Función Estado-Siguiente*) en función de las entradas en el estado presente (*determinado por una entrada periódica en el Registro-Estado*). Así la Máquina, es semejante a una memoria, que generalmente es elaborada con circuitos biestables llamados **Flip-Flops (FF's)**.

Un grupo de **FF's** forman un estado, al cual se le conoce como *Registro-Estado*. Se define un estado diferente para cada combinación de bits almacenados, de donde se pueden tener dos posibles estados para *n Registros-Estado* (**FF's**).

Los **FF's** del estado, son identificados con nombre de variables y se definen como declaración de estado. A cada variable se le puede dar un nombre arbitrario; *A, B, p, IH*, etc. Un grupo de variables forman lo que se conoce como *Código de Estado*, por ejemplo: el estado *DCBA* donde *A, B, C* y *D* son variables de estado, si además, *A=0, B=0, C=1, D=1* entonces el *Código de Estado* será *1100*.

Cada Estado de Máquina se encuentra relacionado con un estado siguiente, así al finalizar el estado presente tenemos un estado siguiente. La Función *Estado-Siguiente (g)* (ver Figura 3.2), depende del estado presente *(X)* y las entradas ó variables *(Q)*, si el estado presente es representado por *T* variable en el tiempo discreto y *k* es un número entero, entonces *X(kT)* representa el estado presente en tiempo discreto *kT*, por lo tanto, se puede definir a la función *Estado-Siguiente (g)* como:

$$X((k-1)) = g[X(kT), Q(kT)]$$

la notación para la operación *Estado-Siguiente* se simplifica notablemente utilizando el *Operador-Retardo* que es una flecha $\rightarrow$ ó $\leftarrow$ con lo cual se indica la transición hacia el estado siguiente, de esta manera la Función *Estado-Siguiente* se define como:

$$X\, g[X, Q]$$

Así, el valor de *X* es alterado al finalizar el estado presente, donde *X* y *Q* son valores en dicho estado. Las modificaciones en el estado presente se mantienen para poder determinar el uso del estado siguiente.

La *Función-Salida*, genera un conjunto de Salida(s) o Instrucción(es) *(I)* del estado y además la información de entrada requerida para cada estado, en semejanza con la Función *Estado-Siguiente*, consiste de un operador de transformación llamado *(f)*, definiéndose como:

$$I(kT) = f[X(kT), Q(kT)]$$

donde *k* es un número entero y *T* es una variable en el tiempo discreto. La notación para la *Función-Salida* puede ser simplificada utilizando el símbolo ' = ' como operador inmediato que indica la operación en el estado presente, de este modo se presenta la expresión simplificada:

$$I = f[X, Q]$$

El Algoritmo de Máquina de Estado (**Carta ASM**), es de gran utilidad dentro del área de diseño digital porque en ella se muestra simultáneamente un algoritmo y los Estados de Máquina, dado que la carta **ASM** es una descripción esquemática de la *Función-Salida* y la Función *Estado-Siguiente* de una máquina general (Figura 3.2), es decir, se separa la fase conceptual del diseño para llevar a cabo la implementación de un circuito electrónico.

Se le proporciona al diseñador la libertad para expresar los más complejos algoritmos para el desarrollo de un circuito. Cuando el algoritmo de la carta **ASM** satisface los requerimientos de diseño, se tiene la opción de implementar la carta utilizando algún tipo de familia lógica **TTL** ó **PLD** .

Carta ASM

La carta **ASM** es un diagrama que describe las operaciones secuenciales en un sistema digital. Esta carta está compuesta de tres elementos básicos :

- Estado
- Decisión
- Condicional

ESTADO. Se representa con una casilla rectangular, donde se escriben operaciones de registro o nombres de señal de salida que el control genera mientras se encuentra en este estado (Figura 3.3). El estado recibe un nombre simbólico, el cual se coloca en la esquina superior izquierda de la casilla. El código binario asignado al estado se coloca en la esquina superior de la derecha.

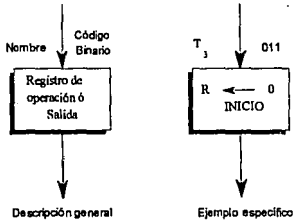


Figura 3.3 Caja de estado

DECISION. Se representa con una casilla en forma de rombo, con dos ó más trayectorias de salida. La condición de entrada que va a probarse está escrita dentro de la casilla. Una trayectoria de salida se toma si la condición es cierta y la otra cuando la condición es falsa. Cuando una condición de entrada está asignada a un valor binario, las dos trayectorias se indican por **1** y **0** (Figura 3.4).

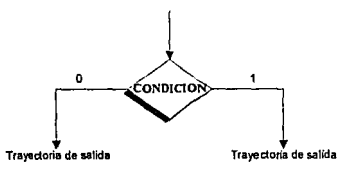


Figura 3.4 Caja de decisión

CONDICION. Es de uso exclusivo en las cartas ASM y la forma de la casilla es ovalada. La trayectoria de entrada a la casilla condicional debe llegar desde una de las trayectorias de salida de una casilla de decisión. Las operaciones de registro o salidas listadas dentro de la casilla condicional se generan durante un estado dado siempre que se satisfaga la condición de entrada (Figura 3.5).

Figura 3.5 Caja condicional

Bloque ASM

Un bloque ASM es una estructura que consta de una casilla de estado y todas las casillas de decisión y condicionales conectadas a sus trayectorias de salida. Un bloque ASM tiene una entrada y cualquier número de trayectorias de salida representadas por la estructura de las casillas de decisión. Un diagrama ASM consta de uno o más bloques interconectados. Un bloque debe ser conectado a un estado siguiente. Cada posible conexión de un estado presente a un estado siguiente es llamado "*camino de unión*" donde el camino de unión depende de la estructura que conforman las casillas de decisión siendo ahí donde se sensan las variables de entrada para así definir la conexión entre estados de la máquina.

Cada bloque ASM representa: un estado presente *(X)*, la salidas en el estado $f[X]$, las salidas condicionales $f[X, Q]$ y el estado siguiente $g[X, Q]$, así como las entradas generales de un estado de máquina $[Q]$.

Las restricciones para la interconexión de bloques en una carta ASM se consideran al momento de elaborar el diseño. Estas son:

- Se debe tener sólo un estado siguiente y variables de entrada estables (*síncronas*) para cada estado presente.
- Si existe más de una casilla de decisión, de las cuales la salida al estado siguiente de cada una de ellas, pasa al mismo estado, esto no es válido, ya que el algoritmo por definición no puede sensar dos variables de entrada simultáneamente.

Cada bloque en el diagrama ASM (Figura 3.6) describe el estado del sistema durante el intervalo de un pulso de reloj. Las operaciones de un estado y sus condicionales se ejecutan con un pulso común de reloj mientras el sistema se encuentra en el estado T_i. El mismo pulso de reloj también transfiere el sistema controlador a uno de los estados siguientes, T_j, T_k o T_l, como dictan los valores binarios de E y F.

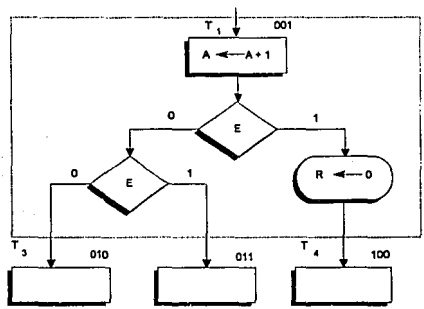


Figura 3.6 Bloque ASM

CONSIDERACIONES DE TEMPORIZADO

El temporizado de todos los registros y **FF's** en un sistema digital se controla por un reloj generador maestro. Los pulsos de reloj se aplican no solo a los registros de la subsección del procesador de datos, sino también a todos los **FF's** en el circuito lógico. Las entradas también están sincronizadas con los pulsos de reloj porque normalmente se generan como salidas de otro circuito que usa las mismas señales del reloj. Si la señal de entrada cambia en un tiempo arbitrario independientemente del reloj, se le llama una *entrada asíncrona*. Para simplificar el diseño se supone que todas las entradas están sincronizadas con el reloj y que cambian de estado como respuesta a una trancisión de borde del pulso de reloj. En forma similar cualquier salida que es una función del estado presente y una entrada síncrona también estarán sincronizadas.

En la carta ASM se considera un bloque entero como una unidad, por lo que todas las operaciones que están especificadas dentro del bloque deben ocurrir en sincronismo durante la transición en borde del mismo pulso de reloj, mientras el sistema cambia desde T_i al estado siguiente. Esto se representa en forma gráfica en la figura 3.7.

1. El registro A se incrementa
2. Si E = 1, el registro R se despeja
3. Dependiendo de los valores de E y F, el control se transfiere al estado siguiente, T_j o T_k o T_l

Figura 3.7 Trancisión entre estados

Secuencia de temporizado

Cada bloque en un diagrama ASM especifica las operaciones que van a realizarse durante un pulso común de reloj. Las operaciones especificadas dentro de las casillas de estado y condicional en el bloque se llevan a cabo en la subsección del procesador de datos. El cambio de un estado al siguiente se lleva a cabo en el control lógico. Con el objeto de apreciar la relación de tiempo implicada, se recomienda hacer una lista de secuencias paso por paso de las operaciones después de cada pulso de reloj desde el tiempo en que ocurre la señal de inicio hasta que el sistema regresa a su estado inicial.

Procesador de datos

La carta ASM da toda la información necesaria para diseñar el sistema digital. Los requisitos para el diseño del subsistema procesador de datos se especifican dentro de las casillas de estado y condicionales. El control lógico se determina mediante las casillas de decisión y las transiciones de estado requeridas.

IMPLANTACION DEL CONTROL

La sección de control de un sistema digital es en esencia un circuito secuencial que puede diseñarse por métodos especializados (*Diseño con flip-flop JK, flip-flops D y decodificador, así como con multiplexores*) que pueden considerarse como una extención del método secuencial clásico combinado con otras suposiciones de simplificación. El **FF's** D es uno de estos métodos especializados que se utiliza para el diseño del control lógico, sin embargo otra alternativa es utilizar una **ROM** o un **PLA** para diseñar el control lógico.

Tabla de estado

La carta **ASM** puede convertirse en una tabla de estado mediante la cual puede diseñarse el circuito secuencial del controlador. Primero deben asignarse valores binarios a cada estado en la carta **ASM**. Para ***n*** **Flip-Flops** en el circuito secuencial de control, la carta **ASM** puede acomodar hasta **2 ^{n}** estados. Un diagrama con tres o cuatro estados requiere un circuito secuencial con dos **FF's**, con cinco a ocho estados, se necesitan tres **FF's**, cada combinación de valores **FF's** representa un número binario para uno de los estados.

Una tabla de estados para un controlador es una lista de los estados y entradas presentes y sus correspondientes estados siguientes y salidas. En la mayoría de los casos hay muchas condiciones "*no importa*" de entrada que deben incluirse, de modo que es aconsejable arreglar la tabla de estado para tomar esto en consideración. El número de renglones en la tabla de estado es igual al número de trayectorias distintas entre los estados en la carta **ASM**.

III.2 REDUCCION DE MINTERMINOS POR EL METODO QUINE-McCLUSKEY

REDUCCION DE FUNCIONES BOOLEANAS

El método de los *mapas de Karnaugh* utilizado para la reducción de funciones booleanas, se complica progresivamente conforme aumenta el número de variables de estado. La técnica de los *mapas de Karnaugh* para funciones con más de cuatro variables de estado, es un reto a la habilidad humana para reconocer los "*encerramientos o cubos*" y en consecuencia, la implementación de un algoritmo por computadora, no es realizable.

Como forma alternativa, el método *Quine-McCluskey* es un procedimiento sistemático tabular, el cual se puede implementar fácilmente en una computadora.

Método de *Quine-McCluskey* para reducción de funciones booleanas

El método comienza con una lista de los mintérminos de la función de los cuales se van a reducir, los mintérminos son almacenados en una tabla formando los 0-cubos (*en lo sucesivo, un cubo se refiere a los agrupamientos de elementos como se realizan en la técnica de mapas de Karnaugh*) los cuales son ordenados para así facilitar el proceso de reducción, el proceso de combinación de 0-cubos para formar cubos de mayor dimensión.

Para facilitar la presentación del procedimiento, se utilizará un ejemplo:
Sea

$$F = \sum m(1,5,7,8,9,10,11,14,15) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.1)$$

así, el arreglo para esta función es:

$$ON = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

a manera de asistencia didáctica del procedimiento, se muestra a continuación el *mapa de Karnaugh* para la función **F** (Figura 3.8 (a)).

x1 x2 \ x3 x4	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	0	1	1
10	1	1	1	1

Figura 3.8 (a) Mapa de Karnaugh para la ecuación 4.1

El primer paso del método, consiste en buscar todas los posibles *1-cubos* agrupados en pares formando así dichos cubos en el mapa de Karnaugh (Figura 3.8 (b))

x1 x2 \ x3 x4	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	0	1	1
10	1	1	1	1

Figura 3.8 (b) Todos los posibles pares de 1's en el Mapa de Karnaugh

Cada par de encerramientos en el mapa de Karnaugh corresponde a un par de 0-cubos en el arreglo **ON**, difiriendo tan solo en una variable, por ejemplo el par (0001, 0101) y (1110,1111). Esto nos indica que se pueden encontrar todos los posibles *1-cubos*, buscando en el arreglo **ON** todos los pares de entradas, los cuales difieren en el valor de una de sus variables. El procedimiento de búsqueda es muy simple pero no todos los *0-cubos* se pueden combinar para formar un *1-cubos* que se distingue por la propiedad particular que a continuación se menciona:

El número de 1's de un 0-cubos a otro debe diferir en una unidad (por *ejemplo: considérense los pares* (0001,0101) y (0111,1111).

El método *Quine-McCluskey*, utiliza la ventaja que nos representa el factor antes mencionado; por lo tanto se debe realizar un ordenamiento al arreglo **ON** de los *0-cubos* de acuerdo al número de 1's que estos contienen. Así el ordenamiento para el ejemplo se muestra en la tabla 3.9 (a):

número de 1's	0-cubos	Equivalencia decimal
1	0001	1
	1000	8
2	0101	5
	1001	9
	1010	10
3	0111	7
	1011	11
	1110	14
4	1111	15

Tabla 3.9 (a) Ordenamiento de 0-cubos

El correcto uso de la tabla 3.9 (a) permite determinar fácilmente los pares de *0-cubos* que se pueden combinar para formar los *1-cubos*.

Los *0-cubos* del grupo superior de la tabla 3.9 (a) no se pueden combinar con los cubos que tengan 3 o más 1's; así, solamente se requieren los *0-cubos* del segundo grupo, los cuales son considerados para realizar las combinaciones con el grupo superior. Por consiguiente, cada *0-cubos* del siguiente grupo, forma los *1-cubos*, en otras palabras, se lleva a cabo una relación entre el elemento seleccionado contra todos los elementos del siguiente grupo. Si en el par de *0-cubos* seleccionado, se identifica que difieren en una sola variable, el correspondiente *1-cubos* se obtiene poniendo una "x" en la posición del elemento, los dos *0-cubos* difieren, obteniéndose de esta manera una tabla de *1-cubos* con su respectivo valor decimal equivalente de los *0-cubos* combinados. Se puede apreciar en la tabla 3.9 (b) de los *1-cubos* que el número de grupos disminuye en uno en comparación con la tabla de los *0-cubos*. A continuación se muestra la tabla 3.9 (b) de los *1-cubos*.

1-cubos	Combinación de 0-cubos
0x01	1,5
x001	1,9
100x	8,9
10x0	8,10
01x1	5,7
10x1	9,11
101x	10,11
1010	10,14
x111	7,15
1x11	11,15
111x	14,15

Tabla 3.9 (b) Ordenamiento de 1-cubos

De lo antes descrito, el procedimiento comienza a comparar el *0-cubos* 0001 con los cubos del siguiente grupo inferior; como se observa, este *0-cubos* puede ser combinado con los cubos 0101 y 1001, ya que solamente con éstos existe una diferencia de una variable, por consiguiente se forman los *1-cubos* 0x01 y x001 respectivamente y son almacenados en la tabla de los *1-cubos*, ver tabla 3.9 (b). Posteriormente, el segundo *0-cubos* en el grupo superior 1000 se compara con los tres *0-cubos* del grupo inferior, pudiéndose combinar solamente con los cubos 1001, 1010 formándose así 100x y 10x0 respectivamente, agregando los *1-cubos* a la tabla.

De esta manera se obtienen todos los *1-cubos* que se pueden formar con los *0-cubos* del segundo grupo (*es decir, el grupo donde el número de 1's es igual a dos, figura 3.9 (b)*) con el tercer grupo, el proceso es exactamente igual y se concluye que la obtención de la tabla completa de los *1-cubos* se realiza en forma similar a como se mencionó anteriormente.

En aplicaciones generales, el método *Quine-McCluskey* dentro del proceso de pares sucesivos en grupos, continúa mientras no se encuentre el último par posible.

Como parte del proceso de obtención de la tabla de los *1-cubos*, los *0-cubos* fueron combinados más de una vez, identificándose con una marca (*los 1-cubos obtenidos corresponden exactamente a los encerramientos en el mapa de Karnaugh de la Tabla 3.8 (b)*); la razón se explica más adelante.

El siguiente paso del procedimiento consiste de una búsqueda a través de la tabla para identificar los pares de los *1-cubos* que se pueden combinar para formar cubos de mayor dimensión (*2-cubos*). Se tienen tres grupos, de donde se principia a realizar la comparación de estos cubos del grupo superior con el segundo grupo. En este caso, los cubos requieren solamente ser comparados con respecto a la posición de la "x". Así, la primer entrada (*0x01*), necesita ser comparada con el cubo 1x10 y como se observa, difiere en sus tres elementos; por lo tanto, no existe combinación, concluyendo la comparación del cubo 0x001 ya que no puede ser combinado con otro *1-cubos*. El siguiente cubo de la lista es x001 el cual no puede ser combinado con ningún otro *1-cubos* porque no existen cubos que tengan "x" en la misma posición dentro del segundo grupo de los *1-cubos*, se concluye por lo tanto, que no existe combinación posible. El tercer cubo en la lista (100x), difiere del 101x en tan solo una posición, así estos dos cubos se pueden combinar formando el *2-cubos* 10xx y se almacena en una nueva tabla marcando los elementos que fueron combinados; finalmente el *1-cubos* del primer grupo 10x0 es comparado con 01x1 y 10x1 (cada uno tiene una "x" en la tercera posición); 10x0 no puede ser combinado con el primero de estos, pero se puede combinar con el segundo, generando el *2-cubos* 10xx, sin embargo, como ya existe éste no se debe agregar a la tabla y se marcan los *1-cubos* que lo generaron. Finalmente se completa la comparación para el primer grupo de los *1-cubos*.

El primer 1-cubo en el segundo grupo es comparado contra los *1-cubos* del tercer grupo de la tabla de la figura 3.9 (b) para determinar los siguientes 2 cubos. Estas comparaciones proporcionan uno o más *2-cubos*, por ejemplo, 1x1x se obtiene de la combinación del par (101x, 111x) y por la combinación del par (1x10, 1x11).

Para este ejemplo, el proceso de combinación de cubos de mayor orden finaliza porque los *2-cubos* de la siguiente tabla 3.10 no es posible combinarlos.

2-cubos	Combinación de *1-cubos*
10xx	8,9,10,11
1x1x	10,11,14,15

Figura 3.10 Tabla de *2-cubos*

En general (*para n variables*), el proceso continúa produciendo cubos de mayor dimensión, mientras la combinación de cubos sea posible.

Antes de explicar la información que nos proporcionan los datos de las tablas obtenidas; la cual es utilizada para formar la mínima expresión de la función F (la cual se utilizó para manejar el ejemplo); es importante mencionar la razón del porque en los *1-cubos* las comparaciones para obtener los *2-cubos*, solamente fue necesario comparar sólo los pares de los *1-cubos* que tengan la "x" en la misma posición. Considérese la formación del término *2-cubos* 10xx, donde se puede apreciar que están formados por los *0-cubos* 8,9,10 y 11, en otras palabras, el *2-cubo* 10xx es formado por la combinación de los mintérminos m8, m9, m10 y m11 mediante procesos algebraicos:

$$m_8 + m_9 + m_{10} + m_{11} = x_4 / x_3 / x_2 / x_1 + x_4 / x_3 / x_2 x_1 + x_4 / x_3 x_2 / x_1 + x_4 x_3 / x_2 x_1 \quad \text{..... (4.2)}$$

$$= x_4 / x_3 / x_2 (/x_1 + x_1) + x_4 / x_3 x_2 (/x_1 + x_1) \quad \text{..... (4.3)}$$

$$= x_4 / x_3 / x_2 + x_4 / x_3 x_2 \quad \text{..... (4.4)}$$

$$= x_4 / x_3 (/x_2 + x_2) \quad \text{..... (4.5)}$$

$$= x_4 / x_3 \quad \text{..... (4.6)}$$

Los términos en (4.4) representan dos *1-cubos* 100x y 101x. en cada uno de estos *1-cubos*, la x nos indica que la variable x_1 puede ser eliminada. Por definición para un problema dado 1-cubo representa un producto de variables de las cuales una puede ser omitida, porque es la combinación de dos 0-cubos con la misma variable, por lo tanto que se pueden combinar solamente *1-cubos* que tengan una variable x en la misma posición. Por esta razón los *2-cubos* se pueden combinar solamente si sus dos "x" se encuentran en la misma posición. En general, la combinación de los *k-cubos* se puede realizar solamente si se tienen sus *k* "x" en la misma posición.

Retomando al problema de minimización de la función F, se pude apreciar que los *2-cubos* corresponden exactamente a dos variables de los encerramientos de cuatro "1s" en la representación de mapas de Karnaugh como se muestra a continuación:

x_1x_2 \ x_3x_4	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	0	1	1
10	1	1	1	1

Donde los *2-cubos* solamente son un subconjunto de los mintérminos dados en la función F. Sin embargo, comparando con la figura 3.8 (b) se aprecia que los *2-cubos* son representados por siete *1-cubos* lo cual indica que debieron checarse siete *1-cubos* en la tabla figura 3.9 (b). Estos *2-cubos* se consideran con lo mínima expresión de la función F (*pero no suficiente*).

Primos implicados

Definición: Un primo implicado es un cubo de una función que no es completamente cubierto por un cubo de mayor dimensión de la función dada.

$x_1 x_2$ \ $x_3 x_4$	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	0	1	1
10	1	1	1	1

Figura 3.11 2-*cubos* representados en el mapa de Karnaugh

Para los cuatro *1-cubos* que no fueron checados estos son considerados como primos implicados de la función **F**. Así en la Figura 3.11 se indican solamente 2 de estos *1-cubos*, los cuales son requeridos para involucrar todos los mintérminos. El método de *Quine-McCluskey*, proporciona una función simplificada directamente de las tablas finales de los cubos requeridos.

La tabla para el ejemplo que se ha estado manejando es la 3.12, donde cada renglón corresponde a uno de los primos implicados y cada columna los relaciona con uno de los mintérminos.

Los primos implicados son ordenados en la tabla de acuerdo a la dimensión del cubo al cual pertenecen. Los cubos de mayor dimensión, que involucran el mayor número de mintérminos, se almacenan primero. Para este caso, solamente existen 2 tipos de cubos (*2-cubos* y *1-cubos*) para ser almacenados pero en un caso general pueden existir varios tipos de cubos almacenándose en la tabla los cubos de mayor a menor dimensión. Los primos implicados son almacenados en la columnas de la izquierda y representados por los mintérminos que involucran; así por ejemplo, cada primo implicado es asignado a un renglón en la tabla, donde cada renglón corresponde a un primo implicado y es marcado en las columnas de la derecha; las cuales corresponden a los mintérminos relacionados con los primos implicados.

La búsqueda para la mínima cobertura - *la mínima cobertura para una función F, es el mínimo conjunto de cubos necesarios para involucrar los mintérminos de una función dada* - comienza con la búsqueda a través de las columnas de la tabla para identificar que mintérminos fueron checados solamente una vez; para este ejemplo, se aprecia que corresponde a la columna 8 y 14. Cuando un mintérmino es checado una sola vez, implica que el 2-cubo 10xx (*que involucra los mintérminos* 8, 9, 10, 11) es el primo implicado identificándose con el mintérmino m8, encontrándose así la mínima cobertura para la ecuación (4.1) dada, la cual incluye el 2-cubo 10xx. En otras palabras, el 10xx es un ***primo implicado esencial.*** Similarmente el 2-cubo 1x1x es también un primo implicado dado que solamente identifica al mintérmino m14. Así se pueden incluir estos dos primos implicados que son marcados con (*) dentro de la mínima cobertura. Los mintérminos identificados con (*) a la izquierda de la columna, son primos implicados esenciales.

En este punto del procedimiento, la tabla de los primos implicados muestran la forma que se observa en la figura 3.11. Los primos implicados esenciales nos proporcionan la información necesaria pero no suficiente

para completar la función **F**, por lo tanto ahora se obtiene la mínima expresión de la función utilizando los primos implicados que han quedado sin marcar en la tabla de los *1-cubos*. Como se puede deducir de la Tabla 3.9 **(b)** estos son los tres posibles pares de *1-cubos* que se pueden seleccionar para completar la cobertura mínima de los mintérminos: (1, 5 y 5,7) (1,5 y 7,5,15) y (1,9 y 5,7); de donde se debe elegir el par de cubos que cubran los mintérminos restantes sin involucrar un mintérmino que exista dentro de otro primo implicado, en otras palabras, si seleccionamos el par de cubos (1, 5 y 7,15), se está abarcando un mintérmino que se encuentra definido en un primo implicado esencial.

Los posibles selecciones de los primos implicados para completar la cobertura puede tener como consecuencia algún incremento en el costo de hardware (porque cada posible selección consiste de un par de *1-cubos*); considerando lo antes descrito, se elige el par (1,5 y 5,7). Estos dos *1-cubos* y los *2-cubos* que fueron identificados como primos implicados esenciales (PIE) proporcionan la mínima expresión de (4.1).

Finalmente se escribe la mínima expresión de suma de productos, en donde los productos asociados a cada uno de estos primos implicados se realizan de la siguiente manera:

8,9,10,11	10xx	x_4/x_3
10,11,14,15	1x1x	x_4x_2
1,5	0x01	$/x_4/x_2x_1$
5,7	01x1	$/x_4x_3x_1$

obteniéndose de esta manera la mínima expresión para 4.1

$$F = x_4 / x_3 + x_4x_2 + /x_4 / x_2x_1 + /x_4x_3x_1$$

Nótese que esta es la mínima suma de productos pero no es única. Dado que se puede elegir otro par de *1-cubos* para proporcionar otra mínima expresión equivalente a la función antes presentada.

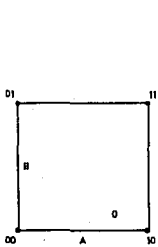
Mintérminos cubiertos por PI	**1**	**5**	**7**	**8**	**9**	**10**	**11**	**14**	**15**
8,9,10,11				+	+	+	+		
10,11,14,15						+	+	+	+
1,5			+						
1,9		+				+			
5,7			+	+					
7,15				+					+

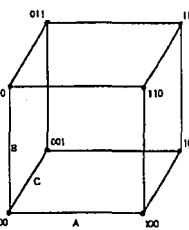
Tabla 3.12 Primos implicados. Tabla inicial

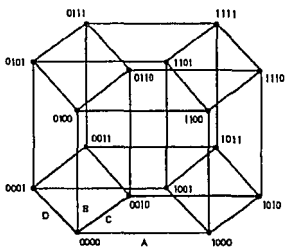
Los métodos *Quine-McCluskey* y Karnaugh, aunque aparentemente distintos, se basan en la misma ley de complementación ($x + /x = 1$); ambos consisten en una agrupación "*la de Karnaugh es geométrica y la de Quine-McCluskey es numérica*" y hacen uso de la ley de tautología, factorizando la solución final.

El método de la representación gráfica es útil hasta las funciones de cuatro variables, mientras que el método numérico se considera para mayor número de variables. Es interesante observar que Karnaugh desarrolló su método a comienzos de la década de los 50 cuando la computadora aún no había tomado fuerza, mientras que *Quine-McCluskey* desarrollaron el suyo a finales de dicha década.

Debemos hacer una observación final acerca del método de simplificación *Quine-McCluskey*, el artificio necesario en este método, especialmente cuando se usa la tabla de términos primos es mucho mayor que el método de la representación gráfica. Sin embargo, el primero se puede extender para simplificar funciones de tantas variables como se desee. Además el papel de la tabla de términos primos se puede sustituir mediante la programación en computadora. Por otra parte, el método de la representación gráfica es muy efectivo para funciones de hasta cuatro variables, se hace complicado para cinco y más o menos imposible de seis en adelante.







Conversión de Logica Positiva a Lógica Negativa

La función **F** obtenida anteriormente, aunque minimizada tiene la desventaja de presentarse en lógica positiva, esto es entendible debido a que por lo general los cursos de diseño lógico, así como la literatura al respecto, conducen a una forma de pensamiento con ese enfoque de lógica positiva. La desventaja consiste en que para programar los **PAL**, se deben obtener mintérminos (*suma de productos*) con la función **F** negada (/F), dadas las caracteristicas del dispositivo programable. Se tiene que programar con lógica negativa para que el dispositivo **PAL** ocupe la corriente de la fuente y disipe menor cantidad de potencia y nos proporcione mayor corriente eléctrica en las salidas, ya sea de tipo registro o de tipo buffer tri-state. Una propuesta alternativa consiste en programar con lógica positiva pero un dispositivo **GAL** (*debido a que la polaridad es programable*).

En caso de no contar con un dispositivo **GAL**, es necesario utilizar los postulados del diseño lógico para

manipulación de funciones booleanas. Entre los teoremas para dicha conversión encontramos:

Teorema 1	A + 0	=	A
Teorema 2	A * 0	=	0
Teorema 3	A + 1	=	1
Teorema 4	A * 1	=	A
Teorema 5	A + A	=	A
Teorema 6	A * A	=	A
Teorema 7	A + /A =	=	1
Teorema 8	A * /A	=	0
Teorema 9	//A	=	A
Teorema 10	A + A * B	=	A
Teorema 11	A * (A + B)	=	A
Teorema 12	(A + B) * (A + C)	=	A+B*C
Teorema 13	A + /A * B	=	A+B

Leyes Conmutativas

$$A+B = B+A$$
$$A*B = B*A$$

Leyes Asociativas

$$A+B+C=(A+B)+C=A+(B+C)$$
$$A*B*C=(A*B)*C=A*(B*C)$$

Leyes Distributivas

$$A+(B*C*D)=(A+B)*(A+C)*(A+D)$$
$$A*(B+C+D)=A*B+A*C+A*D$$

Teoremas de De Morgan

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

Teorema de Expansión de Shannon

F(A,B,C, ...) = A * F(1,B,C, ...) + /A * F(0,B,C, ...)
F(A,B,C, ...) = [A + F(0,B,C,...)] * [/A + F(1,B,C, ...)]

Por lo anteriormente expuesto con referencia a los **PAL**, es necesario que la función minimizada sea cambiada a la forma **SOP** (*suma de productos*) con lógica negada. Esto es posible gracias a la utilización de los teoremas de De Morgan.

A manera de proporcionar un mejor entendimiento se propone el siguiente ejercicio, mismo que será implementado en el apéndice de esta tesis.

Sea la ecuación de lógica positiva:

Q1=Q1/SR/SL + Q2SR/SL + Q0/SRSL + D1SRSL

aplicando teoremas de De Morgan

/Q1=/Q1+SR+SL * /Q2+/SR+SL * /Q0+SR+/SL * /D1+/SR+/SL

aplicando teorema de Shannon o Expansión

/Q1*/Q2*/Q0*/D1	/SR*/Q2*/Q0*/D1	SL*/Q2*/Q0*/D1
/Q1*/Q2*/Q0*/SR	/SR/Q2*/Q0*/SR	SL*/Q2*/Q0/*SR
/Q1*/Q2*/Q0*/SL	/SR*/Q2*/Q0*/SL	SL*/Q2*/Q0*/SL
/Q1*/Q2*SR*/D1	/SR*/Q2*SR*/D1	SL*/Q2*SR*/D1
/Q1*/Q2*SR*/SR	/SR*/Q2*SR*/SR	SL*/Q2*SR*/SR
/Q1*/Q2*SR*/SL	/SR*/Q2*SR*/SL	SL*/Q2*SR/*SL
/Q1*/Q2*/SL*/D1	/SR*/Q2*/SL*/D1	SL*/Q2*/SL*/D1
/Q1*/Q2*/SL*/SR	/SR*/Q2*/SL*/SR	SL*/Q2*/SL*/SR
/Q1*/Q2*/SL*/SL	/SR*/Q2*/SL*/SL	SL*/Q2*/SL*/SL
/Q1*/SR*/Q0*/D1	/SR*/SR*/Q0*/D1	SL*/SR*/Q0*/D1
/Q1*/SR*/Q0*/SR	/SR*/SR*/Q0*/SR	SL*/SR*/Q0*/SR
/Q1*/SR*/Q0*/SL	/SR*/SR*/Q0*/SL	SL/SR*/Q0*/SL
/Q1*/SR*SR*/D1	/SR*/SR*SR*/D1	SL*/SR*SR*/D1
/Q1*/SR*SR*/SR	/SR*/SR*SR*/SR	SL*/SR*SR*/SR
/Q1*/SR*SR*/SL	/SR*/SR*SR*/SL	SL*/SR*SR*/SL
/Q1*/SR*/SL*/D1	/SR*/SR*/SL*/D1	SL*/SR*/SL*/D1
/Q1*/SR*/SL*/SR	/SR*/SR*/SL*/SR	SL*/SR*/SL*/SR
/Q1*/SR*/SL*/SL	/SR*/SR*/SL*/SL	SL*/SR*/SL*/SL
/Q1*SL*/Q0*/D1	/SR*SL*/Q0*/D1	SL*SL*/Q0*/D1
/Q1*SL*/Q0*/SR	/SR*SL*/Q0*/SR	SL*SL*/Q0*/SR
/Q1*SL*/Q0*/SL	/SR*SL*/Q0*/SL	SL*SL*/Q0*/SL
/Q1*SL*SR*/D1	/SR*SL*SR*/D1	SL*SL*SR*/D1
/Q1*SL*SR*/SR	/SR*SL*SR*/SR	SL*SL*SR*/SR
/Q1*SL*SR*/SL	/SR*SL*SR*/SL	SL*SL*SR*/SL

/Q1*SL*/SL*/D1
/Q1*SL*/SL*/SR
/Q1*SL*/SL*/SL

/SR*SL*/SL*/D1
/SR*SL*/SL*/SR
/SR*SL*/SL*/SL

SL*SL*/SL*/D1
SL*SL*/SL*/SR
SL*SL*/SL*/SL

Aplicando A * /A = 0

/Q1*/Q2*/Q0*/D1
/Q1*/Q2*/Q0*/SR
/Q1*/Q2*/Q0*/SL
/Q1*/Q2*SR*/D1
/Q1*/Q2*SR*/SL
/Q1*/Q2*/SL*/D1
/Q1*/Q2*/SL*/SR
/Q1*/Q2*/SL*/SL
/Q1*/SR*/Q0*/D1
/Q1*/SR*/Q0*/SR
/Q1*/SR*/Q0*/SL
/Q1*/SR*/SL*/D1
/Q1*/SR*/SL*/SR
/Q1*/SR*/SL*/SL
/Q1*SL*/Q0*/D1
/Q1*SL*/Q0*/SR
/Q1*SL*SR*/D1

SR*/Q2*/Q0*/D1
SR*/Q2*/Q0*/SL
SR*/Q2*SR*/D1
SR*/Q2*SR*/SL
SR*/Q2*/SL*/D1
SR*/Q2*/SL*/SL
SR*SL*/Q0*/D1
SR*SL*SR*/D1

SL*/Q2*/Q0*/D1
SL*/Q2*/Q0*/SR
SL*/Q2*SR*/D1
SL*/SR*/Q0*/D1
SL*/SR*/Q0*/SR
SL*SL*/Q0*/D1
SL*SL*/Q0*/SR
SL*SL*SR*/D1

Aplicando A * A = A

/Q1*/Q2*/Q0*/D1
/Q1*/Q2*/Q0*/SR
/Q1*/Q2*/Q0*/SL
/Q1*/Q2*SR*/D1
/Q1*/Q2*SR*/SL
/Q1*/Q2*/SL*/D1
/Q1*/Q2*/SL*/SR
/Q1*/Q2*/SL 1
/Q1*/SR*/Q0*/D1
/Q1*/SR*/Q0 8
/Q1*/SR*/Q0*/SL
/Q1*/SR*/SL*/D1
/Q1*/SR*/SL 2
/Q1*/SR*/SL 2
/Q1*SL*/Q0*/D1
/Q1*SL*/Q0*/SR
/Q1*SL*SR*/D1

SR*/Q2*/Q0*/D1
SR*/Q2*/Q0*/SL
SR*/Q2*/D1 3
SR*/Q2*/SL 4
SR*/Q2*/SL*/D1
SR*/Q2*/SL 4
SR*SL*/Q0*/D1
SR*SL*/D1 5

SL*/Q2*/Q0*/D1
SL*/Q2*/Q0*/SR
SL*/Q2*SR*/D1
SL*/SR*/Q0*/D1
SL*/Q0*/SR 6
SL*/Q0*/D1 7
SL*/Q0*/SR 6
SL*SR*/D1 5

Aplicando ley distributiva y reduciendo por absorción

/Q1*/SR*/SL + /Q1*/SR*/SL*D1 = /Q1*/SR*/SL (1+D1)
= /Q1*/SR*/SL

Reacomodando las ecuaciones a partir de los 8 mintérminos numerados:

1. **/Q1*/Q2*/SL**
2. **/Q1*/SL*/SR**
3. **SR*/Q2*/D1**
4. **SR*/Q2*/SL**
5. **SR*SL*/D1**
6. **SL*/SR*/Q0**
7. **SL*/Q0*/D1**
8. **/Q1*/SR*/Q0**

/Q1*/Q2*/Q0*/D1
/Q1*/Q2*/Q0*/SR.....8
/Q1*/Q2*SR*/D1.....3
/Q1*/Q2*SR*/SL.....4
/Q1*/Q2*/SL*/D1.....1
/Q1*/Q2*/SL*/SR.....1
/Q1*/SR*/Q0*/SL.....8
/Q1*/SR*/Q0*/D1.....8
/Q1*/SR*/SL*/D1.....2
/Q1*/SL*/Q0*/D1.....7

/Q1*SL*/Q0*/SR.....8
/Q1*SL*SR*/D1.....5
SR*/Q2*/D1*/Q0.....3
SR*/Q2*/D1*/SL.....4
SR*SL*/D1*/Q0.....5
SL*/Q2*/Q0*/D1.....7
SL*/Q2*/Q0*/SR.....6
SL*/Q2*SR*/D1.....5
SL*/SR*/Q0*/D1.....7

Como puede observarse los mintérminos de 4 variables son anulados por los mintérminos que contienen únicamente 3 variables y que son comunes al mintérmino de cuatro, a excepción del mintérmino que no tiene asociado ningún número. sin embargo, puede aplicarse un truco algebráico válido para anular este mintérmino. Este arreglo algebráico consiste en complementar una variable que tenga como valor 1, de modo tal que al ser posmultiplicada por el mintérmino, éste no altere su valor. Así por ejemplo, se tiene:

/Q0*/Q1*/Q2*/D1 (SR + /SR) = /Q0*/Q1*/Q2*/D1*SR +
/Q0*/Q1*/Q2*/D1*/SR

donde el primer término se anula por el mintérmino 8 y el segundo miembro se anula por el mintérmino 3, con lo cual se puede conformar una nueva función /Q1.

Supóngase que se eliminan cuatro de los ocho mintérminos para que sean implementados con el **PAL16R6**, entonces

/Q1*/SR*/Q0 (SL+/SL)	=	/Q1*/SR*/Q0*/SL + /Q1*/SR*/Q0*SL
SL*/Q2*/D1 (SR+/SR)	=	SL*/Q0*/D1*/SL + SL*/Q0*/D1*/SR
SR*/Q2*/D1 (SL+/SL)	=	SR*/Q2*/D1*/SL + SR*/Q2*/D1*SL
/Q1*/Q2*/SL (SR+/SR)	=	/Q1*/Q2*/SL*SR + /Q1*/Q2*/SL*/SR

A partir de los cuatro mintérminos numerados a continuación se eliminan las cuatro anteriores ecuaciones:

a) /Q1*/SR*/SL
b) /Q0*/SR*SL
c) /Q2*SR*/SL
d) /D1*SR*SL

La elección de estos cuatro mintérminos, de hecho, no fué tan aleatoria sino que se toma en cuenta el criterio de prioridad en la aparición de los estados. Dicha condición se basa en el hecho de que ***una ecuación en la que aparece un único estado siguiente tiene prioridad para anular a otra ecuación donde aparezcan más de un estado siguiente, siempre y cuando tengan al menos un estado común***, es decir, Q1 puede eliminar a un producto Q1*Q2.

Con este tipo de análisis podemos concluir finalmente que la ecuación inversa por leyes de De Morgan de Q1 es:

/Q1 = /Q1*/SR*/SL + /Q0*/SR*SL + /Q2*SR*/SL + /D1*SR*SL

la cual puede ser introducida como **SOP** al dispositivo programable deseado para convertirse al formato **JEDEC**.

Es menester mencionar que esta ecuación no es única, puede tener soluciones múltiples dependiendo de la asociación y decisión para eliminar ciertas variables. Es importante tener en cuenta que si se desea formular un algoritmo computacional que resuelva el problema de conversión de lógica positiva a negativa se pueden encontrar numerosas dificultades, que pondrán a prueba la habilidad de recursos por parte del programador, y que por consiguiente puede llegar a perder generalidad al tomarse ejemplos específicos para la prueba del programa (*benchmark*); esto ocurre debido a la multiplicidad de casos que pueden existir en los diseños digitales, con lo que se vuelve un problema de buen diseño conceptual para minimizar costos.

PROPOSICION DE UN ALGORITMO COMPUTACIONAL PARA CONVERSION DE LOGICA POSITIVA A LOGICA NEGATIVA

La expansión de Shannon es laboriosa y se torna complicada a medida que se incrementa el número de variables, esto es un problema delicado, ya que se puede argumentar que la computadora es quien se encarga de hacer todo el trabajo de multiplicación miembro a miembro para la expansión. El problema real es que al incrementarse el número de variables implicadas en el diseño, el almacenamiento en memoria queda condicionado a los recursos del hardware o a la capacidad del programador para hacer almacenamiento temporal en memoria virtual (*swap*). Nos referimos al hardware en cuanto a memoria RAM y espacio de disco duro, pues el cálculo puede ser lento si no es una máquina de acceso rápido y de procesador de alta velocidad.

Para ejemplificar este problema y puntualizarlo, pensemos en una ecuación de 8 variables por 8

mintérminos que es el caso más extremo de los dispositivos **PAL**. Para elaborar la expansión por Shannon miembro a miembro requiere un almacenamiento de memoria dado por la expresión:

$$8^7 = 2097152 * \text{tipo de variable} * \text{número de ecuaciones}$$

el tipo de variable hace referencia a la extensión en bits o bytes con la que se almacenará en memoria mediante un lenguaje de programación. En nuestro caso si se almacenará en lenguaje " **C** " con una variable de tipo **CHAR** que ocupa 8 bits el número se incrementa a 8^8=16777216 bits. Motivo suficiente para pensar un poco más detalladamente como se programará un algoritmo de expansión de Shannon, en lugar de utilizar simples arreglos matriciales o listas ligadas de apuntadores (*estructuras de datos*) o bien recursividad.

El algoritmo computacional, pensando el posible "*gasto*" y en el tiempo de respuesta del sistema al estímulo de entradas, se ha pensado del siguiente modo:

1. Hacer la expansión por ecuación individual

En lugar de almacenar en memoria directamente el producto de los mintérminos, se establecerán criterior que sirvan de filtro para decidir si se almacenará o no el producto de la operación realizada:
filtro 1: De acuerdo al teorema A*/A=0 el término queda anulado
filtro 2: De acuerdo al teorema A*A=A el término queda reducido en número de variables

en consecuencia el número de productos queda críticamente reducido para su almacenamiento, quizá en un 67 %, debido a las probabilidades de repetición o anulación del producto.
Es cierto que el algoritmo hace más lento el tiempo de respuesta pero a cambio ganamos recursos del sistema y se puede compensar la lentitud si se adquiere un procesador más rápido o bien un coprocesador para la ayuda de las operaciones.

Las condiciones de anulación se irán formado en una lista de comparación para que cuando un producto se ha realizado se haga un chequeo de esta lista para saber si se anula o bien se reduce.

El algoritmo no nos garantiza la minimización de la ecuación pero si nos muestra los mintérminos con los cuáles pueden ser formados Suma de Productos para el dispositivo lógico programable. Lo anterior se argumenta haciendo alusión a que la respuesta de minimización no es única como se mostró.

2. Aplicación del algoritmo

Sea la ecuación

$$Q5 = Q5*/SR*/SL + RILO*SR*/SL + Q4*/SR*SL + D5*SR*SL$$

Aplicando leyes de De Morgan

$$/Q5 = (/Q5+SR+SL) * (/RILO+/SR+SL) * (/Q4+SR+/SL) * (/D5+/SR+/SL)$$

3. Efectuando cambio de variables

A=/Q5
B=SR
C=SL
D=/RILO
E=/SR
F=/Q4
G=/SL
H=/D5

Puede observarse que ninguna de las variables están repetidas, debido a que se han ido formado secuencialmente conforme se realiza la lectura de la ecuación. A partir de esta nueva lista y nuevas variables se establece un nuevo arreglo donde se establecen las condiciones de igualdad y anulación.

$$\begin{bmatrix} BE = 0 \\ CG = 0 \\ AA = A \\ BB = B \\ CC = C \\ EE = E \\ FF = F \\ GG = G \\ HH = H \end{bmatrix}$$

Reacomodando los mintérminos con el cambio de variables:

/Q5+SR+SL	A+B+C
/RILO+/SR+SL	D+E+C
/Q4+SR+/SL	F+B+G
/D5+/SR+/SL	H+E+G

4. Escribiendo los mintérminos como matriz y se haga la multiplicación miembro a miembro

$$\begin{bmatrix} A & B & C \\ D & E & C \\ F & B & G \\ H & E & G \end{bmatrix}$$

se obtiene como resultado los siguientes productos

ADFH	BDFH	CDFH
ADFE	-	CDFE
ADFG	BDFG	-
ADBH	BDBH	CDBH
ADBG	BDG	-
ADGH	BDGH	-
ADGE	-	-
ADG	BDG	-
AEFH	-	CEFH
AEF	-	CEF
AEFG	-	-
AE	-	CE
AEGH	-	-
AEG	-	-
ACFH	BCFH	CFH
ACFE	-	CFE
ACBH	BCH	CBH

El almacenamiento es mcuho menor que el total de la expansión de Shannon, debido a los filtros establecidos por los teoremas.

III.3 GENERACION DEL CODIGO JEDEC

FORMATO ESTANDAR PARA TRANSFERENCIA DE DATOS Y PROGRAMADOR DE DISPOSITIVOS LOGICOS PROGRAMABLES

Un estándar desarrolla y previene la proliferación de formatos de transferencia de datos que ocurren con los desarrollos de sistemas y microprocesadores, el enfoque es en el campo de los dispositivos programables y sus herramientas de soporte, sin embargo no quiere decir que no pueda ser probado para otros dispositivos de diseño semicompleto.

El estándar incluye un protocolo de transmisión basado en formatos de **PROM** tradicionales que almacenan programas compartidos con una terminal de puerto serial de una computadora, pero debemos tomar en cuenta que no es un protocolo de comunicación completo y no puede corregir errores.

Se define un formato de datos para transferir fusibles o celdas de estado a través de sistemas de desarrollo y programación pero no se define la arquitectura del dispositivo, tampoco los tipos de arreglos lógicos o las salidas de las macroceldas. De igual manera el estándar no define el algoritmo de programación o la tecnología del dispositivo especifico para accesar información de los fusibles o celdas.

Los dispositivos lógicos de campo programable pueden requerir más pruebas que las memorias programables, pero el estándar define un formato de prueba funcional que no es un parámetro de propósito general para un lenguaje.

Resumen de programación y prueba de campos

La programación y la información de prueba contiene varios campos. A continuación se listan los campos y una descripción de los identificadores.

Identificador	Descripción
(n.a.)	Especificación de Diseño
N	Notas
QF	Número de fusibles en el dispositivo ***
QP	Número de pines en el vector de pruebas ***
QV	Número máximo de vectores de prueba ***
F	Estado predeterminado (default) de pines *
L	Lista de fusibles *
C	Fusible de checksum
X	Condiciones de Prueba predeterminadas **
V	Vectores de Prueba **
P	Secuencia de Pines **
D	Dispositivo (Obsoleto)
G	Fusible de Seguridad
R,S,T	Análisis de Asignación
A	Tiempo de Acceso
*	Reconocimiento del Programador

**	Reconocimiento para Pruebas
***	Sistemas para Desarrollo

Cambios al estándar de octubre de 1983

El estándar **JEDEC** de julio de1983 define los campos **D, F, L, C, V** y **P**. Otros campos son utilizados y formalmente definidos para este nuevo estándar, el cual es una versión superior que cubre totalmente la versión anterior a 1983. Anteriormente la secuencia y combinación de almacenamiento de varios campos no estaba claramente definida y creaba algunos conflictos en su utilización, es por eso que el campo **D** en este nuevo estándar es obsoleto, en cambio se han agregado pruebas de vectores y condiciones "*no importa*" (dont' care).

PROTOCOLO DE TRANSMISION

Sintáxis del Protocolo

El protocolo **STX-ETX** está basado en formatos tradicionales para **PROM**, dispositivos programables que pueden compartir un puerto serial de computadora con una terminal. La transmisión consiste de un carácter para señalar que está listo el texto **START-OFF-TEXT (STX)**, varios campos, un carácter para señalar el fin de texto **END-OFF-TEXT (ETX)** y chequeo de transmisión. El grupo de caracteres consiste de caracteres **ASCII** y cuatro caracteres de control **(STX, ETX, CR, LF)**.

La sintáxis del protocolo de transmisión es :

$$<formato> ::= <STX> \{ <campo> \} <ETX> <xmit\ check\text{-}sum>$$

Chequeo de la Transmisión

El *Check-sum* de transmisión es una suma de 16 bits de todos los caracteres **ASCII** transmitidos incluyendo el **STX** y **ETX**. El bit de paridad es excluído en esta tesis.

Sintáxis del check-sum de transmisión :

$$<xmit\ check\text{-}sum> ::= <digito\text{-}hexadecimal> : 4$$

Deshabilitando el CHECKSUM de transmisión

Algunas computadoras no almacenan caracteres de control en particular al final de la línea. El equipo receptor algunas veces puede aceptar los valores "0000" como un checksum válido por lo que este método sirve para deshabilitar el checksum de transmisión.

CAMPO DE DATOS

Sintáxis general del Campo

En general, cada campo en el formato se inicia con un identificador seguido por información y

terminado con un asterisco (*). Por ejemplo " C1234 * ", que especifica el checksum para los fusibles de datos 1234.

Las especificaciones de diseño del encabezado no tienen un identificador y pueden ser el primer campo en la transmisión, e inmediatamente seguido por el **STX**.

Sintáxis del campo :

< *campo* > : : = { < *delimitador* > } < *delimitador de campo* >
{ < *campo de carácter* > } ' * '

< *identificador de campo* > : : = 'A' | 'C' | 'D' | 'F' | 'G'
| 'L' | 'N' | 'P' | 'Q' | 'R'
| 'S' | 'T' | 'V' | 'X'

< *identificador reservado* > : : = 'B' | 'E' | 'H' | 'I' | 'J'
| 'K' | 'M' | 'O' | 'U' | 'W'
| 'Y' | 'Z'

Identificadores de campo

Cada campo tiene identificador de tipo carácter el cual identifica el tipo de campo. Múltiples caracteres como identificador pueden ser utilizados creando subcampos (por ejemplo **"A1"**, **"A$"**, ó **"AB3"**). El campo es terminado con un asterisco (*), mientras no se requieran, **CR** (*retorno de carro*) y **LF** (*adelanto de linea*) pueden utilizar formatos de lectura. Los siguientes identificadores han sido reservados y no pueden utilizarse en una función para usos futuros. Los equipos receptores pueden ignorar campos de inicio con identificadores reservados. El significado de los identificadores de campo se especifican en la tabla 3.13 .

A -	Tiempo de acceso	N -	Nota
B -	*	O -	*
C -	Check sum	P -	Secuencia de pines
D -	Tipo de dispositivo	Q -	Valor
E -	*	R -	Vector de resultados
F -	Estado predeterminado del fusible	S -	Vector de inicio
G -	Fusible de seguridad	T -	Ciclos de prueba
H -	*	U -	*
I -	*	V -	Vector de prueba
J -	*	W -	*
K -	*	X -	Condiciones de prueba predeterminadas
L -	Lista de fusibles	Y -	*
M -	*	Z -	*

Tabla 3.13 Identificadores de campo (* indica reservado para futuros usos)

COMENTARIOS Y DEFINICION DE CAMPOS

Especificaciones de diseño

Las especificaciones de diseño son el primer campo en el formato, pueden incluir, no solamente un identificador que señale el inicio, sino también un asterisco que termine el campo y los contenidos de especificación del diseño que no estén definidos pero que consistan de :

1. Nombre y compañía del usuario
2. Fecha, número de parte y revisión
3. Fabricante del dispositivo
4. Otra información

Sintáxis de especificación del diseño :

*< especificación de diseño > : : = { caracter de campo > } ' * '*

Notas (N)

El campo de notas es utilizado para comentarios en el archivo de datos. El campo **note** *(nota)* puede aparecer siempre que en el archivo el equipo receptor ignore este campo.

Sintáxis del campo nota :

*< note > : : = ' N ' < field character > ' * '*

Definición del dispositivo (D) Obsoleto

Este campo ahora es obsoleto y ha sido eliminado para hacer una tecnología independiente del dispositivo.

Valores (QF, QP, QV)

El campo **Q** expresa valores o límites dentro de los cuáles puede estar el equipo receptor. Tres subcampos son definidos :

- El subcampo **F** para el número de fusibles
- El subcampo **P** para el número de pines o condiciones de prueba de un vector
- El subcampo **V** para el máximo número de vectores de prueba

Sintáxis para valores de campo :

*< fusible limite > : : = ' QF ' < número > ' * '*
*< numero de pines > : : = ' QP ' < número > ' * '*
*< vector limite > : : = ' QV ' < número > ' * '*

CAMPOS PROGRAMABLES DEL DISPOSITIVO

Sintáxis y vista general

Cada fusible o celda de un dispositivo es asignada a un número decimal el cual puede tener dos posibles estados:

- Un **0** especifica resistencia baja en la ligadura (*una conexión lógica a través de dos puntos*)
- Un **1** especifica resistencia alta en la ligadura (*no existe conexión lógica entre dos puntos*)

El número de fusibles inicia con un cero y son consecutivos hasta un número máximo de fusibles. Por ejemplo un dispositivo con 2048 fusibles puede tener números entre 0 y 2047. La información de fusibles describe el estado de cada fusible en el dispositivo mediante tres campos:

- El estado predeterminado **(F)**
- La lista de posibles **(L)**
- El fusible del checksum **(C)**

Todos los fusibles programados por el usuario o celdas pueden ser especificadas con un campo **L**.

La sintáxis de los campos de información de fusibles es :

< información del fusible > : : = { < estado de default > } < lista de fusibles >
{ < lista de fusibles > } { < fusible de checksum > }
*< estado predeterminado > : : = ' F ' < digito binario > ' * '*
< listado de fusibles> : : = ' L ' < número > < delimitador >
*{ < digito binario > { < delimitador > } ' * '*
*< fusible de checksum > : : = ' C ' < digito hexadecimal > : 4 ' * '*

Estado del fusible predeterminado o por default (F)

El campo **F** define el estado del fusible que no está explícitamente definido en el campo **L**. Si el campo **F** no es especificado, todos los fusibles tienen un estado definido en el campo **L**, si el estado de default es utilizado puede especificarse después del campo **QF** y antes del primer campo **L**.

Ejemplo: **F0 *** significa que el conjunto por default es **0**

Listado de fusibles (L)

El campo **L** inicia con un número decimal de fusibles y es seguido por un corrimiento de fusibles de estado (**0** y **1**) (*el número de fusible puede incluir ceros*). Por ejemplo "L12" y "L0012" es lo mismo. El espacio **AND/OR** y el retorno de carro (**CR**) pueden estar separados del número de fusibles, el conjunto de fusibles de estado puede ser tan largo como uno quiera (*arriba del máximo número de fusibles almacenados*).

Si el estado es especificado por un fusible más de una vez, el último estado reemplaza todos los unos anteriores especificados para ese fusible. Este almacenamiento o archivo puede ser modificado o "*parchado*" mediante nuevas pruebas de fusibles en el archivo.

Fusible Checksum (C)

El fusible de información checksum es utilizado par detectar errores en la transmisión. El checksum es un dispositivo entero (*fusible numero 0 al fusible de número máximo que puede ser especificado en el campo QF*). Si existen múltiples campos **C** se recibirá únicamente el último como más significativo.

El campo contiene una suma de 16 bits (*módulo 65535*), de palabras de 8 bits que contienen los fusibles de estado para un dispositivo entero. Las palabras de 8 bits son formadas como se muestra en la figura por un bit más significativo (*bit 7*) y uno menos significativo (*bit 0*).

CAMPO DE PRUEBA DEL DISPOSITIVO

Sintáxis y vista general

La información de prueba funcional se lleva acabo mediante vectores de prueba que contienen condiciones (Tabla 3.14) para cada pin del dispositivo.

0	Controlador de entrada baja
1	Manejador de entrada alta
2-9	Controlador de entrada de supervoltaje
B	Registro de precargado
C	Controlador de entrada baja, alta, baja
F	Entrada flotante ó salida
H	Prueba de salida
K	Controlador de entrada alta, baja, alta
L	Prueba de salida baja
N	Pines de encendido y salidas no probadas
P	Prellamadas a los registros
X	Salidas no probadas, entradas con nivel de default
Z	Pruebas de entrada ó salida para alta impedancia

Tabla 3.14 Condiciones de prueba

Condiciones de prueba por default (X)

El campo **X** define un nivel lógico de entrada para vectores de prueba no definidos explícitamente y para la condición de prueba "*no importa*" o "*dont' care*", puede tener un conjunto de vectores através de un máximo (*especificado en QV*) para entradas con pruebas de condición predeterminadas (*por default*) . Si el campo **X** es utilizado, puede especificarse después del **QV** y **QP** y antes del primer vector de prueba.

Ejemplo: **X1*** La condición de prueba por default para **1** (nivel alto).

Vectores de prueba

Cada vector de prueba contiene N condiciones, donde **N** es el nivel de pines en el dispositivo.

El campo **V** inicia con un número decimal de vector seguido por un espacio, posteriormente las series de condiciones de prueba para cada pin y terminan con un *.

Ejemplo : V0001 000000XXXNXXXHHHLXXN*

Los vectores son aplicados en orden numérico al dispositivo que es probado donde el número más alto del vector que se aplica es definido por el campo **QV**. Si un vector no es especificado durante la transferencia de datos, el valor predeterminado para transferencia previa puede ser utilizado; si el mismo número de vector es especificado más de una vez, el dato en el último vector reemplaza todos los datos contenidos en el vector previo con ese número. Esto permite que el conjunto de vectores de prueba sea modificado o "*parchado*" sin transferencia del conjunto completo.

Secuencia de pines

El contenido de condiciones en los vectores de prueba es aplicado a los pines del dispositivo en orden numérico de izquierda a derecha progresivamente antepuestos por el campo **P**.
El tiempo de secuencia no es definido, pero la condición de prueba puede ser aplicada al pin 5 antes o después del pin 4.

```
P 1  2  3  4  5  6  7  8  9  10  11  12  13  18  19  20 *

V 0001  111000HLHHNNNNNNNNNN *
V 0002  111000HLHHNNNNNNNNNN *
```

Condiciones de prueba

Las condiciones de prueba en niveles lógicos son definidos por el tipo de tecnologías del dispositivo (por ejemplo **TTL,CMOS, ECL**). El **0** y el **1** se aplican como condiciones de prueba a un nivel lógico de estado; el dispositivo probado aplica condiciones de entrada que pueden ser bidireccionales (*input, output*) ; la **X** o "*no importa*" se aplica a niveles definidos por default mediante el campo **X** ; El campo **F** aplica condiciones de prueba para alta impedancia en un pin.

La secuencia de las condiciones de entrada no están definidas pero múltiples vectores pueden ser utilizados cuando la secuencia es importante.

Ejemplo :

```
V01 XX00XXXXXNXXXXXXXXXN *
V02 XX00XXXXXNXXXXXXXXXN *
V03 XX00XXXXXNXXXXXXXXXN *
```

- Las condiciones 2-9 no son estándar para un supervoltaje (*voltaje mayor que 5Vcc*) en el dispositivo. Sin embargo pueden ser utilizadas para accesos especiales en algunos modos de

prueba. Los niveles son definidos por cada dispositivo y los vectores de prueba utilizados para supervoltajes pueden utilizarse para dispositivos que demanden una segunda fuente.

- La condición de prueba **C** aplica un nivel lógico **0** a todas las otras entradas que son estables.
- La condición **C** seguida por **1** a **0** a **1** es una manera similar. Para dispositivos con más de una entrada de reloj.
- La condición **N** es utilizada para pines de encendido y otras salidas no probadas.
- Después todas las entradas deben estar estabilizadas incluyendo el reloj; la prueba de salida es llevada acabo y se toma **L** para un nivel lógico **0** o **H** para un nivel lógico **1**.
- La condición de prueba **Z** se aplica a salidas con condición de alta impedancia.

Registro precargado

El registro precargado significa forzar a que un registro tome un estado conocido. Tres tipos de registros precargados se consideran "*in-circuit*" "*output register*" y "*buried register*".

El registro precargado *in-circuit* está acompañado con una entrada dedicada o un control lógico interno para usos normales de niveles lógicos, la entrada estándar y la prueba de reloj pueden ser utilizadas con este tipo de registros. El *output-register* no son niveles para supervoltaje sin embargo, se tiene acceso en modos especiales a este tipo de registros.

PROGRAMADOR / PROBADOR DE OPCIONES

Fusible de seguridad (G)

El fusible de seguridad certifica los dispositivos lógicos para que puedan ser habilitados durante la programación con el mandato de un **1** al campo **G**. El fusible de seguridad previene la lectura del estado de fusibles.

Asignación de pruebas de análisis (S,R,T)

La asignación de prueba de análisis es especificada por medio de los campos **S**, **R**, y **T**. El campo **S** define un vector listo para la prueba. El estado posible es **0** o **1**. El campo **R** contiene el vector resultante. El campo **T** denota el número de ciclos de prueba que fueron corridos.

Tiempo de acceso (A)

El campo **A** define los retardos de propagación por vectores de prueba en un incremento de un nanosegundo. Este campo puede incluir subcampos opcionales.

Sintáxis.

*< tiempo de acceso > : : = ' A ' { < campo de caracteres > } < número > ' * '*

III.4 METODOLOGIA DE DISEÑO CON LOGICA PROGRAMABLE

Los **PLD** ofrecen un gran número de ventajas al programador y para utilizarlos, el diseñador debe aplicar una metodología específica para maximizarla efectividad de las herramientas disponibles para estos dispositivos. A continuación se describe esta metodología de diseño para lograr alcanzar un mejor desarrollo e implementación del diseño.

Proceso de desarrollo del diseño

El proceso de desarrollo del diseño para **PLD** se lleva a cabo en tres fases (Figura 3.15) :

1. DISEÑO LOGICO. En esta fase se debe seleccionar el dispositivo basado en los requerimientos del diseño: número de pines de entrada /salida, número de términos de producto, salida de registro combinacional, polaridad, consumo de potencia y velocidad.

2. IMPLEMENTACION DEL DISEÑO. Consiste básicamente en la selección y uso de herramientas para traducir los resultados de la primera fase a un **PLD** configurado.

3. VERIFICACION LOGICA DEL DISEÑO. Es la fase final en la cual se "*checa*" la programación correcta del dispositivo, seguido de la generación de procedimientos de prueba, que verifican que la implementación del dispositivo es la requerida.

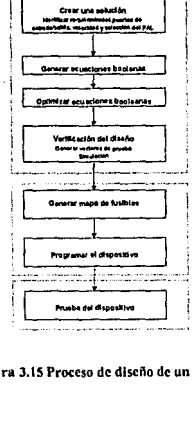


Figura 3.15 Proceso de diseño de un PAL

DISEÑO LOGICO

El procedimiento para llevar a cabo esta fase se describe a continuación.

Definir el problema

Consiste en definir una función lógica en particular que resuelva el problema es el primer paso en la fase de diseño. Resulta natural identificar el problema para el cual una función combinatoria como decodificación de dirección, multiplexaje o generación de señales de control serán la respuesta, también puede ser una función secuencial como: contadores, corrimientos o implementación de una máquina de estado.

Generar un diagrama de bloques

Basados en los criterios anteriores, el diagrama de bloques (*diseño de la lógica*) puede ser generado directamente, como se muestra en la Figura 3.16. Los nombres de las señales se dan para tener una referencia de las funciones que realiza cada una de ellas y para cualquier consideración dentro del contexto del sistema.

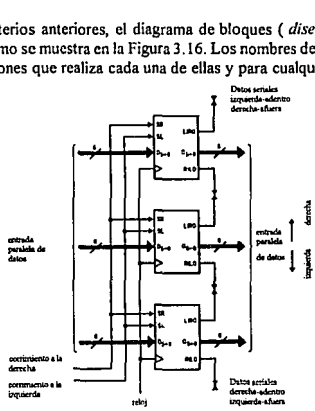


Figura 3.16 Diagrama de bloques para un registro de corrimiento

Función	SL	SR	RILO	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	LIRO
Retén	0	0	Z	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Z
Corrimiento a la derecha	0	1	RI	RI	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
Corrimiento a la izquierda	1	0	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	LI	LI
Llamado en paralelo	1	1	z	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Z

Figura 3.17 tabla de funciones que especifica la operación de un registro de corrimiento de 6 bits

En el diagrama de bloques, se describe con detalle el funcionamiento de lo que se quiere hacer. Puede tomar una serie de formas, dependiendo de la aplicación y de la preferencia del diseñador. Un método común para expresar con detalle la operatividad de esta aplicación con registros, es una tabla de funciones, como la que se muestra en la Figura 3.17.

La función principal puede definirse adicionalmente a través de un esquema lógico detallado (Figura 3.18), de una tabla de verdad combinacional, o por medio de expresiones directas en ecuaciones booleanas.

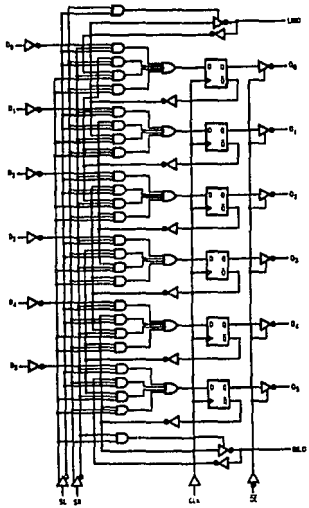


Figura 3.18 Diagrama lógico o esquemático de un registro de corrimiento

Obtener las ecuaciones Booleanas

Para poder proporcionar una definición del circuito que pueda ser manejada por las herramientas de desarrollo que se van a utilizar, es necesario expresar el diseño en términos de ecuaciones Booleanas, las cuales forman la función de transferencia principal de un dispositivo tipo **PAL** con la suma de productos, en contraste con la de un inversor *De Morgan* cuyos términos se expresan en producto de sumas. Las ecuaciones lógicas pueden ser derivadas directamente de la tabla de funciones mostrada en la Figura 3.16, del esquema lógico mostrado en la figura 3.18, o con algún otro método de implementación lógica.

Para cualquier salida tipo Tres-Estados (**TRI-STATE**) incluyendo líneas de Entrada /Salida (**Input /Output**) bidireccionales, es posible que sea necesario especificar las ecuaciones que definan las funciones de control de esas líneas.

- Minimización de la lógica

Generalmente es muy práctico minimizar las ecuaciones lógicas para eliminar cualquier variable extraña o mintérminos redundantes innecesarios. Si no se realiza esta operación no se altera la funcionalidad del dispositivo, sin embargo, podría resultar un diseño que requiera más recursos de los disponibles en un dispositivo determinado, el cual podría satisfacer las necesidades si se redujeran las ecuaciones. Adicionalmente las compuertas y nodos redundantes e innecesarios podrían provocar que no pudiera verificarse el dispositivo programado.

Por otro lado, el empleo intencionado de términos redundantes puede ser un método conveniente para evitar varianzas en funciones lógicas combinacionales (*asincronas*).

IMPLEMENTACION

Seleccionar el dispositivo

Una vez que el diseño lógico está definido, se necesita un **PLD** que pueda acomodarse a la lógica requerida. Algunas herramientas de diseño, como lo es el software **PLAN**, nos proporcionan esta función automáticamente. De otra manera, la selección manual debe hacerse con los dispositivos que se encuentran disponibles.

Algunos criterios de selección son:

- El primer criterio a considerar es el tipo de familia requerido. Los **PLD** vienen en una gran variedad de tecnologías y velocidades y ofrecen un espectro de categorías posibles del dispositivo a seleccionar. Si el dispositivo requiere compatibilidad con un **ECL**, un **CMOS** de baja potencia o a una velocidad muy alta, se limita la selección del dispositivo a uno disponible en esa categoría.

- Una vez que la familia ha sido seleccionada, se debe examinar el diagrama de bloques, la tabla de funciones y las ecuaciones lógicas. Basados en esto, se establecen los siguientes parámetros:

  Número de salidas con registros
  Número de salidas combinacionales
  Número de entradas
  Requerimientos de reloj
  Complejidad de cada ecuación lógica (*número de mintérminos requeridos*)

Adecuar la lógica al dispositivo

Si la lógica es muy complicada para acomodarse dentro de un solo **PLD**, entonces el diseño debe de ser modular en cascada. En este punto se decide el dispositivo, considerando los objetivos del sistema.

Introducir las ecuaciones al software

Antes de proceder, el software que se vaya a utilizar tiene que correr bajo la plataforma seleccionada. Esto normalmente implica la instalación de un ensamblador como lo es el **PLAN** en una **PC** (*Personal Computer*: Computadora Personal) y su preparación para la entrada de la información, en la forma que se requiere.

Para llevarse a cabo, las ecuaciones derivadas anteriormente deben alimentarse al software seleccionado., algunos paquetes de alto nivel proporcionan una interfase muy sofisticada para hacer esto, es por ello que se sugiere utilizar el **PLAN** como un software fácil de usar (existen otros similares como el **PALASM**) . El ensamblador **PLAN** acepta archivos de texto creados con un editor común **(ASCII)** como un archivo de entrada. En todos los casos deberán seguirse algunas reglas de sintáxis.

El método para la entrada de datos al **PLAN**, requiere la conversión de las ecuaciones a la sintáxis del paquete (Figura 3.19)

```
Title     6-Bit cascadable shift register
Pattern   6SHFT
revision A
author    Ing F Engineer
company   National Semiconductor Corporation
Date      11/29/1989

chip 6SHIFT PAL16L6

; pin 1     2     3     4     5     6     7     8     9     10
        CLK   SR    D0    D1    D2    D3    D4    D5    SL    GND
; pin 11    12    13    14    15    16    17    18    19    20
        /G    RILO  Q5    Q4    Q3    Q2    Q1    Q0    LIN0  VCC

equations

        /Q0 := /Q0 * /SR * /SL
             + /Q1 *  SR * /SL
             + /LIN0 * /SR * SL
             +  /D0 *  SR * SL
        /Q1 := /Q1 * /SR * /SL
             + /Q2 *  SR * /SL
             + /Q0 * /SR * SL
             +  /D1 *  SR * SL
        /Q2 := /Q2 * /SR * /SL
             + /Q3 *  SR * /SL
             + /Q1 * /SR * SL
             +  /D2 *  SR * SL
        /Q3 := /Q3 * /SR * /SL
             + /Q4 *  SR * /SL
             + /Q2 * /SR * SL
             +  /D3 *  SR * SL
        /Q4 := /Q4 * /SR * /SL
             + /Q5 *  SR * /SL
             + /Q3 * /SR * SL
             +  /D4 *  SR * SL
        /Q5 := /Q5 * /SR * /SL
             + /RILO *  SR * /SL
             + /Q4 * /SR * SL
             +  /D5 *  SR * SL
        /LIN0 = /Q0
        LIN0.TRST = SR * /SL
        /RILO = /Q5
        RILO.TRST = /SR * SL

; end of 6SHIFT
```

```
Document file for 6SHFT.LNP
Device: 16L6

Pin   Label           Type
---   -----           ----
1     CLK             clock pin
2     SR              com input
3     D0              com input
4     D1              com input
5     D2              com input
6     D3              com input
7     D4              com input
8     D5              com input
9     SL              com input
10    GND             ground pin
11    G               enable pin
12    RILO            reg,tred,com feedback
13    Q5              reg,reg feedback
14    Q4              reg,reg feedback
15    Q3              reg,reg feedback
16    Q2              reg,reg feedback
17    Q1              reg,reg feedback
18    Q0              reg,reg feedback
19    LIN0            reg,trst,com feedback
20    VCC             power pin
```

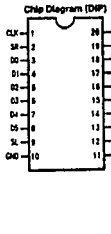


Figura 3.19. Modelo para introducir ecuaciones booleanas al software PLAN

Generación del código JEDEC (compilación)

En esta fase, se hace la asignación de los pines, ya sea en forma manual o automática con la ayuda del software. **PLAN** nos ofrece un asignación automática, la cual puede ser editada manualmente si se desea. Una vez que el archivo de ecuaciones ha sido introducido y se han asignado los pines, el ensamble es ejecutado, resultando un archivo **JEDEC**, que es un mapa de celdas a ser programadas y que alimenta al programador de **PLD**.

Para que las ecuaciones puedan ser convertidas en un patrón de bits, se necesita que las ecuaciones iniciales sean convertidas a un formato conocido como un archivo **JEDEC**. El archivo **JEDEC** es un archivo aceptado ampliamente por todos los diferentes fabricantes de programadores. Consiste en una tabla formateada indicando todas las celdas (*fusibles*) a ser programadas en el **PLD**, para implementar las funciones lógicas especificadas. Los detalles de la operación del ensamblador varían de un paquete a otro, pero cada uno de ellos hace una verificación de sintáxis así como una evaluación de como puede implementar el diseño en el dispositivo seleccionado y generar el archivo **JEDEC**, mismo que es almacenado en un dispositivo listo para ser cargado al programador.

El único propósito de la generación del archivo **JEDEC** es la de establecer una interfase uniforme entre las herramientas comerciales de desarrollo para **PLD**, los programas de desarrollo (*software*) y los programadores universales.

Configurar el programador

Antes de que el patrón de celdas de datos pueda ser transferido, el programador necesita ser conectado a la plataforma de desarrollo y provisto del socket (*enchufe*) apropiado para acomodar el dispositivo virgen a ser programado (**PAL**, **GAL**, *etc.*).

Para realizar esta tarea, se conecta el sistema programador a la plataforma (*base con sockets que sostiene al dispositivo*) a través de un cable **RS232C**, de acuerdo a la documentación del sistema.

Transferir el archivo JEDEC

En este paso, se realiza la transferencia del archivo **JEDEC** a través de una liga de comunicación de la plataforma de desarrollo al programador universal. Cada programador difiere ligeramente uno de otro, requiriendo normalmente en todos los casos la introducción del nombre del archivo, tipo de dispositivo, fabricante, etc. Por lo general, se corre una prueba al dispositivo desde el programador, para asegurar la correcta orientación (*colocación*) del mismo en el socket y si en realidad está en blanco y listo para ser programado.

Programación del dispositivo

Una vez que se le ha proporcionado toda la información necesaria al programador se lleva a cabo la programación del dispositivo, el programador traduce el archivo **JEDEC** en direcciones, patrones de datos y pulsos programados, que se aplican a los pines del dispositivo en el socket, configurando las celdas como un patrón, el cual hará que el dispositivo opere de acuerdo al diseño original. Una vez finalizada la programación, la implementación del diseño en el **PLD** queda terminada.

VERIFICACION

La verificación se hace no sólo para asegurarse de que el dispositivo ha sido configurado correctamente, sino también para saber si el programador universal ha trabajado correctamente y si el diseño original no fue modificado.

Verificación del dispositivo

Esta tarea la lleva a cabo automáticamente el programador universal, de no ser así, se recomienda una verificación manual cuando el dispositivo se encuentre todavía en el programador para asegurarse que el patrón corresponde al mapa de fusión en el archivo **JEDEC**. El programador lee directamente el patrón desde el **PLD**, y lo compara directamente con el archivo **JEDEC** cargado.

Generación de vectores de prueba

Algunos paquetes avanzados generan automáticamente el vector de pruebas con las ecuaciones, de otra forma, los vectores de prueba deben generarse manualmente. Aún cuando se haga la generación automática de vectores, algunos diseñadores prefieren agregar sus propios vectores para verificar cada operación específica de la aplicación.

Simulación del dispositivo

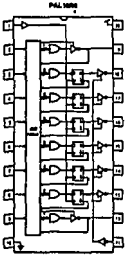
Este paso opcional genera los vectores de salida del dispositivo, los cuales permiten la verificación correcta operación del diseño. Esto pude ser realizado manualmente con la ayuda de un módulo simulador de algún paquete, esto es, para predecir la configuración de salida del dispositivo basado en el modelo original, que se introdujo en forma de ecuaciones Booleanas.

Los vectores de salida del simulador, deben ser examinados para confirmar que el modelo opere correctamente. El sistema proporciona también los estados de salida, que deben acompañar los vectores de prueba para la prueba funcional del dispositivo.

Prueba funcional

El dispositivo debe ser evaluado totalmente para un correcto funcionamiento de la función deseada.
Dependiendo del software que se esté utilizando, se introducen los vectores de prueba y de esta manera lo convierte en un formato propio del archivo **JEDEC**. La generación ó entrada del vector de pruebas normalmente se realiza a continuación de la entrada de la ecuaciones. Después, de la programación del dispositivo y verificación del patrón, el programador lleva a cabo la prueba funcional del **PLD** mientras permanece en el socket. Las señales del vector de entrada son aplicadas a los pines del dispositivo en el modo de operación normal y las señales de salida del dispositivo son comparadas con los vectores de salida esperados.

Como paso final, la mayoría de los **PLD** incluyen "*una celda de fusión de seguridad*" que cuando se programa, no permiten la reprogramación de los patrones lógicos de los circuitos.



CAPITULO IV

DISEÑO DEL SISTEMA SIMULADOR DE PAL

Ya hemos estudiado cómo se genera el archivo de mapa de fusibles en formato JEDEC, sin embargo falta mencionar la forma en como el código es traducido para obtener la información necesaria y realizar una simulación del diseño. En este capitulo nos introduciremos en las técnicas de simulación hasta llegar a la respuesta de operación del circuito secuencial debido a una señal de entrada.

Es necesario mencionar que la forma de traducción de ecuaciones Booleanas al código JEDEC presenta diferencias en el software utilizado de algunas compañias, como *Motorola, National y Texas Instruments.*

IV.1 LAS TECNICAS DE SIMULACION

Introducción

La definición simulación en un diccionario es más o menos cercana a la que a continuación se menciona "hacer una réplica del trabajo de máquinas para demostración o por análisis del problema". Sin embargo, en el contexto del diseño digital los modelos son creados, utilizando modelados matemáticos y con una computadora se investiga la operación de un sistema digital, de cualquiera de las siguientes formas:

a) verificando la operación lógica
b) accesando la información del desempeño en relación a la especificación
c) investigando la operación bajo condiciones predeterminadas

La simulación ha sido utilizada por un largo tiempo en el diseño de sistemas digitales, pero la compatibilidad con el modelo de hardware, asi como la construcción misma, han tenido una atracción natural hacia el diseño en ingeniería. Con el incremento en la utilización de circuitos integrados de diseño semicompleto, los diseñadores pueden trabajar con dispositivos de entradas intermitentes o con pulsos de reloj.

NIVELES DE SIMULACION

Para algunos sistemas digitales existen muchos niveles de sofisticación que pueden ser adoptados por el simulador, para que el diseñador pueda abarcar diferentes niveles de detalle que sean modelados y diseñados conociendo sus ventajas y desventajas.

El costo puede incrementarse cuando se suman niveles de detalle; en un nivel alto de simulación podemos considerar particiones de un número de sub bloques, interconectados mediante algunos de datos, en un nivel bajo de simulación los subbloques se descomponen en una misma estructura funcional, donde típicamente puede ser estructurado por las trayectorias de los datos y bloques lógicos para unidades aritméticas, registros, etc. Un nivel más abajo encontramos que los registros y bloques combinacionales pueden expandirse a sus equivalentes de compuertas, este tipo de simulador es llamado simulador lógico. El último nivel de simulación parte de considerar que el simulador es un circuito, como un sistema de variables que reciben voltajes y corrientes para responder a excitaciones.

TECNICAS EMPLEADAS EN SIMULACION LOGICA

Los modelos digitales pueden ser construídos por hardware o software o bien, por ambos, los modelos de hardware pueden requerir de modificaciones al construirse para ser adaptados al modelo, tomando características comúnes que en la práctica algunos semiconductores ofrezcan para arreglos de compuertas que requieren elementos TTL en el diseño, algunos de los simuladores disponibles son basados en la especificación del sistema digital para ser codificado con algunas formas de entrada al sistema simulador, la especificación de las entradas y el procesamiento del simulador se produce generalmente por software; básicamente existen dos métodos principales de operación de los simuladores, aquellos referidos al compilador de código, y los manejadores de tablas.

SIMULADORES DE COMPILACION DE CODIGO

En este método de operación, la especificación del diseño es trasladada a la máquina de operaciones del procesador donde la simulación es llevada a cabo por compuertas simples, y se solicitan subrutinas que representen las operaciones adicionales.

SIMULADOR DE MANEJO DE TABLAS

Con este tipo de simulador, la descripción del circuito bajo consideraciones, es trasladado por medio de estructuras de datos, que representan las conexiones y estados del sistema que está siendo simulado. La acción de la simulación es transformada mediante un programa que accesa los elementos apropiados de las estructuras y produce nuevos sistemas que pueden ser reescritos en una estructura apropiada. Este método de simulación tiene ventajas considerables en cuanto a los modelos complejos, iteraciones y compuertas que pueden modelarse de manera más real.

MODELO DE TIEMPOS

El tiempo es cuantificado en una simulación digital, las señales del sistema pueden tener algunos cambios en sus intervalos de tiempo. El acotamiento del tiempo de simulación es relativamente cerrado en cuanto se adopta un tiempo, sin embargo existen muchos simuladores con gran variedad de modelos.

El método es muy simple si se toma en cuenta que los retardos son nulos, lo cual implica que la progagación por retardo a través de una compuerta es cero.. Esto significa que el simulador evalúa simplemente las ecuaciones lógicas del sistema, la demás información no es accesada o no es importante para el tiempo que se simula, también se dice que el simulador puede no detectar malos funcionamientos en el dominio del tiempo.

La unidad de retardo es más real si se asume que las compuertas tienen un retardo diferente de cero, pero el tiempo se debe asumir igual para todos los dispositivos. En esta dirección, se pueden habilitar retardos de tiempo y diagnosticar malos funcionamientos pero, el modelo no es simple y los resultados deben interpretarse cuidadosamente por su complejidad; si se intenta hacer un modelo de tiempo real y enumerar diferentes factores se debe tener en cuenta: el retardo de propagación en cada compuerta que puede ser afectado debido a la compuerta y las capacitancias de los cables. Aún así, es importante tomar en cuenta las siguientes consideraciones:

- En una compuerta real, para el pulso mínimo no debe existir salida.
- Para cada entrada es posible calcular el tiempo que tomará llegar a la señal el camino de salida.
- Los datos deben describir futuras transiciones para el chequeo del tiempo en cuanto halla un cambio en el estado del sistema simulador.

MODELO DE SEÑALES

El modelo de señales es simple para evaluar de modelos de estados de los nodos en un circuito, la principal desventaja es la inicialización donde cada nodo tiene una señal con un valor inicial y los demás valores pueden ser o no cero, y caer entonces en inconsistencias lógicas. Un avance de estos modelos es

utilizar un tercer valor como modelo de señal, con la instrucción de un valor no conocido llamado "X". Este resuelve el problema de inicialización para habilitar la simulación mediante caminos razonables, sin condiciones de carrera y las intermitencias.

El siguiente nivel consiste en adoptar el modelo de cinco valores, considerando los estados con los valores de verdad (0,1), un rizo, una transición y una condición indefinida, estos valores posteriormente son utilizados como inicialización siempre y cuando el resultado de la operación de los campos sea una condición de intermitencia (*hazzard*).

OPERACION DEL SIMULADOR

La descripción de un circuito puede ser una tabla que contenga información de las funciones de la compuerta, o bien, otra tabla puede contener los valores actuales para cada nodo en el sistema. Por cada tipo de compuerta en la simulación, habrá una tabla que almacene los detalles del número de entradas y salidas, y de los valores de rizo, y retardos de tiempo, finalmente deberá existir un proceso donde las primeras entradas del sistema hagan unas transiciones que causen cambios en las salidas y pueda servir como un avance de tiempo.

SIMULACION HIBRIDA

Hemos podido apreciar como la computación ha transformado los modelos de respuesta a un sistema digital con un cambio único de entrada, es por eso que no nos sorprende que una simulación de circuitos de alta escala de integración tome un tiempo apreciable para las transiciones de entrada que llevar algunas horas para una simulación completa. En algunas áreas el tiempo es un factor limitante para el diseño y los recursos de cómputo representan un alto costo para tratar de reducir el proceso, sin embargo, en pocos años se ha incrementado la simulación híbrida para reducir los tiempos que requiere la simulación de un circuito de gran escala.

El hardware también tiene un propósito especial mediante interconexiones y funciones lógicas que puede acelerar el almacenamiento en memoria, los valores de la señal para cada nodo son almacenados en el área de memoria. Mientras se efectúa la simulación de un sistema, el hardware ejecuta una serie de comandos, cada comando normalmente es equivalente a una compuerta que escribe un resultado en memoria, cada compuerta es una dirección específica que será operada de acuerdo a la función evaluada. Cuando el siguiente ciclo de operación es modelado, entonces el resultado de las operaciones anteriores es almacenado para formar el primer operando de una nueva función.

SIMULACION DE UN CIRCUITO

Cuando se implantan sistemas digitales utilizando familias lógicas (TTL, CMOS u otras) para funciones entonces se necesitan facilidades para obtener ventajas en el diseño de tipo semicompleto, debido a las técnicas para estos diseños que indican la necesitad de investigar la operación interna de un simulador lógico que puede ser adaptado.

Los simuladores de circuitos son representados mediante ecuaciones que determinan las corrientes y los voltajes en los nodos. Los componentes pasivos (resistencias, capacitancias e inductancias) pueden estar contenidos en modelos de dispositivos activos (diodos, transistores, amplificadores operacionales), entonces apreciamos que es posible incorporar un modelo de información sobre otros dispositivos y tratar

el modelo por capas. Esto quiere decir que es posible modelar por partes un diseño de un circuito de niveles, el problema consiste en cómo definir las secuencias de entrada válidas para habilitar el circuito de un modo real y poder efectuar una operación en la simulación.

SELECCION DE UN SIMULADOR DISPONIBLE

No es posible explicar en un corto espacio el funcionamiento detallado de los simuladores, sin embargo, explicaremos brevemente algunos de los utilizados comúnmente para simuladores lógicos y simuladores de circuitos.

HARTRAN (1): Desarrollado por GEC Hirst Research Laboratory. Es un compilador capaz de admitir simulaciones de reloj y sistemas síncronos. La estructura del lenguaje es similar al FORTRAN y puede ser montado en máquinas pequeñas.

HILO (2). Es un desarrollo de Cirrus Computers Limited & Brunel University. El simulador soporta muchas primitivas lógicas y un gran número de interconexiones.

DIGSIM (3). Elaborado por SCANIA-SAAB & Linkopping University. Es un simulador estocástico para obtener transiciones de señales de tiempo.

SPICE (4). Inventado por la Univesidad de Berkeley California/ Es un simulador de circuitos disponible para muchos tipos de computadoras. Se basa en la descripción de nodos.

IV.2 ALGORITMO PARA SIMULACION DE DISPOSITIVOS PAL CON 20 PINES

CONSIDERACIONES PARA LA FORMULACION DEL ALGORITMO

El tema central de la tesis es la simulación de los **PAL** de 20 pines, es decir, el algoritmo cubre solamente los dispositivos **PAL16L8, PAL16R4, PAL16R6** y **PAL16R8**. Atendiendo a los casos particulares dentro de una gran diversidad de aplicaciones que pueden ser implementados en sus arquitecturas.

A pesar de que la información sobre simulación de dispositivos electrónicos puede llegar a ser abundante, nuestro caso requiere un tipo diferente para poder llevar a cabo su simulación, en otras palabras, se presentaron los modelos de simulación existentes en la literatura acorde al tema, pero ninguno de los modelos mencionados anteriormente nos sirve por completo para la simulación del **PAL**, en cambio, por separado proporcionan algunos elementos para conjuntar ideas que nos relacionen una directriz hacia la solución. Es por esta razón que no se amplió mucho la discusión de los diversos modelos de simulación, sino que se propuso un escudriñamiento de los tópicos respectivos.

El algoritmo propuesto tendrá módulos que simulen por separado cada una de las partes integrantes de la arquitectura del dispositivo de estudio, es decir, se requieren módulos de programas que semejen los siguientes puntos:

- Módulo de pines de entrada
- Módulo de almacenamiento de fusibles
- Módulo de lectura de código JEDEC para almacenarse en el arreglo OR.
- Módulo de simulación del arreglo OR
- Módulo de simulación del arreglo AND
- Módulo de simulación del buffer tres-estados
- Módulo de simulación del flip-flop "D"
- Módulo de pines entrada/salida
- Módulo de pines de salida
- Módulo de simulación gráfica de señales.

Debido a la estructura modular del sistema, puede observarse que ninguno de los modelos expuestos en el capítulo IV.1 se adapta completamente a las características que se desean simular, sin embargo, sirven como marco de referencia para las consideraciones siguientes:

Se deben inicializar los flip-flop's con un estado conocido, que se marcará como predeterminado o por default. La simulación de los pines de entrada, así como los pines de salida, no considera retardos de propagación, márgen de ruido, ni alteración de la señal por alta frecuencia, debido a que pueden consultarse estas especificaciones en las hojas de datos técnicos del fabricante del dispositivo.

Tal vez, en un desarrollo futuro, implementado otro modelo de simulación, que por el momento se sale del objetivo de esta tesis, sea anexado a la estructura del sistema los puntos anteriores, con sus respectivas consideraciones.

Nuestro algoritmo no se considera como un modelo de simulación de circuito puesto que no existen resitencias, ni capacitores, ni transistores u otros dispositivos pasivos/activos que nos lleven a pensar en un modelos de simulación matemática; más adecuado resulta pensar que se simular circuitos electrónicos con ciertas propiedades no matemáticas sino propiamente descriptivas del funcionamiento del dispositivo.

En otras palabras, podemos decir que este simulador es descriptivo del funcionamiento de los **PAL**, que puede ser manejado, en un momento dado, como una caja negra donde se reciben estímulos digitales y nos entrega, ya sea una simulación de señales digitales en forma gráfica o bien en forma numérica.

No se consideran circuitos asíncronos, dado que se considera únicamente una señal de reloj por dispositivo (normalmente en el pin número uno está destinado para tal función).

Como consecuencia del número de variables que puede admitir en su combinación de lógica negada, es decir, 8 mintérminos con 8 ecuaciones en el caso extremo, sin garantizar un buen funcionamiento, el número máximo de estados permisibles por el sistema serán 120, debido al funcionamiento interno del manejo de memoria del ambiente Windows, teniendo en consideración que estarán presentes en memoria mientras se efectúa la simulación gráfica, o mientras se almacena el resultado numérico de la simulación.

FORMALIZACION DEL ALGORITMO

El algoritmo propuesto funciona de acuerdo a la siguiente estructura, adaptándose al tipo de **PAL**:

Identificación del dispositivo PAL que será simulado

Inicializa el número de flip-flop's

El **PAL16L8** no tiene flip-flop's, mientras que el **PAL16R4** tiene 4, el **PAL16R6** cuenta con 6 y el **PAL16R8** tiene 8 salidas con registro y no hay entradas bidireccionales.

Lectura de archivo de tabla de estados (un renglón en ASCII equivale a un ciclo de reloj)

El archivo de tabla de estados deberá contener solamente 1's y 0's, y lee uno a la vez de acuerdo a los pines de entrada de cada dispositivo particular.

Lectura del archivo con el código JEDEC que será almacenado en el arreglo OR.

El arreglo OR queda definido como una estructura de 2048 fusibles (máximo), dividido en 32 columnas por 64 renglones.

Simulación del campo programable OR

Una vez almacenado el código JEDEC en la estructura de datos, se comparan las entradas de los pines y se realiza la operación lógica OR por cada uno de los 32 columnas de fusibles que se supone, han sido programados por las funciones booleanas que los mapean, el resultado es almacenado en otro arreglo. De aquí saldrán 8 resultados por salida, es decir en total se efectuarán 64 resultados como combinación de sus respectivas columnas de fusibles. Se asume que cada 8 renglones mapean una salida. En caso de que exista salida con buffer tipo tres-estados, se sensa primero la ecuación correspondiente al primero de los 8 renglones consecutivos, para saber si el buffer está habilitado o deshabilitado. Los otros siete renglones son enviados al arreglo AND con sus respectivos resultados

Cuando se presenta un **PAL** que contiene registros y salidas con buffer tres estados juuntos, primero se realiza la operación OR sobre los arreglos que tienen salidas de registros para actualizar el estado siguiente del flip-flop, y posteriormente, se llevan a cabo los arreglos asociados con la salida tipo buffer. De no ser realizado así, tenemos un retraso de un ciclo de reloj y las salidas simuladas serán falsas, lo que nos llevaría a pensar que el diseño no es correcto. En consecuencia, si el **PAL** sólamente tiene registros de salida o puros buffer tres-estados, la operación se realiza simultáneamente como si fueran salidas paralelas.

Simulación del campo no programable AND

Se reciben como entradas, 8 renglones ordenados para efectuar la operación lógica AND, el resultado será enviado, ya sea, a un buffer de salida tipo tres-estados o bien a un registro tipo "D".

Simulación de las estructuras de registro o flip-flop tipo "D"

La estructura definida para simular a este dispositivo deberá contener los siguientes campos:

Estado Q presente
Estado Q siguiente
Estado /Q siguiente
Estado D.

La actualización de esta estructura esta condicionada de acuerdo a la variación del estado D, así, por ejemplo, si después de realizarse la operación AND el resultado es "1", este es trasladado al flip-flop, la entrada D tiene un "1" y por lo tanto Q siguiente cambiará a uno si Q presente (estado anterior) tenía el valor de "0", o se conservará en "1" si Q presente valía "1", pero /Q siguiente tomará el valor opuesto al resultado de Q.

Estructura de simulación para un inversor.

Esta estructura se encargará de invertir el resultado que le sea mandado como parámetro siempre que sea necesario o que se encuentre el elemento "INVERSOR".

Estructura de simulación para un buffer tres-estados.

Básicamente se encarga de revisar si el renglón asignado a la ecuación del buffer está activa. Si el valor de la operación OR de renglón para la ecuación del buffer es "1", entonces el resultado de la operación AND es anulado por considerarse en condición de "Alta Impedancia", si por el contrario, el resultado de la ecuación es "0", el resultado AND simplemente es invertido para ser destinado a un pin de salida.

Implementación de rutina para almacenamiento de información en un archivo.

El procedimiento se encargará de actualizar los parámetros necesarios para el ambiente de Windows y concatenará la información para mandarla hacia un archivo respetando los protocolos de almacenamiento de información y también de darle formato accesible de comprender.

Implementación de rutina para mapeo gráfico de la señal de salida

La rutina se encargará de actualizar los controladores y adecuar el ambiente gráfico por medio de instrucciones para el uso del GDI propio de Windows, para que se pueda desplegar la información numérica en forma de pulsos digitales mostrando el funcionamiento de la simulación..

CAPITULO V

PROGRAMACION DEL SISTEMA SIMULADOR

La implementación de las herramientas de diseño en sistemas amigables, mediante el uso de interfases gráficas en computadora, hacen que la unión de software y hardware faciliten las tareas de desarrollo (el entorno gráfico *Microsoft Windows* hace más sencilla la tarea del usuario).

Con la finalidad de automatizar el proceso de diseño, se desarrolló el sistema simulador en el entorno gráfico *Microsoft Windows con Borland C++* ya que cuenta con características que reducen la curva de aprendizaje mediante el uso de interfases de usuario familiares. Ambas plataformas (Windows y Borland) se describen en este capítulo.

Partiremos de una visión de Windows para introducirnos al desarrollo de la aplicación y llegar al código fuente necesario para construir y ejecutar el sistema.

V.1 PROGRAMACION EN WINDOWS

UNA VISION DE WINDOWS

A finales de los 60's, la compañía Xerox fundó el **PARC** (*Palo Alto Research Center* : Centro de Investigaciones de Palo Alto) en California, Estados Unidos, cuyo objetivo fué y sigue siendo, el de crear nuevas tecnologías en software y hardware. En los primeros días en que inició sus labores el **PARC**, su equipo contaba con terminales que permitían una mejor interacción hacia el usuario o el programador; pero para la comunicación, el operador de computadoras debía aprender los complejos comandos del sistema operativo y de los compiladores que existían dentro de un sistema cómputo.

El **PARC** atacó este problema desarrollando la filosofía **WYSIWYG** pronúnciese Wiz-ee-wig (*What You See Is What You Get* : Lo Que Tú Ves es lo Que Tú Obtienes) que realiza un mapeo de coordenadas de un dispositivo físico al área de visualización por medio de la ecuación de conversión Ventana-Puerto:

$$\frac{Vmax - Vmin}{Xmax - Xmin}$$

La base de esta Filosofía es "*una imágen dice más que mil palabras*"; por lo que se fijó el objetivo de crear sistemas de software usando símbolos que significaran comandos y procesos. Estos símbolos no necesitaban ser parte de un idioma, podrían ser señales de tránsito o figuras de objetos, lo cual implicó el desarrollo de hardware para permitir gráficos complejos. A este sistema de software se le llamó **GUI** (*Graphic User Interface* : Interface Gráfica para el Usuario). Entre los primeros desarrollos en hardware, el **PARC** rediseñó el dispositivo conocido como *mouse* (*ratón*), bajo las bases del sistema **GUI**.

En agosto de 1981, La compañía IBM lanzó al mercado una computadora basada en el microprocesador 8086 : la computadora XT que incluía como software el sistema operativo **PC-DOS** (*Personal Computer -Disk Operating System* : Disco de Sistema Operativo Para Computadora Personal), desarrollado por una compañía de software desconocida hasta ese momento : ***Microsoft***

Microsoft había sido fundada en 1975 por Bill Gates y Paul Allen, dos extraordinarios programadores, ambos crearon el **PC DOS** que derrocó al rey de los sistemas operativos a nivel personal de aquél momento : **CP/M** (*Sistema operativo para computadoras basadas en 8086*). A pocas semanas de ser lanzada comercialmente la XT, Microsoft firmó un contrato, casi en exclusividad con IBM para el desarrollo de cualquier proyecto relacionado con sus computadoras personales.

En noviembre de 1985, IBM y Microsoft anunciaron la salida comercial de la *versión 1.01* de Windows que causó entusiasmo al utilizar ventanas y menús controlados por el mouse, sin embargo, en Windows 1.01 no existía ninguna forma viable para desarrollar software en este entorno gráfico.

Con la idea de tener compatibilidad en periféricos sin importar el tamaño del bus de datos a usar, IBM anunció el desarrollo de un nuevo sistema operativo compatible con cualquier sistema de cómputo, que permitiera el uso normal de la memoria RAM de cualquier tamaño, implementar multiprocesamiento y llevar a cabo el control de acceso a red. Este sistema sería realizado en conjunto por IBM y Microsoft

y su nombre sería **OS/2** (*Operating System/2* : Sistema Operativo 2). Sin embargo para lograr el pleno desarrollo del OS/2; Microsoft debía primero lanzar su nueva versión de Windows (*Windows versión 2.0*).

El lazo entre Windows 2 y OS/2 se debe a la introducción de las normas **SAA** (*Systems Application Architecture* : Arquitectura de Aplicación de Sistemas) y **CUA** (*Common User Acces* : Acceso Común al Usuario). La **SAA** permite la comunicación entre las computadoras personales tipo IBM con mainframes, en tanto que el **CUA** marca el establecimiento de bibliotecas conteniendo funciones y estructuras comunes entre los sistemas para permitir una estandarización entre el diseño y programación de sistemas.

Microsoft lanzó *Windows 3.0* y *MS-DOS versión 5.0* en 1991. En ambos sistemas se presentaba nuevas caracteristicas para usarlas en modo real y protegido, al mismo tiempo que aparecen nuevas versiones de OS/2 también hicieron su aparición, aunque no con mucha aceptación. De hecho la poca aceptación de OS/2 motivó a Microsoft a generar la *versión 3.1* de Windows en menos de un año de haber aparecido la *versión 3.0*.

AMBIENTE EN WINDOWS

Ventana

Una ventana es una zona reservada de trabajo del usuario. Puede contener información relacionada a procesos y documentación referente a otras ventanas de trabajo, en la siguiente figura se describen los elementos que sirven de herramientas de trabajo y la aplicación dentro de la ventana.

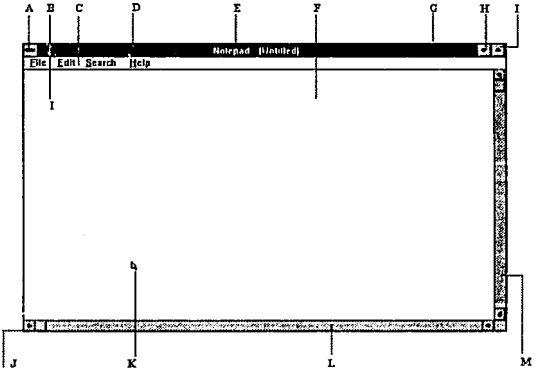


Figura 5.1 Ventana típica de Windows

A Cuadro del menú Control
B Punto de inserción
C Barra de menús
D Barra de Título
E Título de la ventana
F Area de trabajo
G Borde de la ventana
H Botón "*Minimizar*"
I Botón "*Maximizar*"
J Esquina de la ventana
K Puntero del Mouse
L Barra de desplazamiento horizontal
M Barra de desplazamiento vertical

• El *cuadro del menú Control* también llamado menú Sistema; resulta de mayor utilidad cuando se utiliza el teclado para mover, maximizar, minimizar, cerrar ventanas y modificar su tamaño, así como cambiar a otras aplicaciones.

• La *barra de título* muestra el nombre de la aplicación o documento.

• El *título de la ventana*, puede ser el nombre de la aplicación y el nombre de un documento, o bien el nombre de un grupo, un directorio o un archivo de datos.

• En la *barra de menús* se muestran los menús disponibles. Un menú contiene una lista de comandos o acciones que pueden realizarse con Windows.

• Las *barras de desplazamiento* permiten mostrar en pantalla partes de un documento, cuando el documento completo no cabe en una sola ventana.

• El *Botón Maximizar* es para ampliar la ventana de la aplicación activa de tal modo que ocupe toda la pantalla, o bien seleccionar el botón Minimizar para reducir la ventana a un icono.

• El *punto de inserción* muestra el lugar en el que se encuentra dentro del área de trabajo, marcando el lugar en el cual aparecerá el texto ó gráficos.

Iconos

Un icono es un pequeño gráfico que representa a un elemento de Windows (*para una definición general ver apéndice B*). En el sistema de simulación existen varios tipos de iconos generalmente representados de acuerdo a la tarea que realizan.

CARACTERISTICAS ESPECIALES DEL PROGRAMA

Esta tesis ha sido diseñada para que pueda pasar a formar parte de sus herramientas de diseño, análisis y simulación, en un contexto donde emerge la **computadora** como herramienta del diseño, teniendo aplicación fundamental en la **simulación**.

Formato del código fuente

En el desarrollo del sistema se involucraron varios archivos con su respectivo indicador de código mismos que a continuación se describen:

Indicador de código. Los indicadores de código, como se muestran en la Figura 5.2, sirven para identificar cada uno de los archivos que componen el programa. La extensión **.DEF** identifica el archivo de definición de módulo. La extensión **.H** se coloca junto al archivo de cabecera (*#include*) que contiene los prototipos de las funciones y las definiciones de constantes. La extensión **.RC** se refiere a los archivos de especificación de los recursos que la aplicación utiliza. La extensión **.C** identifica el archivo fuente en C que contiene el punto de entrada a la aplicación y las funciones propias de cada programa.

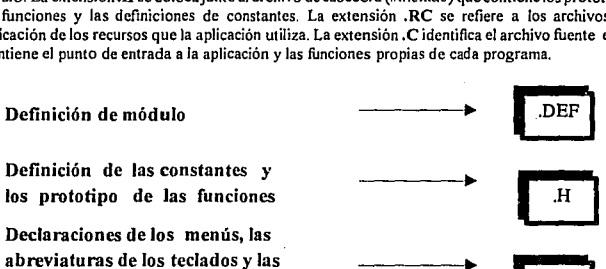


Figura 5.2 Indicadores de código. El sistema simulador se generó a partir de cuatro archivos fuente

Archivos de recursos. Los archivos fuente que manipulan los recursos que Windows pone a disposición de las aplicaciones también siguen un formato consistente. El programa contiene un archivo de especificación de recursos que describe el menú del sistema, las abreviaturas de teclado y las cadenas de texto que la aplicación utiliza al tiempo de ejecución.

Compatibilidad hardware y software

Si en la plataforma que se tiene se puede ejecutar la *versión 3.1* de Windows o alguna posterior, entonces el correspondiente hardware y el sistema operativo serán compatibles con las aplicaciones del programa. La plataforma de desarrollo es la computadora donde se programan las aplicaciones, mientras que en una plataforma objeto es el sistema que utiliza el cliente o usuario final de la aplicación.

La aplicación de esta tesis se realizó y verificó en una computadora personal equipada con tarjeta gráfica **VGA** (cuando Windows se ejecuta en una **VGA**, utiliza el modo por defecto de 640 X 480 a 16 colores, mientras que en una **Super VGA** puede utilizar distintos modos, entre los que incluye el de 640 X 480 a 256 colores).

El programa ofrece compatibilidad nominal con compiladores de **C** o **C++** que se ajusten al estándar **ANSI C**, estén orientados a Windows, dispongan de un compilador de recursos compatible Microsoft y de un archivo de cabecera **WINDOWS.H**

Hardware y el Sistema Operativo

Todos aquellos que dispongan de una computadora personal equipada con una tarjeta **VGA** y sobre la que se pueda ejecutar **MS-DOS** y *Microsoft Windows 3.1*, es muy probable que tengan el hardware y el sistema operativo necesarios para el programa de simulación, mismo que se desarrolló en una computadora personal 80486SX a 33 MHz con una tarjeta **VGA**, con el sistema operativo **MS-DOS 6.0** y la *versión 3.1* de *Windows*.

V.2 DESARROLLO DE LA APLICACION (*PROGRAMACION EN WINDOWS*)

PLATAFORMA

Windows es un ambiente con entorno gráfico y es menester aprender unos principios básicos acerca de este entorno particular, para empezar a construir aplicaciones que se ajusten al modo de trabajo, Windows ejecuta los programas a la vez que los programas lo ejecutan, debido a que las aplicaciones cooperan como compañeros en tiempo de ejecución, esta dependencia mutua aparece en la figura 5.3

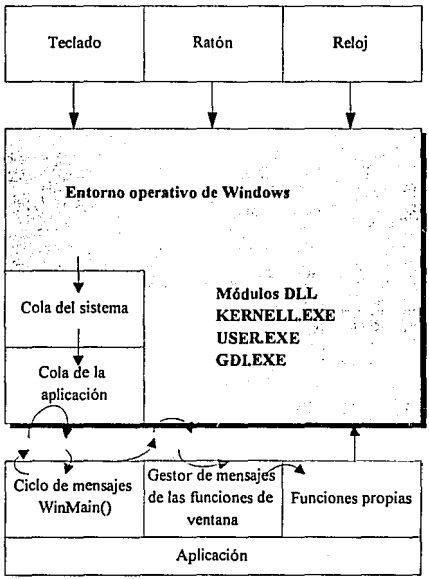


Figura 5.3 El modelo de programación de Windows

EJECUCION DE LOS PROGRAMAS EN WINDOWS

En la ejecución de los programas, Windows lleva a cabo tres tareas diferentes para proporcionar a la aplicación un entorno de ejecución:

1. Carga y ejecuta el programa

2. Gestiona la memoria de la computadora de manera que tanto la aplicación particular como el resto de aplicaciones Windows, que en ese momento se están ejecutando, tengan acceso a la memoria y a los demás recursos del sistema que puedan necesitar

3. Gestiona la mayoría de las entradas de bajo nivel que la aplicación realice

Activación de una aplicación

Cuando el usuario selecciona el icono de una aplicación, Windows asigna una cierta cantidad de memoria al programa. Después carga el código ejecutable y determina una pila local. A continuación, se ocupa de cargar los datos de la aplicación y de proporcionar cualquier recurso particular que sea requerido en tiempo de ejecución, estos recursos especiales son propios del entorno e incluyen objetos tales como un menú de sistema, abreviaturas de teclado, caja de diálogo, caja de mensaje, iconos y mapas de bits, aunque es importante decir que no se debe confundir los recursos de Windows con los recursos del sistema, como la memoria, el acceso a disco, las entrada de ratón y de teclado, y el acceso a la pantalla, entre otros.

Todos los programas para aplicaciones contienen una función **WinMain( )**, puesto que sirve como punto de partida para el programa que corresponde, en cierta manera, con la función **main ( )** de la programación **C** estándar.

Gestión de los recursos del sistema

Cuando se ejecuta más de una aplicación al mismo tiempo, puede ocurrir que la memoria esté limitada, Windows se ocupa de garantizar que cada aplicación disponga de la memoria que necesite; por eso, en algunas ocasiones esto implica el desalojo de fragmentos de código, datos y recursos fuera de la memoria para dejar el espacio libre que otra aplicación necesita, estos tres componentes (*código, datos y recursos*) están organizados en fragmentos llamados *segmentos*.

Windows también decide que mensaje debe ir a que aplicación, porque existe una cola de mensajes asociada explícitamente con cada aplicación, que el propio Windows se encarga de gestionar.

Paso de mensajes. La entrada de la aplicación son los mensajes que toma de su cola de aplicación, gestionada para el programa correspondiente, además, Windows está continuamente sondeando la actividad del reloj, el teclado y el ratón.

Todos los mensajes están formados por un núcleo de datos uniforme. El archivo *#include* de Windows, **WINDOWS.H**, que debe estar incluido al principio de todas las aplicaciones para declarar una estructura de datos denominada **MSG**(*message-mensaje*). Esta estructura está formada por seis miembros o elementos que se describen a continuación :

- **Gestor de ventana hwnd.** La aplicación posee una ventana principal, que normalmente se implementa como una superficie rectangular dentro del área de visualización de la pantalla. Las ventanas se identifican por un elemento llamado gestor, expresado como *hwnd*. Este primer miembro de la estructura **MSG** contiene un gestor de ventana *hwnd (handle windows)*.

- **Identificador de mensaje WM.** Especifica el tipo de suceso. Todos los identificadores de mensaje comienzan con los caracteres **WM_** (*Windows-message*).

- **wParam e lParam.** Contienen información relevante asociada al mensaje (*elemento del menú, tecla pulsada, etc.*).

- **time.** Indica el instante en que el mensaje fue ubicado dentro de la cola.

- **pt.** Contiene las coordenadas del cursor del mouse en el momento en que Windows ubicó el mensaje dentro de la cola.

Ciclo de mensajes. El programa contiene un ciclo repetitivo que se encarga de leer los mensajes, denominado bucle de mensajes (message loop), el ciclo de mensajes esta contenido en la función **WinMain( )** descrita anteriormente como el punto de partida de las aplicaciones.

Cuando la aplicación detecta un mensaje, abandona temporalmente el bucle para realizar la acción que juzgue oportuna y regresa a la espera de nuevos mensajes, mientras tanto, el ciclo de mensajes llama a la función predefinida **GetMessage( )** para comprobar si existe otro mensaje pendiente. Al hacer la llama da a la función **GetMessage( )**, Windows comprueba primero si existe algún mensaje en la cola de la aplicación, a continuación busca en la cola del sistema si existe algún mensaje del teclado o el ratón y, por último, comprueba si hay algún mensaje del reloj.

Gestor de mensajes. Cuando la aplicación detecta la llegada de un mensaje, abandona el ciclo de mensajes para ejecutar las funciones correspondientes incluídas en el programa. Este proceso se lleva a cabo con el envío de un mensaje, a una parte especial del programa denominado *gestor de mensajes*, asociando cada ventana (*o aplicación*) con un gestor del procedimiento, designádose como **WinProc( ).**

LOS PROGRAMAS EJECUTAN WINDOWS

La mayoría de las operaciones fundamentales realizadas por un aplicación están implementadas con llamadas a las funciones predefinidas por Windows. Estas rutinas se encuentran en la interfaz de programación de aplicaciones (**API:** *Application Programming Interface*).

Bibliotecas de enlace dinámico

Las funciones ejecutables del **API** están contenidas en las *denominadas bibliotecas de enlace dinámico* (**DLL** : *Dynamic-Link Libraries*). Existen tres **DLL** distintas: **KERNELL.EXE, USER.EXE, GDI.EXE**.

- **KERNEL.EXE** proporciona las funciones para la gestión de ventanas ,que constituyen el soporte del entorno Windows.

- **USER:EXE** enlaza los servicios del sistema con la gestión de memoria o la gestión de la multitarea.

- **GDI.EXE**, es la interfase de dispositivos gráficos, proporcionando una serie de rutinas gráficas que explotan las potencialidades del ambiente gráfico

Funcionamiento de las bibliotecas DLL

Las tres bibliotecas **API** se denominan de enlace dinámico debido a que no son enlazadas a la aplicación hasta el momento de la ejecución porque el enlace se realiza dinámicamente.

Este método contrasta fuertemente con el enlace estático utilizados por los programas de C estándar, al compilar y enlazar un programa de C estándar, se inserta el código objeto de las funciones de biblioteca en el archivo ejecutable resultante y, por su parte, el enlazador inserta un bloque de código objeto denominado *biblioteca de importación (contiene información acerca de la ubicación de las funciones ejecutables incluidas en KERNEL.EXE, USER.EXE y GDI.EXE.* En Borland C se llama **IMPORT.LIB**).

COMPONENTES DE LA APLICACION EN WINDOWS

La programación en Windows esta basada en una aproximación multimodular donde se compilan y enlazan una serie de archivos fuente para generar el programa completo denominado *ejecutable*, formado por cuatro tipos de archivos fuente conocidos como: archivo de definición del módulo, un archivo *#include*, un archivo de especificación de recursos y un archivo fuente en C.

- **archivo de definición del módulo**

Contiene información técnica acerca de la estructura del archivo ejecutable completo, denotado por la extensión **.DEF**, describe caracteristicas tales como el tamaño de la pila local, el nombre de la función gestora de mensajes, y así sucesivamente.

- **archivo #include**

Es un archivo de cabecera típicoque incluye los prototipos de las funciones, con la declaración de la lista de parámetros y los valores a devolver por las funciones que constituyen el núcleo de la aplicación; define también variables de concordancia, denominadas constantes identificadoras de menú que permiten asociar las funciones con elementos específicos de un menú. El archivo *#include* tiene extensión **.H**.

- **archivo de especificación de recursos**

Este archivo utiliza combinaciones de teclado especiales para describir los recursos que la aplicación necesita como el menú del sistema y sus mnemónicos de teclado. Cuenta con una extensión **.RC**

- **archivo fuente en C**

Contiene las funciones que constituyen el núcleo del programa, también la función indispensable para el inicio de una aplicación conocida como **WinMain( )**, que constituye el punto de partida de la ejecución, el gestor de mensajes y el archivo fuente en C conmutan el control a las funciones específicas de la aplicación. Este archivo fuente utiliza la extensión .C.

COMPILACION Y ENLACE DEL SISTEMA EN WINDOWS

En la Figura 5.4 se ilustra de manera gráfica el proceso de compilación y enlace que se utiliza para construir una aplicación Windows típica, mismo que se consideró en la elaboración del sistema simulador.

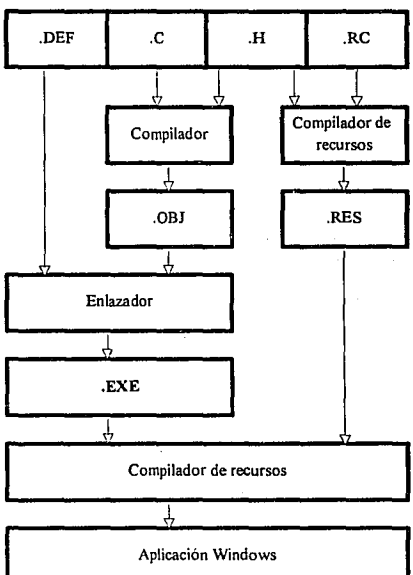


Figura 5.4 Plataforma para el desarrollo de aplicaciones Windows

Compilación

El compilador de C reúne los archivos *#include*, extensión **.H**, y los archivos fuente en C que forman la aplicación y los compila en un archivo de código objeto con la extensión **.OBJ**.

El compilador de recursos reúne los archivos de especificación de recursos, extensión **.RC** y los archivos *#include* **.H** de la aplicación, y los compila en un archivo de código objeto con la extensión **.RES**.

Enlace

El enlazador combina los módulos del código **.OBJ** y los módulos de código **.LIB** para generar el archivo ejecutable con extensión **.EXE**, consultando la información del archivo de definición del módulo **.DEF** y, a continuación enlaza el código **.OBJ** con la biblioteca de importación y las rutinas de la biblioteca en C que aparezcan referenciadas en el código. El enlazador realiza un proceso de enlace estático para insertar en el programa los módulos **.OBJ** de la biblioteca en C apropiados, a continuación inserta en el programa la biblioteca de importación **.OBJ** completa y la información acerca de la ubicación del **API** que contiene la biblioteca de importación permite el enlace dinámico de la aplicación y las bibliotecas **DLL** en tiempo de ejecución.

A pesar de todo, el archivo **.EXE** generado por el enlazador todavía no esta listo para ejecución.

Asignación de recursos

Se utiliza el compilador de recursos para añadir el código objeto **.RES** (*que contiene los recursos que la aplicación necesita en tiempo de ejecución*) al archivo ejecutable, el resultado es un archivo ejecutable Windows.

Identificación

El compilador de recursos, al mismo tiempo que asocia los recursos al archivo ejecutable, identifica al archivo **.EXE** con el número de la versión Windows, todos los ejecutables deben incluir un número de versión en su encabezamiento de lo contrario se rechaza el ejecutable, es decir se utiliza un proceso compilación-enlace-identificación, donde el compilador genera el código **.OBJ** a partir de los archivos fuente **.H** y **.C**.

V. 3 COMPILACION CON BORLAND C++

Se puede construír una aplicación siguiendo una de las indicaciones que se presentan : con línea de órdenes del **DOS** de Borland C++ para compilar y enlazar los programas, con el entorno integrado **IDE** (*Entorno Integrado de Desarrollo*) ó utilizar de un modo apropiado programas **CASE**.

Disponibilidad de archivos

Durante una sesión de trabajo los compiladores y enlazadores de Borland C++ son capaces de encontrar los archivos fuente, cabecera y archivos de biblioteca. Los compiladores y enlazadores se almacenan en **C:\BORLANDC\BIN**, los archivos de cabecera se encuentran en **C:\BORLANDC\INCLUDE**, las bibliotecas ejecutables de **C** se encuentran en **C:\BORLANDC\LIB**.

- **Construcción de la aplicación con DOS**

Construír la aplicación Windows en Borland C++ consiste de una operación directa que sigue el proceso compilar-enlazar-identificar que se uso a lo largo de todo el desarrollo. Esta aproximación de tres pasos proporciona un entorno de desarrollo consistente y fiable para contruír la aplicación en Windows.

Compilación

Este proceso requiere de una linea de órdenes para construír cualquier programa.Sin embargo también se puede compilar dentro del software de Borland C++ para Windows desde el menú ó el icono de compilación. En la Figura 5.5 se proporciona una explicación de las diversas opciones.

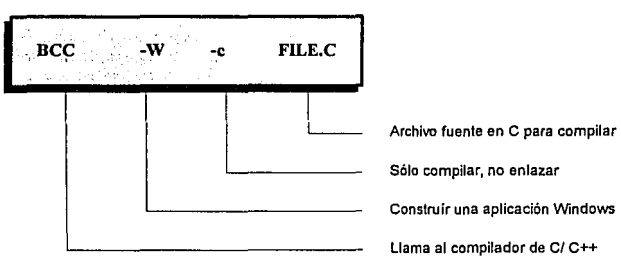


Figura 5.5 Línea de órdenes genérica para el compilador Borland C++

Enlazado

Para enlazar los archivos **.OBJ** con las bibliotecas ejecutables y que estas se importen, se utiliza la línea de órdenes con los nombres apropiados de la aplicación (Figura 5.6).

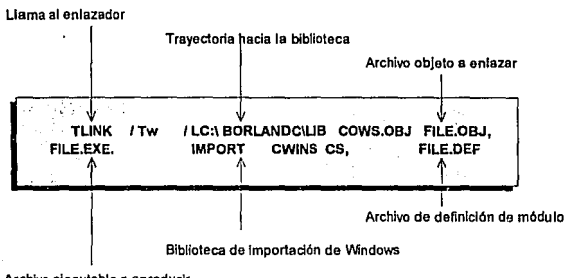


Figura 5.6 Línea de órdenes genérica para el enlazador Borland C++

- **Construcción de una aplicación con IDE**

Como la plataforma de Borland nos ofrece un entorno más integrado, el archivo ejecutable lo construimos creando un proyecto utilizando el **IDE** de Borland C++ en vez de utilizar la línea de órdenes del **DOS** (anteriormente descrita). El **IDE**, se maneja mediante menús, permite editar y compilar aplicaciones sin abandonar el editor. Para la construcción de la aplicación con el **IDE** de Borland C++, se creó un proyecto añadiendo los nombres de los archivos fuente en **C** del archivo de especificación de recursos **.RC** y del archivo de definición de módulo **.DEF** en el archivo **.PRJ**.

- **Construcción utilizando herramientas de rápido - prototipo (WindowsMAKER)**

La ingeniería de software asistida por computadora (**CASE**) se puede utilizar de modo apropiado en el desarrollo de aplicaciones para Windows. Al utilizar programas **CASE** estos se encargan de muchas de las tareas relacionadas con la construcción de sistemas de menús, cuadros de diálogos y control de ventanas de interfaz de documento múltiple (**MDI**), pudiendo enfocar la atención en las funciones propias de la aplicación.

Bluesky incluye un programa **CASE** llamado **WindowsMAKER** que está disponible como paquete autónomo con el nombre de **WindowsMAKER Professional**. Esta herramienta es particularmente útil para situaciones explícitas.

Con **Resource Workshop** se generó código fuente para los menús, cuadros de diálogo y controles, de esta manera se estudia el código fuente generado para ver que funciona y soluciona específicamente los problemas de programación. Esta capacidad de rápido prototipo significa que se puede afinar la presentación e interacción de la aplicación.

EJECUCION DEL PROGRAMA

Una vez finalizado con éxito el proceso de compilación-enlace-identificación, es posible ejecutar la aplicación del simulador en el entorno Windows, para experimentar con la aplicación del sistema se puede utilizar el mouse o el teclado; basta apuntar y pulsar con el mouse para explorar los menús. También se puede accesar los menús mediante el teclado empleando las flechas direccionales, utilizando la tecla mnémonica que corresponda con la tecla subrayada en el elemento del menú, los menús también admiten el uso de abreviaturas de teclado que son combinaciones de teclas que permiten activar las acciones correspondientes desde cualquier punto del programa.

Menús y cuadros de mensaje

El documento *System Application Architecture* (**SAA**) *Common User Acces* (**CUA**) de IBM recoge una serie de estándares par los programas que se ejecutan en un entorno de ventanas. El programa de simulación **SimPAL** posee una serie de menús y facilidades que siguen el estándar **SAA CUA**.

About. Esta aplicación en Windows contiene un cuadro About en donde se tiene el nombre del programa e información sobre derechos de autor, entre otras cosas.

El menú File. Los elementos que contiene, e incluso su orden, son los que se recomiendan específicamente en el citado documento. En este molde tan solo está activado el elemento *Exit* del menú *File*. Se puede utilizar para salir del sistema.

El menú Edit. Todos los elementos se pueden activar con sus correspondientes abreviaturas.

El menú de sistema. La ventana principal de la aplicación contiene un botón de menú Sistema, localizado en el vértice superior izquierdo de la misma, una pulsación sobre dicho botón llamará al menú de Sistema que se muestra en la figura 5.7. El sistema de simulación intercepta los cinco primeros elementos del menú Sistema. (con la selección de *Close* terminará la ejecución del programa)

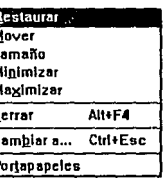


Figura 5.7 El menú de Sistema

PROGRAMAS FUENTE

El programa SimPAL proporciona una interfase genérica para lo cual utiliza al archivo **.RC** donde define un menú de sistema, abreviaturas de teclado y una tabla de cadenas de texto.

Los identificadores de cadenas y menú que necesita el código fuente en **C** se definen en el archivo **.H**, El archivo fuente en **C** utiliza un bucle de mensajes para recoger los mensajes del mouse y el teclado, y después los envía de uno en uno al gestor de mensjes. El gestor de mensajes utiliza una sentencia *switch()* para llamar a la función apropiada dentro del archivo fuente en **C**.

Para construir el código fuente se requiere del archivo de definición del módulo **SimPAL.DEF** que aparece en la Figura 5.8

```
1   NAME          SimPAL

2   DESCRIPTION   ' SimPAL :  Copyrigth 1994  M. G. Serna L., A. Rios M. '

3   STUB          'c:\borlandc\bin\WINSTUB.EXE'

4   CODE          PRELOAD MOVEABLE DISCARDABLE
5   DATA          PRELOAD MOVEABLE MULTIPLE

6   EXETYPE       WINDOWS

7   SEGMENTS
8                 _RES  PRELOAD MOVEABLE DISCARDABLE

9   HEAPSIZE      1024
10  STACKSIZE     8192

11  EXPORTS       WndProc
12                AboutDlgProc
13                PickDlgProc
```

Figura 5.8 Archivo de definición de módulo del programa de simulación

Funcionamiento del archivo .H

El listado del archivo *#include* **.H** es el que aparece en la figura 5.9. Las constantes de identificación de menú son las que se encuentran desde la línea 7 a la línea 37. Se considera una buena practica de programación anteponer **IDM_** a los nombres de las constantes. Estas constantes se utilizan en el gestor de mensajes del archivo fuente **.C**, donde una sentencia *switch()* las compara con los mensajes de entrada. El archivo **.RC**, utiliza identificadores cuando define las cadenas de texto propiamente, dichas y el archivo fuente **.C** las utiliza cuando carga los recursos cadena en tiempo de ejecución.

```
1   /*-------------------------------------------------
2    SimPAL.H Archivo de cabecera (header file):
3    incluye los prototipos de las funciones,
4    los identificadores de los elementos del
5    menú y  de las cadenas.
6    -------------------------------------------------*/

7   #define IDM_NEW             101
8   #define IDM_OPEN            102
9   #define IDM_SAVE            103
10  #define IDM_SAVEAS          104
11  #define IDM_PRINT           105
12  #define IDM_PAGSETUP        106
13  #define IDM_PRINSETUP       107
14  #define IDM_EXIT            108

15  #define IDM_UNDO            201
16  #define IDM_CUT             202
17  #define IDM_COPY            203
18  #define IDM_PASTE           204
19  #define IDM_DEL             205
20  #define IDM_SELALL          206

21  #define IDM_FIND            701
22  #define IDM_NEXT            702
23  #define IDM_REPLACE         703

24  #define IDM_FONT            401

25  #define IDWHELP             14
26  #define IDM_ABOUT           10

27  #define IDD_FNAME           20

28  #define IDM_CONV            301

29  #define IDM_ITDATA          501

30  #define IDM_ITOP            601

31  #define IDM_TILE            801
32  #define IDM_CAS             802


33  #define IDM_ABOUT           10
34  #define IDM_Tools           12

35  #define IDWHELP             14      // Id of help button

36  #define ID_VERSION          100

37  #define EXE_NAME_MAX_SIZE 128
```

Funcionamiento del archivo .RC

El listado del archivo de especificación de recursos aparece en la figura 5.9. Este archivo especifica los recursos que la aplicación necesitará en tiempo de ejecución, entre los que se incluyen los menús, las abreviaturas de teclado y las cadenas.

```
1    SimPal MENU
2    BEGIN
3    POPUP "&File"
4    BEGIN
5          MENUITEM "&New",              IDM_NEW
6          MENUITEM "&Open...",          IDM_OPEN
7          MENUITEM "&Save",             IDM_SAVE
8          MENUITEM "Save &as...",       IDM_SAVEAS
9          MENUITEM SEPARATOR
10         MENUITEM "&Print...",         IDM_PRINT
11         MENUITEM "Page se&tup...",    IDM_PAGSETUP
12         MENUITEM "P&rinter setup...", IDM_PRINSETUP
13         MENUITEM SEPARATOR
14         MENUITEM "E&xit",             IDM_EXIT
15   END

16   POPUP "&Edit"
17   BEGIN
18         MENUITEM "&Undo\tCtrl+Z",     IDM_UNDO
19           MENUITEM SEPARATOR
20         MENUITEM "Cu&t\tCtrl+X",      IDM_CUT
21         MENUITEM "&Copy\tCtrl+C",     IDM_COPY
22         MENUITEM "&Paste\tCtrl+V",    IDM_PASTE
23         MENUITEM "&Delete\tDel",      IDM_DEL
24         MENUITEM SEPARATOR
25         MENUITEM "&Select All",       IDM_SELALL
26   END

27   POPUP "Fo&rmula"
28   BEGIN
29         MENUITEM "&Conversion DeML",  IDM_CONV
30   END

31   POPUP "&Character"
32   BEGIN
33         MENUITEM "&Font...",          IDM_FONT
34   END

35   POPUP "&Data"
36   BEGIN
37         MENUITEM "Item",              IDM_ITDATA
38   END

39   POPUP "&Options"
40   BEGIN
41         MENUITEM "Item",              IDM_ITOP
42   END
```

```
43  POPUP "&Search"
44  BEGIN
45          MENUITEM "&Find...",                IDM_FIND
46          MENUITEM "Find &Next\tF3",          IDM_NEXT
47          MENUITEM "&Replace",                IDM_REPLACE
48  END

49  POPUP "&Window"
50  BEGIN
51          MENUITEM "&Tile",                   IDM_TILE
52            MENUITEM "&Cascade",              IDM_CAS
53  END

54  POPUP "\a&Help"
55  BEGIN
56          MENUITEM "&Help",                   IDWHELP
57          MENUITEM "&About SimPal ...",       IDM_ABOUT
58  END

59  END

60  //———Especificación de las abreviaturas de teclado—

61  Simpal ACCELERATORS
62    BEGIN

63    VK_BACK,  IDM_UNDO,  VIRTKEY, ALT
64    "^X",     IDM_CUT
65    VK_DELETE, IDM_CUT,  VIRTKEY, SHIFT
66    "^C",     IDM_COPY
67    VK_INSERT, IDM_COPY,  VIRTKEY, CONTROL
68    "^V",     IDM_PASTE
69    VK_INSERT, IDM_PASTE, VIRTKEY, SHIFT
70    VK_DELETE, IDM_DEL,  VIRTKEY
71    VK_F3,    IDM_NEXT,  VIRTKEY
72    VK_F1,    IDWHELP,  VIRTKEY
73    END

74  /*STRINGTABLE

75  BEGIN

76          IDS_Source "SimPal.c"
77          IDS_SignOn "Here-is-how-it's-done source code taken from the "
78          IDS_Warning "Warning"
79          IDS_NoMouse "No mouse"
80          IDS_Caption "Programa de windows"
81          END
82          */
```

Figura 5.9 Parte del programa original de los recursos utilizados

Menús. Las líneas 1 a 59 describen el menú de la aplicación. Observe como se asigna una constante de identificación de menú, **IDM_** a cada elemento del mismo,de manera que el archivo **.C** disponga de las herramientas necesarias para identificar los elementos de menú específicos que el usuario seleccione en tiempo de ejecución. Las teclas mnemónicas subrayadas tan sólo se pueden utilizar desde el sistema de menús.

El recurso cadena. La palabra clave **STRINGTABLE** que aparece en la línea 74 señala el principio de las definiciones de cadenas, que van de la línea 75 a la 81, cada cadena lleva asignada una constantes de identificación de cadena, **IDS_**. El archivo fuente **.C** utilizará la constante de identificación de cadena siempre que necesite cargar una cadena para su visualización en tiempo de ejecución.

Funcionamiento del archivo .C

En la figura 5.10 aparece una parte de el listado del archivo fuente **.C**. En este archivo se incluye la función *WinMain()*, que es el punto de entrada a la aplicación.

```
1  /*============================================================
2  WinMain — main window procedure for SimPAL

3  File    : SIMPAL.C

4  Prototype : int PASCAL WinMain (HANDLE, HANDLE, LPSTR, int);

5  Call    : WinMain (hInstance, hPrevInstance, lpszCmdParam, nCmdShow);

6  Library  :

7  ------------------------------------------------------------------------*/
8  #pragma wam -par                /* nCmdShow, lpszCmdParam no son usados */

9  int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
10                                   LPSTR lpszCmdParam, int nCmdShow)
11     {
12     MSG  msg;
13     WNDCLASS WndClass;
14     HDC  hdc;
15     TEXTMETRIC      tm;
16     HMENU      hMenu;
17     GLOBALHANDLE  MyStructHandle;
18     FPWYNAPSESTRUCT MyStruct;
19     static char    szMenuTitle[21 + 1];
20     HANDLE     hAccel;

21     if (!hPrevInstance)
22          {

23     WndClass.style       = CS_HREDRAW | CS_VREDRAW;
24     WndClass.lpfnWndProc   = (WNDPROC)WndProc;
25     WndClass.cbClsExtra   = 0;
26     WndClass.cbWndExtra   = 0;
```

```
27      WndClass.hInstance     = hInstance;
28      WndClass.hIcon         = LoadIcon(hInstance, szAppName);
29      WndClass.hCursor       = LoadCursor(NULL, IDC_ARROW);
30      WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
31      WndClass.lpszMenuName  = szAppName;
32      WndClass.lpszClassName = szAppName;

33      RegisterClass(&WndClass);
34          }

35      hdc = CreateIC("DISPLAY", NULL, NULL, NULL);
36            GetTextMetrics(hdc, &tm);
37      DeleteDC(hdc);

38      cxClient = GetSystemMetrics(SM_CXSCREEN);
39      cyClient = GetSystemMetrics(SM_CYSCREEN);

40      hClickBar = LoadLibrary("SIMBAR.DLL");
41      if (hClickBar < 32)
42        {
43        FreeLibrary(hClickBar);
44        MessageBox(GetFocus(),"Dynamic Link Library SIMBAR.DLL must be present",
45                              szAppName, MB_ICONEXCLAMATION | MB_OK);
46        return 0;
47        }

48      hClickDlg = LoadLibrary("SIMDLG.DLL");
49      if (hClickDlg < 32)
50        {
51        FreeLibrary(hClickDlg);
52        MessageBox(GetFocus(),"Dynamic Link Library SIMDLG.DLL must be present",
53                              szAppName, MB_ICONEXCLAMATION | MB_OK);
54        return 0;
55        }

56      lpfnDlgSetup    = GetProcAddress(hClickDlg,  "ClickBarDialogSetup");
57      lpfnDlgMove     = GetProcAddress(hClickDlg,  "ClickBarDialogMove");
58      lpfnDisplayDlg  = GetProcAddress(hClickDlg,  "ClickBarDialogProc");

59      lpfnClickBarSetup = GetProcAddress(hClickBar,  "ClickBarSetup");

60      MyStructHandle = GlobalAlloc(GHND, sizeof(WYNAPSE_STRUCT));
61      MyStruct = (FPWYNAPSESTRUCT)GlobalLock(MyStructHandle);
62        lstrcpy(MyStruct->RegName, (LPSTR)"Your Name");  /* ClickBar RegName */
63        MyStruct->RegNumber = 0;                         /* ClickBar RegNumber */
64      GlobalUnlock(MyStructHandle);

65      (*lpfnClickBarSetup)(MyStructHandle);

66      GlobalFree(MyStructHandle);

67      hWndMain = CreateWindow(szAppName,  /* window class name */
68                  szAppName,    /* window caption */
69                  WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
70                  CW_USEDEFAULT,    /* initial x pos */
71                  CW_USEDEFAULT,    /* initial y pos */
```

```
72                      CW_USEDEFAULT,          /* initial x size */
73                      CW_USEDEFAULT,          /* initial y size */
74                      NULL,           /* parent window handle */
75                      NULL,           /* window menu handle */
76                      hInstance,      /* program instance handle */
77                    lpszCmdParam /*NULL*/);           /* creation parameters */

78      hMenu = GetSystemMenu(hWndMain, FALSE);
79      AppendMenu(hMenu, MF_SEPARATOR, 0, NULL);
80      AppendMenu(hMenu, MF_STRING,    IDM_ABOUT, "About...");
81      AppendMenu(hMenu, MF_STRING,    IDM_Tools, "Tools...");
82      AppendMenu(hMenu, MF_STRING,    IDWHELP,   "Help....");

83      hInst = hInstance;

84      MakeHelpPathName(szHelpFileName);

85      ShowWindow(hWndMain, nCmdShow);
86      UpdateWindow(hWndMain);

87      hAccel = LoadAccelerators (hInstance, szAppName);

88      hWndDlg = CreateDialog(hInst, "CLICK", hWndMain,
89                              MakeProcInstance((*lpfnDisplayDlg), hInst));

90      lstrcpy(szMenuTitle, "AbNormal");
91      SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);
92      lstrcpy(szMenuTitle, "AsNormal");
93      SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);
94      lstrcpy(szMenuTitle, "ProNormal");
95      SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);
96      lstrcpy(szMenuTitle, "Not Normal");
97      SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);
98      lstrcpy(szMenuTitle, "Huh?");
99      SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);
100     lstrcpy((LPSTR)szMenuTitle, "Normal");
101     SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_ADDSTRING, 0, (LONG)(LPSTR)szMenuTitle);

102     SendDlgItemMessage(hWndDlg, IDXCOMBO, CB_SELECTSTRING, -1,
    (LONG)(LPSTR)szMenuTitle);

103     SendMessage(hWndMain, WM_NCACTIVATE, 1, 0L);

104     while (GetMessage(&msg, NULL, 0, 0))
105      {
106      if ((hWndDlg == 0 || !IsDialogMessage(hWndDlg, &msg)) && (hDlgModeLess == 0 ||
    !IsDialogMessage(hDlgModeLess, &msg)))
107         {
108         if (!TranslateAccelerator (hWndMain-10, hAccel, &msg))
109               {
110                     TranslateMessage(&msg);
111               DispatchMessage(&msg);
112                 }
113         }
114      }
```

Figura 5.10 Parte del programa original SimPAL.C

Archivos incluídos. En la línea 3 del archivo de cabecera (**SimPAL.C**) se incluye el archivo **WINDOWS.H**, que contiene la declaración de variables, la definición de constantes y los prototipos de funciones necesarios para todas las aplicaciones Windows.

```
1   /*--------------------Archivos de cabecera--------------------*/
2   #define WINVER 0x0300
3   #include <windows.h>
4   #include <string.h>
5   #include <commdlg.h>
6   #include <stdlib.h>
7   #include <c:\prog\testclik\simdlg.h>
8   #include <c:\prog\testclik\simbar.h>
```

El bucle de mensajes. El bucle de mensajes del programa esta localizado entre las líneas 104 y 114. Una llamada a *GetMessage( )* comprueba la llegada de mensajes. A continuación se llama a la función *DispatchMessage( )* para inciar un mensaje al gestor de la aplicación.

Gestor de mensajes. El gestor de mensajes de SimPAL comienza con la sentencia *switch()* que se utiliza para identificar el tipo de mensajes recibido (*del programa original se tomaron líneas representativas de esta sentecncia*). La línea detecta los mensajes **WM_COMMAND**, que implican la selección de un elemento del menú por parte del usuario.

```
1       switch (wParam)
2           {
3               case IDM_ABOUT :
4               lpfnAboutDlgProc = MakeProcInstance(AboutDlgProc, hInst);
5               DialogBox(hInst, "About", hWnd, lpfnAboutDlgProc);
6               FreeProcInstance(lpfnAboutDlgProc);
7               return 0;

8               case IDWHELP :
9               WinHelp(hWnd, szHelpFileName, HELP_INDEX, 0L);
10              return 0;
```

La clase de ventana. Los atributos de la clase que describe la ventana principal se definen con la función que aparece entre las líneas 23 y 34 del archivo **.C** .

Creación de la ventana principal. La función que va de la línea 67 del archivo **.C** crea la ventana principal.

CAPITULO VI

PRUEBAS Y LIBERACION

La tendencia en el diseño de circuitos requiere el uso de herramientas computacionales con interfase gráfica para que la interacción *hombre-máquina* sea más amigable, logrando que los sistemas incrementen la productividad al disminuir costos y tiempo de producción.

Bienvenido a **SimPAL**, el sistema que facilita la tarea del diseño de sistemas digitales. Aprenderlo a utilizar es muy sencillo, ya que su interfaz gráfica es la misma que utiliza Windows. En este capítulo se describen los módulos que lo integran y las operaciones que realiza.

VI.1 VISUALIZACION DE MODULOS

SimPAL es la herramienta útil para el Diseño de Sistemas digitales (Figura 6.1). Este sistema orientado al empleo de PAL o GAL facilita las tareas de desarrollo ofreciendo entre sus principales características :

- Ambiente gráfico
- Herramientas para un fácil diseño
- Reducción del número de circuitos integrados
- Reducción de costos
- Verificación de la lógica Booleana
- Edición y modificación de ecuaciones Booleanas
- Simulación
- Disminución de tamaño físico del sistema digital

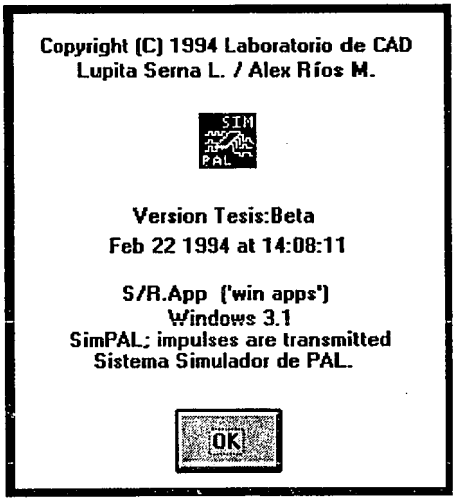


Figura 6.1 Derechos reservados del sistema SimPAL (About)

En Windows, todas las aplicaciones se realizan dentro del límite del área de trabajo, es decir, el espacio en pantalla que Windows dedica al usuario.

Una vez iniciada una aplicación, ejecutará en la ventana de la aplicación que aparecerá en el área de trabajo. Cuando se ejecuten varias aplicaciones al mismo tiempo, se podrán reducir a iconos de aplicación. **SimPAL** aparece como un pequeño icono en la pantalla de Windows y se puede accesar al sistema como se hace con cualquier otro icono de Windows (*pulsando doble-clic el botón izquierdo del mouse*).

Otra opción para trabajar con **SimPAL** es dar una línea de comandos de Windows que permite ejecutar una determinada aplicación o comando inmediatamente después de entrar en Windows. Por ejemplo, la línea de comandos siguiente iniciará con Windows y a continuación activará el programa SimPAL :

win c:\ directorio de trabajo del usuario\ SimPAL.exe

Nota: También puede establecerse la aplicación al iniciar Windows, agregando el icono de programa a un grupo del Administrador de Programas.

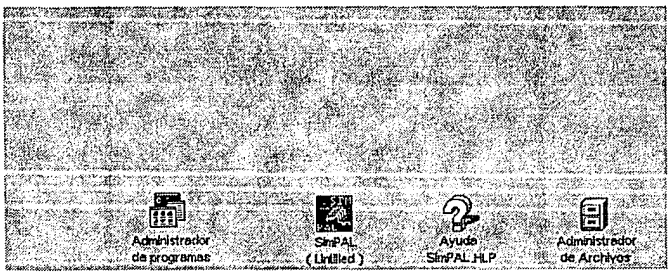


Figura 6.2 Ventana de aplicaciones en Windows

Descripción de la ventana

Los elemento de la ventana que sirven de herramientas para trabajar con la aplicación se describen en la figura 6.3, donde aparece la pantalla inicial que se presenta cuando iniciamos con **SimPAL**.

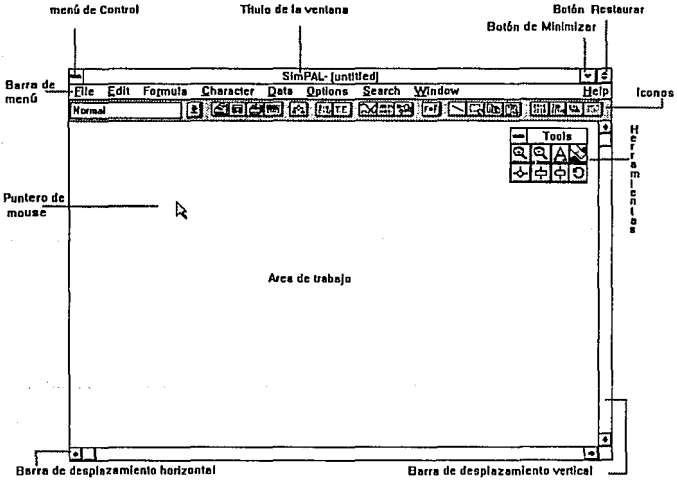


Figura 6.3 Ventana del sistema SimPAL

Barra de herramientas

Un icono es un pequeño gráfico que representa a un elemento de Windows. En el simulador se presentan varios tipos de iconos cuyo símbolo gráfico esta relacionado con la función que deben realizar.

Abrir un documento existente. SimPAL presentará el cuadro de diálogo Abrir, donde se muestran los nombres de todos los archivos de un directorio. En el cuadro "*Mostrar archivos de tipo*" se tiene la opción *.* para visualizar aquellos de nuestro interés como son los de extensión **.JED, .SIM**.

Guarda el archivo en la unidad y directorios específicos. Para los archivos de cualquier extensión (*.*).

Imprime el documento activo con las especificaciones seleccionadas. Incluye la opción de configurar o instalar impresoras que son archivos controladores para las impresoras utilizadas con más frecuencia (*.DRV).

Imprime el documento activo en el ploter. En caso de ser almacenado en un archivo, asigna extensión **.PLT**

Permite estructurar la carta **ASM** del circuito digital.

Introduce las **Ecuaciones Booleanas** y muestra el archivo con extensión **.BEQ**

Proporciona la **Tabla de Estados** utilizando la extensión **.STT**

Realiza la simulación a partir del código **JEDEC** generado guardando un archivo **.SIM**

Muestra el código **JEDEC**.

Presenta de modo gráfico el diagrama lógico del **PAL** con su respectiva asignación de ecuaciones.

Cambia la lógica positiva a negativa de las Ecuaciones Boleanas utilizando leyes de De Morgan.

Son las líneas que unen los bloques para cartas **ASM** ó módulos **ASM**.

Inserta módulos **ASM**.

Copia textos y gráficos seleccionados y los guarda en el Portapapeles.

Inserta el contenido del Portapapeles en el punto de inserción.

Traduce las Ecuaciones Booleanas al formato JEDEC.

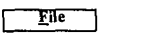
Traduce el Formato JEDEC hacia Ecuaciones Booleanas

Convierte el código equivalente de **GAL** a **PAL**.

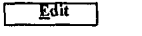
Convierte el código equivalente de **PAL** a **GAL**.

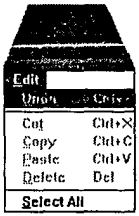
Barra de menú

Contiene todos menús y comandos de **SimPAL**. Para elegir alguno de ellos se puede utilizar el mouse o presionando combinaciones de teclas (*Aceleradores*).



Este menú contiene los comandos que se usan para abrir, imprimir y guardar o cerrar los documentos y/o archivos.





En edición seleccionamos todo o una parte de texto para borrar, cortar, copiar y restaurar (*cancelar el último comando ejecutado*) . El texto copiado o cortado se coloca en el Portapapeles (*área de almacenamiento temporal*). Los elementos que se coloquen en el Portapapeles permanecerán en él hasta que se vuelvan a elegir **Cortar** (*Copy*) o **Copiar** (*Cut*).

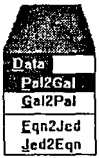
Formula

Con esta opción convertimos lógica Positiva a Negativa por leyes de De Morgan

Character

Este menú es para seleccionar el tamaño, tipo y color de fonts (*estilo de letra*).

Data



El menú Data despliega opciones para la conversión de diferentes formatos entre dispositivos

Options

En Options tenemos las formas de visualización para diseño con PAL.

Search

El comando Search permite hacer una búsqueda y reemplazo de cadenas

Window

La utilidad de este menú es la de organizar las ventanas del sistema en forma de títulos o cascada

Help

La selección de Help nos proporciona datos de la elaboración de SimPAL, así como una ayuda en línea

VI.2 ELABORACION DE PRUEBAS

DISEÑO LOGICO

Se desea diseñar un Registro de corrimiento donde se requieren las siguientes caracteristicas:

- Registro de corrimiento de 6 bits izquierda/derecha
- Entrada paralela y puertos de salida
- Entrada de reloj
- Líneas de control para selección de modo
- Facilidad para encadenarlos por medio de dos puertos seriales bidireccionales

El diagrama de bloques expresado en una tabla de funciones es (Figura 6.4) :

Función	SL	SR	RILO	Q_5	Q_4	Q_3	Q_2	Q_1	Q0	LIRO
Retén	0	0	Z	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Z
Corrimiento a la derecha	0	1	RI	RI	Q_5	Q_4	Q_3	Q_2	Q_1	Q0
Corrimiento a la izquierda	1	0	Q_4	Q_4	Q_3	Q_2	Q_1	Q_0	LI	LI
Llamado en paralelo	1	1	z	D_5	D_4	D_3	D_2	D_1	D_0	Z

Figura 6.4 Tabla de funciones del corrimiento de registros

Para el Registro de corrimiento de 6 bits, las ecuaciones son :

$$Q_0 = Q_0 /SR^* /SL + Q_1 *SR /SL + LIRO */SR *SL + D_0 * SR *SL$$

$$Q_i = Q_i */SR *SL + Q_{i+1} *SR*/ SL + D_i *SR*SL_{\ i=1...4}$$

$$Q_5 = Q_5*/SR*/SL + RILO *SR*/SL + Q_4* /SR *SL + D_5*SR * SL$$

LIROoutput = Q_0
LIROenable = SR* /SL
RILOoutput = Q_5
RILOenable = /SR * SL

IMPLEMENTACION DEL DISEÑO

Una vez que la familia ha sido seleccionada, se debe examinar el diagrama de bloques, la tabla de funciones y las ecuaciones lógicas para establecer los siguientes parámetros :

- Número de salidas con registros
- Número de salidas combinacionales
- Número de entradas
- Requerimientos del reloj
- Número de mintérminos

Así podemos observar que se necesita :

- Seis salidas con regsitro (*para las líneas de salida paralelas*)
- Dos líneas de I/O bidireccionales combinatorias (*para los puertos serie*)
- Seis entradas de datos paralelos más dos salidas de control de modo
- Un reloj maestro

Selección del dispositivo

Examinando las características de los diferentes dispositivos en el manual de **PLD**, se identifica que el **PAL16R6** y el **GAL16V8**, cumplen con los requerimientos. Otros dispositivos como el **PAL20R6** tambien pueden ser utilizados, pero tendría 24 pines en vez de 20 y por lo tanto mayor disipación de potencia, además de un exceso de entradas y salidas, es por eso que el **PAL16R6** resulta el más adecuado.

En nuestro ejemplo del registro de corrimiento de 6 bits, el método para la entrada de datos al **PLAN**, requirió la conversión de las ecuaciones a la sintáxis del paquete.

Para el ejemplo, se utilizó el Programador universal UP600 para el PAL. Para realizar esta tarea, se conectó el sistema a la plataforma (*base que sostiene al dispositivo*) a través de un cable **RS232C**, de acuerdo al manual del sistema.

Introducir Ecuaciones Booleanas

Se editan las ecuaciones en el **SimPAL** como a continuación se muestra (Figura 6.5).

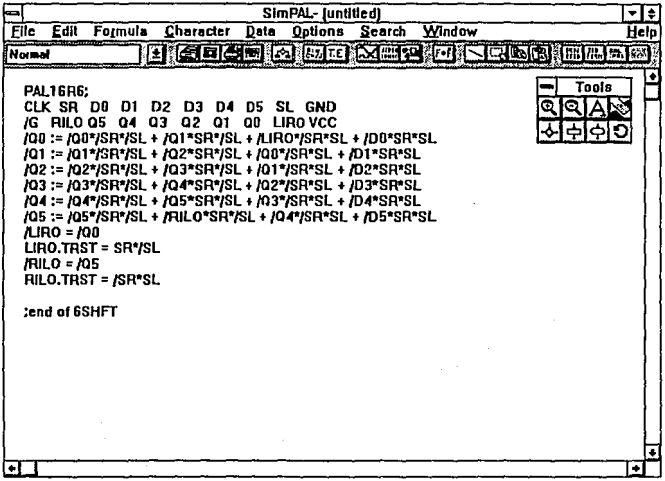


Figura 6.5 Edición de Ecuaciones Booleanas con el sistema SimPAL

Gereración del código JEDEC

Las Ecuaciones Booleanas anteriormente editadas, son convertidas a un formato como un archivo **JEDEC** que consiste en una tabla de fusibles programados para las funciones lógicas (Figra 6.6).

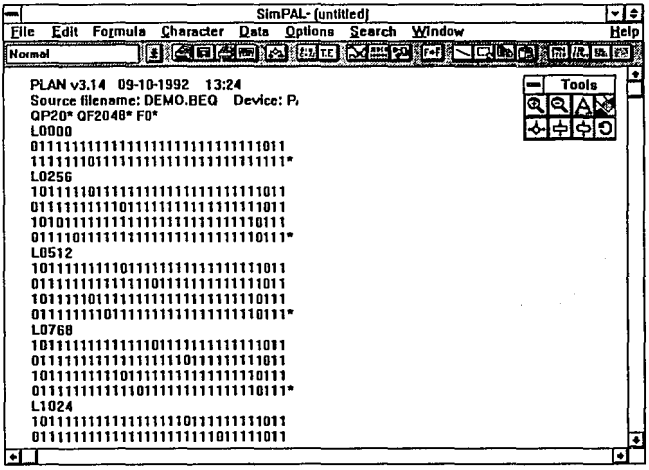


Figura 6.6 Generación del código JEDEC con el sistema SimPAL

VERIFICACION LOGICA

Para el diseño de nuestro registro de corrimiento de 6 bits, los vectores de prueba fueron realizados manualmente como se muestra en la Figura 6.7 . En este caso, una prueba de 1's y 0's es probablemente suficiente para comprobar la funcionalidad del dispositivo.

ENTRADAS								Bidireccional I / O		SALIDAS						COMENTARIOS
SL	SR	D5	D4	D3	D2	D1	D0	LIRO	RILO	Q5	Q4	Q3	Q2	Q1	Q0	Carga de ceros
1	1	0	0	0	0	0	0	1	1	L	L	L	L	L	L	Congela ceros
0	0	1	1	1	1	1	1	1	1	L	L	L	L	L	L	Corrimiento en uno a la izquierda
1	0	1	1	1	1	1	1	1	L	L	L	L	L	L	H	Seguimiento de ceros
1	0	1	1	1	1	1	1	0	L	L	L	L	L	H	L	
1	0	1	1	1	1	1	1	0	L	L	L	L	H	L	L	
1	0	1	1	1	1	1	1	0	L	L	L	H	L	L	L	
1	0	1	1	1	1	1	1	0	L	L	H	L	L	L	L	
1	0	1	1	1	1	1	1	0	H	L	L	L	L	L	L	Un corrimiento de RILO
1	0	1	1	1	1	1	1	0	L	L	L	L	L	L	L	Desvanecimiento
1	1	1	1	1	1	1	1	0	0	H	H	H	H	H	H	Carga de unos
0	0	0	0	0	0	0	0	0	0	H	H	H	H	H	H	Congelamiento de unos
0	1	0	0	0	0	0	0	H	0	L	H	H	H	H	H	Corrimiento a la derecha seguido de ceros
0	1	0	0	0	0	0	0	H	1	H	L	H	H	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	L	H	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	H	L	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	H	H	L	H	
0	1	0	0	0	0	0	0	L	1	H	H	H	H	H	L	Corrimiento de LIRO
0	1	0	0	0	0	0	0	H	1	H	H	H	H	H	H	Desvanecimiento

Figura 6.7 Tabla de funciones del Registro de corrimiento de 6 bits

Generación de Tablas de Estado

De acuerdo a la tabla 6.7, las entradas necesitan ser secuenciales, en SimPAL, los estados serán leídos por renglón y de acuerdo a la numeración de los pines del dispositivo PAL; para el caso del PAL16R6 se comienza por el número 2 debido a que el 1 es el reloj (Figura 6.8).

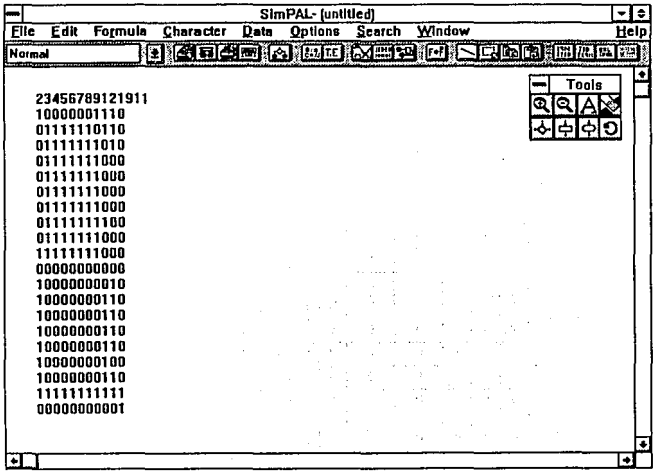


Figura 6.8 Tablas de Estado del Registro de Corrimiento de 6 bits

El archivo se guarda con extensión ***.STT** (*State table* : tabla de estados) y cada renglón esta condicionado a un ciclo de reloj.

Resultados de Entrada/Salida

Al seleccionar el icono de simulación o bien seleccionar el menú Options (opción) y elegir " Wave Form ", se despliega la simulación por tiempos generándose automáticamente un archivo con extensión *.SIM (*de simulación*) (ver Figura 6.9).

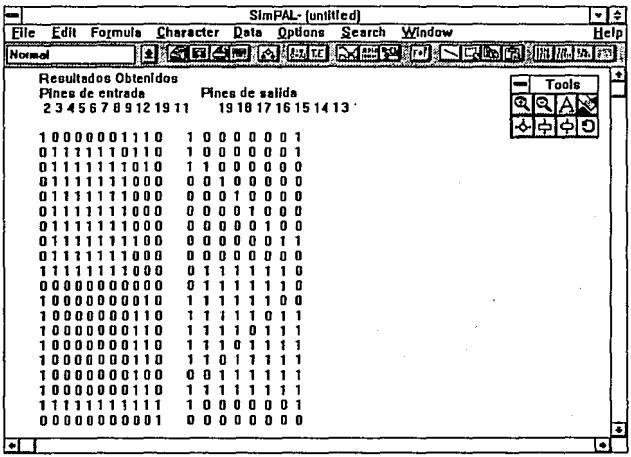


Figura 6.9 Resultados de Entradas/Salidas

En el archvio ***.SIM** pueden consultarse las respuestas de salida provocadas por los estímulos de entrada del archivo ***.STT**. La respuesta también está condicionada a un renglón por cada ciclo de reloj; esto permite cotejar los resultados obtenidos con la tabla de estados de los valores esperados. De coincidir la comparación, se asume que el archivo **JEDEC** proveniente de las ecuaciones Booleanas ha permitido un buen diseño, de lo contrario, es menester la elaboración de correcciones al diseño o bien revisar las ecuaciones Booleanas.

Simulación gráfica.

No solamente es almacenado el archivo con respuesta al estímulo de tabla de estados, sino que la misma información es desplegada en la pantalla, gracias a la utilización del dispositivo GDI suministrado por Windows, con lo cual se puede observar en la figura 6.10 los resultados gráficos por tiempos del reloj, respecto a la tabla de la figura 6.9

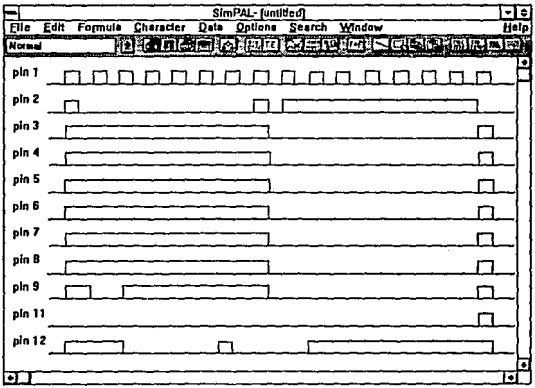


Figura 6.10 Despliegue gráfico de los resultados del registro de corrimiento de seis bits

Opciones

Ahora que ya estamos listos para programar nuestro **PAL**, antes de salir de una sesión de trabajo podemos guardar, iniciar un archivo nuevo (abrir uno existente) e imprimir, todas estas y otras alternativas se pueden llevar acabo en el sistema simulador **SimPAL**.

Save : Registra todos los cambios efectuados durante el diseño del sistema digital. Cuando elegimos esta opción se nos presenta una caja diálogo para salvar en unidad y directorios especificos.

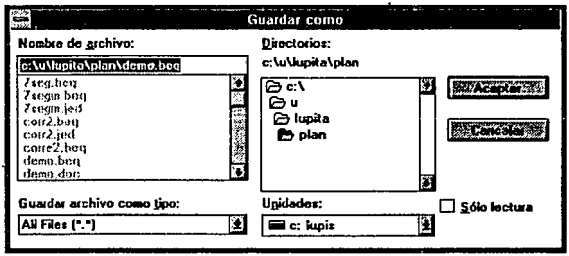


Figura 6.11 Caja de diálogo del comando Save

Open : Abre un archivo existente, para esto, muestra una caja de diálogo para indicarle el lugar donde se encuentran los archivos (Figura 6.12).

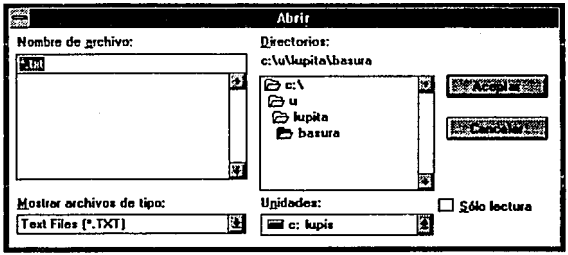
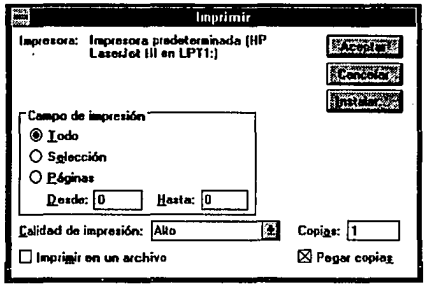
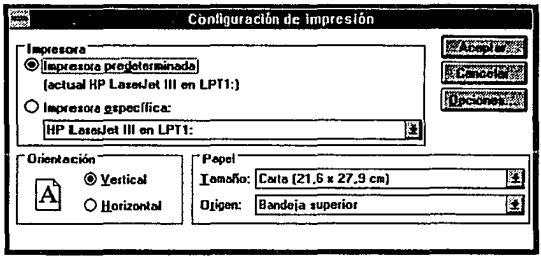


Figura 6.12 Caja de diálogo del comando Open

Print ...
Con esta opción se puede mandar a la impresora pre-establecida los resultados del diseño.



Printer setup
Si ni se tiene el controlador de impresoras adecuado, desde la aplicación SimPAL, puede configurarse el dispositivo de salida.



CONCLUSIONES

Ante la necesidad de programar **PLD** para un diseño digital, se tiene el problema de hacer esta actividad manualmente identificando las celdas que van a ser fundidas, de esta manera tenemos la consecuencia de perdernos o confundirnos en el mapa de fusibles de estos dispositivos. De esta problematica surge la idea de crear un sistema tipo ensamblador que proporciona la facilidad de manejo y que sirva como herramienta al momento de diseñar.

Al principio entre las principales dificultades a las que nos enfrentamos fue en buscar una forma de traducir las ecuaciones Booleanas a código **JEDEC**. Las consultas a las que nos dirijimos (*Motorola, Texas Instruments, National, etc*) nos proporcionaron diversas respuestas, sin embargo, después de un análisis exhaustivo nos vimos en la necesidad de proponer un método que funcionara para lograr nuestro fin.

El objetivo de la tesis fue crear un sistema con interfase gráfica para facilitar las tareas de desarrollo evitando un lenguaje descriptivo como se acostumbra a la hora de utilizar un ensamblador de **PAL**, es decir, se prefiere la presentación gráfica a la descripción por medio de palabras y formalismos de un lenguaje ensamblador. Nos adecuamos al estándar de Windows por que es un ambiente al cual nos podemos familiarizar sin problemas.

El uso de tcnología **PLD**, implica que los sistemas digitales diseñados en base a estos, sean de un costo de fabricación muy bajo, así como la disminución en cuanto al tamaño físico y aumento en la integración del mismo, además de que estos dispositivos **PAL** o **GAL** son fáciles de programar y de bajo costo.

El sistema está orientado al empleo de dispositivos **PAL** o **GAL** (serie 20), debido a que proporcionan una reducción considerable de un circuito lógico y de los cuales se tienen ventajas como: reducción física del número de elementos de circuitos integrados, reducción de tiempo de respuesta al sustituii elementos **TTL** por dispositivos **PLD**. Cabe mencionar que tenemos la opción de respaldar nuestra información e imprimir los archivos editados o de simulación.

SimPAL es una alternativa en el área de diseño que facilita la tarea en sistemas digitales, desde el diseño mismo hasta la prueba, edición, verificación y simulación, de esta manera se tiene como aportación un sistema integral de diseño para dispositivos lógicos programables.

Esperamos que a los compañeros de la facultad de Ingeniería les resulte útil el sistema para la elaboración de sus proyectos académicos y profesionales.

APENDICES

APENDICE
NOTACIONES ESPECIALES Y DEFINICIONES

Convenciones de notación

Los ejemplos de este documento utilizan la forma tradicional *Backus-Naur* (**BNF**) que define la sintaxis para los formatos de tranferencia de datos. **BNF** es una notación corta que se puede manejar con las siguientes reglas:

- "**: : =**" denota "*es definido como*"
- los caracteres encerrados con apóstrofe significan algún acotamiento de literales (requerida)
- Los picoparentesis "**<>**" encierran identificadores
- Los paréntesis cuadrados o corchetes "**[ ]**" encierran identificadores opcionales
- Los paréntesis curvos "**( )**" pueden aparecer repetidas veces. Cada vez puede aparecer cero o más veces.
- La barra vertical " **|** " indica un cambio de identificador.
- Repetidas veces viene un subfijo denotado por : n. Por ejemplo, seis numero dígitos pueden estar definidos como " **< number > : : = < digit > : 6.**"

sintaxis BNF :

< full name > : : = {< title >} < f. name > {< m. name >} < l. name >
< title > : : = ' Mr. ' | ' Mrs. ' | Ms. ' | ' Miss ' | ' Dr. '

Reglas BNF y definiciones

En el estándar siguiente las definiciones utilizadas para el resto del documento son :

< digit > : : = '0' | '1' | '2' | '3' | '4'
'5' | '6' | '7' | '8' | '9'

< hex-digit > : : = *< digit >*
| 'A' | 'B' | 'C' | 'D' | 'E' | 'F'

< binary-digit > : : = '0' | '1'
< number > : : = *< digit > { < digit > }*
< del > : : = *< space > | < carriage return >*
< delimiter > : : = *< del > { < del > }*
< printable character > : : = *< ASCII 20 hex . . . 7E hex >*
< control character > : : = *< ASCII 00 hex . . . IF hex >*
| *< ASCII 7F hex >*
< STX > : : = *< ASCII 02 hex >*

< ETX > : : = *< ASCII 03 hex >*
< carriage return > : : = *< ASCII 0D hex >*
< line feed > : : = *< ASCII 0A hex >*
< space > : : = *< ASCII 20 hex >*

< valid character > : : = *< printable character >*
| *< carriage return > | < line feed >*

< field character > : : = *< ASCII 20 hex . . . 29 hex >*
| *< ASCII 2B hex . . . 7E hex >*
| *< carriage return > | < line feed >*

7.11 Hexadecimal 7-Segment Display Encoder (Continued)

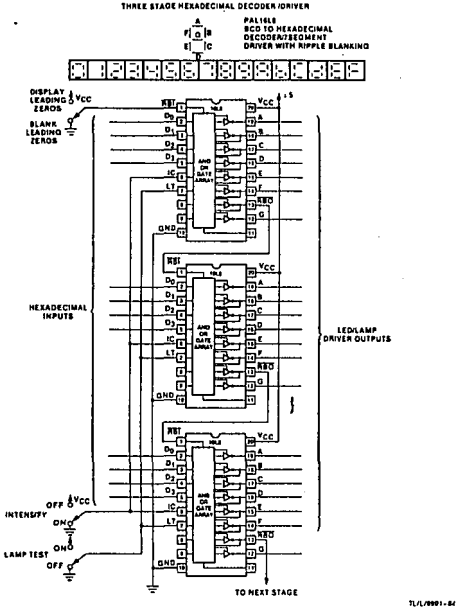


FIGURE 7.11.1. Hex Display Decoder-Driver Combinational Logic Diagram

7.11 Hexadecimal 7-Segment Display Encoder (Continued)

```
title   7-segment display encoder
pattern ENC
revision A
author  Iarif Engineer
company National Semiconductor Corporation
Date    10/28/1989

chip ENC PAL16L8

; pin 1    2    3    4    5    6    7    8    9    10
    /A0  D0   D1   D2   D3   IC   LT   NC   NC   GND
; pin 11   12   13   14   15   16   17   18   19   20
    NC   B    /BB0 F    E    D    C    B    A    VCC
equations
    /A  =  /A80 * /D0 * /D2
         + /A80 * /D1 * D3
         + /A80 * D1 * D2
         + /A80 * D1 * D2 * /D3
         + /A80 * D0 * D2 * /D3
         + /A80 * /D1 * /D2 * D3 * LT

    A.TEST = /IC

    /B  =  /A8C * /D2 * /D3
         + /A8C * D2 * /D3
         + /A80 * D0 * D1 * /D3
         + /A80 * D0 * /D1 * D3
         + /A80 * D0 * D1 * /D3 * LT

    B.TEST = IC

    /C  =  /A80 * D0 * /D1
         + /A80 *  D0 * /D2
         + /A80 * D1 * /D2
         + /A80 * D2 * /D3
         + /A80 * /D2 * D3 * LT

    C.TEST = IC

    /D  =  /A80 * /D1 * D3
         + /A80 *  /D0 * /D2 * /D3
         + /A8C * D0 * D1 * /D2
         + /A80 * /D0 * D1* D2
         + /A8D * D0  * /D1 * D2 * LT

    D.TEST = IC

    /E  =  /A8C * /D0 * /D2
         + /A80 *  D2 * D3
         + /A80 * /D0 * D1
         + /A80 * D1 * D3

    E.TEST = IC

    /F  =  /A80 * /D0 * /D1
         + /A80 * /D2 * D3
         + /A80 * D1 * D3
         + /A8D * /D0 * D2
         + /A8D * /D1 * D2 * /D3 * LT

    F.TEST = IC

    /G  =  /A8D * D1 * /D2
         + /A8I * DC * D3
         + /A80 * /D2 * D3
         + /A80 * /D0 * D1
         + /A80 * /D1 * D2 * /D3 * LT

    G.TEST = /IC

    BB0 = /D0 * /D1 * /D2 * /D3 * /BB1
    BB0.TEST = VCC
; end of ENC
```

TL/L/8981-84

7.11 Hexadecimal 7-Segment Display Encoder (Continued)

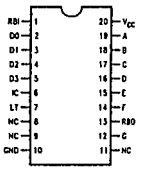
```
PAL16L8
title   7-segment display encoder
pattern ENC
evision A
author  Iarif Engineer
company National Semiconductor Corporation
Date    11/28/1989
*
QF2048*QP20*FD*
L0000
111111111111111101111111111111
101111111101111111111111101111
101111111111011111111111011111
111101111011111111111111011111
111011110111111111111111011111
111111101110111111111111011111
111110111011101111111111011111
111111111111111111101111111111*
L0256
111111111111111101111111111111
111111111011111111111111011111
101111111011111111111111101111
101111011111011111111111011111
011111011111011111111111011111
011110111111011111111111011111
111111111111111111101111111111
000000000000000000000000000000*
L0512
111111111111111101111111111111
011110111111111111111111011111
011111111011111111111111011111
111110111011111111111111011111
111111101111011111111111011111
111111111111111111101111111111
000000000000000000000000000000*
L0768
111111111111111101111111111111
111110111111011111111111011111
101111111101111111111111011111
011101111101111111111111011111
101101111111111111111111011111
011110111111111111111111011111
111111111111111111101111111111
000000000000000000000000000000*
                                TL/L/9981-Q6
L1024
111111111111111101111111111111
101111111101111111111111011111
111111101111011111111111011111
101110111111111111111111011111
111101111101111111111111011111
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000*
L1280
111111111111111101111111111111
101111011111111111111111011111
111111111101101111111111011111
111101111111011111111111101111
101111111011111111111111011111
111101101111011111111111011111
111111111111111111101111111111
000000000000000000000000000000*
L1536
111111111111111101111111111111
100111011101111111111111101111
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000*
L1792
111111111111111101111111111111
111101111011111111111111011111
011111111111011111111111011111
111111111101101111111111011111
101101111111111111111111011111
111110111011101111111111011111
111111111111111111101111111111
000000000000000000000000000000*
CC1F9*
QQQQ
                                TL/L/9981-61
```

7.11 Hexadecimal 7-Segment Display Encoder (Continued)

```
Document file for ENC.inp
Device: 16L8

Pin   Label                 Type
---   -----                 ----
1     RBI                   com input
2     D0                    com input
3     D1                    com input
4     D2                    com input
5     D3                    com input
6     IC                    com input
7     LT                    com input
8     NC                    unused
9     NC                    unused
10    GND                   ground pin
11    NC                    unused
12    G                     neg,trst,com output
13    RBO                   neg,trst,com feedback
14    F                     neg,trst,com output
15    E                     neg,trst,com output
16    D                     neg,trst,com output
17    C                     neg,trst,com output
18    B                     neg,trst,com output
19    A                     neg,trst,com output
20    VCC                   power pin
```

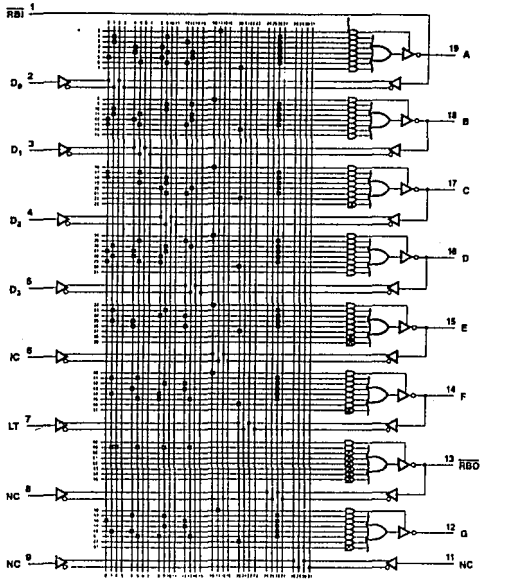
TL/L/9991-62

Chip Diagram (DIP)



TL/L/9991-60

7.11 Hexadecimal 7-Segment Display Encoder (Continued)



TL/L/5991-72

FIGURE 7.11.2. PAL16L8 Logic Diagram Showing Lamp Driver Pattern

While generating the JEDEC file, the compiler may find errors in your program. When it does, it describes what seems to be wrong on the computer display. The error text completely describes the problem, so there is no need for you to look up error codes in the manual. Another file called NAND.LST is also created which allows you to read the NAND3.JED file. The part of the file which describes pin19 of the 16R4 looks like this:

```
proLogic Compiler (Fuse Plot A100) V2.00
Serial bxaw0
Copyright (C) 1988 INLAB, Inc.

p16r4 revision 89.2.11

                  11 1111 1111 2222 2222 2233
     0123 4567 8901 2345 6789 0123 4567 8901

  0 ---- ---- ---- ---- ---- ---- ---- ----   OE
  1 X--- X--- X--- ---- ---- ---- ---- ----   +
  2 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +
  3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +   !pin19
  4 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +
  5 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +
  6 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +
  7 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX   +

     |  |  |  |  |  |  |  |  |  |  |  |  |
  pin2  3  4  5  6  7  8  9
     |  |  |  |  |  |  |  |  |  |  |  |  |
     pin1   18   17   16   15   14   13   11

     Legend:

   X  : Cell intact      (JEDEC 0)
   -  : Cell programmed (JEDEC 1)

   X- : True input term
   -X : Complement input term
    XX : Any XX pair in a product term yields product term
LOW.
    -- : No input term (don't care).  A product term com-
prised
        entirely of -- yields product term HIGH.
```

Compare the NAND3.LST file to Figure 5. This format, called a fuse plot, is just a printer oriented version of the logic diagram. It has some new words in it – an input term is another word for a signal; a product term is another word for AND gate. Other than that, the fuse plot is about the same as a logic diagram.

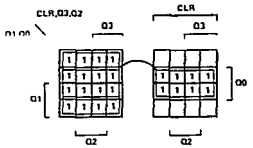
4-Bit Binary Counter Details

The same basic procedure used in determining the equations for the clock selector we are dealing with a present state, next state situation. This means a D-type flip-flop will be required in actual circuit implementation. As before, the truth table is generated first and is shown in Table 3.

Table 3. Truth Table

	PRESENT STATE				NEXT STATE			
CLR	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	X	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	1	1
1	0	0	1	1	0	1	0	0
1	0	1	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0
1	0	1	1	0	0	1	1	1
1	0	1	1	1	1	0	0	0
1	1	0	0	0	1	0	0	1
1	1	0	0	1	1	0	1	0
1	1	0	1	0	1	0	1	1
1	1	0	1	1	1	1	0	0
1	1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	1	0
1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	0	0

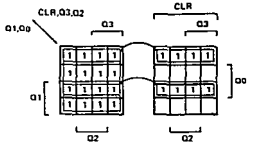
From the truth table, the equations for each output can be derived from the Karnaugh map. This is shown in Figure 16. Note that the inverse of the truth table is being solved so that the equation will come out in AND-NOR logic form.



$\overline{Q0} = \overline{CLR}\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + CLR\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,Q0$

$\overline{Q0} = \overline{CLR} + Q0$

(a) Karnaugh Map for $\overline{Q0}$

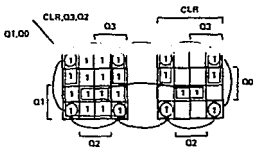


$\overline{Q1} = \overline{CLR}\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + CLR\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,Q0 + CLR\,\overline{Q3}\,\overline{Q2}\,Q1\,\overline{Q0} + CLR\,\overline{Q3}\,\overline{Q2}\,Q1\,Q0$

$\overline{Q1} = \overline{CLR} + \overline{Q1}\,\overline{Q0} + Q1\,Q0$

(b) Karnaugh Map for $\overline{Q1}$

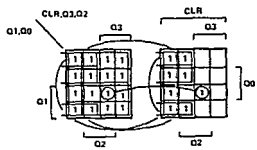
Figure 16. Karnaugh Maps



$\overline{Q2} = \overline{CLR}\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + Q3\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + Q3\,\overline{Q2}\,Q1\,Q0 + Q3\,\overline{Q2}\,Q1\,\overline{Q0}$

$\overline{Q2} = \overline{CLR} + \overline{Q3}\,\overline{Q1} + Q3\,Q1\,Q0 + \overline{Q3}\,\overline{Q0}$

(c) Karnaugh Map for $\overline{Q2}$



$\overline{Q3} = \overline{CLR}\,\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + Q3\,\overline{Q2}\,\overline{Q1}\,\overline{Q0} + Q3\,\overline{Q2}\,Q1\,\overline{Q0} + Q3\,\overline{Q2}\,Q1\,Q0$

$\overline{Q3} = \overline{CLR} + \overline{Q3}\,\overline{Q2} + \overline{Q3}\,\overline{Q1} + \overline{Q3}\,\overline{Q0} + Q3\,Q2\,Q1\,Q0$

(d) Karnaugh Map for $\overline{Q3}$

Figure 16. Karnaugh Maps (Continued)

Binary/Decade Count Details

Recalling from the example requirements that the counter should count in decade [illegible]

Since the counter is already designed to count in binary, we can use this feature to simplify our design. What we desire is a circuit whose output goes low, whenever the BD input is equal to a logic level "0", and the counter output is equal to "9". This output can then be fed back to the CLR input of the counter so that it will reset whenever the BD input is low. Whenever the BD input is high, the output of the circuit should be a high since the counter will automatically count in binary. Notice that $\overline{Q}$ shown in the truth table is the function we desire.

Table 4. Truth Table

BD	Q3	Q2	Q1	Q0	Q	$\overline{Q}$	BD	Q3	Q2	Q1	Q0	Q	$\overline{Q}$
0	0	0	0	0	0	1	1	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	0	1	0	0	1
0	0	0	1	1	0	1	1	0	0	1	1	0	1
0	0	1	0	0	0	1	1	0	1	0	0	0	1
0	0	1	0	1	0	1	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	1	1	0	0	1
0	0	1	1	1	0	1	1	0	1	1	1	0	1
0	1	0	0	0	0	1	1	1	0	0	0	0	1
0	1	0	0	1	0	1	1	1	0	0	1	0	1
0	1	0	1	0	0	1	1	1	0	1	0	0	1
0	1	0	1	1	0	1	1	1	0	1	1	0	1
0	1	1	0	0	0	1	1	1	1	0	0	0	1
0	1	1	0	1	0	1	1	1	1	0	1	0	1
0	1	1	1	0	0	1	1	1	1	1	0	0	1
0	1	1	1	1	0	1	1	1	1	1	1	0	1

In this particular example, a Karnaugh map is not required because the equation cannot be further simplified. The resulting equation is given below.

$\overline{BD\ OUT} = \overline{BD}\,Q3\,\overline{Q2}\,\overline{Q1}\,Q0$

Fuse Map Details

Now that the logic equations have been defined, the next step will be to specify which fuses need to be programmed. Before we do this however, we first need to label the input and output pins on the device by using figure 16 as a guide. We can make the following pin assignments in Figure 17.

PIN:

1 CLK	20 V_{CC}
2 SEL0	19 CLKOUT
3 SEL1	18 NC
4 CLKA	17 Q0
5 CLKB	16 Q1
6 CLKC	15 Q2
7 CLKD	14 Q3
8 CLR	13 NC
9 BD	12 BD OUT
10 GND	11 OE

With this information defined, we now need to insert the logic equations into the logic diagram as shown in Figure 17.

It is now probably obvious to the reader, that inserting the logic equations into the logic diagram is a tedious operation. Fortunately, several software programs are available to perform this task automatically. All that is required is telling the program which device has been selected and defining the input and output pins with their appropriate logic equations. The program will then generate a fuse map for the device selected. This information can then be down loaded into the selected device programmer.

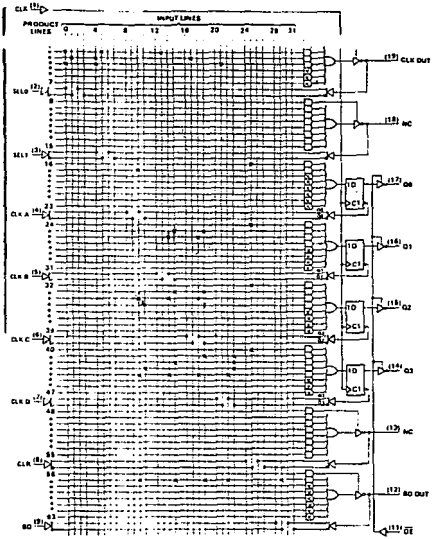


Figure 17. Programmed TIBPAL16R4

PLD Design Software

Software packages such as ABEL™, CUPL™, and proLogic™ not only generate the fuse maps, but they also help in developing the logic equations. In most cases, these packages use logic equations from simply providing the program with either a truth table or state diagram. In addition, they can test the logic equations against a set of test vectors. This helps to ensure the that designer gets the desired function.

Several software packages are described in further detail in this data book. The Programmable Logic Device Design Software Support section provides a detailed summary of the capabilities of several of these popular design tools.

As an example, we will approach our previous design utilizing DATA I/O's ABEL™ package. The purpose here is not to teach the reader how to use ABEL™, but rather to give them a basic overview of this powerful software package. Figure 18 shows the source file required by ABEL™. Note that the 4-bit counter has been described with a state diagram table. When the ABEL™ program is compiled, the logic equations will be generated from this. The equations for CLK OUT and BD OUT are given in their final form to demonstrate how ABEL™ will handle these. Also notice that test vectors are included for checking the logic equations. This is especially important when only the logic equations are given.

Figure 19 shows some of the output documentation generated by the program. Notice that the equations generated for the counter match the ones generated by the Karnaugh maps. A pinout for the device has also been generated and displayed. The fuse map for the device has not been shown; however, the standard JEDEC fuse map thus generated can be down loaded into the device programmer to program the selected PLD.

ABEL is a trademark of Data I/O Corporation
CUPL is a trademark of LOGICAL DEVICES, INC.
proLogic is a trademark of Itdah Inc.


```
module BD_COUNT flag 'r-2'
title '4-bit binary/decade counter

" pin assignments and constant declarations
     CLK_IN,SELO,SEL1,CLKA    pin  1,2,3,4;
     CLKB,CLKC,CLKD           pin  5,6,7;
     CLR,BD_IN,OE             pin  8,9,11;
     BD_OUT,CLK_OUT           pin  12,19;
     Q3,Q2,Q1,Q0              pin  14,15,16,17;
     OK, L, H, X, Z =         .C., 0, 1, .X., .Z.;
     OUTPUT                   =    [Q3,Q2,Q1,Q0];

" counter states
     S0="b0000;   S4="b0100;   S8="b1000;   S12="b1100;
     S1="b0001;   S5="b0101;   S9="b1001;   S13="b1101;
     S2="b0010;   S6="b0110;   S10="b1010;  S14="b1110;
     S3="b0011;   S7="b0111;   S11="b1011;  S15="b1111;

equations
" clock selector
     CLK_OUT = CLKA & !SEL0 & !SEL1 # CLKB & !SEL0 & SEL1
             # CLKC & SEL0 & !SEL1 # !SEL0 & CLKD & SEL1 & SEL0;

" count mode indicator for decade counting
     BD_OUT = !SEL0 & !SEL1 & Q3 & !Q2 & !Q1 & Q0;

state_diagram [Q3,Q2,Q1,Q0]
     State  S0:  IF CLR == 0 THEN S0 ELSE  S1;
     State  S1:  IF CLR == 0 THEN S0 ELSE  S2;
     State  S2:  IF CLR == 0 THEN S0 ELSE  S3;
     State  S3:  IF CLR == 0 THEN S0 ELSE  S4;
     State  S4:  IF CLR == 0 THEN S0 ELSE  S5;
     State  S5:  IF CLR == 0 THEN S0 ELSE  S6;
     State  S6:  IF CLR == 0 THEN S0 ELSE  S7;
     State  S7:  IF CLR == 0 THEN S0 ELSE  S8;
     State  S8:  IF CLR == 0 THEN S0 ELSE  S9;
     State  S9:  IF CLR == 0 THEN S0 ELSE S10;
     State S10:  IF CLR == 0 THEN S0 ELSE S11;
     State S11:  IF CLR == 0 THEN S0 ELSE S12;
     State S12:  IF CLR == 0 THEN S0 ELSE S13;
     State S13:  IF CLR == 0 THEN S0 ELSE S14;
     State S14:  IF CLR == 0 THEN S0 ELSE S15;
     State S15:  IF CLR == 0 THEN S0 ELSE  S0;

test_vectors   'clock selector'
     ([CLKA, CLKB, CLKC, CLKD, SEL1, SEL0] -> CLK_OUT)
     [ L  ,  X  ,  X  ,  X  ,  L  ,  L ] ->  L;
     [ H  ,  X  ,  X  ,  X  ,  L  ,  L ] ->  H;
     [ X  ,  L  ,  X  ,  X  ,  L  ,  H ] ->  L;
     [ X  ,  H  ,  X  ,  X  ,  L  ,  H ] ->  H;
     [ X  ,  X  ,  L  ,  X  ,  H  ,  L ] ->  L;
     [ X  ,  X  ,  H  ,  X  ,  H  ,  L ] ->  H;
     [ X  ,  X  ,  X  ,  L  ,  H  ,  H ] ->  L;
     [ X  ,  X  ,  X  ,  H  ,  H  ,  H ] ->  H;
```

Figure 18. Source File for ABEL

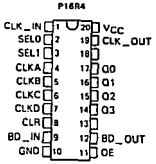
```
test_vectors   'counter'
      ([CLK_IN, OE, CLR, BD_IN] -> [OUTPUT, BD_OUT])
      [ .CK.   L .  L.    X  ] -> [  $0.   H ];
      [ .CK.   L .  H.    X  ] -> [  $1.   H ];
                                  .
                                  .
      [ .CK.   L .  H.    X  ] -> [  $9.   H ];
      [ .CK.   L .  H.    X  ] -> [  $A.   H ];
      [ .CK.   L .  H.    X  ] -> [  $5.   H ];
      [ .CK.   L .  H.    X  ] -> [  $6.   H ];
      [ .CK.   L .  H.    X  ] -> [  $7.   H ];
      [ .CK.   L .  H.    X  ] -> [  $8.   H ];
      [ .CK.   L .  H.    L  ] -> [  $9.   L ];
      [ .CK.   L .  H.    X  ] -> [  $10.  H ];
      [ .CK.   L .  H.    X  ] -> [  $11.  H ];
      [ .CK.   L .  H.    X  ] -> [  $12.  H ];
      [ .CK.   L .  H.    X  ] -> [  $13.  H ];
      [ .CK.   L .  H.    X  ] -> [  $14.  H ];
      [ .CK.   L .  H.    H  ] -> [  $15.  H ];
      [ .CK.   H .  H.    X  ] -> [  $0.   H ];
      [  Z.    H .  X.    X  ] -> [  Z     H ];
end BD_COUNT
```

Figure 18. Source File for ABEL (Continued)


```
ABEL(tm) Version  1.00  - Document Generator
4-bit binary/decade counter

Equations for Module BD_COUNT

Device IC1

    Reduced Equations:

     CLK_OUT = !((SEL1 & SEL0 & !CLKD
               # (SEL1 & !SEL0 & !CLKC
               # (!SEL1 & SEL0 & !CLKB
               # !SEL1 & !SEL0 & !CLKA))));


     BD_OUT = !(!Q3 & !Q2 & !Q1 & Q0 & !BD_IN);

     OE = !((Q3 & Q2 & Q1 & Q0
           # (!Q3 & !Q2
           # (!Q3 & !Q1
           # (!Q3 & !Q0
           # !CLR))));


     Q2 := !((Q2 & Q1 & Q0 # (!Q2 & !Q1 # (!Q2 & !Q0 # !CLR))));

     Q1 := !((Q1 & Q0 # (!Q1 & !Q0 # !CLR)));

     Q0 := !((Q0 # !CLR));
```

Figure 19. ABEL Output Documentation


```
ABEL(tm) Version  1.00  - Document Generator
4-bit binary/decade counter

Device IC1
```




```
end of module BD_COUNT
```

Figure 19. ABEL Output Documentation (Continued)

Reference

1. H. Troy Nagle, Jr., B.D. Carroll, and David Irwin, *An Introduction to Computer Logic*. New Jersey: Prentice-Hall, Inc., 1975.

EXAMPLES

[illegible] to the [illegible] of PALASM as a tool to aid in the development of PAL [illegible] designs were generated by [illegible] a technique which depended upon fuse maps. [illegible] technique the designer was required to actually [illegible] the individual fuses which needed to be blown to [illegible]

[illegible] a firm foundation in physical reality of how a PAL implements a function.

As a demonstration of this [illegible] the arbitrary circuit shown in Figure 2 is an example for selecting and coding a PAL. This example shows the technique of [illegible] a PAL once the logic [illegible] has been developed. Again, this technique is no longer used, as it is much easier to simply generate the equations directly as input to PALASM.

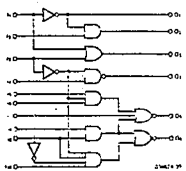


Figure 2. Design Example, Logic Diagram

The first step is to generate the Boolean equations for this function. These are [illegible] from the logic diagram. It should be noted that [illegible] to use meaningful pin names rather than [illegible]. This will help in providing good design documentation.

The following symbols will be used for all logic equations

* = AND   + = OR   / = NOT [illegible]

Boolean equations for Figure 2 are:

01 = /I1

02 = [illegible]

03 = I1 + I3

04 = /(/I3*I4)

05 = /(/I3*I5*I6 + I7 + I8*I9)

06 = /(I8*I9 + /I3*/I7*I9*I10)

The next step is to select the particular PAL we want to use for this function. Since no registers are required, we should select from the combinatorial devices (AmPAL16L8, AmPAL16H8, AmPAL16LD8 or AmPAL16HD8). Since 3 outputs have AND-OR functions and 3 outputs have AND-OR-INVERT functions, we could still select from either active HIGH or active LOW (H or L) parts, but since the more complex functions are AND-OR-INVERT, the active LOW (L) series is most likely. Finally, we see that no output enable is required, thus we could use either the AmPAL16L8 or the AmPAL16LD8. For the purposes of this example we will select the AmPAL16L8.

Now, since we have selected an AmPAL16L8 (which has inverting outputs) we need to apply DeMorgan's theorem to convert these equations from the active HIGH to the active LOW output. DeMorgan's theorem can be used to convert any logic equation in any form into the AND-OR structure used in PALs. Applying DeMorgan's theorem gives the active LOW form of the equation

/01 = I1

/02 = [illegible]

/03 = /I1*/I3

/04 = /I3*I4

/05 = /I3*I5*I6 + I7 + I8*I9

/06 = [illegible]I8*I9 + /I3*/I7*I9*I10

[illegible] for the PAL to perform this function. Figure 3 shows the conventions which are used when coding fuse maps.

Figure 4 shows the logic diagram of the AmPAL16L8. We will assign outputs O_1-O_6 to pins 14-19, and inputs I_1-I_{10} to pins 2-9, 11 and 13.

O_1 is assigned to pin 19. To make this output the inverse of I_1 [illegible]

[illegible] dicated by the X at the intersection of input line 0 and product term 1 in Figure 4.

Since the other inputs to the OR gate are unused, they are forced to zero by leaving all the fuses intact on product terms [illegible] the [illegible] inputs.

The [illegible] consideration for O_1 is the output enable. By referring back to the logic diagram for the circuit (Figure 2) or the Boolean equation, we see that there is no output enable function for O_1 [illegible] O_1 should [illegible] enabled.

[illegible] the output enable function for O_1. To have the output always enabled, we blow all the fuses in product term [illegible]. A product term with all fuses blown is always HIGH, so this will leave the three-state gate always on, and the output will always be enabled.

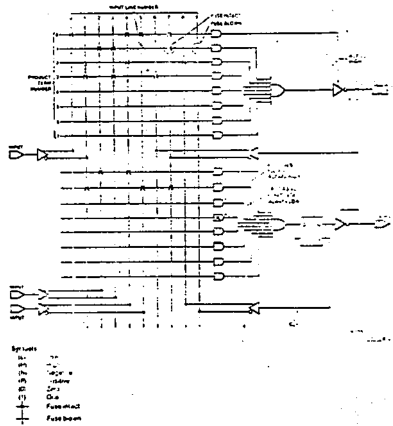


Figure 3. Coding Conventions

S1	S0	SERIAL LEFT	SERIAL RIGHT	Q_3	Q_2	Q_1	Q_0	SERIAL RIGHT	SERIAL LEFT
0	0	X	X	Q_3	Q_2	Q_1	Q_0	X	X
H	0	X	X	0	Q_3	Q_2	Q_1	Q_0	X
H	H	X	X	L	Q_3	Q_2	Q_1	Q_0	X
L	H	X	X	Q_2	Q_1	Q_0	0	X	Q_3
H	H	X	X	Q_2	Q_1	Q_0	H	X	Q_3
H	H	X	X	Q_2	Q_1	Q_0	L	X	Q_3

Figure 16. Shift Register Function Table

Figure 17. 4-bit Bidirectional Shift Register

```
PAL16R6                                          [illegible]
PAL-3                                            [illegible]
SHIFT REGISTER
ADVANCED MICRO DEVICES
/S   S1   S0   D3   D2   D1   D0   SR   SL   CLK

;SHIFT REGISTER OUTPUT SIGNALS

/Q3   := /S1*/S0*/D3          +
         /S1* S0*/SRISLO      +
          S1*/S0*/Q2          +
          S1* S0*/Q3

/Q2   := /S1*/S0*/D2          +
         /S1* S0*/Q3          +
          S1*/S0*/Q1          +
          S1* S0*/Q2

/Q1   := /S1*/S0*/D1          +
         /S1* S0*/Q2          +
          S1*/S0*/Q0          +
          S1* S0*/Q1

/Q0   := /S1*/S0*/D0          +
         /S1* S0*/Q1          +
          S1*/S0*SRISRO       +
          S1* S0*/Q0

IF(/S1*/S0) /SLISRO = /Q0

IF(S1*/S0) /SRISLO = /Q3
```

SHIFT REGISTERS

Shift registers are useful for multiplication, division and serial communications.

An easy way to construct a shift register is to think of it as [illegible]

right, shift data to the left, and "hold" (leave unchanged) the data. There would be two select lines that control the multiplexers and choose the desired function. If data is to be loaded into the shift register, this path is enabled and the data is passed through and clocked into the registers (see Figure 15a). Figures 15b and 15c illustrate how the shift functions are performed. When shifting right, bit i will clock in bit i + 1. This is accomplished by feeding back the output of register i + 1 to the multiplexer of bit i and selecting this path. Similarly, shifting left is done by clocking bit i − 1 into bit i using similar feedback techniques. "Holding" data is easily done by feeding back register I to itself and selecting the path, as shown in Figure 15d. Thus keeping data unchanged is accomplished by feedback, not by gating the clock (which

This information is now used to design a 4-bit shift register using an AmPAL16H8. The function has four data inputs, four outputs, and two select lines, a left serial input, a right serial input, an output enable, and a clock input. The serial inputs are bidirectional and are used to input new data while shifting. The function table for the shift register is shown in Figure 16 and the logic diagram in Figure 17. The Design Specification and Logic Diagram for these shifters are shown in Tables 3(a) and 3(b).

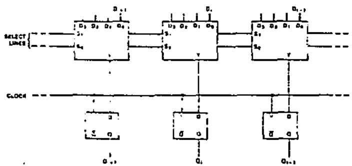


Figure 15a. Loading the Shifter

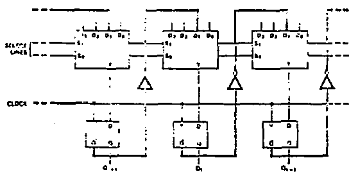


Figure 15b. Shifting Right

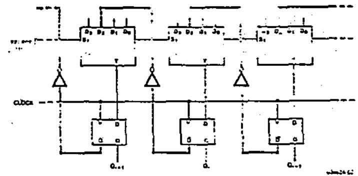


Figure 15c. Shifting Left

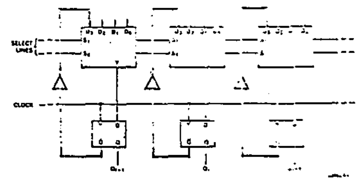
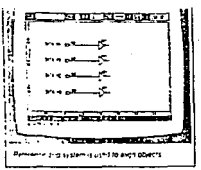


Figure 15d. "Holding" Data

This means that all symbols and lines can be moved, and yet the connection lines retain 90-degree angles as they move to maintain the connectivity of the schematic. This allows the designer to "clean up" a design and make first-class drafting quality schematics.



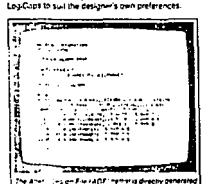
Removing and replacing symbols and objects

ALTERA PRIMITIVE SYMBOL LIBRARY

Over 50 logic and I/O symbols provide the basic building blocks for logic schematics. Familiar names like "AND2" and "XOR" identify the logic symbols and mnemonic symbols like "PORT" (Register Output, Negative Pre-Load) define the EPLD I/O architecture configurations. See figure on page 6.

USER DEFINABLE MACROS

Frequently used command sequences can be saved by the user as macro recordings. These recordings may then be executed by a single keypress. This speeds design entry and allows customization of LogCaps to suit the designer's own preferences.



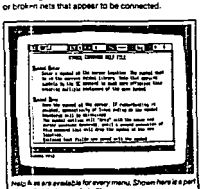
The Altera design file (ADF) netlist is directly generated by LogCaps, ready for processing by A+PLUS.

positioning the cursor on the object and pressing a button. An outline of the object can then be moved to the new location where another press of a button will move the object there. A different button can be used for making copies of the object if that is desired.

Whole areas of a schematic can be moved, copied, erased, saved, or loaded. This makes construction of repetitive sections of a design a snap. Through this technique a designer can build up his own library of sub-circuit functions that may be used in many different EPLD designs.

"WHAT YOU SEE IS WHAT YOU GET" DESIGN PHILOSOPHY

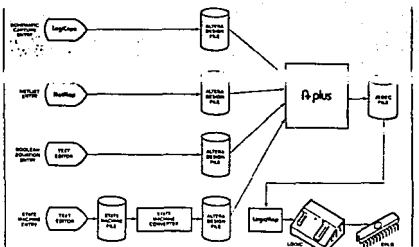
LogiCaps derives the netlist directly from the graphical drawing database. There are no hidden connections or broken nets that appear to be connected.



Help A is are available for every menu. Shown here is a part of the *Symbol* menu help file.

EXTENSIVE ON-LINE DOCUMENTATION

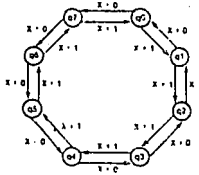
Every menu includes a HELP function with explanation of all selections available in that menu. The primary menu HELP text also explains key functions and other aspects of the system.



LogiCaps operates in conjunction with, and is a natural enhancement to the Altera Programmable Logic User System (A+PLUS). It provides a complete path from the engineer's design concept to a programmed EPLD.

Window	Drawing	Symbol	Line	Text	Area	Macro	Quit	Help
MoveXY	Load	Enter	Enter	Enter	Boundary	Record		
Pan	Write	Move	Move	Move	Move	Play		
Tag	Delete	Copy	Copy	Copy	Copy	Stop		
Recall	Size	Delete	Delete	Delete	Delete	Assign		
Zoomset	Files	Reflect	Join	Select	Toggle	Clear		
Split	ADF	List	Select	Find	Load	Files		
Grid	Help	Find	Bend	Import	Write	Help		
Auto		Numbers	Rubberband	Bolders	Files			
Color		Help	Help	Help	Help			
Help								

LogiCaps Main Command Menu (across the top) and sub-menus (columns) under selections. All selections are made by a single keypress (first letter of name) and each menu includes a HELP selection.

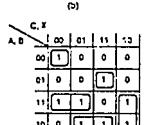


(a)

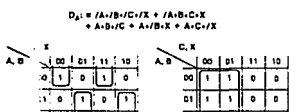
PRESENT STATE	NEXT STATE X = 0	NEXT STATE X = 1
Q_7	Q_7	Q_1
Q_1	Q_0	Q_2
Q_2	Q_1	Q_3
Q_3	Q_2	Q_4
Q_4	Q_3	Q_5
Q_5	Q_4	Q_6
Q_6	Q_5	Q_7
Q_7	Q_6	Q_0

PRESENT STATE	NEXT STATE X = 0	NEXT STATE X = 1
000	111	001
001	000	010
010	001	011
011	010	100
100	011	101
101	100	110
110	101	111
111	110	000

(b)



$D_A = /A•/B•/C•/X + /A•B•C•X + A•B•/C + A•/B•X + A•C•/X$



$D_B = /B•C•/X + /B•C•X + B•C•/X$

$D_C = /C$

(c)

Figure 3-20. 3-Bit Up/Down Counter (a) State Diagram (b) State Transition Tables (c) Karnaugh Maps

We can easily implement this counter using a PAL device. The PAL device for this design will require at least three flip-flops. Since there is no PAL device available that has only three flip-flops, the best PAL device will be the PAL16R4, because it has four flip-flops and contains the AND-OR gates needed to realize the combinational circuit of the counter. The schematic of the 3-bit up-down counter using the PAL16R4 is shown on page 6-31. Note that there are plenty of unused inputs and outputs for other functions if desired.

Design of a binary counter can be complicated by adding new features to it. Counters can be made to count up or down, load new values in, or clear the present state of the counter and reset to 0's.

As we try to design bigger counters, using the state tables and K-maps becomes more difficult. In the design of the 3-bit counter, K-maps were used. If we try to design a counter bigger than this, summarizing the equations will be a very tedious process. Therefore, we try to find a general solution for solving the counter design problem. Let's try to write the equation for the most significant bit (MSB) of an n-bit binary counter (Q_n).

First we look at the case where the counter is counting up. The new value of Q_n will depend on the carry-in from bit Q_{n-1} into Q_n. If all less significant bits (LSBs) are high when we count up, we will have a carry-in from C_{n-1} into Q_n.

$$C_{n-1} = Q_{n-1}•Q_{n-2}•...Q1•Q0•UP$$

Now let's look at the following table:

UP	Q_n	CARRY-IN	NEW Q_n
H	L	L	L
H	L	H	H
H	H	L	H
H	H	H	L

Figure 3-21. Carry-in Table

Examining the above table, it is easily concluded that:

$Q_n := Q_n$:+: carry-into Q_n

$Q_n := Q_n$:+: $(Q_{n-1}•Q_{n-2}•...Q0•UP)$

Now that we have calculated the equation for the count-up case, let's look at the count-down case. Borrow-out from Q_n to Q_{n-1} will be high if all the LSBs are low and we are counting down.

$B_{OUT} := /Q_n - 1•/Q_n - 2• .../Q0•/UP$

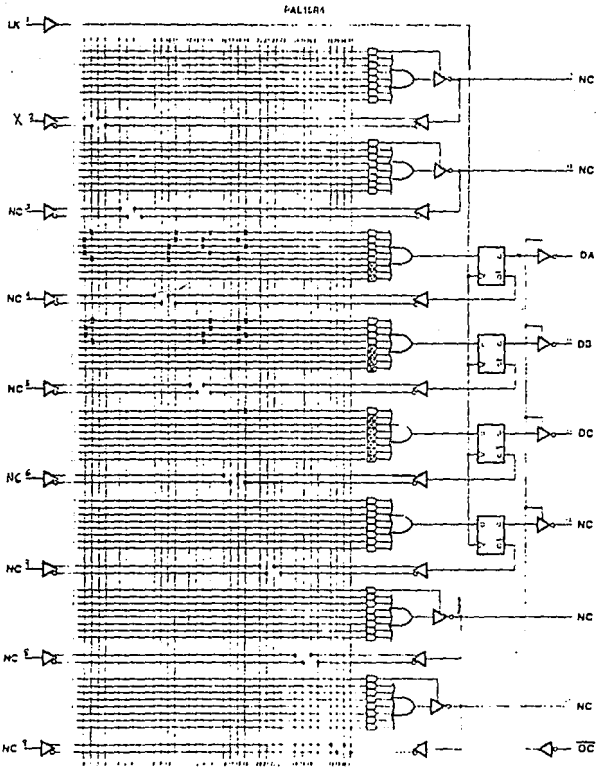
Let's look at the following table:

UP	Q_n	BORROW OUT	NEW Q_n
L	L	L	L
L	L	H	H
L	H	L	H
L	H	H	L

Figure 3-22. Borrow-out Table

Logic Tutorial

Diagram



So

C_n = Q_n + Borrow-out D_n

D_n = D_n + (/D_{n-1} + ... + D_1+D_0+UP)

Thus,

C_n = D_n + (/D_{n-1}+...+D_1+D_0+UP)

= D_n + /(/D_{n-1}+...+/$D1$+/$D0$+UP)

Another way to look at a counter is to formulate the following statement about the nth bit of the counter:

On clock hold UNLESS all lower order bits are HI and we are counting UP.

The "UNLESS" operation is implemented by an XOR gate. Thus the statement translates directly into

D_n = (D_{n-1}+...+D_1+D_0+UP)

This is, of course, the same equation we obtained by looking at the nth bit table. The expression for a down-counter can be found in the same way.

This formulation makes it easy to combine these two expressions into a single equation for an up/down counter.

On clock hold UNLESS:

All lower order bits are HI and counting UP

OR all lower order bits are LO and NOT counting UP.

This translates to

D_n = ... + (D_{n-1}+D_{n-2}+...+D_1+D_0+UP) + .../D_{n-1}+/D_{n-2}+.../D_1+/D_0+/UP) Eq. 5.1

Example 5.4

Using the solution discussed above, let's try to design an n-bit counter that can count up, count down, RESET and LOAD new values into the counter. RESET overrides LOAD, count and HOLD. LOAD overrides count. RESET sets all of the Qs to 0.

Solution

The above operations are summarized below:

RESET	LOAD	HOLD	UP	D	Q	OPERATION
H	X	X	X	X	X	Set all LOW
L	L	X	X	D	D	Load D
L	H	H	X	X	Q	Hold Q
L	H	L	L	X	Q plus 1	Increment
L	H	L	H	X	Q minus 1	Decrement

We will try to generate a general equation that can be used for any bit in the counter. If we look at the nth bit, we have D_n as input and Q_n as output. There are five operations that can happen for any given bit: LOAD, HOLD, RESET, count up or count down.

If RESET is HI, then all other operations are overridden. Overriding the other operations is sufficient to provide a LO into the flip-flop. So every product term will contain /RESET. We LOAD the new value in the counter's register if RESET is OFF (/RESET = HI). Since Q_n is replaced by D_n if (/RESET = HI, LOAD = HI) the expression that will allow loading the new value will be: /RESET•LOAD•D_n. In order to be able to HOLD the Q_n value, we should have the following conditions: (RESET = L, LOAD = L, HOLD = HI). So the expression for holding the old value is:

/RESET•/LOAD•HOLD•Q_n

There are two more functions for the counter: count up and count down. These two functions have been calculated in Eq. 5.1. Using this equation and the calculation for the HOLD and LOAD cases, the final equation for the nth bit is

D_n = /RESET•LOAD•D_n Eq. 5.2

+ /RESET•/LOAD•/D_n

+ /RESET•/LOAD•HOLD•D_n•Q_n•C_{n-1}

+ /RESET•/LOAD•HOLD•/UP•Q_n•/C_{n-1}•.../Q_{n-1}

Using the above general equation, any large counter can be designed.

GLOSARIO

GLOSARIO A
(Acronimos de PLD)

ABEL Advanced Boolean Expression Languaje (*Lenguaje Avanzado de Expresiones Booleanas*). Lenguaje universal para diseño de **PLD** con entrada y salida de datos con software de soporte.

AMAZE Automatic Map And Zap of Equations (*Autómata de Mapeo de ecuaciones*). Marca registrada por Signetics como herramientas de diseño de dispositivos **FPLA**.

CHMOS Complementary High-Speed Metal Oxide Semiconductor (*Semiconductor de óxido de metal complementario de alta velocidad*).

CMOS Complementary Metal Oxide Semiconductor (*Semiconductor de óxido de metal complemen tario*), tecnología caracterizada por requerimientos mínimos de potencia.

E^2 Eléctricamente borrable

EEPLD PLD eléctricamente borrable.

EEPROM PROM eléctricamente borrable.

EPROM PROM borrable. Se asume que es borrable mediante luz ultravioleta.

FPLA Field Programmable Logic Array (*Arreglo lógico de campo programable*). Dispositivo de Signetics donde se pueden programar los arreglos **AND** y **OR**.

FPLD Field Programmable Logic Device (*Dispositivo Lógico de Campo Programable*).

GAL Generic Array Logic (*Arreglo Lógico Genérico*). Dispositivo eléctricamente borrable que fueron diseñados para reemplazar los dispositivos tipo **PAL**.

JEDEC Joint Electron Device Engineering Council (*Concilio de Dispositivos de Juntura en Ingeniería Electronica*).Comité formado por miembros de la asociación de industrias eléctronicas quienes fijaron unos estándares arbitrarios para los dispositivos electrónicos y formatos de transferencia de datos.

MOS Metal Oxide Semiconductor (*Semiconductor de Oxido de Metal*). Uno de las dos formas básicas de un transistor (*la otra es bipolar*).

MSI Medium-Scale Integration (*Integración de media escala*).Circuito integrado que contiene entre doce y cien compuertas.

PAL Programmable Logic Array (*Arreglo Lógico Programable*). Dispositivo con arreglo programable **AND** y uno fijo de **OR**.

PALASM PAL Assembler (*Ensamblador de PAL*). Software de desarrollo para el diseño de **PAL** desarrollado por John Birkner.

PLA Programmable Logic Array (*Arreglo Lógico Programable*). Estructura para implementar suma de productos de funciones lógicas. Caracterizado por tener un arreglo programable de **AND** seguido por un arreglo programable de **OR**.

PLAN Programmable Logic Analysis by National (*Análisis de Lógica Programable por National*). Herramienta de diseño para **PLD** desarrollada por National Semiconductor.

PLD Programmable Logic Device (*Dispositivo Lógico Programable*). Circuito integrado para aplicaciones de lógica digital.

PLDS Programmable Logic Development System (*Sistema para desarrollo de Lógica Programable*). Cada sistema programable puede ser programado y ensamblado mediante **PALASM** para una aplicación específica.

PROM Programmable Read-Only Memory (*Memoria programable sólo de lectura*). Es un circuito integrado programable que contiene una suma de productos lógicos en un arreglo con el campo **AND** fijo y campo **OR** programable.

TTL Transistror-Transistor Logic (*Lógica Transistor Transistor*). Es una familia de dispositivos lógicos bipolares tipificada por las series 7400 con funciones fijas en el circuito integrado.

VLSI Very Large Scale Integration (*Integración de muy alta escala*). Circuito integrado que contiene el equivalente a 3000 compuertas o más.

GLOSARIO B
(Términos utilizados)

3D. Tridimensional. Se refiere a las imágenes por computadora que representan objetos con las dimensiones de anchura, altura y profundidad mutuamente ortogonales.

Abreviaturas de teclado. Pulsación de teclas que se utilizan en lugar del mouse (*ratón*), para seleccionar elementos del menú de una aplicación Windows en tiempo de ejecución (también conocidos como aceleradores).

Algoritmo. Es la metodología de diseño para la solución de un problema. Secuencia de acciones para un fin determinado.

Analógica. Una señal continua y variante en el tiempo que puede tomar valores no acotados, mientras que una señal digital es discreta y con valores finitos en el tiempo.

Archivo binario. Un archivo almacenado en formato binario compuesto por 1's y 0's.

Archivo de cabecera. Un archivo de texto que se mezcla lógicamente pero no físicamente en tiempo de compilación. Generalmente se denota por la definición y declaración de variables.

Archivo fuente. Archivo que contiene la descripción de los datos y funciones a procesar en un programa através de un lenguaje de programación.

Archivo objeto. Es el archivo que contiene todos los comados para enlazar un programa en tiempo de ejecución y formar el ejecutable.

Area de trabajo. El espacio en el interior de la ventana principal de la aplicación en Windows que está disponible para el usuario durante la aplicación.

Barra de título. Es la barra de la ventana principal que contiene el nombre de la aplicación.

Biblioteca. Archivo que contiene módulos de código objeto que representan funciones disponibles para el desarrollador de programas en Lenguaje C o C++.

Biblioteca en tiempo de ejecución. Un archivo que contiene las rutinas que necesita un programa en tiempo de ejecución.

Biblioteca de enlace dinámico. Todas las aplicaciones Windows pueden llamar a esta biblioteca de rutinas y datos que se compone por la interfase de dispositivos gráficos GDI.EXE , KERNEL.EXE y USER.EXE, también crearse librerías DLL propias.

BIOS (Basic Input / Output System). Rutinas en lenguaje ensamblador almacenadas como código de máquina en ROM. Estas rutinas proporcionan los servicios básicos de entrada / salida para el sistema operativo y para los programas de aplicación que utilizan interrupciones. También se llama ROM BIOS.

Buffer. Area de almacenamiento temporal de datos o imágenes.

Char. Tipo de variable en Lenguaje C o C++ almacenada en un byte de memoria, capaz de representar valores entre -128 y +127. Con frecuencia se utiliza un tipo *char* para almacenar el valor entero de un miembro del conjunto de caracteres alfanuméricos ASCII o ANSI.

Clase de ventana. Definición lógica que describe los atributos de una ventana. La ventana principal de una aplicación Windows pertenece a una determinada clase de ventana.

Código objeto. Código de máquina que contiene las instrucciones de las operaciones del procesador. Un compilador de C o C++ toma el código fuente escrito por un programa y lo convierte en el código objeto que la máquina entiende, el enlazador se encarga luego de combinar ese código objeto con otros módulos OBJ para generar el código ejecutable.

CPU. Acrónimo de unidad central de procesamiento.Lugar donde se llevan acabo las operaciones de la computadora.

CUA. (Acceso común de usuario). Colección de especificaciones estándar de IBM para los desarrolladores de interfases de usuario para aplicaciones que se ejecutan en sistemas operativos de ventanas, como Windows u OS/2.

DLL. Ver Bliblioteca de enlace dinámico.

GDI. Interfaz de dispositivos gráficos (*Graphics Device Interface*), es el mecanismo gráfico predefinido en Windows como una biblioteca de enlace dinámico (DLL) y a cuyas rutinas pueden llamar las aplicaciones Windows.

Gestor. Identificador que establece Windows para una aplicación en tiempo de ejecución que permite que manipule una ventana, contexto de visualización, mapa de bits, datos u otro objeto identificado.

GUI. Interfase gráfica de usuario (*Graphical User Interface*).

Hexadecimal. El sistema de numeración y operaciones que utiliza como base el número16.

Icono. Un icono es una representación gráfica o pictografica de un objeto, una acción, una propiedad, o algun otro concepto.

Instancia. Es el número de aparición de una entidad gráfica en una escena o imagen

KERNEL.EXE. La biblioteca de enlace dinámico de Windows que proporciona los recursos del sistema tales como gestión de memoria y gestión de recursos para las aplicaciones Windows en tiempo de ejecución. Vease USER.EXE y GDI.EXE.

Mapa de bits. Una disposición de bytes en memoria de pantalla o memoria convencional cuyos bits corresponden a los pixeles de la pantalla del monitor. Un mapa de bits dependiente del dispositivo se puede mostrar en un dispositivo particular (*adaptador gráfico*). Un mapa de bits independiente del dispositivo contiene una descripción generalizada de su contenido, permitiendo que la aplicación o Windows la modifique para su presentación en diversos dispositivos.

SAA. Arquitectura de sistemas de aplicación (*System Application architecture*). Consiste en un conjunto de especificaciones de IBM para la estandarización de los protocolos, interfaces y convenios de las aplicaciones que se ejecutan en un entorno de ventanas.

Super VGA. Adaptador gráfico que mejora las capacidades del VGA. Una tarjeta Super VGA normal incluye los modos de 640X480 a 256 colores, 800X600 a 16 colores, 1024 X768 a 16 colores y 1024 X 768 a 356 colores. Normalmente los controladores (*dispositivos para la entrada de discos*) se encargan de la gestión de la tarjeta.

USER.EXE. Biblioteca de enlace dinámico de Windows que contiene las funciones para la gestión de ventanas que se utilizan en las aplicaciones Windows al tiempo de ejecución. Véase KERNEL.EXE y GDI. EXE.

Vector. Un valor matemático que tiene módulo y dirección.

VGA. Arreglo gráfico de visualización (*Video Graphics Array*). En general se ejecuta en el modo 640 X 480 a 16 colores sobre una tarjeta VGA.

WINDOWS.H. Archivo de cabecera donde se definen y declaran las variables, estructuras de datos, funciones y constantes que luego estarán disponibles para una aplicación Windows.

WinMain. La función que sirve como punto de entrada a una aplicación Windows. WinMain contiene un ciclo de mensajes para atender y despachar peticiones.

WM_COMMAND. Variable que, en tiempo de ejecución, almacena los mensajes de entrada relativos a las acciones del usuario sobre los menús de la aplicación.

BIBLIOGRAFIA


Programmable Logic Devices
National Semiconductor
E.U.,1983, 1056 pp.

PAL/PLE Device Programmable Logic Array Hand book
Monolithic Memories (MMI)
E.U.,1984, 456 pp.


The Prologic Compiler User's guide
Texas Instruments
E.U., 1988, 207 pp.

The Programmable Logic Databook
Texas Instruments
E.U., 1990, 840 pp.


Programmable Logic Devices
Data Book and Design Guide
National Semiconductor
E.U., 1990, 450 pp.

JEDEC Standar No. 3-A
Solid State Product Engineering Council
E.U., 1986, 37 pp.

P.J. Hicks
Semi-Custom IC Design and VLSI
Ed.Peter Peregrinus
London, 1987, 218 pp.


M. Morris Mano
Diseño Digital
Ed. Prentice Hall
México, 1984,491 pp.

Roger C. Alford
Programmable Logic Designers Guide
Ed. Haward W. SAMS and Company
E.U., 1989, 304 pp.


Programmable Array Logic
Avanced Micro Device (AMD)
E.U., 1984, 220 pp.

The Logical Alternative
Altera Corp.
E.U., 1985, 187 pp.


William I. Fletcher
An Engineering Approach to digital design
Ed. Prentice Hall
E.U., 1980, 763 pp.


Sergio Ambriz M., Martín Pérez M.
Ambiente CAD para Diseño de Sistemas Digitales Orientado a PAL's
UNAM, 1990

McCoord
Borland C++ Tools
Ed. SAMS
E.U., 1992, 645 pp.

Borland C++
User's Guide
Ed. Borland Inc.
E.U., 1992, 211 pp.


Borland C++
Tools and Utilities Guide
Ed. Borland Inc.
E.U., 1992, 241 pp.

Borland C++
Library Reference
Ed. Borland Inc.
E.U., 1992, 655 pp.

[18] Borland C++
Programmers Guide
Ed. Borland Inc.
E.U., 1992, 467 pp.

[19] Scott Boggan, David Farkas & Joe Welinske
Developing on Line Help For Windows
Ed. SAMS
E.U., 1993, 459 pp.

[20] Alex Leavens
Windows Programmer's Guide To Resources
Ed. SAMS
E.U., 1992, 667 pp.

[21] Ben Ezzell
Windows 3.1 Graphics Programming
Ed. PC Magazine
E.U., 1992, 475 pp.

[22] Hughes, Freiner, Van Dame
Computer Graphics : Principles and Practice
2a. edición
Ed. Adison Wesley
E.U., 1991, 265 pp.

[23] William H. Murray and Chris H. Pappas
Windows Programming An Introduction
Ed. McGraw-Hill
E.U., 1990, 651 pp.

[24] Borland C++
Resource Workshop User's Guide
Ed. Borland Inc.
E.U., 1992, 275 pp.

[25] Lee Adams
Programación Avanzada de Gráficos en C para Windows
Ed. McGraw-Hill
México, 1993, 566 pp.

[26] Windows 3.1 Programming Tools
Ed. Microsoft Press
E.U., 1991, 265 pp.

[27] Windows 3.1 Programmers Reference
Vol.2 Function
Ed. Microsoft Press
E.U., 1992, 1001 pp.

[28] Windows 3.1 Guide To Programming
Ed. Microsoft Press
E.U., 1992, 567 pp.

[29] Windows 3.1 Programmers Reference
Vol 1 Overview
Ed. Microsoft Press
E.U., 1992, 487 pp.

[30] Windows 3.1 Programmers Reference
Vol. 3 Messages, Structures and Macros
Ed. Microsoft Press
E.U., 1992, 601 pp.

[31] Windows 3.1 Programmers Reference
Vol. 4 Resource
Ed. Microsoft Press
E.U., 1992, 300 pp.

[32] Peter Norton & Paul Yao
Windows Power Programming Techniques
Ed. Bantam Computer Books
E.U., 1990, 939 pp.

[33] Charles Petzold
Programming Windows 3.1
3a. edición
Ed. Microsoft Press
E.U., 1993, 983 pp.