25
24

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO DE SISTEMAS DIGITALES ASISTIDO POR COMPUTADORA (DISDAC)

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A N:

EDUARDO CARDENAS HERNANDEZ
FERNANDO MAR OLIVARES
ARTURO RINCON PARTIDA

Director: I.Q. Manuel Vázquez Islas

México, D. F. 1991

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

INTRODUCCION

En la actualidad, los sistemas digitales son ampliamente utilizados por el hombre. Su aplicación se encuentra desde juegos de entretenimiento hasta complejos equipos de telecomunicaciones.

Por la amplia variedad de componentes digitales existentes en el mercado el diseño de un circuito digital específico, puede tener infinidad de soluciones y corresponde al diseñador elegir, dependiendo de su criterio y experiencia, una combinación de módulos que ofrezcan una solución eficiente y de costo razonable.

Este trabajo trata sobre un sistema de cómputo que ayuda a diseñar módulos digitales a partir de módulos existentes en el mercado, para ello se ha incorporado un apartado en el sistema en el que se describe un módulo básico, que puede corresponder con un circuito integrado o con un diseño antes desarrollado. Lo anterior quiere decir que un diseño puede lograrse desarrollando sus partes constituidas terminando con la unión de todas ellas o directamente en teminos de circuitos integrados.

El programa utiliza un método de diseño y análisis de circuitos secuenciales temporizados que da inicio con un diagrama de estados en la que se describe la dinámica del circuito requerido, posteriormente, de manera automática se obtiene la tabla de estados que representa todas las situaciones posibles.

Como siguiente etapa deben elegirse los elementos de memoria o módulos básicos sobre los que descansará el diseño, después de lo cual, el programa tiene los elementos para completar la tabla de estados con un grupo de funciones de Bool que especifiquen las entradas lógicas a los módulos básico.

A partir del punto en que se tiene completa la tabla de estados, el problema secuencial se ha transformado en combinacional, pues sólo resta implementar cada una de las funciones de Bool. La forma de lograrlo va desde la utilización de circuitos complejos que pueden albergar la tabla completa, hasta la implementación basada en compuertas elementales Y, O y NO

Como apoyo a la etapa combinacional, el programa cuenta con una sección de simplificación de tablas de verdad para la cual se utiliza el método de Quine-McCluskey, el cual permite encontrara una ecuación boolean mínima que represente cada función. Finalmente se proporciona un opción que utiliza las ecuaciónes booleanas como entrada y emite un reporte en el que se listan los módulos básicos necesarios y la forma en que deben ser interconectadas sus terminales para lograr el objetivo inicial.

Como se ha podido apreciar el sistema de cómputo esta formado por secciones bien delimitadas, cada una realiza una tarea especifica. La comunicación entre secciones se lleva a cabo mediante archivos de texto en formatos legibles por el diseñador para obtener las siguientes ventajas:

a) Se permite alterar los datos de entrada de una sección y con ello observar las repercusiones en el resto del desarrollo.
b) Permitir utilizar el sistema por partes, por ejemplo, para resolver problemas meramente combinacionales.
c) Facilitar el seguimiento del desarrollo al poder estudiar los resultados de cada etapa (análisis de bitacora de diseño).

1. METODOS DE SIMPLIFICACION DE CIRCUITOS COMBINACIONALES

1.1 METODO DEL MAPA DE KARNAUGH

El mapa de Karnaugh es un metodo gráfico que se usa para simplificar ecuaciones lógicas o para convertir una tabla de verdad a su circuito lógico correspondiente en un proceso simple y ordenado. Aun cuando un mapa de Karnaugh puede usarse para problemas que involucran cualquier numero de variables de entrada, su utilidad practica esta limitada a cuatro variables. Los problemas con cinco y seis entradas en general son complejos y requieren de un programa de computadora para ser resueltos.

El mapa es un diagrama hecho de cuadros. Cada cuadro representa un término mínimo. Como cualquier función de Boole puede ser expresada como una suma de términos mínimos, dicha función se reconoce gráficamente en el mapa a partir del área encerrada por aquellos cuadros cuyos términos mínimos se incluyen en la función. El mapa presenta un diagrama visual de todas las posibles formas en que puede ser expresada una función en la forma normalizada. Al reconocer varios patrones el usuario puede derivar expresiones algebraicas alternas para la misma función de las cuales se puede escoger la más simple. La expresión algebraica más simple es aquella en la que una suma de productos o producto de sumas tiene el mínimo numero de literales.

1.1.1 Formato del mapa de Karnaugh

El mapa de Karnaugh, como una tabla de verdad, es una manera para mostrar la relación entre entradas lógicas y la salida deseada. Un mapa de dos variables se muestra en la figura 1.1. El mapa contiene cuatro términos mínimos para dos variables, es decir cada cuadro es para cada término mínimo. El mapa de la figura 1.1(b) sirve para demostrar la relación entre los cuadrados y las dos variables. Los ceros y unos marcados para cada fila y columna designan los valores x y y respectivamente.

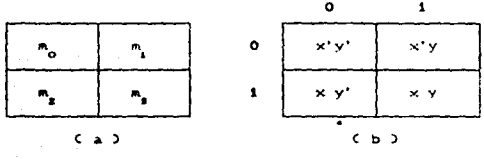


Figura 1.1. Mapa de dos variables

Nótese que la x aparece tildada en la fila 0 y no tildada en la fila 1. De manera similar, y aparece tildada en la columna 0 y no tildada en la columna 1. Si se marcan los cuadrados cuyos términos mínimos pertenecen a una función dada, el mapa de dos variables se convierte en otro método útil para representar una de las 4 funciones de Boole de dos variables. Como ejemplo, la función xy se muestra en la figura 1.2(a). Como xy es igual a m_3, se coloca un 1 dentro del cuadro que pertenece m_3. De manera similar, la función x + y se representa en el mapa de la figura 1.2(b) por medio de tres cuadros marcados con unos. Estos cuadrados se escogen de los términos mínimos de la función :

$$x + y = x'y + xy' + xy = m_1 + m_2 + m_3$$

Los tres cuadrados pudieron haberse determinado de la intersección de la variable x en la segunda fila y la variable y en la segunda columna, lo cual cubre el área perteneciente a x o y.

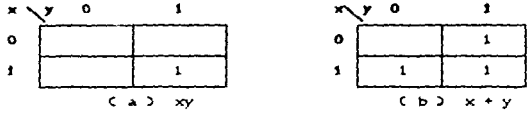


Figura 1.2. Representación de las funciones xy y x+y en un mapa

En la figura 1.3 se ilustra un mapa de tres variables. Existen ocho términos mínimos para las tres variables. Por lo tanto se tienen ocho cuadrados. Cabe hacer notar que, los términos mínimos se arreglan en una secuencia similar al código reflejado o gray, y no en una secuencia binaria. La característica de esta secuencia es que solamente cambia un bit de 1 a 0 o de 0 a 1, en la secuencia del listado. El mapa dibujado en la parte (b) se marca con los números de cada fila o cada columna para mostrar la relación entre los cuadrados de las tres variables. Observando la figura, vemos que, el cuadrado asignado a m_5, corresponde a la fila 1 y columna 01. Cuando se concatenan estos dos números darán el número binario 101, cuyo equivalente decimal es 5. Otra manera de mirar el cuadrado m_5 =xy'z es considerar que está en la fila marcada x y en la columna que pertenece a y'z (columna 01). Nótese que existen cuatro cuadrados donde cada variable es igual a 1 y cuatro donde son iguales a 0. La variable aparece no tildada en aquellos cuatro cuadrados donde sea igual a 1 y tildada en los que sea igual a 0. Por conveniencia se escribe la variable usando un símbolo de letra que abarca aquellos cuatro cuadrados donde la primera no esté tildada.

Para entender la utilidad del mapa en la simplificación de funciones de Boole, se debe de reconocer la propiedad básica que

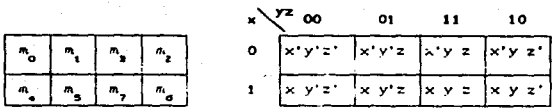


Figura 1.3. Mapa de tres variables

tienen los cuadrados adyacentes en el mapa que es la de diferir por una variable tildada en un cuadro y no tildada en el otro. Por ejemplo, m_5 y m_7 están en dos cuadrados adyacentes. La variable y está tildada en m_5 y no tildada en m_7, mientras que las otras dos variables son iguales en ambos cuadrados. De los postulados del álgebra de Boole, se desprende que la suma de los términos mínimos en cuadrados adyacentes pueden ser simplificados a un simple término AND consistente en dos literales. Para mostrar lo anterior, considérese la suma de dos cuadrados adyacentes tales como m_5 y m_7

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Aquí los dos cuadrados difieren en la variable y, que puede ser eliminada cuando se forme la suma de los términos mínimos. Así cualquier par de términos mínimos en cuadrados adyacentes a los cuales se les aplica la función OR se les quitará la variable diferente.

El mapa para las funciones de Boole de cuatro variables binarias se muestra en la figura 1.4 . En la figura (a) se listan los 16 términos mínimos y los cuadrados asignados a cada uno. En (b) se redibuja el mapa para para demostrar la relación con las cuatro variables. Las columnas y las filas se enumeran en la secuencia del código reflejado con un dígito que cambia de valor entre dos columnas o filas adyacentes. El término mínimo correspondiente a cada cuadro puede obtenerse por la concatenación del número de la fila con el número de la columna. Así, los números en tercera fila (11) y la segunda columna (01) una vez concatenados dan el número binario 1101, equivalente a número

decimal 13. Por lo tanto, el cuadrado en la tercera fila y la segunda columna representa el término mínimo m_{13}.

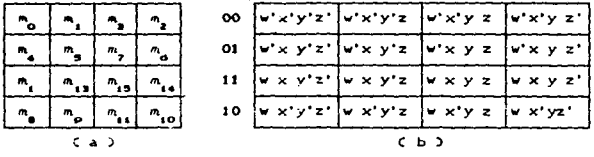


Figura 1.4. Mapa de cuatro variables

Los mapas de más de cuatro variables no son simples de usar. El número de cuadrados se hace muy grande y la geometría de combinar cuadrados adyacentes se complica. El número de cuadrados es siempre igual al número de términos mínimos. Para mapas de cinco variables se necesitan 32 cuadrados y para seis variables se necesitan 64 cuadrados. Mapas de siete variables en adelante necesitan muchos cuadros y son muy impracticos de usar. En las figuras 1.5 y 1.6 se muestran los mapas para cinco y seis variables respectivamente. Las columnas y las filas se enumeran de la misma forma que la secuencia del código reflejado. El término mínimo asignado a cada cuadrado se lee de esos números. De esta manera el cuadrado en la tercera fila (11) y la segunda columna (001) en el mapa para cinco variables se enumera 11001 y es el equivalente al decimal 25. Por tanto este cuadrado representa el término m_{25}. El símbolo de letra de cada variable se marca abarcando aquellos cuadrados donde el valor del bit correspondiente al número del código reflejado es 1. Por ejemplo, en el mapa de cinco variables, la variable A es un 1 en las últimas dos filas y B es un 1 en las dos filas de en medio. Los números reflejados en las columnas muestran a la variable C con un 1 en las cuatro columnas de la extrema derecha, la variable E, no adyacentes físicamente, se dividen en dos partes. La asignación de las variables en los mapas de dos, tres, cuatro y seis variables es de manera similar.

AB \ CDE	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

Figura 1.5. Mapa de cinco variables

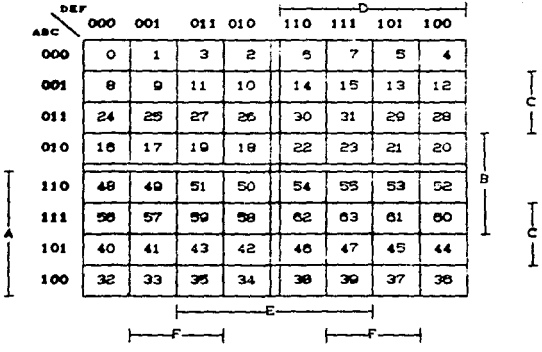


Figura 1.6. Mapa de seis variables

La definición de los cuadrados adyacentes para los mapas de las figuras 1.5 y 1.6 deben modificarse de nuevo para tener en consideración el hecho de que algunas variables están divididas en dos partes. Debe pensarse que el mapa de cinco variables consiste en dos mapas de cuatro variables y el mapa de seis variables consiste en cuatro mapas de cuatro variables. Cada uno de estos

mapas de cuatro variables se reconocen por las líneas dobles en el centro del mapa; cada uno de ellos conserva la cercanía definida cuando se toma individualmente. Además, la línea doble del centro debe ser considerada como el centro de un libro con cada mitad del mapa como página. Cuando se cierra el libro, los dos cuadrados adyacentes coinciden uno sobre el otro. En otras palabras, la línea doble del centro actúa como un espejo ya que cada cuadrado es adyacente, no solamente con sus cuatro cuadrados vecinos, sino con su imagen de espejo. En el mapa de cinco variables, el término mínimo 31 es adyacente a los términos mínimos 30, 15, 29, 23 y 27.

Por inspección y teniendo en cuenta la nueva definición de cuadrados adyacentes, es posible mostrar que cualquier 2^k cuadrados adyacentes para k=0, 1, 2,..., n, en un mapa de n variables, representan un área para un término de n-k literales. Para que la afirmación anterior tenga algún significado, n debe ser mayor que k. Cuando n=k el área total del mapa se combina para dar una función de identidad. La tabla 1.1 muestra la relación entre el número de cuadrados adyacentes y el número de literales en el término. Por ejemplo, ocho cuadrados adyacentes se combinan en una área del mapa de cinco variables para dar un término de dos literales.

1.1.2 Condiciones de no importa

Los unos y ceros en el mapa significan la combinación de variables que hacen la función igual a 1 ó 0 respectivamente. Las combinaciones se obtienen comunmente de una tabla de verdad que lista las condiciones bajo las cuales la función es 1. Se asume que la función sea igual a cero bajo cualquier otra condición. Esta suposición no es siempre verdadera ya que hay aplicaciones donde ciertas combinaciones de variables de entrada nunca ocurren. Un código decimal de cuatro bits, por ejemplo, tiene seis combinaciones que no se usan. Cualquier circuito digital que use este código, opera bajo la suposición de que esas combinaciones no usadas nunca ocurren, siempre y cuando el sistema funcione correctamente. Como resultado, no importa lo que sea la salida de la función para estas combinaciones de variables ya que se garantiza que nunca ocurrirán. Estas combinaciones de no importa

pueden usarse en un mapa para lograr una mejor simplificacion de la función.

Número de cuadrados adyacentes		Número de literales de un término en un mapa de n variables					
k	2^k	n=2	n=3	n=4	n=5	n=6	n=7
0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	4	0	1	2	3	4	5
3	8		0	1	2	3	4
4	16			0	1	2	3
5	32				0	1	2
6	64					0	1

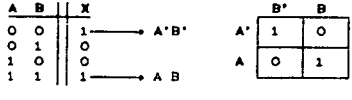
Tabla 1.1. Relación entre el número de cuadrados adyacentes y el número de literales en el término.

Se puede hacer énfasis en que la combinación de no importa no puede ser marcada con un uno en el mapa ya que ello implica que la función sea uno para esa combinación de entrada. De la misma manera colocar un cero requiera que la función sea cero. Para diferenciar las condiciones de no importa de los unos y ceros se usará un *.

Cuando se escogen cuadrados adyacentes, para simplificar la función en el mapa, se asume que * sea 1 ó 0 segun lo que produzca la expresión más simple. Además, no se necesita usar * si esta no contribuye al cubrimiento de una área mayor. En cada caso, la alternativa depende solamente de las simplificación que se puede lograr.

1.1.3 Método para simplificar los mapas de Karnaugh

La expresión de salida puede simplificarse combinando en forma apropiada aquellos cuadrados en el mapa de Karnaugh que contiene unos. El proceso para combinar estos unos se llama enlace.



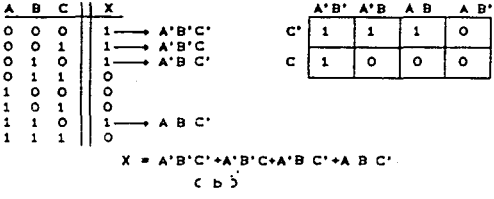


Figura 1.7. Tabla de verdad y mapa de Karnaugh para dos y tres variables

1.1.3.1 Enlazando grupos de a dos

En la figura 1.7 se muestra una tabla de verdad de tres variables, con su respectivo mapa de Karnaugh. Este mapa contiene un par de unos que son adyacentes horizontalmente el uno con el otro; el primer uno representa A'BC' y el segundo representa ABC'. Note que en estos dos términos sólo la variable A aparece en ambas formas: normal y complementada (B y C' permanecen invariables). Estos dos términos pueden ser enlazados (combinados) para dar un uno resultante que elimina la variable A puesto que aparece en ambas formas no complementada y complementada. Esto se prueba

fácilmente como sigue :

$$X = A'B\,C' + A\,B\,C'$$
$$= B\,C'\,(A' + A)$$
$$= B\,C'\,(1)$$
$$= B\,C'$$

Este mismo principio se mantiene verdadero para cualquier par de unos adyacentes verticalmente o horizontalmente, en mapas de dos, tres o más variables. Se puede presentar el caso de que aparezcan unos en los cuadrados que se encuentran en los cuadros en los extremos del mapa. Estos unos son considerados adyacentes y pueden simplificarse de la manera anterior descrita (figura 1.8).

El enlace de un par de unos adyacentes en un mapa de Karnaugh elimina la variable que aparece en la forma complementada y no complementada.

	A'B'	A'B	A B	A B'
C'	1	0	0	1
C	0	0	0	0

$X = A'B'C' + A\,B'C'$
$= B'C'$

Figura 1.8.

1.1.3.2 Enlace de grupos de cuatro

Un mapa de Karnaugh puede contener un grupo de cuatro unos que sean adyacentes con otros. La figura 1.9 muestra varios ejemplos de enlaces de cuatro unos. En (a) están adyacentes horizontalmente y en (b) los cuatro unos están adyacentes verticalmente. En (c) el mapa contiene cuatro unos en un cuadrado se consideran adyacentes unos con otros. Los cuatro unos en (d) son también adyacentes, como lo son los de (e) debido a que las filas, superior e inferior y las columnas en los extremos derecha e izquierda se consideran adyacentes las unas con las otras.

Cuando se enlazan cuatro unos, el término resultante contendrá solo las variables que no cambian de forma para todos los cuadrados de los cuatro unos. Por ejemplo en la figura 1.9 (a) los cuatro cuadrados que contienen un 1 son A'B'C, A'BC, ABC y A'BC. El examen de estos términos revela que solo la variable C permanece sin cambio (en tanto A y B aparecen en forma complementada y no complementada). Así, la expresión resultante para X es simplemente X =C. Esto, puede probarse como sigue:

$$
\begin{aligned}
X &= A'B'C + A'BC + ABC + AB'C \\
&= A'C(B' + B) + AC(B + B') \\
&= A'C + AC \\
&= C(A' + A) \\
&= C
\end{aligned}
$$

Para resumir: el enlace de cuatro unos elimina las dos variables que aparecen en forma complementada y no complementada.

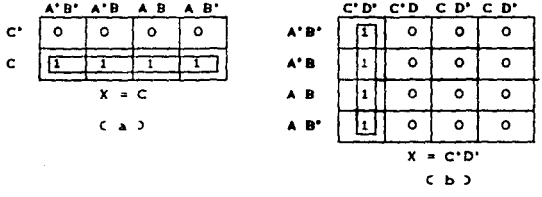


Figura 1.9. Ejemplos de enlaces de grupos de cuatro unos

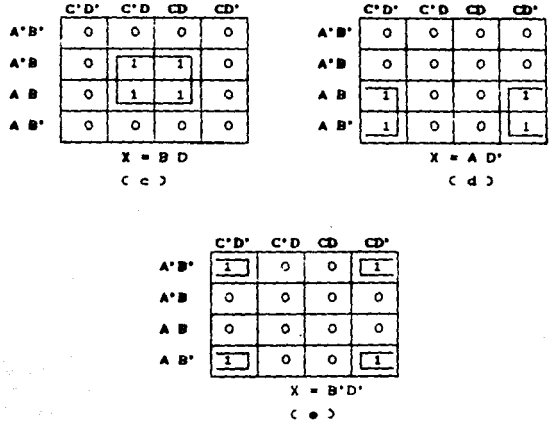


Figura 1.9. Ejemplos de enlaces de grupos de cuatro unos

1.1.3.3 Enlace en grupo de ocho

Cuando ocho unos se enlazan en un mapa de cuatro variables, tres de las cuatro se eliminan porque sólo una variable permanece sin modificar, es decir, sólo aparece complementada o no complementada. Por ejemplo en la figura 1.10 se muestran ocho unos enlazados y muestra que la variable B está en la misma forma para los ocho cuadrados; las otras variables aparecen en la forma complementada y no complementada. Así, para este mapa X = B.

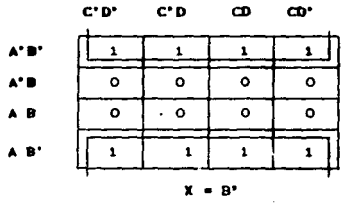


Figura 1.10. Enlace de ocho unos

1.2. PROCEDIMIENTO COMPLETO DE SIMPLIFICACION

Hemos visto que el enlace de unos en pares, cuartetos y octetos en un mapa de Karnaugh puede usarse para obtener una expresión simplificada. Podemos resumir la regla para enlaces de cualquier tamaño: *Cuando una variable aparece en forma complementada y no complementada dentro de un anillo, esa variable se elimina de la expresión. Las variables que son las mismas para todos los cuadrados del anillo deben aparecer en la expresión final.*

Debe quedar claro que un anillo mayor de unos elimina más variables. Para ser exactos, un anillo de dos elimina una variable, un anillo de cuatro elimina dos variables y un anillo de ocho elimina tres variables. Este principio será usado ahora para obtener una expresión logica simplificada desde un mapa de Karnaugh que contiene cualquier combinación de unos y ceros.

El procedimiento que se sigue para simplificar una expresión booleana en un mapa son :

1. Construya el mapa de Karnaugh y coloque unos en aquellos cuadrados que corresponden a los unos en la tabla de verdad coloque ceros en los demás cuadros.

2. Examine el mapa para los unos adyacentes y coloque un anillo alrededor de aquellos unos que no son adyacentes a cualesquiera otros unos.

3. A continuación, busque aquellos unos que son adyacentes a cualquier otro uno solamente. Enlace cualquier par que contenga tal uno aún si el mismo uno es enlazado más de una vez.

4. Enlace cualquier grupo de cuatro unos que contenga uno o más unos que no hayan sido ya enlazados.

5. Enlace cualquier grupo de ocho unos aún si algunos de los unos han sido ya enlazados.

6. Enlace cualquier par necesario para incluir cualesquiera unos que no hayan sido enlazados aún, asegurándose de usar el número mínimo de anillos.

7. Forme la suma OR de todos los términos generados por cada anillo.

1.2.1 Ejemplos de simplificación de mapas de Karnaugh

Ejemplo 1.1 : Simplificar la función de Boole:

$$F = A'B'C + C'BC' + ABC + AB'C + ABC$$

El mapa para simplificar esta función se muestra en la figura 1.11. La función de este ejemplo tiene cinco términos mínimos, como se indica con los cinco cuadrados marcados con un 1. El mapa se obtuvo de la tabla de verdad que da por resultado la función de Boole (paso 1). Los cuadrados están numerados para facilitar la identificación de cada anillo.

Paso 2. No hay cuadrados con un 1 que no tenga adyacencia con otro.

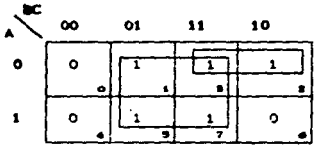


Figura 1.11.

Paso 3. El cuadrado 2 es adyacente unicamente al cuadrado 3. Este par es enlazado y su simplificación da como resultado A'B.

Paso 4. Los cuadrados 1,3,5 y 7 forman un conjunto de cuatro unos. Este cuarteto es enlazado y su simplificación es igual a C. Note que el cuadrado 3 se usa nuevamente.

Paso 7. La simplificación del mapa es la suma OR de los resultados generados por cada anillo enlazado. Por tanto :

$$F = A'B + C$$

Ejemplo 1.2 : La figura 1.12 muestra un mapa de Karnaugh con la siguiente función de Boole :

$$F(W,X,Y,Y) = \sum_{4} (0,1,2,3,5,6,8,9,12,13,14)$$

Paso 4. Hay dos anillos de cuatro cuadrados que contienen un 1. Un anillo está formado por los cuadrados 0, 2, 4 y 6, que dan como resultado W'Z'. El segundo anillos formado por los cuadrados 4, 6, 12, y 14, dan como resultado XZ'.

Paso 5. Se tiene un anillo de ocho unos, formado por los cuadros 0, 1, 4, 5, 8, 9, 12 y 13 que nos da como resultado Y'.

Paso 7. La función simplifacada es:

$$F(0,1,2,4,5,6,8,9,12,13,14) = Y' + W'Z' + XZ'$$

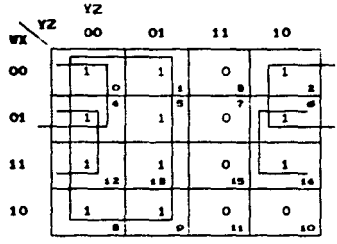


Figura 1.12.

Es importante hacer notar que, en este ejemplo, se pueden formar mas anillos, pero, se tendrian más productos a implementar, lo cual llega a ser redundante en la mayoria de los casos.

Ejemplo 1.3 : Simplificar la función de Boole:

$$F(w,x,y,z) = \sum (1,3,7,11,15)$$

y las condiciones de no importa:

$$d(w,x,y,z) = \sum (0,2,5)$$

Los terminos minimos de F son las combinaciones de variables que hacen la función igual a 1. Los términos minimos de d son las combinaciones de no importa que se conocen que nunca ocurren. La minimización se muestra en la figura 1.13. Los términos minimos de F se marcan con unos y aquellos de d se marcan con un # y los cuadrados restantes se llenan con ceros. En (a) los unos y los # se combinan de una forma conveniente tal que se abarque el mayor numero de cuadrados advacentes. No es necesario incluir todos o

algunos de los * sino aquellos que sean útiles para la simplificación de un término. Una función que da una función mínima incluye un * y deja dos por fuera. Esto dará como resultado una función simplificada en suma de productos.

$$F = W'Z + YZ$$

WX \ YZ	00	01	11	10
00	*	1	1	*
01	0	*	1	0
11	0	0	1	0
10	0	0	1	0

Figura 1.13. Considerando un * como 1

Las funciones de Boole minimizadas, derivadas del mapa de Karnaugh son expresadas en la forma de suma de productos. Con una pequeña modificación se puede obtener un producto de sumas.

El procedimiento para obtener una función minimizada en producto de sumas se desprende de las propiedades básicas de las funciones de Boole. Los unos colocados en los cuadrados del mapa representan los términos mínimos de la función. Los términos mínimos no incluidos en la función denotan el complemento de una función y se representan en un mapa por cuadrados no marcados por unos. Si se marcan los cuadrados vacíos con ceros y se combinan en cuadrados adyacentes válidos, se obtiene una expresión simplificada del complemento de la función, es decir de X'. El complemento de X' dará de nuevo la función X. Debido al teorema generalizado de De Morgan el producto así obtenido queda automáticamente en la forma de producto de sumas.

Ejemplo 1.4 :simplificar la siguiente función de Boole en (a) suma de productos y (b) producto de sumas.

$$F(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Figura 1.14.

Los unos marcados en la Figura 1.14 representan todos los términos mínimos de la función. Los cuadrados marcados con ceros representan los términos mínimos no incluidos en F y por tanto denotan el complemento de F. Combinando los cuadrados con unos se obtendrá una función simplificada en suma de productos:

$$(a) \quad F = B'D' + B'C' + A'C'D$$

Si se combinan los cuadrados marcados con ceros, como se muestra en el diagrama, se obtiene la siguiente función simplificada de complemento:

$$F' = AB + CD + BD'$$

Aplicando el teorema de Morgan, se obtiene una función simplificada en producto de sumas:

$$F = (A' + B')(C' + D')(B' + D)$$

1.3 METODO DE QUINE-McCLUSKEY

El método del mapa para simplificación es conveniente siempre y cuando el número de variables no exceda de cinco o seis. A medida que el número de variables aumenta el número de cuadrados impide una selección razonable de cuadrados adyacentes. La desventaja obvia del mapa es esencialmente el procedimiento de prueba y error que depende de la habilidad del usuario humano para reconocer ciertos patrones. Para funciones de seis o más variables es muy difícil estar seguro que realmente se hizo la mejor selección.

El método de Quine-McCluskey elimina la anterior dificultad. Este método se puede aplicar a problemas con muchas variables y tiene la ventaja de ser adecuado para cómputos con máquina. Sin embargo es un poco tedioso para uso humano y propenso a errores debido a un proceso rutinario y monótono.

El método de simplificación consiste en dos partes. La primera en encontrar mediante una búsqueda muy completa de todos los términos candidatos de inclusión en la función simplificada. Estos términos se llaman *primeros-implicados*. La segunda operación es escoger entre los primeros implicandos aquellos que dan una expresión con el menor número de literales. A este método se le conoce con el nombre del método del tabulado.

1.3.1 Determinación de los primeros implicados

El punto de partida del método de Quine-McCluskey es la lista de términos mínimos que especifican la función. La primera operación de tabulado es buscar los primeros implicados para usarlos en el proceso de apareamiento. Este proceso compara cada término mínimo con cada uno de los restantes términos mínimos. Si dos términos mínimos difieren en solamente una variable, esa variable se elimina para encontrar un solo término con un literal menos. Este proceso se repite para cada término mínimo hasta que se complete el proceso completo de busqueda. El ciclo del proceso de apareamiento se repite para aquellos términos nuevos encontrados. Se continúa con el tercer y subsiguientes ciclos hasta el paso por un ciclo que no produzca nuevas eliminaciones de literales. Los términos restantes y todos los términos que no se

aparearon durante el proceso, constituyen los primeros implicados.

Ejemplo 1.5: Simplificar la siguiente función de Boole:

$$F = \sum (0,1,2,8,10,11,14,15)$$

Paso 1: Agrupar la representación binaria de los términos mínimos de acuerdo al número de unos contenido de la manera mostrada en la tabla 3.2 columna (a). Esto se hace agrupando los términos mínimos en cinco secciones separadas por líneas horizontales. La primera sección contiene el número sin unos en él. La segunda sección contiene aquellos números que tienen solamente un uno. La tercera, cuarta y quinta sección contienen aquellos números binarios con dos, tres y cuatro unos respectivamente. Los decimales equivalentes de los términos mínimos se colocan a todo lo largo para identificación.

Paso 2: Cualquier par de términos mínimos que difieren entre sí solamente por una variable, se pueden combinar y las variables no apareadas se pueden eliminar. Dos números de término mínimo caen dentro de esta categoría si ambos tienen el mismo valor de bit en todas las posiciones excepto en una. Los términos mínimos en una sección se comparan con aquellos de la siguiente en adelante ya que dos términos que se diferencian en más de un bit no se pueden aparear. El término mínimo de la primera sección se compara con cada uno de los tres términos mínimos de la segunda sección. Si hay dos términos iguales en todas las posiciones excepto en una, se marcan a la derecha de ambos términos mínimos para indicar que han sido usados. El término resultante, conjuntamente con los equivalentes decimales, se lista en la columna (b) de la tabla. La variable eliminada durante el proceso de apareamiento se reemplaza por un guión en su posición original. En este caso m(0000) se combina con m(0010) para formar (00-0). Esta combinación es equivalente a la operación algebraica :

$$w'x'y'z' + w'x'yz' = w'x'z'$$

El término mínimo *m0* se combina para formar (00-0) y con *m8* para formar (-000). El resultado de esta comparación se coloca en la primera sección de la columna (b). Los términos mínimos de las secciones dos y tres de la columna (a) se comparan en seguida para producir los términos listados en la segunda sección de la columna (b). Todas las otras secciones de (a) se comparan de manera similar y las secciones subsecuentes se forman en (b). Este proceso de comparación dará como resultado cuatro secciones de (b).

(a)		(b)		(c)	
	W X Y Z		W X Y Z		W X Y Z
0	0 0 0 0 *	0,1	0 0 0 - *	0,2,8,10	- 0 - 0
		0,2	0 0 - 0 *	0,8,2,10	- 0 - 0
		0,8	- 0 0 0 *		
1	0 0 0 1 *	2,10	- 0 1 0 *	10,11,14,15	1 - 1 -
2	0 0 1 0 *	8,10	1 0 - 0 *	10,14,11,15	1 - 1 -
8	1 0 0 0 *				
10	1 0 1 0 *	10,11	1 0 1 - *		
		10,14	1 - 1 0 *		
11	1 0 1 1 *	11,15	1 - 1 1 *		
14	1 1 1 0 *	14,15	1 1 1 - *		
15	1 1 1 1 *				

Tabla 1.2. Determinación de los primeros implicados

Paso 3: Los términos de la columna (b) tienen solamente tres variables. Un 1 debajo de la variable significa que no es tildada, un 0 significa que es tildada y un guión significa que no se incluye en el término. El proceso de búsqueda y comparación se repite para los términos en la columna (b) para formar los dos términos variables de la columna (c). De nuevo, los términos en cada sección necesitan compararse solamente si tienen guiones en la misma posición. Nótese que el término 10,11 no se aparea con

cualquier otro término. Por consiguiente, este no tendrá marca a su derecha. Los equivalentes decimales se escriben a mano derecha de cada entrada para propósitos de identificación. El proceso de comparación debe llevarse a cabo de nuevo en la columna (c) y en las columnas subsecuentes siempre y cuando se consiga el apareamiento adecuado. En este ejemplo, la operación realizada para la tercer columna.

Paso 4: Los términos no marcados en la tabla forman los primeros implicados. En este ejemplo tenemos el término w'x'y'(000-) en la colúma (b) y los términos x'z'(-0-0) y wy(1-1-) en la columna (c). Nótese que cada término aparece dos veces en la tabla y cuando el término forme un primer implicado es innecesario usar el mismo término dos veces. La suma de los primeros implicados dará una expresión simplificada de la función. Esto es debido a que cada término marcado en la tabla se ha tenido en cuenta para la entrada de un término más sencillo en la columna subsecuente. Así, las entradas no marcadas(primeros implicados) constituyen los términos que forman la función. Para este ejemplo, la suma de los primeros implicados dará la función minimizada en suma de productos :

$$F = w'x'y' + x'z' + wy$$

Es importante señalar que el ejemplo anterior da una función simplificada a partir de una suma de primeros implicados. En la mayoría de los casos la suma de los primeros implicados no necesariamente forman la expresión con el número mínimo de términos. Esto lo comprobaremos en el ejemplo 1.6.

La tediosa manipulación que se debe hacer cuando se usa el método de Quine-McCluskey se reduce si la comparación se hace con números decimales en vez de binarios. Se mostrará ahora un método que usa la resta de números decimales en vez de comparar y aparear números binarios. Hay que notar que cada 1 en un número binario representa el coeficiente multiplicando por una potencia de dos. Cuando dos términos mínimos son iguales en todas las posiciones excepto en una, el término mínimo con el 1 extra debe ser más

grande, que el número del otro término mínimo, en una potencia de dos. Por tanto, dos términos mínimos se pueden cambiar si el número del primer término mínimo difiere por una potencia de dos de un segundo número mayor de la siguiente sección inferior de la tabla. Se ilustrará este procedimiento repitiendo el ejemplo 1.5.

Ejemplo 1.6:
Como se ilustra en la tabla 1.3 columna (a), los términos mínimos se arreglan en secciones como se hizo anteriormente excepto que se listan solamente los decimales equivalentes a los términos mínimos. El proceso de comparar los términos mínimos es como sigue: inspeccionar todo par de números decimales en secciones adyacentes a la tabla. Si el número de la sección inferior es mayor que el número de la sección superior por una potencia de dos (por ejemplo 1,2,4,8,16,etc.) marquese ambos números para indicar que han sido usados y escríbalos en la columna (b). El par de números transferidos a la columna (b) incluyen un tercer número en paréntesis que designa la potencia de dos por la cual difieren los números. El número en paréntesis dice la posición del guión en la notación binaria. El resultado de la comparación de la columa (a) se muestra en la columa (b).

La comparación entre secciones adyacentes en la columna (b) se realiza de manera similar, excepto que solamente se comparan aquellos términos con el mismo número entre paréntesis. El par de números en una sección debe diferir por una potencia de dos del par de números en la siguiente sección. Y los números en la sección inmediatamente inferior deben ser mayores para poder lograr la combinación. En la columa (c) se escriben todos los cuatro numeros decimales, con los dos números en paréntesis como indicadores de la posición de los guiones. Una comparación de las tablas 1.2 y 1.3 podría ser útil para comprender las derivaciones de la tabla 1.3.

(a)			(b)			(c)	
0000	1	*	1,9	(8)		8,9,10,11	(1,2)
0100	4	*	4,6	(2)		8,9,10,11	(1,2)
1000	8	*	8,9	(1)	*		
			8,10,	(2)	*		
0110	6	*					
1001	9	*	6,7	(1)			
1010	10	*	9,11	(2)	*		
			10,11	(1)	*		
0111	7	*					
1011	11	*	7,15	(8)			
			11,15	(4)			
1111	15	*					

PRIMEROS IMPLICADOS

DECIMAL		BINARIO W X Y Z	TERMINOS
1,9	(8)	- 0 0 1	x'y'z
4,6	(2)	0 1 - 0	w'y'z'
6,7	(1)	0 1 1 -	w'x y
7,15	(8)	- 1 1 1	x y z
11,15	(1,2)	1 - 1 1	w y z
8,9,10,11	(1,2)	1 0 - -	w x'

Tabla 1.4. Determinación de los primeros implicados del ejemplo 1.7

Los números de los términos mínimos se agrupan en secciones de la manera mostrada en la tabla 1.4 columna (a). El binario equivalente de un término mínimo se incluye con el propósito de contar el numero de unos. Los números de los términos mínimos se comparan por el método decimal y se hacen parejas, si el número de la sección inferior es mayor que aquel de la sección superior. Si el número de la sección inferior es más pequeño que el de la sección superior no se tiene en cuenta la pareja aunque los dos números difieran por una potencia de dos. La búsqueda minuciosa en la columnna (a) dará como resultado los términos de la columna (b), con todos los términos mínimos en la columna (a) marcados. Hay solamente dos parejas de términos en la columna (b) los cuales

darán el mismo término de dos literales en la columna (c). Los primeros implicados consisten en todos los términos no marcados en la tabla. La conversión de notación binaria a decimal se muestra en la parte inferior de la tabla. Los primeros implicados encontrados son : x'y'z,w'xz',w'xy,xyz,wyz y wx'.

La suma de todos los primeros implicados dará una expresión algebraica válida para la función. Sin embargo esta expresión no es necesariamente la que contiene el mínimo número de términos. Esto puede demostrarse inspeccionando el mapa de la función del ejemplo 1.7, en la figura 1.15; donde la función minimizada reconocida es:

F = x'y'z + w'xz' + xyz + wx'

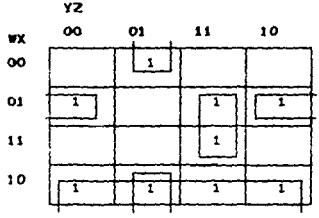


FIGURA 1.15. Mapa de la función del ejemplo 1.7

La cual consiste en la suma de cuatro de los seis implicados derivados en el ejemplo 1.7. El procedimiento de tabulado para la selección de los primeros implicados se da a continuación.

La selección de los primeros implicados que forman la función minimizada se hace a partir de una tabla de primeros implicados. En esta tabla, cada primer implicado se representa en una fila y cada término mínimo en una columna. Se colocan cruces en cada fila para indicar la composición de los términos mínimos que constituyen los primeros implicados. Un mínimo grupo de de primeros implicados se escoge de manera que abarque todos los términos mínimos de la función. Esto lo ilustraremos con el ejemplo 1.8.

Ejemplo 1.8 : Minimizar la función obtenida en el ejemplo 1.7.

El tabulado de los primeros implicados para este ejemplo se muestra en la tabla 1.5. Hay seis filas, una para cada primer implicado(obtenido en el ejemplo anterior) y nueve columnas que representan cada una un término mínimo de la función. Se colocan cruces en cada fila para indicar los términos mínimos contenidos en el primer implicado de esa fila. Por ejemplo, las dos cruces en la primera fila indican que los términos mínimos 1 y 9 están contenidos en el primer implicando x'y'z. Es aconsejable incluir el equivalente decimal del primer implicado en cada fila y conveniente dar los términos mínimos contenidos en él. Una vez que se hayan marcado todas las cruces se procederá a seleccionar un número mínimo de primeros implicados.

La tabla completa de primeros implicados se inspecciona para obtener columnas que contengan solamente una cruz. En este ejemplo hay cuatro términos mínimos cuyas columnas tienen una sola cruz: 1,4,8 y 10. El término mínimo 1 está cubierto por el primer implicado x'y'z; es decir,la selección del primer implicado x'y'z garantiza que el término mínimo 1 está incluido en la función. De manera similar el término mínimo 4 está cubierto por el primer impicado w'xz' y los términos mínimos 8 y 10 por el primer implicado wx'. Los primeros implicados que cubren los términos mínimos con una sola cruz en su columna se llaman *primeros implicados esenciales*. Para permitir que la expresion final simplificada contenga todos los términos mínimos no queda otra alternativa que incluir los primeros implicados esenciales. Se coloca una marca en la tabla a continuación de los primeros implicados esenciales para indicar que han sido seleccionados.

		1	4	6	7	8	9	10	11	15
* x'y'z	1,9	X					X			
* w'x z'	4,6		X	X						
w'x y	6,7			X	X					
x y z	7,15				X					X
* w x'	8,9,10.					X	X	X	X	
		*	*	*		*	*	*	*	

TABLA 1.5

Tabla de primeros implicados del ejemplo 1.8

En seguida se observa cada columna cuyo término mínimo está cubierto por los primeros implicandos esenciales seleccionados. Por ejemplo, el primer implicando seleccionado x'y'z cubre los términos mínimos 1 y 9, entonces se coloca una marca en la parte inferior de las columnas. De manera similar, el primer implicando w'xz' cubre los términos mínimos 4 y 6 y w'x' cubre 8,9,10 y 11 respectivamente. La inspicción de la tabla de primeros implicados cubre todos los términos de la función con excepción de 7 y 15. Estos dos términos mínimos deben ser incluidos por la selección de uno o más primeros implicados. En este ejemplo, el primer implicado xyz cubre ambos términos mínimos y es por tanto seleccionado. Así ,se ha encontrado el conjunto mínimo de primeros implicados cuya suma da la función minimizada requerida:

$$F = x'y'z + w'xz' + wx' + xyz$$

Las enpresiones simplificadas deducidas de los ejemplos anteriores estaban expresadas en la forma de suma de productos. El método de Quine-Mc Cluskey puede adaptarse para dar una expresión simplificada en producto de sumas. De la misma manera que el método del mapa se tiene que comenzar con el complemento de la función tomando los ceros como la lista inicial de términos mínimos. Esta lista contiene aquellos términos mínimos no incluidos en la función original, los cuales son numéricamente iguales a los términos máximos de la función. El proceso de tabulación se lleva a cabo con los ceros de la función para

terminar con una expresión simplificada en suma de productos del complemento de la función. Obteniendo de nuevo el complemento se consigue la expresión simplificada en producucto de sumas.

Una función con condiciones de no importa puede ser simplificada por el método del tabulado despues de una pequeña modificación. Los términos de no importa se incluyen en la lista de los términos mínimos cuando los primeros implicados se determinan. Esto permite la deducción de primeros implicados con el mínimo número de literales. Los términos de no importa no se incluyen en la lista de los términos mínimos cuando se prepara la tabla de los primeros implicados ya que los términos de no importa no tienen que estar cubiertos por los primeros implicados seleccionados.

2. DESCRIPCION DE LOS SISTEMAS SECUENCIALES Y SU REPRESENTACION POR MEDIO DE DIAGRAMAS

Un *circuito secuencial* es aquel que contiene flip-flops y compuertas. Este es llamado así porque opera en secuencia através de una sucesión de estados durante periodos sucesivos de reloj.

Un diagrama de bloque de un circuito secuencial se muestra en la figura 2.1. Este consiste en un circuito combinacional al cual se le conectan elementos de memoria para formar un camino de realimentación. Los elementos de memoria son capaces de almacenar información binaria dentro de ellos. La información binaria almacenada en los elementos de memoria en un tiempo dado define el estado del circuito secuencial. El circuito secuencial recibe la información binaria de las entradas externas. Estas entradas, conjuntamente con el presente estado de los elementos de memoria, determinan el valor binario de los terminales de salida. También determinan la condición de cambio de estado en los elementos de memoria. El diagrama de bloque demuestra que las salidas externas en un circuito secuencial son una función no solamente de las entradas externas sino del presente estado de los elementos de memoria. El siguiente estado de los elementos de memoria es también una función de las entradas externas y del estado presente. Así, un circuito secuencial se especifica por medio de una secuencia de tiempo de las entradas, salidas y estados

internos.

Un sistema o circuito secuencial puede categorizarase como tal por la existencia de dos propiedades, que son:

1. El sistema debe de tener elemento de memoria.
2. El sistema debe tener una ruta de realimentación del elemento de memoria al sistema de entradas.

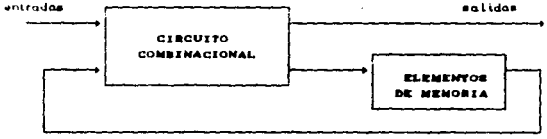


Figura 2.1. Diagrama de bloque de un circuito secuencial

Hay dos tipos de circuitos secuenciales. Su clasificación depende del tiempo de sus señales. Un circuito secuencial síncrono es un sistema cuyo comportamiento puede definirse a partir del conocimiento de sus señales en instantes discretos de tiempo. El comportamiento de un circuito asíncrono depende del orden en que cambien las señales de entrada y puedan ser afectadas en un instante dado de tiempo. Los elementos de memoria comúnmente usados en los circuitos secuenciales asíncronos son mecanismos retardadores de tiempo. Un circuito secuencial asíncrono es basicamente lógica combinacional con relimentación y son ciclados por transiciones de cada entrada y usan estrictamente el retardo de propagación del siguiente estado decodificador de memoria. Los circuitos síncronos usan dispositivos hardware llamados flip-flops para memoria y son ciclados por una forma de onda especial como entrada para sincronización llamada *sistema de reloj*. El *reloj* es la "señal comando" que causa que el elemento de memoria (flip-flop) lea y almacene el código de sus entradas. El flip-flop por si mismo es un circuito asíncrono diseñado especificamente para proveer los requerimientos de memoria de un circuito

síncrono. Un sistema lógico secuencial síncrono, por definición, puede usar señales que afecten los elementos de memoria solamente en instantes de tiempo discreto. Una forma de lograr este propósito es usar pulsos de duración limitada a través del sistema de tal manera que la amplitud de un pulso represente un 1 lógico y otra amplitud de pulso (o la ausencia de un pulso) represente un 0 lógico.

2.1 OPERACION FUNDAMENTAL DE LA MAQUINA SECUENCIAL

La figura 2.2 es un diagrama de bloques del modelo general de una máquina secuencial o máquina de estado-finito.

Las máquinas secuenciales fueron clasificadas como tales primeramente por Huffman y Moore cuando estos introdugeron el concepto de "*estado*" como el más significativo caracterizador de la memoria de estos circuitos. Ellos hicieron esto para definir el código de salida (patrón de bit) de la memoria en tiempos elegidos apropiadamente como el *estado* de la máquina. El tiempo elegido apropiadamente es cualquier tiempo del circuito que *no* esté en condición de transición, esto es, cambiando de estado. Dicho de otra manera, definimos el estado de un circuito secuencial de acuerdo al código presentado directamente de la salida del elemento de memoria. Hay que hacer notar que otros nombres han sido dados a el estado de la máquina. Estos son el *estado presente* o el de *estado variable*.

La siguiente definición de estado es más formal; sin embargo esta soporta el concepto Huffman-Moore también como el concepto de control de la máquina secuencial.

El ESTADO de una máquina es la propiedad de esa máquina la cual relaciona las entradas a las salidas de tal manera que reconozca el momento de entrada de la función f(t) para t ≥ to y el estado en t=to determinan completamente la salida para todo t > to. Es decir, la salida es mantenida durante todo el tiempo de duración del estado, hablando de ciclos de reloj.

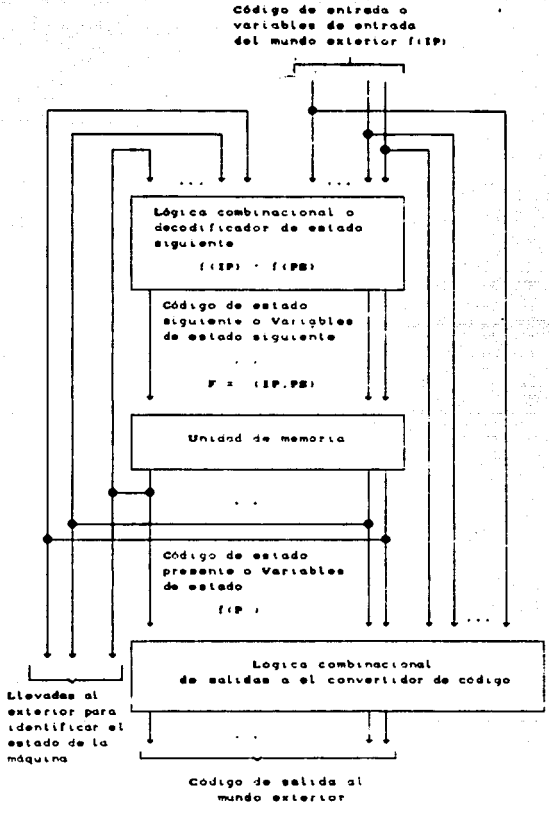


Figura 2.2. Modelo general de una maquina secuencial o de estado-finito.

Definiendo el estado de la máquina de esta manera, el diseño de un circuito secuencial se basa en algo más que en un diseño combinacional con ciertos forzamientos. Es posible describir la operación en un proceso discreto paso-a-paso usando una gráfica como ayuda llamada *diagrama de estado*.

De la figura 2.2 y considerando la función del bloque combinacional etiquetado como "decodificador de Estado Siguiente", algunas veces llamado "entradas o lógica de formación del estado siguiente". La función de esta red lógica es decodificar las entradas del mundo exterior y el estado presente de la máquina (almacenado por la memoria) y generar como salida un código llamado el código de estado siguiente. Este código de estado siguiente llegará a ser el código de estado presente cuando la memoria lo carge y almacene.[1] Este proceso es definido como un estado de cambio o un *cambio de estado*. El Cambio de estado es un proceso continuo con cada nuevo estado y las condiciones de estado presentes son decodificadas para formar el nuevo código de estado siguiente(patrón de bits).

El bloque lógico combinacional de la figura 2.2 etiquetado como "salidas al convertidor de código de mundo exterior" tiene la función basica de decodificar el estado presente de la máquina y la condición de entrada presente para el proposito de generar las salidas de control deseadas al mundo exterior.

En resumen, una máquina secuencial es aquella en la que los estados pasan (códigos de salida de memoria) de manera secuencial. Los codigos de estado de una máquina son generados por el decodificador de estado siguiente que convierte las entradas del mundo exterior combinadas con el código de estado presente dentro de un patrón de bits llamado código de estado siguiente de la máquina. Tal combinación no es necesariamente una operación básica booleana, podrían ser varias.

[1] ESTO ES VERDAD UNICAMENTE CUANDO ELEMENTOS ESPECIALES DE MEMORIA SON USADOS.DE OTRA MANERA EL CODIGO DE ESTADO SIGUIENTE ES DECODIFICADO EN EL ELEMENTO DE MEMORIA PARA GENERAR EL VERDADERO CODIGO DE ESTADO PRESENTE.

2.1.1 Clasificación de las máquinas secuenciales

Los siguientes modelos pueden ser derivados del modelo básico mostrado en la figura 2.2 por un proceso de degeneración, como se muestra atraves de las figuras 2.3 a 2.7.

La figura 2.3 muestra el diagrama de una máquina secuencial definida como máquina de MEALY en honor a G.H. Mealy, uno de los pioneros en el diseño secuencial. La distinción básica de la máquina de Mealy es que las salidas a el mundo exterior son una función de dos tipos de variables: (1) las condiciones de entradas presentes y (2) el estado presente de la máquina.

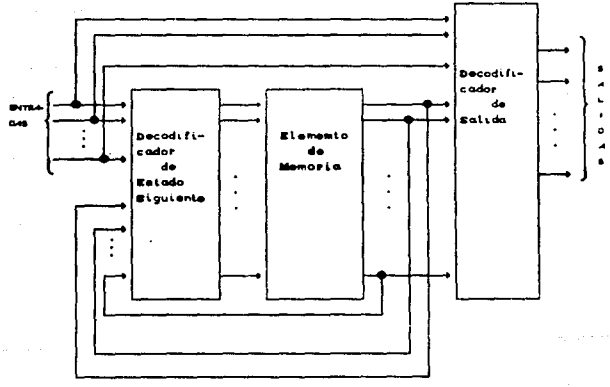


FIGURA 2.3 Máquina de Mealy

Las figuras 2.4 y 2.5 muestran las máquinas definidas como máquinas de MOORE en honor a E.F. Moore, otro pionero en el diseño de circuitos secuenciales. La distincion básica de una máquina de Moore es que sus salidas están estrictamente en función de un estado de la máquina. Tanto como la máquina de Mealy como la de Moore son ampliamente usadas y es posible derivar máquinas que son combinaciones de ambas; en otras palabras, algunas salidas son condicionadas a las entradas y el estado de la máquina, mientras en otras son dependientes unicamente del estado de la máquina.

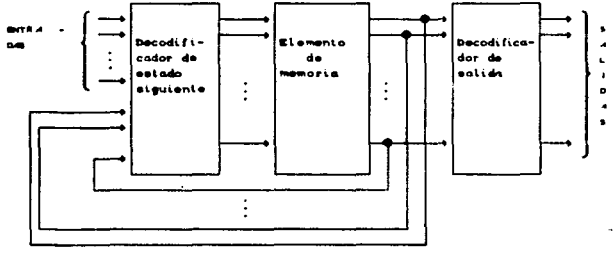


FIGURA 2.4 Máquina de Moore

Los circuitos mostrados en las figuras 2.3, 2.4,y 2.5 con una salida simple son máquinas interesantes. Estas forman el módelo general para un circuito en el cual los eventos a ser contados son introducidos directamente dentro del elemento de memoria o atraves del decodificador de estado siguiente.

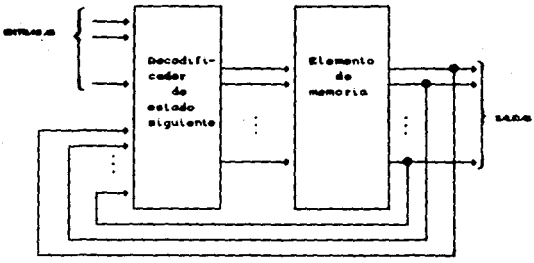


FIGURA 2.5 Máquina de Moore sin decodificador de salida

Es importante notar que las máquinas anteriormente descritas son igualmente aplicables para circuitos síncronos y asíncronos, y el número mínimo de entradas para cualquiera de estas máquinas es uno. Para los circuitos síncronos esa entrada puede ser el sistema de reloj.

2.2 REPRESENTACIÓN POR MEDIO DE DIAGRAMAS

Para la representación de los circuitos secuenciales existen dos métodos gráficos:

1. Diagrama de estados,
2. Carta ASM.

2.2.1 Diagrama de estado

El diagrama de estado es una ayuda pictórica usada para describir o definir gráficamente las transiciones de estado a estado de una máquina secuencial. El diagrama de estado es para el diseño de circuitos secuenciales lo que la tabla de verdad es para el diseño de circuitos combinacionales. Un ejemplo de un diagrama de estados se ilustra en la figura 2.6, se muestra un estado cualquiera con cinco características importantes para información relativa para cada estado de la máquina. El diagrama de estados es un arreglo de círculos conectados por segmentos de líneas con una punta de flecha. Cada círculo representa un estado de la máquina y

los segmentos de líneas son indicaciones gráficas de los cambios del estado.

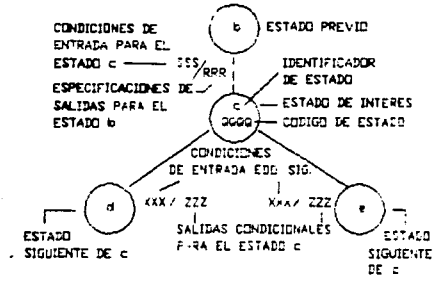
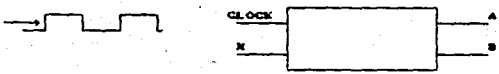
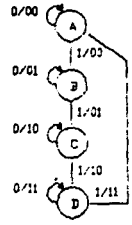


Figura 2.6. Segmento de un diagrama de estado en un formato definido para describir los cinco componentes más importantes para cada estado de la máquina secuencial

Para ilustrar el uso del diagrama de estado, considere el siguiente ejemplo:
Suponga que una máquina secuencial tiene una sola entrada de control llamada X y el reloj, y dos salidas A y B. Cada que se presenta un flanco de subida del reloj, el código sobre A y B cambia de 00 a 01 a 10 a 11 y entonces repite por si misma si X esta presente; si en cualquier instante de tiempo X no está presente, esta máquina supone una retención del estado presente. El diagrama pictórico de esta máquina secuencial se muestra en la figura 2.7.



(a)



FORMATO : X/AB

* EL RELOJ ES CONSIDERADO IMPLICITO

(b)

Figura 2.7. (a) Diagrama de bloque y (b) diagrama de estado definiendo la operación de la maquina secuencial

La máquina es descrita con mayor claridad por el diagrama de estado que por medio de palabras. El diagrama permite examinar cada estado y obtener información referente al estado previo, las condiciones para el estado siguiente, y la salida para cada estado.

Las caracteristicas descriptivas del diagrama de estado es una ayuda invaluable en las siguientes tres áreas:

1. La definición de sistemas secuenciales.
2. el analisis de circuitos secuenciales.
3. el diseño de circuitos secuencialet.

2.2.2 Carta ASM

La carta ASM (Algorithmic State Machine) provee una representación gráfica de la transición de estado y salidas de una máquina de estado. Se usa como ayuda en el diseño de una máquina de estado para la implementación de un algoritmo y llega a ser parte complementaria de la documentación del diseño. La carta se conforma de tres símbolos básicos. Estos son:

1. La caja de estado,
2. la caja de decisión, y
3. la caja de salidas condicionales.

La caja de estado representa un estado del ASM. La máquina permanece en una caja de estado durante un tiempo de estado. El símbolo se muestra en la figura 2.8 e indica que cada estado tiene un *nombre*, usualmente un mnemónico o un número, y un *código de estado*. El código de estado representa una combinación única de valores de las variables de estado durante el proceso de *asignación de estado*. Por lo consiguiente, es probable desconocer este cuando la carta ASM es primero dibujada y debe ser incorporado subsecuentemente. Las salidas generadas durante el estado, también están representadas en forma mnemónica, son listadas en la caja de estado y constituyen la *lista de salidas de estado*. Una salida de estado es activa unicamente cuando la maquina reside en un estado en la que su lista de salidas de estado contiene la salida listada. Las salidas de estado están activas durante un tiempo de estado. Cada caja de estado tiene una ruta de entrada y una ruta de salida. La ruta de salida pude cargar a otra caja de estado, en el caso de una transición de estado directa o incondicional, o puede cargar a una caja de decisión en el caso de una transición condicional.

La caja de decisión, mostrada en la figura 2.9, involucra las entradas a la máquina de estado y da las condiciones que controlan o qualifican las transiciones de estado condicional y salidas condicionales. La caja contiene una expresion booleana, involucrando las entradas, las cuales definen la condición, y tiene dos rutas de salida. Si la condicion es verdadera, la *ruta*

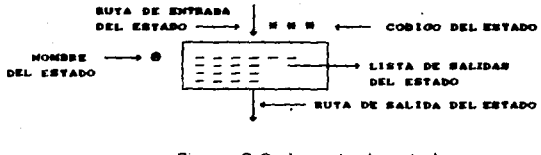


Figura 2.8. La caja de estado

de salida verdadera es seguida, si es falsa la *ruta de salida falsa* es usada. Estas rutas son indicadas por V y F, pero si las condiciones lógicas S o N prefijadas son omitidas de la lógica establecida las rutas de salida pueden ser etiquetadas con 1 y 0, asumiendo una lógica positiva como convención se tiene que V = 1, F = 0. Las rutas de salida pueden ser cargadas con otras cajas de decisión, cajas de estado, o cajas de salidas condicionales.

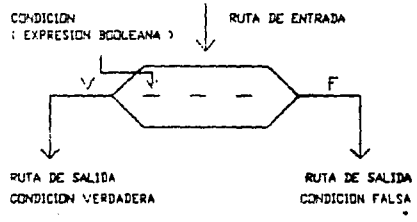


Figura 2.9 La caja de decisión

La caja de salida condicional describe aquellas salidas que llegan a ser activas unicamente si ciertas condiciones, definidas en términos de las entradas del sistema, son verdaderas. Por consiguiente una caja de salidas condicionales contiene una *lista de salidas condicionales* y siempre estan asociadas con una caja de decisión la cual describe la condición. La ruta de entrada puede originarse de una caja de decisión pero la ruta de salida puede cargar a otra caja de decisión o una caja de estado. El simbolo

esta dado en la figura 2.10.

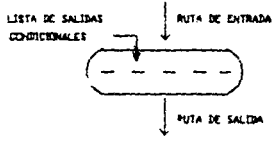


Figura 2.10 La caja de salida condicional

Estos elementos básicos o primitivos pueden ser combinados juntos para formar la unidad básica de una carta ASM la cual es el *bloque ASM*. Esta es una estructura conformada por una caja de estado y una red de cajas de decisión y cajas de salidas condicionales tal como se representa en la figura 2.11. Un bloque ASM tiene una ruta de entrada y muchas rutas de salida. Cada ruta de salida puede cargar a otro estado y por tanto ser una ruta de entrada a otro bloque ASM o a su propio bloque. Cada ruta posible de un estado a el siguiente es llamada una *ruta ligada*. Cada ruta ligada corresponde a una expresión booleana simple la cual contribuye a la expresión completa para una salida condicional o a una función de estado-siguiente. Sin un bloque ASM la caja de estado es el único elemento que indica un factor de tiempo y todas las otras cajas son asumidas por ser corrientemente activadas. Por lo tanto este es un paso cuantico en tiempo cuando el bloque es accedido y el orden de las cajas de decisión y cajas de salidas condicionales es inmaterial desde un punto de vista de tiempo como todas las funciones condicionales y salidas son evaluadas simultaneamente sin considerar sus posiciones en el bloque. En este aspecto la carta ASM difiere de un programa o carta de flujo de señal. Una carta ASM consiste de uno o más bloques ASM interconectados de una manera consistente y describe el comportamiento de una máquina de estado.

En la construcción de cartas ASM deben de ser tomadas precauciones para asegurar que estas sean físicamente realizables. Primero, nosotros debemos asegurar siempre que cada estado cargo a un único estado siguiente para cada set estable de condiciones estables. La figura 2.12 muestra algunos arreglos inválidos. La figura 2.12a es claramente imposible porque dos estados siguientes son indicados simultaneamente pero la ambigüedad de la figura 2.12b es menos obvia. Si X y Y son verdaderas hay una única transición de estado. Sin embargo, si cualquiera de las dos es falsa entonces dos estados siguientes son indicados. La conducta probable pensada esta dada en la figura 2.13 la cual ha removido la ambigüedad por medio de un simple reordenamiento de las cajas de decisión. Segundo, no deben de haber lazos cerrados los cuales no incluyan o contengan al menos una caja de estado. Esto es, que todos los lazos deben abandonar el bloque ASM por una de las rutas

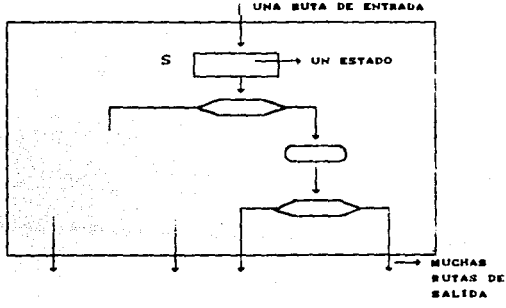


Figura 2.11. Diagrama de un bloque ASM

de salida y reentrar al bloque por la ruta de entrada normal. Entonces el arreglo de la figura 2.14a es corregido por el dibujo de la figura 2.14b.

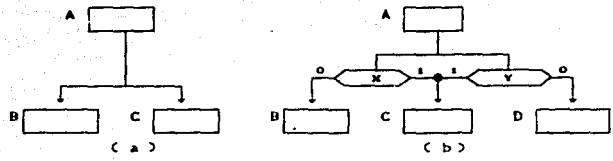


Figura 2.12 Algunas estructuras inválidas para cartas ASM

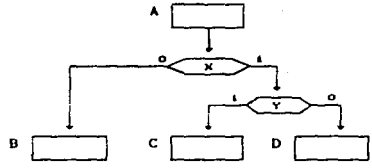


Figura 2.13 Una estructura ASM correcta para la figura 2.12b

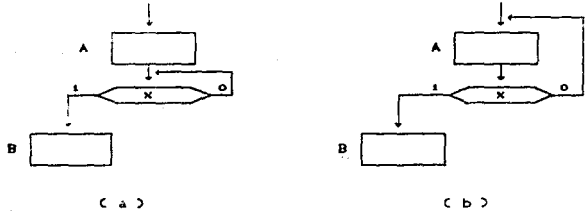


Figura 2.14 Representación de lazo invalido (a) y válido (b)

Otros errores comunes son debidos a arreglos indebidos con cajas de decisión los cuales acarrean condiciones lógicas imposibles, rutas ligadas ficticias y transiciones de estado ambiguas. En la figura 2.15a y 2.15b, las aparentes rutas ligadas entre los estados A y B no pueden existir porque estas requieren que X sea verdadera y falsa simultaneamente. De manera similar,

pero de una manera menos obvia, es el arreglo de la figura 2.15c donde la ruta de liga de A a B es imposible porque se requiere que

$$\bar{X}\,\bar{Y}\,(X + Y) = \bar{X}\,\bar{Y}\,X + \bar{X}\,\bar{Y}\,Y = 0$$

lo cual nunca será cierto. Es perfectamente posible que un calificador o condición pueda estar presente en algunas cajas de decisión en un bloque ASM y dar una construcción válida. La figura 2.16 ilustra dos de estas situaciones. Nosotros tenemos la libertad de usar series o paralelos equivalentes para interconectar cajas de decisión, dentro de un bloque ASM. Como no existe una relación de tiempo entre las cajas de salida condicional o decisión los dos arreglos de la figura 2.17 son equivalentes y las tres salidas llegan a estar disponibles simultaneamente, previendo que las condiciones son correctas, en ambos arreglos. Las estructuras en serie son menos propensas a acarrear ambigüedades en transiciones de estado pero la estructura en paralelo puede hacer la carta mas compacta.

Las cartas ASM contienen más información que los diagramas de estado para diseño de circuitos secuenciales tradicionales porque despliegan el comportamiento secuencial de la ejecucion del estado de la máquina y su algoritmo, también revelan los componentes de las funciones requeridas en la síntesis de el ASM.

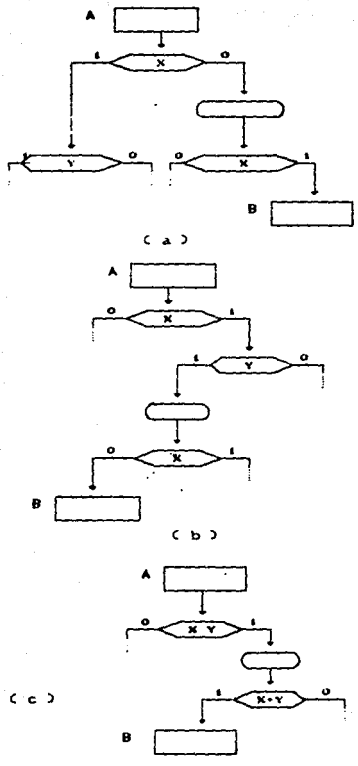


Figura 2.15 Cartas ASM con rutas de liga imposibles

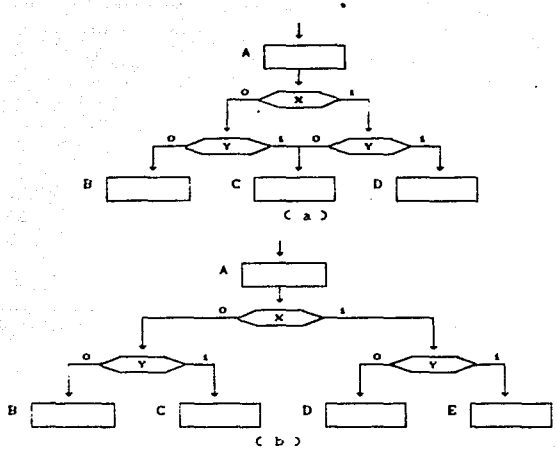


Figura 2.16 Algunas estructuras de cartas ASM válidas

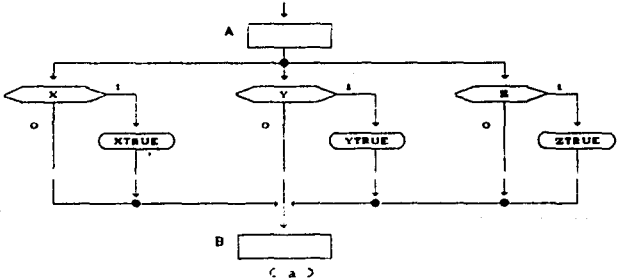


Figura 2.17 Estructuras serie y paralelo equivalentes

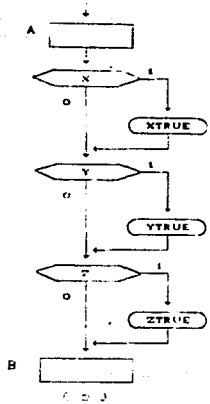


Figura 2.17 (continuación) Estructuras serie y paralelo equivalentes

3. *TECNICAS DE DISEÑO.*
(FLIP-FLOPS, CONTADORES, REGISTROS)

Los sistemas secuenciales síncronos son aquellos en los que las variables de estado interno o las variables de entrada no actúan directamente sobre el sistema, sino que lo hacen en los instantes en los que éste recibe impulsos procedentes de un generador.

En estos sistemas no existe el problema de las transiciones críticas porque la transición entre estados internos se realiza al producirse un impulso de sincronización que es activo en uno de los dos flancos de cambio, el de subida o el de bajada.

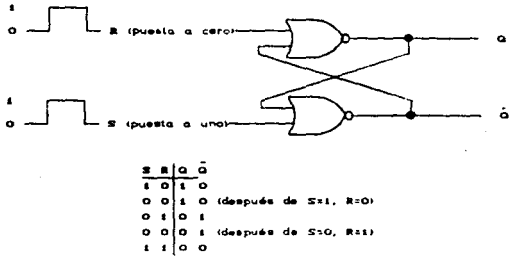
El elemento de memoria en este tipo de circuitos secuenciales se llaman *flip-flops*. Estos circuitos son celdas binarias capaces de almacenar un bit de información. Un circuito flip-flop tiene dos entradas, una para el valor normal y una para el valor complemento del bit almacenado en él.

3.1. FLIP-FLOPS

Un circuito flip-flop puede mantener un estado binario indefinidamente (siempre y cuando se este suministrando potencia al circuito) hasta que se cambie por una señal de entrada para cambiar estados.

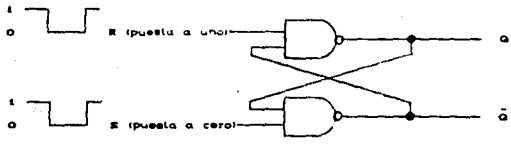
3.1.1. Circuito basico de un flip-flop

Un circuito flip-flop se puede construir con dos compuertas NAND o dos compuertas NOR (figuras 3.1 y 3.2). La conexión de acoplamiento intercruzado de la salida de una compuerta a la entrada de la otra constituye un camino de realimentación. Cada flip-flop tiene dos salidas, Q y $\bar{Q}$ y dos entradas, S (set) y R (reset).



S	R	Q	$\bar{Q}$	
1	0	1	0	
0	0	1	0	(después de S=1, R=0)
0	1	0	1	
0	0	0	1	(después de S=0, R=1)
1	1	0	0	

Figura 3.1. Circuito de flip-flop básico con compuertas NOR

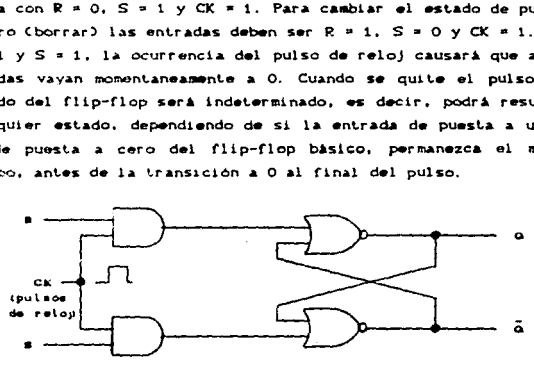


S	R	Q	$\bar{Q}$	
1	0	0	1	
1	1	0	1	(después de R=1, R=0)
0	1	1	0	
1	1	1	0	(después de S=0, R=1)
0	0	1	1	

Figura 3.2. Circuito de flip-flop básico con compuertas NAND

3.1.2. Flip-flop RS temporizado

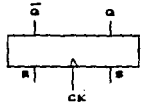
El flip-flop básico por si solo es un circuito secuencial asincrono. Agregando compuertas a la entrada del circuito básico, puede hacerse que el flip-flop responde a los niveles de entrada durante la ocurrencia del pulso del reloj. El flip-flop RS temporizado mostrado en la figura 3.3 consiste en un flip-flop básico NOR y dos compuertas AND. Las salidas de las compuertas AND permanecen en cero mientras el pulso del reloj (CK) sea cero, independientemente de los valores de entrada R y S. Cuando el pulso del reloj vaya a 1, la información de las entradas R y S se permiten llegar al flip-flop básico. El estado de puesta a uno se logra con R = 0, S = 1 y CK = 1. Para cambiar el estado de puesta a cero (borrar) las entradas deben ser R = 1, S = 0 y CK = 1. Con R = 1 y S = 1, la ocurrencia del pulso de reloj causará que ambas salidas vayan momentaneamente a 0. Cuando se quite el pulso, el estado del flip-flop será indeterminado, es decir, podrá resultar cualquier estado, dependiendo de si la entrada de puesta a uno o la de puesta a cero del flip-flop básico, permanezca el mayor tiempo, antes de la transición a 0 al final del pulso.



(a) Diagrama lógico

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminado
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminado

(b) Tabla de caracteristica



(c) Simbolo gráfico

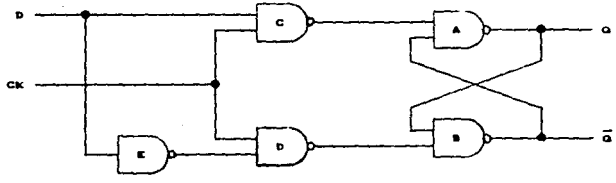
Figura 3.3. Flip-flop RS temporizado

El símbolo gráfico del flip-flop RS sincronizado tiene tres entradas: R, S y CK y dos salidas: Q y $\bar{Q}$.

3.1.3. Flip-flop D

El flip-flop D mostrado en la figura 3.4 es una modificación del flip-flop RS sincronizado. La entrada D va directamente a la entrada S y su complemento a la entrada R a través de la compuerta F. Mientras que el pulso de reloj sea 0, las compuertas C y E tienen un 1 en sus salidas, independientemente del valor de las otras entradas. La entrada D se comprueba durante la ocurrencia del pulso de reloj. Si es 1, la salida de la compuerta C va a 0, cambiando el flip-flop al estado de puesta a 1. Si es 0 , la salida de la compuerta E va a 0, cambiando el flip-flop al estado de borrado.

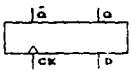
El flip-flop tipo D recibe su nombre por la habilidad de transmitir "datos" a un flip-flop. Es basicamente un flip-flop RS con un inversor en la entrada R. El inversor reduce el número de entradas de dos a uno. El símbolo del flip flop D sincronizado se muestra en la figura 3.5 al igual que la tabla característica.



(a) Diagrama lógico con compuertas NAND

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

(b) Tabla característica



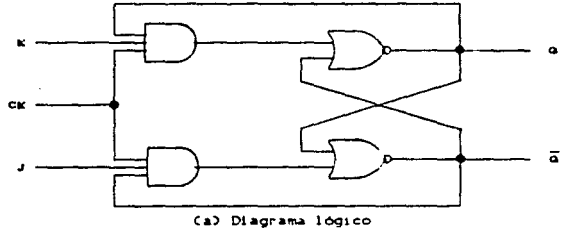
(c) Símbolo gráfico

Figura 3.5. Flip-flop D temporizado

3.1.4. Flip-flop JK

Es un refinamiento del flip-flop RS ya que el estado indeterminado del flip-flop RS se define en el tipo JK. Las entradas J y K se comportan como las entradas R y S para poner a uno o cero al flip-flop, la J se usa para la entrada de puesta a 1 y la K para la entrada de puesta a 0. Cuando ambas entradas se aplican a J y K simultaneamente, el flip-flop cambia a su estado de complemento, esto es, si Q = 1 cambia a Q = 0 y viceversa.

Un flip-flop sincronizado se muestra en la figura 3.6. La salida Q se aplica junto con K y CK a una compuerta AND de tal manera que el flip-flop se ponga a cero (clear) durante un pulso de reloj, solamente si Q fue 1 previamente. De manera similar la salida $\bar{Q}$ se aplica con J y CK a una compuerta AND de tal manera que el flip-flop se ponga a uno con un pulso de reloj, solamente si $\bar{Q}$ fue 1 previamente. Además en la figura 3.6 se muestra la tabla característica, y se puede observar que el flip-flop JK es como un flip-flop RS excepto cuando J y K son igual a 1. Cuando esto ocurra el pulso de reloj se transmite a través de una compuerta AND solamente; aquella cuya entrada se conecta a la salida del flip-flop que presente un 1. El simbolo gráfico de este flip-flop también es mostrado en la figura 3.6.

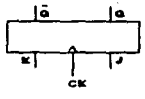


(a) Diagrama lógico

Figura 3.6. Flip-flop JK temporizado

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(b) Tabla característica

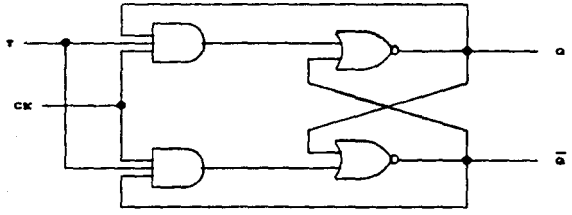


(c) Símbolo gráfico

Figura 3.6. Flip_flop JK temporizado

3.1.5. Flip-flop T

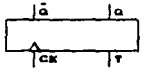
El flip-flop T es la versión de una entrada del flip-flop JK (figura 3.7), se logra uniendo las dos entradas de un flip-flop tipo JK. El nombre se deriva de la habilidad del flip-flop de variar ("toggle") o cambiar de estado. Independientemente del presente estado del flip-flop, este asume el estado de complemento cuando ocurre el pulso del reloj mientras que la entrada T esté en 1 lógico.



(a) Diagrama lógico

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(b) Tabla característica



(c) Símbolo gráfico

Figura 3.7 Flip-flop T temporizado

3.1.6. Analisis de los circuitos secuenciales temporizados

El comportamiento de los circuitos secuenciales se determina de las entradas, las salidas y los estados de los flip-flops. Ambas salidas y el siguiente estado son una función de las entradas y el presente estado. El análisis de los circuitos secuenciales consiste en obtener una tabla o un diagrama de la secuencia de tiempo de las entradas, salidas y estados internos. Es posible escribir expresiones de Bool que describan el comportamiento de los circuitos secuenciales. Sin embargo, estas expresiones deben incluir la secuencia de tiempos necesaria directa o indirectamente.

En un diagrama se puede reconocer como circuito secuencial, a aquel que contenga flip-flops. Los flip-flops pueden ser de cualquier tipo y el diagrama lógico puede o no incluir compuertas combinacionales.

3.1.7. Tablas de estado

La secuencia de tiempo de las entradas, salidas y estados de los flip-flops pueden enumerarse en una *tabla de estados* (o tabla de transición). Consiste en tres secciones llamadas *estado presente*, *estado siguiente* y *salida*. El *estado presente* designa los estados de los flip-flops antes de la ocurrencia de un pulso de reloj. El *estado siguiente* muestra los estados de los flip-flops después de la aplicación del pulso de reloj y la sección de *salida* lista los valores de las variables de salida durante el presente estado.

La deducción de la tabla de estado comienza a partir de un estado inicial asumido. El estado inicial de la mayoría de los circuitos secuenciales practicos se define como el estado con ceros en todos los flip-flops. Aunque este estado inicial depende del diseñador y puede ser cualquier otro, el que considere óptimo para disminuir su circuiteria combinacional o la lógica de control. En general, un circuito secuencial con m flip-flops y n variables de entrada tendrá 2^m renglones, una para cada estado. Las secciones del siguiente estado y de salida tendrán cada una 2^n columnas, una para cada entrada.

3.1.8 Diagrama de estado

La información disponible en la tabla de estado puede representarse gráficamente en un *diagrama de estado*. En este diagrama se representa un estado por un círculo y la transición entre estados se indica por líneas dirigidas que conectan los círculos. En la figura 3.8 se muestra un diagrama de estado, el número binario mostrado a un lado de cada círculo identifica el estado presentado por este. Las líneas dirigidas se marcan con la entrada que indica tal transición, indicando el nivel lógico que posee. Dentro de cada círculo es indicada la salida que se presenta durante ese estado. Una línea curva dirigida que conecta un círculo a sí mismo indica que no hay cambio de estado. Por ejemplo, en la figura 3.8, la línea dirigida del estado 00 al estado 01 marcada con un 1, significa que el circuito secuencial esta en el estado presente 00 mientras la entrada es 1 y la salida 0 y que al finalizar el siguiente pulso de reloj el circuito va al siguiente estado. El diagrama de estado suministra la misma información que la tabla de estado.

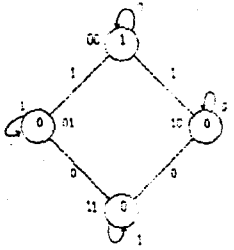


Figura 3.6. Diagrama de estados

Un ejemplo más claro de diagrama de estado se muestra en la figura 3.9, en donde se le da un nombre específico a los estados, a las entradas y a las salidas, para un mejor identificación de lo que sucede.

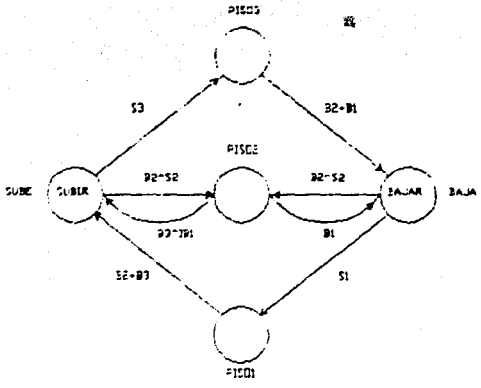


Figura 3.9. Diagrama de estados de un elevador

Como se puede observar las palabras PISO1, PISO2, PISO3, SUBE y BAJA, se encuentran en el exterior del circulo, lo que significa que es el nombre del estado. SUBIR y BAJAR, se encuentran dentro de un circulo respectivo, esto indica que son las salidas durante ese estado, y en las lineas de transición son presentadas las entradas o combinaciónes de estas. B2*S2 significa que es una AND entre B2 y S2, B2+B1 indica una operación OR, el simbolo "!" previo a una salida indica que dicha salida es negada, es decir, activa baja ó cero lógico.

No hay diferencia entre una tabla de estado y un diagrama de estado excepto en la forma de la presentación. La tabla de estado es más fácil de deducir a partir de una diagrama de lógica dado y el diagrama de estado se desprende directamente de la tabla de estado. El diagrama de estado da una vista pictorica de las transiciones de estado y está en forma disponible para la interpretación binaria de la operación del circuito. El diagrama

de estado se usa a menudo como la especificación de diseño inicial de un circuito secuencial.

3.1.9. Ecuaciones de estado

Una *ecuación de estado* es una expresión algebraica que especifica las condiciones para la transición de estado de un flip-flop. El lado izquierdo de la ecuación denota el estado siguiente del flip-flop y el lado derecho una función de Boole que especifica las condiciones del presente estado que hacen el siguiente estado igual a 1.

3.1.10. Funciones de entrada de un flip-flop

El diagrama lógico de un circuito secuencial consiste en elementos de memoria y compuertas. La clase de flip-flops y la tabla característica especifican las propiedades lógicas de los elementos de memoria. Las interconexiones entre las compuertas forman un circuito combinacional y se pueden expresar algebraicamente con funciones de Boole. Así, un conocimiento del tipo de flip-flop y una lista de funciones de Boole del circuito combinacional darán toda la información necesaria para dibujar el diagrama lógico de un circuito secuencial. La parte del circuito combinacional que genera las salidas externas se describe algebraicamente por *funciones de salida del circuito*. La parte del circuito que genera las entradas de los flip-flops se describe algebraicamente por un conjunto de funciones de Boole llamadas *funciones de entrda del flip-flop* o algunas veces ecuaciones de entrada.

En algunos textos se ha adoptado la convención de usar dos letras para designar una variable de entrada de un flip-flop: la primera designa el nombre de las entradas y la segunda el nombre del flip-flop. Por ejemplo:

$$JA = BC'x + B'Cx'$$

$$KA = B + y$$

la primera letra representa las entradas J y K de un flip-flop JK, la segunda letra identifica el flip-flop en el circuito. El lado derecho de cada ecuación es una función de Boole para la correspondiente variable de entrada del flip-flop. La

configuración de las funciones anteriores es presentada en la figura 3.10.

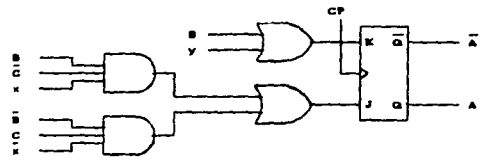


Figura 3.10. Configuración de las funciones de entrada de un flip-flop

3.1.11. Procedimiento de diseño

El diseño de un circuito secuencial temporizado comienza a partir de un conjunto de especificaciones y culmina en un diagrama lógico o una lista de funciones de Boole de las cules se puede obtener el diagrama lógico. El primer paso en el diseño de los circuitos secuenciales es obtener una tabla de estado o una representación equivalente tal como un diagrama de estado o ecuaciones de estado.

Un circuito secuencial sincrono se hace de flip-flops y compuertas combinacionales. El diseño del circuito consiste en escoger los flip-flops y luego encontrar una estructura de compuertas combinacional, la cual, conjuntamente con los flip-flops, produce un circuito que satisface las caracteristicas enunciadas. El número de flip-flops se determina por el número de estados necesarios en el circuito. El circuito combinacional se deriva de la tabla de estados, de hecho una vez que el tipo y número de los flip-flops se determinen, el proceso de diseño envuelve una transformación del problema del circuito secuencial al problema del circuito combinacional. De esta manera las técnicas de diseño de los circuitos combinacionales pueden ser aplicadas mediante una serie de pasos consecutivos que se recomiendan como sigue:

1. Se establece la decripción en palabras del comportamiento del circuito. Esto puede acompañarse por el diagrama de

estado, un diagrama de tiempos, u otra información pertinente.

2. De la información dada del circuito se obtiene la tabla de estado.
3. Se asignan valores binarios a cada estado si la tabla de estado obtenida anteriormente contiene símbolos de letras.
4. Se determina el número de flip-flops necesarios para asignar una letra a cada uno.
5. Se escoge el tipo de flip-flop que se va a utilizar.
6. A partir de la tabla de estado, se deduce la excitación del circuito y las tablas de salida.
7. Usando un método de simplificación, se deducen las funciones de salida del circuito y las funciones de entrada del flip-flop.
8. Se dibuja el diagrama lógico.

Es necesario que el diseñador utilice su intuición y experiencia para llegar a la correcta interpretación de las especificaciones del circuito, una vez que se hayan establecido tales especificaciones así como la tabla de estado, es posible hacer uso del procedimineto formal de diseño.

El número de flip-flops se determina por el número de estados. Un circuito puede tener estados binarios sin usar si el número de estado es menor que 2^m. los estados no usados se consideran como condiciones de no importa durante el diseño del circuito combinacional del circuito, lo cual lo hace más fácil.

El tipo de flip-flop a utilizarse, puede ser incluido en las especificaciones de diseño, o depende de la disposición del diseñador. Cabe hacer notar que los flip-flops tipo JK son los mas utilizados, por su mayor versatilidad y disponibilidad. Cuando hay varios tipos de flip-flops disponibles se recomienda usar el tipo RS o D para transferencia de datos; el tipo T para aplicaciones que incluyen complementación, y el JK para aplicaciones en general.

La información de salida externa se epecifica en la sección de salida de la tabla de estado. De ella se pueden deducir las funciones de salida del circuito. La tabla de exitación del circuito es simular a la de los flip-flops individuales, excepto

que las condiciones de entrada son dictadas por la información disponible en el estado presente y las columnas del estado siguiente de la tabla de verdad.

La tabla de excitación de un circuito secuencial con m flip-flops, k entradas por flip-flop y n entradas externas, consiste en $m + n$ columnas para el estado presente y las variables de entrada y hasta 2^{m+n} renglones listados en alguna cuenta binaria conveniente. La siguiente sección de estado tiene m columnas, una para cada flip-flop. Los valores de entrada de los flip-flops se listan en mk columnas, una para cada entrada de cada flip-flop. Si el circuito contiene j salidas, la tabla debe incluir j columnas. La tabla de verdad del circuito combinacional se toma de la tabla de excitación considerando el estado presente $m+n$ y las columnas de entrada como *entradas*, y los valores de entrada del flip-flop $mk+j$ y las salidas externas como *salidas*.

3.2. CONTADORES

Los contadores son sistemas secuenciales con una sola entrada de impulsos, cuyo estado interno en cada instante representa el número de impulsos que se hayan aplicadoa a aquélla. El número de estados internos debe ser igual al máximo de impulsos que se desean contar, más uno, que será el estado inicial el cual indicará la ausencia de impulsos.

Para la realización de los contadores se utilizan flip-flops sincronizados que poseen dos estados internos. Esto es si se desean realizar en forma "discreta", pero nosotros no abocaremos a el diseño utilizando a los contadores en forma integrada, es decir, un contadores en un circuito integrado.

3.2.1. Síntesis de los sistemas secuenciales síncronos con contadores.

Los diagramas de flujo de los sistemas secuenciales síncronos más utilizados en la práctica evolucionan entre estados internos de dos formas:

a) Sin intervención de las variables externas de entrada.

b) Dependiendo del estado de una o más variables externas.

Por lo tanto, el diagrama de flujo es una combinación de un contador (diseñado con flip-flops en forma discreta) y de un sistema secuencial síncrono general.

Los contadores síncronos con entradas en paralelo síncronas pueden ser utilizados como bloques funcionales de la síntesis de los sistemas secuenciales síncronos.

Debido a que no existe una unidad de criterio para la denominación de las terminales de estos circuitos (contadores), lo mostramos como sigue:

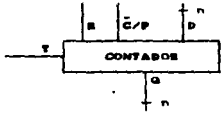


Figura 3.11.

La entrada R permite poner al contador en sus estado inicial mediante la aplicación en ella de un determinado nivel activo. Se supone que este nivel es uno lógico cuando se representa un símbolo sin invertir (R) y que es un cero lógico cuando se indica invertido ($\bar{R}$). La entrada $\bar{C}/P$ permite seleccionar la forma de actuación del contador cuando recibe los flancos (inicios finales de pulsos) activos en la entrada T. Si se encuentra en un determinado estado, el flanco activo aplicado a la entrada T es contado en un cierto código que suele ser el binario natural o el BCD natural, dependiendo del tipo de contador; si se encuentra en el estado activo opuesto, la información en las entradas D se introducen en paralelo en el contador. En general las entradas de un bloque funcional cuyo nivel decide la forma de actuación se indicarán siempre mediante dos letras separadas por una diagonal.

El esquema de bloques del sistema secuencial síncrono mediante un contador se representa en la figura 3.12. El sistema combinacional recibe como entradas las salidas Q del contador y las variables externas y genera la señal $\bar{C}/P$ y las entradas D del contador.

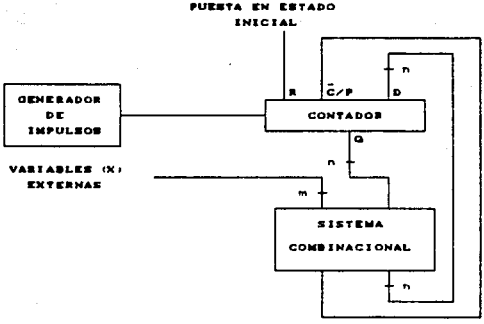


Figura 3.12

En primer lugar se observa el número de estados del diagrama de flujo para deducir el número de flip-flops n que ha de tener el contador (número de bits). El número n debe de ser tal que $2^n \geq m$, siendo m el número de estados del diagrama de flujo

La asignación de códigos binarios a los estados se debe realizar de tal forma que el número de saltos entre estados que no tengan asignados codigos consecutivos sea mínimo.

Siempre que sea posible, a los estados consecutivos se les asignan números decimales consecutivos, por ejemplo; E_0 y E_1. Una vez teniendo el diagrama de estados se procede a obtener la tabla de verdad. En ella se representa como variables independientes el estado del contador en el instantye t, antes de recibir el flanco activo y las variables externas, y como variables dependientes del estado del contador en el instante $t + 1$, después de recibir el flanco activo y la variable $\bar{C}/P$.

Las ecuaciones de las variables se pueden deducir con un metodo de simplificación tal como mapas de Karnaugh o algún método numérico.

3.3. REGISTROS DE DESPLAZAMIENTO

Un registro de dezplazamiento es un circuito secuencial formado por flip-flops sincronizados e interconectados entre si uniendo las salidas del primero a las entradas del segundo y asi sucesivamente. Además se unen también entre si las entradas de impulsos de todos los flip-flops. Al suministrar un impulso a esta conexión común se introduce en el primer flip-flop del registro la información presente en las entradas sincronizadas, la que habia en este flip-flop se desplaza al segundo y asi sucesivamente. Según la forma de interconectar los flip-flops, el desplazamiento puede ser a la izquierda o a la derecha o reversible, si se puede seleccionar externamente con una linea de control.

Las terminales de un registro de desplazamiento son:

a) Entrada serie: Está constituida por las entradas sincronizadas del primer flip-flop.

b) Entradas paralelo: Permiten poner todos los flip-flops del registro en un estado determinado.

c) Salida serie: Está constituida por la salida del último flip-flop del registro.

d) Salida en paralelo: Son todas las salidas de los flip-flops que forman el registro.

La clasificación por su modo de operación es la siguiente:

1. Entrada paralelo-salida patalelo.
2. Entrada paralelo-salida serie.
3. Entrada serie-salida paralelo.
4. Entrada serie-salida serie
5. Universal.

El más simple de todos los registros de corrimiento es el mencionado en primer lugar, entrada paralelo-salida paralelo. Que como su nombre lo indica al introducir datos en paralelo después de un pulso de reloj nos los entrega en paralelo.

El de entrada paralelo-salida serie, se entregan los datos de igual manera pero nos entrega una secuencia de bits, es decir, la

salida es en serie, un dato (bit) a la vez.

El siguiente realiza la operación inversa al anterior.

El de entrada serie-salida serie, tanto las entradas como las salidas son en secuencia, es decir, un dato tras otro.

El más complejo es el universal, ya que involucra todas las formas de operar anteriormente mencionadas con solo enviar cierta combinación de entradas a las líneas de control. Un circuito de registro de desplazamiento comercial muy común es el 74194, cuyo diagrama es mostrado a en la figura 3.13.

De esta manera el diseño de circuitos secuenciales utilizando registros de desplazamiento se fácilita, ya que la asignación de estados no sería forzada a ser consecutiva, y con la versatilidad de los registros universales esta fácilidad aumenta, debido a que los corrimientos pueden ser hacia ambos lados y además permite cargas en paralelo y corrimientos en serie, lo cual podría disminuir la circuitería combinacional para la lógica de control.

S0, S1	ENTRADAS DEL MODO DE CONTROL
P0-P3	ENTRADA DE DATOS EN PARALELO
DER	ENTRADA DE DATOS EN SERIE (DESPLAZAMIENTOA LA DERECHA)
DEL	ENTRADA DE DATOS EN SERIE (DESPLAZAMIENTOA LA IZQ.)
CK	ENTRADA DEL RELOJ
MR	RESET MAESTRO
Q0-Q3	SALIDAS EN PARALELO

Figura 3.13

De esta manera el diseño de circuitos secuenciales utilizando registros de desplazamiento se fácilita, ya que la asignación de estados no sería forzada a ser consecutiva, y con la versatilidad de los registros universales esta fácilidad aumenta, debido a que los corrimientos pueden ser hacia ambos lados y además permite

cargas en paralelo y corrimientos en serie, lo cual podría disminuir la circuitería combinacional para la lógica de control.

3.3.1. Aplicación de los registros de desplazamiento

1. *Contadores en codigos especiales.* Conectando la salida del último flip-flop a la entrada del primero y poniendo inicialmente en estado uno un solo flip-flop se obtiene un contador en anillo. Si se conecta la salida inversa del último flip-flop a la entrada del primero, se obtiene un contador en código continuo y cíclico.

2. *Conversión paralelo-serie.* La información contenida en un registro de salida en paralelo, se introduce en un registro de desplazamiento a través de sus entradas en paralelo y aplicando impulsos de desplazamiento se obtiene en serie a través de su salida en serie (ver figura 3.14).

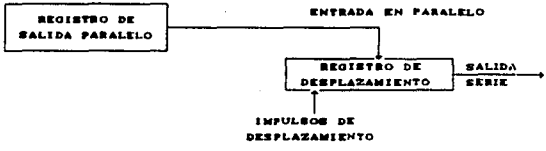


Figura 3.14

3. *Conversión serie-paralelo.* La información se introduce en el registro a través de su entrada serie y se obtiene en paralelo por medio de las salidas en paralelo, tal como se indica en la figura 3.15.

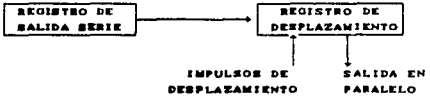


Figura 3.15

4 DESARROLLO DEL SISTEMA DE COMPUTO

En este capitulo, se describen cada uno de los algoritmos de funcionamiento de los módulos que componen al sistema. Para que exista una mayor comprensión tales algoritmos son descritos a manera de diagrama de bloques explicando después cada uno de éstos.

Debido a que el sistema lleva una secuencia de utilización, los módulos serán descritos de acuerdo a esta secuencia. Tal se puede observar en la figura 4.1. Aunque es importante mencionar que la descripción del funcionamiento del módulo circuitos se antepone a la del módulo resuelve debido a que es de gran utilidad el primero para el segundo.

Primero se crea un archivo de texto, en el cual se describe un diagrama de estados o una carta ASM del sistema secuencial en cuestión. Se lleva a cabo un análisis sintáctico y semántico del archivo creado. Una vez verificada la perfecta descripción del sistema se genera la tabla correspondiente al mismo. Ya obtenida la tabla se solicita la indicación de los circuitos, almacenados en ese módulo, con los cuales se desea resolver la tabla y hace lo propio. Seguido esto, con la tabla ya resuelta, la simplifica a su mínima expresión en forma de funciones booleanas, indicando una para cada linea de control y entrada de cada circuito indicado. Por ultimo, con la simplificación efectuada, se impementan dichas

funciones booleanas.

4.1. MODULO MENU.

En este módulo se lleva a cabo la parte del sistema de cómputo que muestra el formato general del sistema, el cual indica el nombre del mismo en la parte superior, y en la inferior las teclas de funciones válidas con su respectiva tarea a ejecutar.

Cuando se oprima la tecla F8, se despliega en la pantalla un recuadro en el cual se encuentran todos los módulos del sistema, excepto el módulo menu, por supuesto. De este recuadro se selecciona aquel con el cual se desee trabajar.

La figura 4.1. muestra el menu principal del sistema de cómputo.

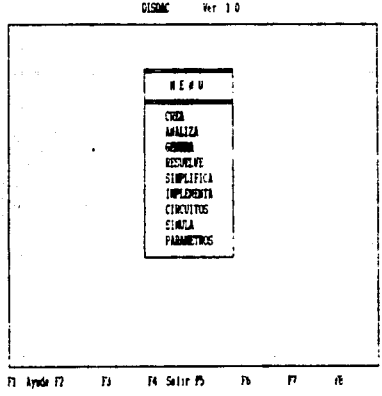


Figura 4.1. Menu principal del sistema de computo (DISMAC)

Es importante mencionar que cada módulo tiene teclas de funciones especificas según sus posibilidades. Por tanto cada uno de ellos carga sus funciones colocandolas en uno de los números de teclas predefinidas (F1 a F8).

4.2. MODULO CREA.

* Este módulo es un editor de linea, con el cual se puede escribir, la descripción del diagrama de estado o carta ASM.

Esta parte del programa primero llena de espacios la pantalla, para después colocarnos en el renglón número 5, y en la columna numero 4, esto es debido a los margenes que permite el módulo menú. Ademas de inicializar una variable llamada salir, que es de tipo booleano y permite la repetición de una secuencia de eventos programados. A continuación se muestra la pantalla formada para el editor.

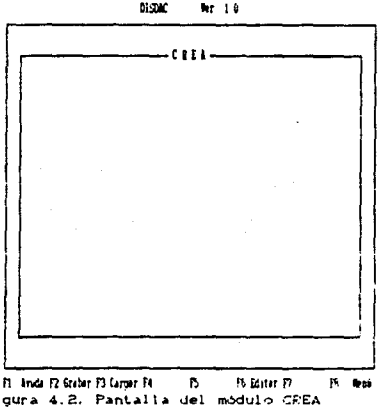


Figura 4.2. Pantalla del módulo CREA

Una vez llevada a cabo esta secuencia se ejecuta un procedimiento que coloca en la parte inferior de la ventana las teclas de funciones habilitadas para cada caso, es decir, dependiendo del modo y módulo en que se encuentre las teclas de función son distintas, las que coloca cuando estamos en el módulo CREA son: F1 Ayuda, F2 Grabar, F3 Cargar, F8 Editar y F8 Menú.

Mientras que la variable salir sea un "false", se inicia el funcionamiento en si del editor. La parte más importante la lleva una función llamada Captura, que es una entrada por teclado programada, con la cual le indicamos al editor cuales son las teclas permitidas, la longitud de línea permitida y la hoja de ayuda que se desplegará cuando F1 sea presionada. Después se tiene una toma de decisión controlada por un CASE para tomar el retorno, es decir, para saber que función debe realizar el editor de acuerdo a las teclas de control determinadas:

Si se presiona la tecla subir (↑), si esta colocado el cursor en el renglón 3 se emite un "bip", si el renglón es diferente de 3 decrementa el renglón.

Si se presiona la tecla bajar (↓) o enter (↵), si esta el cursor colocado en el renglón maxarch, que es el último del archivo, se emitirá un bip, en caso contrario se incrementará en uno el renglón.

Si es presionada la tecla "proxima pagina", se da un valor a despl = reng + pridat, esto significa que el renglón más el primer dato de la pantalla es igual al despliegue, después se lleva a cabo el procedimiento presenta de pridat + maxpant, lo cual hace que pridat sea igual a pridat + maxpant, esto es debido a que en el procedimiento Presenta(linea) la variable pridat es igualada a linea, y ahora reng = pridat + despl. Con lo cual se avanza una pantalla completa.

Si es presionada la tecla "pagina anterior", se da un valor a despl = reng - pridat, esto significa que el renglón menos el primer dato de la pantalla es igual al despliegue, después se lleva a cabo el procedimiento presenta de pridat - maxpant, lo cual hace que pridat sea igual a pridat - maxpant, esto es debido a que en el procedimiento Presenta(linea) la variable pridat es igualada a linea, y ahora reng = pridat + despl. Con lo cual se

retrocede una pantalla completa.

Si se presiona la tecla "inicio" de inmediato será presentado el primer renglón del archivo.

Si se presiona la tecla "final" de inmediato será presentado el último renglón del archivo.

Si es presionada la tecla escape, de inmediato se leera lavariable salir como verdadera (true), y por tanto se abortará de este módulo.

Si la tecla F2 que se refiere a grabar, se lleva a cabo el procedimiento que permite almacenar la información en un archivo cuya extensión será por descontado .CRE.

Si es presionada la tecla F3 entonces será llamado el procedimiento que carga la información de un archivo *.CRE y es desplegado en la pantalla.

Si es la tecla F6, entonces se llama a un procedimiento que permite la edición del renglón en el cual se encuentre en el momento de ser presionada esta tecla. Para poder salir de este modo de edición se debe presionar nuevamente F6.

Si es presionada la tecla "insertar" se inserta un renglón entre líneas, es decir, en el renglón en el que se encuentra el cursor se introduce un renglón en blanco, y el resto de la información es recorrida hacia abajo.

Si se presiona la tecla "borrar", entonces el renglón en el que se encuentre el cursor será borrado, y el resto de la información se recorrerá hacia arriba.

4.3. MODULO ANALIZA.

El algoritmo del módulo que ANALIZA se describe a continuación tratando de cubrir todos los aspectos involucrados.

El principal objetivo de este módulo es crear un archivo *.ANA, que garantiza el hecho de que el diagrama de estados descrito en el archivo *.CRE no contenga algún error sintáctico o semántico. Para esto se toman en cuenta varias reglas para describir el diagrama de estados o la carta ASM. Tales reglas son fácilmente visibles en la tabla 4.2.2.

A continuación se muestra la pantalla principal de éste

módulo.

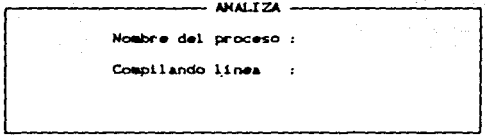


Figura 4.3. Pantalla principal del módulo ANALIZA

Primero describiremos todo el módulo a manera de diagrama de bloques, y después se detallará cada uno de los mismos.

El primer bloque limpia las listas, es decir, las listas de etiquetas, salidas y condiciones se llenan con espacios, para de esta manera preparar la entrada a las palabras que sean clasificadas y destinadas a estas listas.

El siguiente paso que realiza el programa es asignar a la variable "previo" el valor "NULO". De esta manera el primer valor de esta variable será "NULO" cada vez que inicie un análisis. El análisis del diagrama de estados descrito en el archivo *.CRE se basa en dos variables, "previo" y "actual", más adelante se describirá una tabla de sintaxis que muestra las combinaciones para una mayor comprensión del funcionamineto del programa.

Limpia_Estructura es un procedimiento que filtra el archivo *.CRE, es decir, elimina los renglones en los cuales no se haya escrito algo y elimina los comentarios (aquellos escritos entre corchetes { }), de tal manera que sólo sea analizada la parte coherente del diagrama de estados.

Se asigna el valor de la función "TOKEN" a la variable "palabra". La función "Token" es de tipo string, y obtiene una cadena de caracteres delimitada por un separador (espacio, coma, punto y coma, cambio de linea, ^ y }).

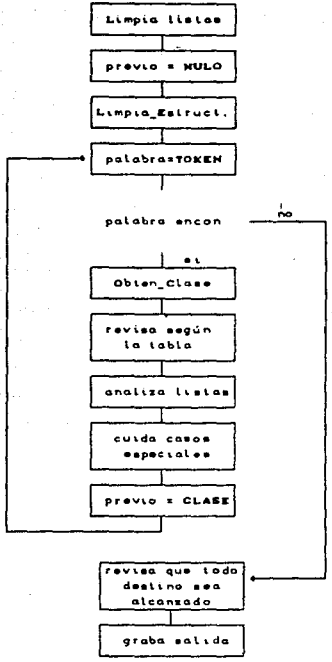


Figura 4.4. Diagrama de bloques del algoritmo del módulo ANALIZA

Una vez asignado el valor de la función "TOKEN", se realiza el procedimiento Obten_Clase, el cual cubre el objetivo de clasificar a la palabra de entrada. La forma de obtener esta clasificación es de acuerdo a comparaciones sucesivas con las palabras reservadas, hasta localizarla. Se cuenta con una etapa adicional, la cual es recurrida cuando la clasificación de la

palabra de acuerdo al criterio anterior no es lograda. En esta etapa se busca la palabra en cada una de las listas de "etiquetas", "salidas" y "condiciones", suspendiendo la busqueda en caso de encontrar la palabra en una de estas listas, en tal caso la clase de la palabra es el nombre de la lista. Si la busqueda termina sin éxito, entonces se asigna la clase IDENT (identificador) a la palabra, la cual será considerada para actualizar el grupo de listas según los siguientes bloques:

SI LA PALABRA ES	ASIGNA EL VALOR
#	NADA
SI	SI
BRINCA	BRINCA
ESTADO	ESTADO
IMPOSIBLES	IMPOSIBLE
:	PUyCO
,	COMA
^	TAMBIEN
)	NO
NULO	NULO
<identificador>	IDENT
en lista de etiquetas	ETIQUETA
en lista de salidas	SALIDA
en lista de condiciones	CONDICION

Tabla 4.3.1. Asignación de CLASE a PALABRAS determinadas.

Como se puede observar en el diagrama de bloques mostrado en el inicio de esta descripción del funcionamiento del algoritmo del módulo ANALIZA, el siguiente bloque indica que se lleva a cabo una revisión de acuerdo a la tabla de sintaxis antes mencionada. Esta tabla lleva a cabo un análisis de acuerdo al "Token" actual y al previo. La tabla contiene todas las posibles formas de encontrar la sucesión (previo-actual), e indica, para cada caso, si la pareja es válida o no. Por ejemplo, si el diagrama de estados a analizar es el siguiente:

```
PISO0   ESTADO;
   SI  B2  BRINCA  BAJA;
   SI  B1  BRINCA  BAJA;
BAJA  ESTADO  BAJAR;
  SI  B2'B2  BRINCA  PISO2;
  SI  B1        BRINCA  PISO1;PISO2  ESTADO;
  SI  B0'B1  BRINCA  SUBE;
  SI  B1  BRINCA  BAJA;
SUBE  ESTADO  SUBIR;
```

```
  SI  SI'SI  SBINCA  PISO3;
  SI  SS     SBINCA  PISO5;
PISO4   ESTADO;
  SI  SS  SBINCA  SUBE;
  SI SS SBINCA SUBE;
```

Digamos que se desea iniciar el análisis desde el primer renglón, recordando que la variable "previo" fue inicializada con el valor "NULO", y la variable "actual" está apuntando a la palabra "PISO3", en este momento debe verificar si su longitud no excede a 10 caracteres los cuales deben estar comprendidos entre la A y la Z y entre el 0 y el 9. Una vez verificado se le asigna la clase IDENT, según lo expuesto. Observando la tabla 4.3.2. en la cual se puede apreciar con facilidad lo siguiente: si el "previo" es "NULO" entonces el "actual" válido sólo puede ser IDENT, ETIQUETA o IMPOSIBLE. Cuando la clase resulta ser IDENT, se revisa la palabra previa y de acuerdo a esta, la palabra actual es introducida a la lista correspondiente, ya sea de etiquetas, salidas o condiciones, en este caso, debido a que el previo es "NULO", es introducida a la lista de etiquetas.

De la misma manera se llena la lista de SALIDAS y CONDICIONES, es decir, de acuerdo a la "palabra" "actual" y a la "previa"; por ejemplo para que una palabra sea considerada como SALIDA, primero debe ser un identificador válido, una vez cumplido esto, el previo debe ser ESTADO o COMA. Del diagrama de estados descrito anteriormente podemos observar en el tercer renglón que después de la palabra ESTADO hay una SALIDA, cuyo nombre es BAJAR:

```
PISO8   ESTADO;
   SI  SI  SBINCA  BAJA;
   SI  SI  SBINCA  BAJA;
BAJA  ESTADO  BAJAR;
  SI  SI'SI  SBINCA  PISO3;
  SI  SI        SBINCA  PISO1;PISO2  ESTADO;
```

En cuanto a la lista de condiciones de igual manera, se verifica que sea un identificador válido y si el "previo" es "SI" "TAMBIEN" o "NO" entonces es enviado a la lista de condiciones, cuando es "COMA" es introducido a la lista de condiciones locales, ya que las condiciones, seguidas de una coma sólo pueden darse en la instrucción "IMPOSIBLES". Por ejemplo, en el diagrama de

estados descrito en los ejemplos anteriores tenemos que en la quinta y hasta la octava líneas se presentan condiciones:

```
PISO8   ESTADO;
   SI  S1  BRINCA  BAJA;
   SI  S1  BRINCA  BAJA;
BAJA  ESTADO  BAJAR;
  SI  S1'S2  BRINCA  PISO2;
  SI  S1        BRINCA  PISO1;PISO2  ESTADO;
  SI  S2'S1  BRINCA  SUBE;
  SI S1 BRINCA BAJA;
```

A C T U A L

P R E V I O	SI	BRINCO	ESTADO	PUyCO	COMA	TAMBIEN	NO	IDENT	ETIQUETA	SALIDA	CONDICION	NULO	IMPOSIBLE
SI	X	X	X	X	X	X	V	V	X	X*	V	*	X
BRINCO	X	X	X	X	X	X	X	V	V	X	X	*	X
ESTADO	X	X	X	V	X	X	X	V	X	V	X	*	X
PUyCO	V	V	X	X	X	X	X	V	V	X	X	*	V
COMA	X	X	X	X	X	X	X	V	X	V	V	*	X
TAMBIEN	X	X	X	X	X	X	V	V	X	X	V	*	X
NO	X	X	X	X	X	X	X	V	X	X	V	*	X
IDENT	X	V	V	V	V	V	X	X	X	X	X	*	X
ETIQUETA	X	X	V	V	X	X	X	X	X	X	X	*	X
SALIDA	X	X	X	V	V	X	X	X	X	X	X	*	X
CONDICION	X	V	X	V	V	V	X	X	X	X	X	*	X
NULO	X	X	X	X	X	X	X	V	V	X	X	*	V
IMPOSIBLE	X	X	X	X	X	X	X	V	X	X	V	*	X

Tabla 4.3.2. Casos válidos de sucesión de clases.

La X significa que es un caso que no es válido, es decir, la consecución entre esas clases no es permitida por el algoritmo del programa. En cambio la V son consecuciones permitidas, y de acuerdo a esta tabla, es más fácil identificar cuales son las consecuciones de clases válidas. El asterisco significa que no se presenta la revisión para estos casos (el actual nunca es nulo).

Después de un "SI" hay una condición, pero se pueden presentar dos condiciones para poder realizar un brinco por eso después de un "TAMBIEN" (TAMBIEN = ^), puede ir otra condición, como es el caso del ejemplo. En el sexto renglón sólo se presenta una condición que es SI. En la septima linea se presenta que hay un identificador cuyo previo es un "NO", esto es cuando una condición es negada, hablando en niveles lógicos es un cero.

En el siguiente bloque, de acuerdo al diagrama de bloques de la figura 4.4, actualiza listas, esto en realidad se lleva a cabo a la vez que la revisión según la tabla. Conforme se localiza una ETIQUETA, SALIDA o CONDICION, se coloca en la ultima posición de la lista correspondiente. Además existen listas adicionales las cuales son:

etiq_usadas: se refiere a las etiquetas que ya fueron utilizadas, para identificar estados, y se llena de todas las etiquetas cuyo "previo" sea "NULO" o "PUyCO", cuando el "previo" es un "BRINCA" no son llevadas a esta lista, es decir, cuando son destinos de brincos.

sali_locales: esta es la de las salidas, y se llena cada vez que el "previo" es "IDENT" o "ETIQUETA", es decir cada nueva instrucción.

cond_locales: se llena cada vez que el "previo" sea "SI", "NO", "TAMBIEN", "IMPOSIBLE" o "COMA". Es decir, dentro del bloque que pertenece a un mismo ESTADO.

cond_reales: es utilizada si el "previo" es "SI", "NO" o "TAMBIEN". Esta lista indica las condiciones que realmente son utilizadas en la descripción del diagrama de estados. Debido a que pueden ser declaradas como imposibles al inicio de la descripción, y no ser utilizadas a lo largo de la misma.

Estas listas son muy utiles para cuidar los casos especiales que es el siguiente bloque de nuestro diagrama para describir a este módulo. En dichos casos especiales se describen las condiciones de error que no son detectadas con el procedimiento de secuencia de palabras ya descrito.

No puede haber dos estados con la misma etiqueta. Para revisar que no se presente esta situación, es necesario construir una lista de las etiquetas que se usan como nombre de un ESTADO. Antes de aceptar una ETIQUETA asociada a un ESTADO, se busca en esta lista y en caso de encontrarla es reportado el error. Para actualizar la lista (etiq_usadas) se considera que: El previo válido a una ETIQUETA es un BRINCA, PUyCO y un NULO, entonces se revisa el previo a un identificador, y si se encuentra que es un PUyCO o NULO entonces se coloca en la lista de etiquetas_usadas. En caso de que el previo sea un BRINCA no es introducida a la lista de etiquetas_usadas, sólo a la de etiquetas.
Por ejemplo, en el diagrama de estados descrito en los ejemplos anteriores, podemos observar como se introducen en la lista las etiquetas encontradas.

```
PISO3   ESTADO;
   SI  S2  BRINCA  BAJA;
   SI  S1  BRINCA  BAJA;
BAJA  ESTADO  BAJAB;
  SI  S2*S2  BRINCA  PISO2;
  SI  S1       BRINCA  PISO1;PISO2  ESTADO;
  SI  S3*!S1  BRINCA  SUBE;
  SI  S1  BRINCA  BAJA;
SUBE  ESTADO  SUBIB;
  SI  S2*S2  BRINCA  PISO2;
  SI  S3       BRINCA  PISO3;
PISO1   ESTADO;
  SI  S2  BRINCA  SUBE;
  SI  S3 BRINCA SUBE;
```

PISO3 es la primera etiqueta introducida a la lista de etiquetas y también a la lista de etiquetas usadas ya que el previo es NULO, la siguiente es BAJA, pero como la precede un BRINCA no es introducida a la lista de etiquetas usadas, como se puede observar después de la primer etiqueta llamada BAJA, se encuentra otra etiqueta con el mismo nombre, pero también precededida por un BRINCA y por tanto tampoco se introduce en la lista de etiquetas usadas. En la lista de etiquetas ya había sido introducida. Después aparece de nuevo la etiqueta BAJA pero ahora precedida por un PUyCO (,), por tanto si es introducida en la lista de etiquetas usadas.

Para un ESTADO, no pueden repetirse sus SALIDAS. Cuando en un ESTADO se presentan SALIDAS, entonces son colocadas en una lista de salidas locales, que es revisada cada vez que se introduce una nueva SALIDA, y si ésta ya se encuentra en la lista se indica el error de "salidas redundantes". Por ejemplo en el diagrama de estados utilizado a lo largo de esta descripción del funcionamiento del algoritmo del módulo ANALIZA, observamos que en la cuarta línea se presenta una salida que es BAJAR y es precedida de un ESTADO por tanto es introducida en la lista de salidas y de salidas locales. En este ejemplo no se presentan dos salidas en un mismo estado, para poder aclarar la previsión de este problema, para un estado no pueden repetirse sus salidas, digamos que tenemos la siguiente línea en la descripción de un diagrama de estados:

```
001 ESTADO ROJO,AZUL,ROJO;
```

De acuerdo con lo expuesto hasta ahora, al encontrar la palabra ROJO se sabe que es una SALIDA, y es introducida a la lista de salidas y a la de salidas_locales, la siguiente salida es AZUL, la cual es buscada en la lista de salidas locales y al no ser encontrada es introducida en ella, la siguiente salida es ROJO, y al ser encontrada en la lista de salidas_locales indica la existencia del error.

Brincos condicionales e incondicionales no pueden ir juntos. Recordando que después de la palabra ESTADO todos los brincos que se encuentren parten de él, debe cuidarse que no existan brincos condicionales e incondicionales simultaneamente. Esto se cuida de la siguiente manera. Se declara una bandera llamada brinco_edo que puede tomar los siguientes valores:

brinco_edo:	N	Ninguno	(lo pone ESTADO)
	S	Si	(lo pone SI)
	C	Condicional	(lo pone BRINCA si estaba en S)
	I	Incondicional	(lo pone BRINCA si estaba en N)

Cuando encuentra la palabra ESTADO actualiza la bandera con el valor "N", lo cual nos indica que el estado no tiene brincos declarados. Cuando se identifica la palabra SI se estudia el valor

de la bandera "brinco_edo", si es "N" ó "C", lo cual significa que no hay brincos declarados ó los declarados son condicionales, obliga a la bandera a cambiar su valor por "S". Por otro lado si el valor de la bandera es "I", entonces se despliega el mensaje de error "Brincos incondicional y condicional en el mismo estado".

Cuando encuentra la palabra BRINCA, es necesario considerar la palabra previa para deliverar si es condicional o incondicional el brinco, para el primer caso debe reportarse error cuando la bandera sea "I", lo cual manifiesta la existencia de un brinco incondicional declarado para el mismo ESTADO. Si se trata de un brinco incondicional el valor de la bandera debe ser "N" para lograr la aceptación. En cualquier caso cuando la revisón es realizada con éxito la bandera se actualiza con "C" o "I" dependiendo de la naturaleza del brinco.

En un brinco condicional no se pueden repetir condiciones. De acuerdo a la tabla 4.3.2, la clase CONDICION es dada siempre y cuando sea un identificador válido y además el previo sea SI, TAMBIEN o NO. Una vez identificado de esta manera se coloca en una lista de condiciones y una de condiciones locales. En caso de que ya se encuentre en la lista de condiciones locales, el módulo indicará el error "Variable con redundancia". Por ejemplo si la línea a analizar es:

```
SI A"JA"A BRINCA BETA
```

Coloca en la lista de condiciones y de condiciones locales a "A", después al encontrar "JA", e intentar almacenar "A", como esta condición se encuentra ya en la lista de condiciones locales, al identificarla como ya usada despliega el mensaje de error.

En los casos especiales, se cuida la situación en la que en un mismo estado se presentan dos condiciones que derivan a dos estados distintos.

Esto lo logra actualizando una lista de condiciones compuestas y una lista con los destinos de cada condición. Seguido esto, genera una lista de condiciones, buscando en la lista de condiciones compuestas y tambien de la lista de condiciones

locales, cada uno de los elementos e introduciendolo a esta lista de condiciones. Una vez completa la lista de condiciones generada, se calcula el número de renglones de la tabla de verdad de dichas condiciones. Ahora compara los destinos de cada condición, si son iguales, no existe problema alguno. En cambio si son diferentes, forma la tabla de verdad, cuyos renglones fueron calculados con anterioridad, observando primero que se cumplan los destinos de las condiciones locales y después los de las condiciones compuestas, en caso de que dos destinos se encuentren en un mismo renglón de la tabla de verdad despliega un mensaje de precaución que indica al usuario la colisión. Por ejemlo, si el estado analizado es el siguiente:

```
BAJA  ESTADO  BAJAB;
  SI  S2*B2   BRINCA   PISO2;
  SI  S1      BRINCA PISO1;PISO2 ESTADO;
```

Si se forma la tabla de veradad, ésta queda así:

S2	B2	S1	DESTINO
0	0	0	———
0	0	1	PISO1
0	1	0	———
0	1	1	PISO1
1	0	0	———
1	0	1	PISO1
1	1	0	PISO2
1	1	1	PISO1, PISO2

Como se puede observar en el octavo renglón de la tabla se presentan ambas condiciones, es decir, S2^B2 y S1, por tanto se presentan dos destinos en este renglón, PISO1 y PISO2, lo cual presenta una colisión y el programa la indica.

Para estos casos en el módulo que GENERA la tabla se asignan asteriscos en el renglón que presente este tipo de colisiones, asumiendo que no se presentan estos casos.

El siguiente bloque indica que la "clase" es cargada a la variable "previo". Lo cual indica que se cierra una malla para dar inicio al proceso de nuevo, asignando a la variable "palabra" la función "Token".

El diagrama de bloques posee una toma de decisión cuando

existe palabra encontrada continúa con el proceso anteriormente descrito, pero cuando no hay más palabras encontradas entonces, el diagrama nos lleva a un bloque que indica que se lleva a cabo una revisión de todos los destinos y esto se puede hacer gracias a las listas de etiquetas y de etiquetas usadas. Y esto nos lleva a resolver el problema de que algún brinco tiene indicada una etiqueta que no fue designada a estado alguno, esto es:
Todas las etiquetas usadas en brincos deben tener destinos. Gracias a que se llevan dos listas de etiquetas, una de las cuales es de etiquetas usadas, actualizada cuando es usada una etiqueta después de un brinco y esta no es encontrada en la lista de etiquetas usadas, significa que no hay estado con dicha etiqueta, y por tanto se presenta el error de "No se encontró estado etiquetado".

El último bloque indica que la salida es grabada, es decir, el resultado del análisis es grabado en un archivo *.ANA, que nos garantiza que no hay algún error en la descripción del diagrama de estados del archivo *.CRE.

La forma en la cual este módulo almacena la información es la siguiente:

```
Estados.....5
Entradas....8
Salidas.....2
Situacion...COMPLETO

IMPOSIBLES-S1-,-S3-;
IMPOSIBLES-S1-,-S2-;
IMPOSIBLES-S2-,-S3-;
PISO3-ESTADO-;
SI-B2-BRINCA-BAJA-;
SI-B1-BRINCA-BAJA-;
BAJA-ESTADO-BAJAR-;
SI-S2-^-B2-BRINCA-PISO2-;
SI-S1-BRINCA-PISO1-;
PISO2-ESTADO-;
SI-B3-^-}-B1-BRINCA-SUBE-;
SI-B1-BRINCA-BAJA-;
SUBE-ESTADO-SUBIR-;
SI-B2-^-S2-BRINCA-PISO2-;
SI-S3-BRINCA-PISO3-;
PISO1-ESTADO-;
SI-B2-BRINCA-SUBE-;
SI-B3-BRINCA-SUBE-;
```

4.4. MODULO GENERA.

El principal objetivo de este módulo, como su nombre lo indica, es generar una tabla representativa del sistema. Dicha tabla es generada de acuerdo a los datos obtenidos del archivo producido por el módulo anterior.

El primer paso que se realiza en este módulo es el de limpiar las listas de entradas, salidas y estados.

En segundo término despliega la pantalla del proceso, que indica el numero de estados, entradas y salidas. El cual es mostrado en la siguiente figura.

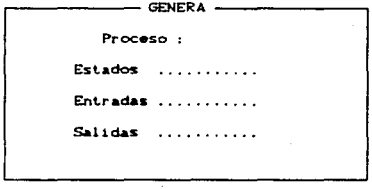


Figura 4.5. Pantalla principal del módulo genera GENERA

Después se lleva a cabo el procedimiento principal que es seccionado de acuerdo a los procedimientos que llamará de acuerdo a las necesidades del proceso.

Primero se realiza el procedimiento Carga_Proceso, es decir, carga el archivo *.ANA del cual se desea generar su tabla.

Una vez cargado el archivo *.ANA, se llenan las tablas. Primero inicializa la variable "palabra" con el valor ";", y la bandera "status" con el valor "ESPERA". Una vez hechas estas asignaciones, obtiene la clase de la primer palabra encontrada. Dicha clase es asignada de acuerdo a la siguiente tabla:

SI LA PALABRA ES	ASIGNA EL VALOR
IMPOSIBLES	IMPOSIBLE
SI	SII
BRINCA	BRINCA
ESTADO	ESTADOO
;	PUyCO
,	COMA
^	ADEMAS
]	NO

Tabla 4.4.1. Asignación de CLASE a PALABRAS determinadas.

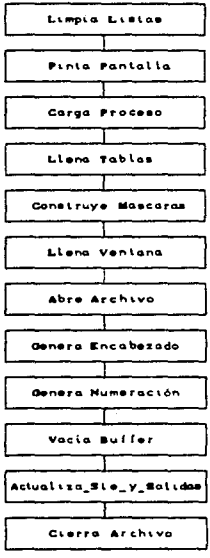


Figura 4.6. Diagrama de bloques del algoritmo del módulo GENERA

De acuerdo a el "status" actual y a la "clase" de la palabra en cuestión, se adquiere un nuevo "status" o se actualizan las listas de entradas, salidas y estados. A continuación se muestra una tabla que indica el "status" actual y la clase de la palabra siguiente, en la intersección se da la el "status" siguiente, para seguir con la actualización de listas, y la lista que es actualizada.

STATUS ACTUAL

CLASE SIGUIENTE	ESTADO	SI	BRINCO	ESPERA	IMPOS
IMPOSIBLE	——	——	——	IMPOS	——
SII	——	——	——	SI	——
BRINCA	——	BRINCO	——	BRINCO	——
ESTADOO	——	——	——	ESTADO	——
PUYCO	ESPERA	——	ESPERA	——	ESPERA
COMA	separador	——	——	——	——
ADEMAS	——	Conectivo	——	——	——
NO	——	Conectivo	——	——	——
IDENT	Act_L_Sal	Act_L_Ent	——	Act_L_Edo	——

Tabla 4.4.2. Actualización de listas de acuerdo al "status"

Una vez terminado el barrido del archivo *.ANA, se rellena de espacios hasta alcanzar la longitud de 10 caracteres, que es el máximo de caracteres permitido para nombrar a las entradas, salidas y estados; para que en el archivo *.TBL, que es el generado por este módulo, sean presentados en forma recorrida hasta la maxima derecha, que en realidad es hasta la máxima parte inferior. Debido a que son presentados en columnas.
Seguido esto ordena las listas en orden alfabético.

El bloque que indica construir máscaras, utiliza las llamadas "IMPOSIBLES" y genera una máscara por cada una para llenarlas con asteriscos. Inicializa la variable palabra con un ";", el apuntador de renglón con 7 y el apuntador de máscara lo pone en cero. Además limpia el vector de máscaras y la bandera "band" la declara falsa.

Ahora mientras exista una siguiente palabra, y esta es un punto y coma (;) la bandera sigue siendo falsa. Si la palabra es "IMPOSIBLES", entonces la bandera cambia a verdadera y se incrementa el apuntador de máscara. Si la bandera es verdadera y la palabra es diferente de una coma (,) entonces la palabra es justificada a 10. Al operador "ps" se le asigna el valor que resulte de la sustracción del resultado de la función Encuentra al número de entradas. En la función encuentra se busca en la lista de entradas la palabra, con un tope del número de entradas. Una vez calculado el primer valor de "ps" se recalcula elevando dos a la potencia marcada por sí mismo. Y en el el vector de máscaras se coloca el valor indicado por el apuntador de máscara OR el valor de "ps" (donde OR es la operación booleana). Esto es para calcular el valor de la sentencia "IMPOSIBLES", o mejor expresado el valor de la máscara en la cual cada vez que se encuentre con este valor, asignará asteriscos al estado siguiente así como a las salidas.

El siguiente procedimiento llena la ventana inicialmente presentada, es decir despliegua los datos del archivo a procesar, estos datos son: número de salidas, entradas y estados así como el nombre de éste.

A continuación se lleva a cabo el procedimiento Abre_Salida. El cual, como su nombre lo indica, abre un archivo con el mismo nombre del archivo cargado M.ANA, sólo que ahora la extensión es TBL.

Después de esto, genera el encabezado del archivo. Determina el número de bits que corresponden a un estado, calculando el logaritmo en base 2 del número de estados. El número de renglones de la tabla elevando dos, el número de bits que corresponden a estados más el número de entradas. También calcula el tamaño del

renglón sumado el número de entradas, más el de salidas más dos veces el número de bits correspondientes a estados más 14, este 14 se obtiene de la justificación a once de los primeros renglones del encabezado, y los espacios entre estados, entradas y salidas. A continuación se muestra un ejemplo del encabezado.

```
TamRen=030
BitEdo=003
NumRen=512
NumEnt=006
NumSal=002
                                    BS
                                    AU
                                    JB
                EEE BBBSSS SSS AI
                210 123123 210 RR
```

Una vez generado el encabezado del archivo, procede a generar la numeración. Lo cual es realizado de la siguiente manera.

La primer actividad que realiza es llenar la lista de asteriscos con asteriscos (*) y la lista de ceros con ceros (0). Después, para cada renglón desde cero hasta el número de renglones menos uno, define la longitud del renglón e inicializa la "mascara", por la que se debe efectuar la operación booleana AND con el número de renglón, con el valor unitario. La generación de columnas se realiza por medio de un contador de 1 hasta la suma del número bit correspondientes a los estados con el número de entradas.

El resultado de la AND entre el número del renglón y la "mascara" es escrito en forma binaria en la posición correspondiente del renglón. Dicha operación booleana es realizada de la siguiente manera: se toma el número del renglón y se realiza la AND entre este número y las potencias de dos de la máscara, hasta llegar al máximo número de que puede tomar ésta. Es decir, se lleva a cabo la operación booleana del número del renglón y la primer potencia de dos, que es uno, la segunda potencia de dos y así sucesivamente hasta llegar a la ultima potencia de dos. El resultado de la operación booleana se coloca en la posición correspondiente a la potencia de dos utilizada. Y de esta manera la representación binaria del número del renglón es realizada.

Por ejemplo si el número de bits correspondientes a los estados es tres y el número de entradas es seis, el maximo número de renglones es dos elevado a la suma de los antes mencionados (2^{3+6}) y esto es igual a 512, que es el máximo número que puede tomar la "máscara". La AND se realiza del número del renglón y cada una de las potencias de dos, primero 1, después 2, después 4, y asi sucesivamente hasta llegar a 512 que es 2^9 y el resultado de la operación booleana en esta potencia es colocado en la posición número nueve del renglón, es decir el bit más significativo.

Hasta este momento se ha llenado la parte correspondiente a el estado actual y las entradas. EL siguiente estado es simplemente copiado del estado actual, por el momento.

En cuanto a las salidas de la lista de ceros, es copiada la cadena de ceros igual al número de salidas.

Verifica el cambio de estado comparando los bits del estado escritos en el renglón con el la representación binaria del estado actual, si son diferentes y además el estado actual es menor al número de estados, entonces la representación binaria del estado actual es actualizada con la del renglón en cuestión. Seguido esto, de la lista de estados copia el nombre del estado correspondiente en la parte izquierda del renglón.

Si el estado actual es mayor que el número de estados, de la lista de asteriscos toma el número que corresponde a el estado siguiente y a las salidas y las introduce en la lista, debido a que estos estados jamás se presentan en el sistema cuyo diagrama de estados fue descrito. Una vez realizado todo este proceso se envia al archivo que fue abierto con anterioridad.

El siguiente bloque de nuestro diagrama de descripción del algoritmo, indica que el buffer es vaciado. Esto es para seguir trabajando sobre la misma tabla, sólo haciendo las modificaciónes necesarias, de acuerdo al diagrama de estados descrito en el archivo anterior. (*.CRE Y *.ANA)

En el bloque actualiza siguiente y salidas se actualiza la variable "palabra" con un punto y coma ";", el "status" con "ESPERA" y la condición actual con un espacio en blanco.

Mientras exista una palabra que analizar, obtiene la clase de esta de acuerdo a lo anteriormente visto en el bloque llena tablas, y de acuerdo al "status" y a la "clase" da un nuevo valor al "status". Y con esto poder calcular la posición a partir de las que debe hacer inserciones al archivo para actualizarlo según las salidas. Actualizar las columnas, si con el número binario correspondiente al destino, se cumple la condición.
Para una mayor comprensión de este procedimiento se elaboró la siguiente tabla.

STATUS ACTUAL

CLASE SIGUIENTE	ESTADO	SI	BRINCO	ESPERA
IMPOSIBLE	——	——	——	——
SII	——	——	——	SI
BRINCA	——	BRINCO	——	BRINCO
ESTADOO	——	——	——	ESTADO Act_Edo
PUyCO	ESPERA	——	ESPERA	——
COMA	separador	——	——	——
ADEMAS	——	——	——	——
NO	——	——	——	——
IDENT	Guar_Sali	Act_Cond	Guar_Ste	——

Tabla 4.4.3. Actualización de las columnas de estado siguiente y de salidas.

La forma en la que realiza el procedimiento Guar_Ste; que en realidad lo que hace es actualizar las columnas del estado siguiente, con el número binario correspondiente al destino solo cuando la condicion se cumpla.

Después de un BRINCA la palabra siguiente es el destino de éste, por lo tanto esta es la palabra destino. Ya conocida esta palabra la busca en la lista de estados y de acuerdo a la posición que guarda en ésta le resta la unidad, recordando que se inicia del estado cero, y la asigna a un operador llamado "ps". Una vez

realizado esto se lleva a cabo una operación booleana (AND) entre una máscara y el operador "ps" para de esta manera obtener su representación binaria.

Después Transforma cada condición simple por la máscara que activa el bit de la condicion.

Esto se lleva a cabo de la siguiente manera. Inicializa la variable que corresponde a las condiciones simples, mientras que un contador inicializado con cero, sea menor que la longitud de la condición realizara un incremento en la cuenta, seguido esto, si el caracter de la posición de la condición que indica el contador es un "]" no realiza operación alguna, si es un "^" entonces realiza un procedimiento adicional llamado Sust_Cond_Simp, que cambia una condicion simple por la mascara necesaria para hacer la evaluacion del bit que le corresponde.

Este procedimiento adicional asigna a la cadena de salida la condición a tratar y la justifica a 10 (recordando que en la lista están justificadas de esta manera), a la máscara se asigna el valor de la sustracción del resultado obtenido por la función Encuentra, al número de entradas. La función Encuentra, devuelve el valor, según la posición que guarde en la lista, y en caso de no encontrar la palabra en la lista indicada devolverá un cero. Por ejemplo si la función es declarada de la siguiente manera Encuentra(entradas,str_salida,ult_ent), busca en el vector de entradas el str_salida ul_ent veces.

El valor finalmente asignado a la máscara es dos elevado al valor anteriormente calculado para la máscara. Después borra la condición y a la cadena de salida le asigna el valor en número entero de la máscara e incrementa el número de condiciones simples.

Después evalúa la condición y sustituye. A la variable "tamano" se asigna el valor dos elevado al numero de entradas, que es el tamaño de cada partición. Para el renglón de cero a "tamano" - 1, si no existe condición alguna, significa que el brinco es incondicional y por tanto asigna el valor del estado siguiente a todo el bloque en el cual se presente el estado que obliga a tal brinco.

Mientras exista la condición, esta es monitoreada, si el primer caracter es un "!" la bandera llamada "negada" se hace verdadera. Esto es para saber que el bit que representa a la condición es activo bajo (cero lógico). Si es un "^", se asigna a la máscara el valor de la condición, y evalúa la condición, ya que esta es compuesta, y por cada vez que se cumpla la asignará el valor del estado siguiente, es decir, el destino del brinco.

A este destino le agrega un punto del lado derecho, el cual nos indica que biene de una condición. En caso de que en el mismo bloque se encuentre otra condición que nos lleve a un estado (o destino) diferente, este punto nos es de gran ayuda, ya que lo primero que se realiza es buscar si tiene el punto, si lo tiene se compara el destino y si es igual no hay problema alguno, pero en caso de que no coincidan los destinos, se asume que es una colisión entre dos condiciones y le asigna asteriscos tanto al estado siguiente como a las salidas, ya que se asume que no se presentará dicho suceso.

Ahora el procedimiento Guarda_Salida, con el número de estado calcula la posición a partir de la cual debe hacer inserciones al archivo de salida para actualizarlo.

Primero se busca la salida en la lista para saber que lugar ocupa en la misma, calcula el tamaño de la partición, elevando dos al número de entradas, y para todo el bloque (o partición) del estado en el cual es activada la salida colocado el valor lógico correspondiente.

Por último cierra el archivo, es decir, se envía la instrucción que cierre el archivo, con la extensión .TBL que es ya toda la tabla generada para el sistema descrito en el diagráma de estados del archivo *.CRE.

Este módulo permite al diseñador modificar el orden de los estados, lo cual se traduce en un cambio en la secuencia lógica del sistema digital y por lo tanto en la generación de la tabla, así como de el resto del procesode diseño.

4.5. MODULO CIRCUITOS.

Aquí se maneja el mismo editor del módulo CREA, pero se sigue un formato distinto para describir el circuito integrado, es decir, nombre, tipo (doble, simple, etc.), numero de salidas, de terminales, descripción de cada una de sus terminales, instrucciones e implementación, entre otras cosas.

Tales circuitos son importantes para poder, en base a ellos, resolver la tabla generada.

Para describir el algoritmo del módulo CIRCUITOS, nos abocaremos a la parte que revisa la coherencia en la descripción del circuito integrado, ya que la parte que se refiere al editor es exactamente la misma del algoritmo del módulo CREA, que es el editor de texto para describir el diagrama de estados del problema a resolver.

De acuerdo al diagrama de bloques mostrado en la figura 4.7. La primer actividad que realiza el programa es limpiar las listas de terminales, salidas, entradas, funciones, funciones locales (funs_loc) y códigos de operación (cops).

Una vez limpias tales listas, inicia la revisión de la descripción del circuito integrado. El primer parámetro que es revisado es el NOMBRE, checando que exista esta palabra en primer lugar, y después que exista, como definición del nombre, una cadena de caracteres que no exceda 50. Seguido esto verifica la existencia de la palabra NUMERO, aceptando después una cadena de 10 digitos, incluyendo letras. Acto seguido, observa la existencia de la palabra SALIDA, y seguida a ésta el número correspondiente. La salidas son revisadas para un sólo circuito del encapsulado, es decir si se encuentran encapsulados dos circuitos iguales en el chip se toma cuantas salidas tiene uno de ellos y no el circuito integrado en forma global. Después verifica la existencia de la palabra TIPO, y seguido a esta palabra revisa la palabra adecuada que puede ser: SIMPLE, DOBLE o TRIPLE. El tipo se refiere a cuantos circuitos iguales se encuentran en el mismo encapsulado. Después revisa la existencia de la palabra TERMINALES, y el numero

correspondiente seguido a ésta palabra.

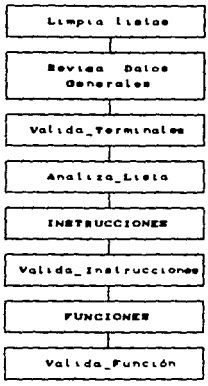


Figura 4.7. Diagrama de bloques del algoritmo del módulo CIRCUITOS

Ya revisado esto, da paso al bloque Valida_Terminales, el cual realiza dicha validación por columnas, de acuerdo fue capturada la descripción del circuito. En la primera columna revisa que se encuentre un número el cual es el de la terminal, verificando que tal número no exceda al máximo de terminales definidas anteriormente. En la segunda columna verifica el nombre de la terminal, el cual debe ser no mayor a 10 caracteres e iniciar tal cadena con letra. La tercera columna revisa la clase de la terminal, la cual puede ser: ENTRADA, SALIDA, CONTROL, VCC, TIERRA, RELOJ, FUNCION, NO_CON, MAL. La clase NO_CON significa que la terminal no requiere de conexión alguna y, MAL significa que la clase que fue asignada a la terminal no es alguna de la válidas. En la cuarta, y última, columna revisa el orden de la terminal, esto es a cual bloque pertenece, si es terminal de polarización o de reloj, el orden asignado debeá ser cero (0), las demás terminales deberán ser identificadas de acuerdo al bloque o al

número de circuito dentro del encapsulado.

Realizada esta revisión, verifica la clase de la terminal, si es SALIDA, ENTRADA o FUNCION, la coloca en la lista correspondiente; en cambio si es CONTROL incrementa el contador de las terminales de control, el cual es utilizado para determinar el número de bits del código de operación. Si es identificada la clase como MAL despliega un error que indica "Clase de terminal no reconocida".

El siguiente bloque indica que es analizada una lista, esta lista es formada por la estructura en la que fueron descritas las terminales. Primero es ordenada de acuerdo al bloque que se indicó en la descripción, después revisa la repetición de nombres de terminales en un mismo bloque, por ultimo revisa la existencia de las terminales en todos los bloques. Esto es en base a que si se tiene más de un circuito idéntico en un mismo encapsulado, debe tener el mismo número de terminales coincidiendo en la clase de las mismas.

El siguiente bloque se llama "INSTRUCCION", esto es debido a que revisa la existencia de la palabra "INSTRUCCIONES"y después un número que corresponda a las mencionadas.

El bloque "Valida_Instrucciones", el nombre indica su labor. Revisa las cuatro componentes de una instrucción. Esta validación se lleva a cabo por columnas. En la primer columna revisa el nombre de la instrucción; en la segunda, el código de operación; revisando que el número de bits corresponda al número de terminlaes de control indicadas con anterioridad, el orden de los bits es el mismo en que aparecen descritas las terminales de control. Después revisa que el bit debe ser 1, 0 ó #, a continuación checa la lista de códigos de operación, si encuentra el código en cuestión envía un mensaje de error, indicando que corresponde a otra instrucción, en caso de no encontrarlo, es introducido en esta lista. Por último valida la implementación: Primero inicializa la variable previo con el valor "NULO", después limpia la lista de variables temporales y la de salidas locales, si no hubo una signación previamente y la clase de la palabra no

es TEMPORAL (@), SALIDA o ASIGNA (<0, entonces despliega un mensaje de error que indica "se espera una variable de salida o temporal". De no hallarse error realiza un análisis de acuerdo a una tabla de sintaxis en la cual se observan las secuencias válidas de clases de palabras. En esta tabla las "X" indican secuencia no válida y las "V" secuencia válida, ver tabla 4.5.1. Cuando el actual es TEMPORAL o SALIDA, se lleva a cabo una revisión de la lista de variables temporales y de salidas locales respectivamente, en caso de no encontrar la palabra en estas listas las introduce en ellas. Cuando el actual es un punto y coma (PUyCO), compara la última salida detectada en la descripción del circuito con la última salida local, esta salida local se refiere a cada una de las instrucciones, si no son iguales se despliega un mensaje de error "La implementacion no considera todas salidas". En caso de encontrar una palabra una palabra que no sea reconocida con alguna de las clase vistas en la tabla se indicará un error de sintaxis.

ACTUAL

		TEMPORAL	ENTRADA	SALIDA	NUMERO	ASIGNA	OP_BIN	NO	COMA	PUYCO
	TEMPORAL	X	X	X	X	V	V	X	V	V
	ENTRADA	X	X	X	X	V	V	X	V	V
P	SALIDA	X	X	X	X	V	V	X	V	V
R	NUMERO	X	X	X	X	X	V	X	V	V
E	ASIGNA	V	V	V	V	X	X	V	X	X
V	OP_BIN	V	V	V	V	X	X	V	X	X
I	NO	V	V	V	V	X	X	X	X	X
O	COMA	V	V	V	V	X	X	X	X	X
	PUYCO	V	V	V	V	X	X	X	X	X
	NULO	V	V	V	X	X	X	X	X	X

Tabla 4.5.1 Análisis sintáctico del bloque Valida_Instrucciones

El siguiente bloque verifica la existencia de la palabra "FUNCIONES", y seguida a esta un número que determine a las mismas.

En cuanto al bloque Valida_Función, primero inicializa la variable previo, la cual se refiere a la clase de la palabra anterior a la que se analiza en el momento, con el valor o la clase "NULO", después inicializa la variable actual con la clase de la primera palabra encontrada. Después revisa si el hubo un ASIGNA antes y si el actual no es FUNCION o ASIGNA, entonces hay un error "Antes de asignación debe ir un nombre de función". En caso contrario se lleva a cabo un análisis por medio de una tabla de sintaxis, en la cual se observa que las "X" indican que la secuencia entre esas clases de palabras no es válida, en cambio la "V" indica que si es permitida la secuencia entre esas clases de palabras. Dicha tabla es mostrada a continuación:

		ACTUAL					
		FUNCION	SALIDA	OP_BIN	NO	ASIGNA	FUYCO
P	FUNCION	X	X	X	X	V	X
R	SALIDA	X	X	V	X	X	V
E	OP_BIN	X	V	X	V	X	X
V	NO	X	V	V	X	X	X
I	ASIGNA	X	V	X	V	X	X
O	FUYCO	X	V	V	V	X	X
	NULO	V	V	V	V	X	X

Tabla 4.5.2. Sintaxis para Valida_Funciones.

Si la primer palabra es clasificada como FUNCION, entonces revisa la lista de funciones, si es encontrada indica la existencia de error, en caso contrario, la coloca en dicha lista.

4.6. MODULO RESUELVE

Para resolver la tabla generada, es necesario tomar un circuito integrado, debido a que es tomado el estado actual de la tabla, para que por medio de alguna instrucción del circuito elegido, sea obtenido el estado siguiente.

Así que el resultado de este módulo es la tabla pero ahora en la columna de estado siguiente, se coloca el estado obtenido por medio de alguna de las instrucciones del circuito integrado (secuencial), así como por supuesto el nombre de dicha instrucción y su código de operación.

El primer bloque, llamado pantalla, lo que realiza es desplegar en el monitor el formato principal, en el cual se especifican el nombre de la tabla que se desea resolver, el número de renglones que contiene dicha tabla, el número de bits utilizados por los estados, y lo referente a los circuitos con los cuales se ha decidido resolver la tabla, es decir, el nombre, número de bits, número de funciones y función que se resuelve con los diferentes circuitos, así como el número de renglón que se esta resolviendo.
Dicha pantalla es mostrada en la figura 4.9.

El diagrama de bloques es el siguiente:

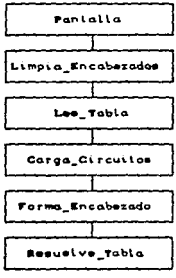


Figura 4.8. Diagrama de bloques del algoritmo del módulo RESUELVE

El siguiente bloque limpia los encabezados, es decir, limpia la lista de encabezados, dichos encabezados son los nombres de las terminales de control de los circuitos, las cuales forman los códigos de operación, así como los de sus entradas.

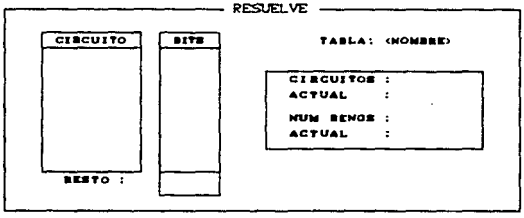


Figura 4.9. Formato principal del módulo RESUELVE

El siguiente bloque indica la lectura de la tabla, es decir, pide el nombre de la tabla a resolver, revisa el nombre que fue capturado y si no encuentra el archivo despliega el respectivo error. En caso de existir el archivo, lo abre para tomar la lecturas del número de los bits de estado, el número de entradas y el de salidas. Así como el número de renglones de la tabla y lo despliega en el respectivo renglón del formato principal.

El siguiente bloque especifica la carga de los circuitos, es decir, la lectura de tantos circuitos como sean necesarios para lograr la implementación del circuito secuencial que representa el diagrama de estados descrito. Además de actualizar una lista que contiene el nombre del circuito, el número de bits de salida y el número de bits del código de operación en cada renglón. Esta lista es llamada *vec_clos*.

Esta función del programa verifica la existencia del circuito propuesto para resolver la tabla, en caso de no encontrarse en el catálogo de circuitos indica el respectivo error. Por el contrario, si es encontrado, abre el archivo del circuito para extraer la información necesaria para resolver la tabla.

En el lugar que corresponde al nombre, en la lista *vec_ctos*, se coloca el nombre del circuito capturado. Seguido esto abre el archivo del circuito para tomar del número de salidas del mismo y cloca la información en la columna correspondiente de la lista. Y en el renglón llamado resto del formato principal coloca la diferencia entre los bits de estado del sistema (descrito en el diagrama de estados) y los bits de salida del circuito. Para de esta manera saber si solicita otro circuito o da por culminada la misma.

Para conocer el número de bits del código de operación. Primero carga el circuito en memoria, limpiando la estructura, es decir, borrando los renglones en blanco, y eliminando los comentarios. Después extrae el tipo de circuito (simple, doble, etc.), el número de terminales y vacía las listas de entradas, salidas y controles, esto es en cuanto a las terminales del circuito.

Una vez inicializadas dichas listas, hace un barrido por todas la terminales del circuito, para de esta manera ocupar las listas con los datos correspondientes. Cada vez que encuentre una entrada, salida o control será introducido el nombre de la terminal en la lista.

Ahora extrea el número de instrucciones que posee el circuito. El siguiente paso es llenar la lista de instrucciones. Para realizar esto, barre la parte de las instrucciones del circuito, tomando nombre, y código de operación. Así como cada una de las operaciones booleanas que son asignadas a cada entrada.

El número de bits del código es tomado de la suma de las terminales de control y de las terminales de entrada del circuito. Tomando los nombres de cada una de estas terminales, y colocandolos en la lista de encabezados.

El renglón correspondiente a número de circuitos es ocupado, y de acuerdo a cuantos circuitos sean utilizados para implementar el sistema, serán colocados en dicho renglón.

El siguiente bloque abre el archivo de salida, es decir, el archivo con el mismo nombre de la tabla pero con la extensión .RES. Forma el encabezado, toma el tamaño del renglón y lo suma a

los bits del código de operación de cada circuito para de esta manera obtener la longitud del renglón de salida. Lee los bits de estado, el número de entradas, de salidas y de circuitos, así como el nombre y los bits de código de operación de cada uno de estos, escribiendolo en el archivo de salida.

```
TamRen=045
BitEdo=003
NumEnt=006
NumSal=002
NuCtos=001
  CON74181
  BitsCO=008
```

```
                                        H
                                      R A
                                      E B
                              BS      S _
                              AU      E P
                              JB      TCAC
          EEE BBBSSS SSS AI           _EREPPPP
          210 123123 210 RR           MPAT3210
```

Figura 4.10. Encabezado y subencabezado del elevador

Acto seguido, forma el subencabezado, este se refiere al nombre de los estados (actual y siguiente), entradas, salidas, así como de las terminales de control y de entrada del circuito o circuitos propuestos para resolver la tabla. En la figura 4.10. se muestra el encabezado y subencabezado del elevador que se ha estado manejando a lo largo de la descripción de los algoritmos de cada módulo.

Además genera los renglones de la tabla, esto lo realiza leyendo del archivo *.TBL cada renglón de la tabla y aumentando los espacios necesarios para introducir los bits del código de operación de cada circuito.

El siguiente bloque es el de mayor importancia en este módulo, ya que como su nombre lo indica resuelve la tabla. Abre el archivo de salida, calcula el número de columna a partir de la cual recibirá adiciones la tabla generada anteriormente. Indicando

en la ubicación correspondiente del formato principal el número de circuito para el que está resolviendo la tabla.

A partir de este momento el procedimiento se encuadra a un ciclo repetitivo para todos y cada uno de los renglones de la tabla.

Primero carga las características del circuito, toma el circuito a resolver, quita comentarios y renglones vacios, y por último configura los datos del circuito. Esta configuración la efectua colocando en una lista dinámica, para cada instrucción, el código de operación y la dirección de otra lista dinámica en la que se encuantra cada operación booleana asignada a cada una de sus salidas.

Calcula los bits de interes, es decir, de los bits de estado cual de ellos será resuelto. Esto se realiza de la siguiente manera, se leen los bits de estado y se restan los acumulados, los bits acumulados son aquellos para los cuales ya fue resuelta la tabla. Por obvias razones para el primer bit a resolver, los bits acumulados son cero. Si la diferencia mencionada es mayor o igual al número de bits de salida del circuito, los bits de interes son calculados elevando dos a la número de bits del circuito menos 1 (bits de interes = 2^{bits_cto} - 1). En caso de que la diferencia sea menor al número de bits de salida del circuito, los bits de interes serán calculados elevando dos a la diferencia menos uno (bits de interes = $2^{bits\_edo-bits\_acum}$ - 1).

Inicializa la lista de valores, esta lista contiene, las salidas con su respectivo valor binario, una vez ocupadas todas las salidas, en los siguientes lugares de la lista se ubican las entradas también con sus respectivos valores binarios. Aunque en esta parte del proceso como su nombre lo indica son inicializados y el valor es cero.

Seguido esto entra en un ciclo llamado de solución de renglones, primero indica en el lugar correspondiente del formato principal el número del renglón que se encuentra resolviendo y apaga la bandera resuelto, es decir, le da el valor de falso, o cero lógico. Indicando, de esta manera, que tal renglón no se ha resuelto.

Ahora, del archivo lee el renglón en cuestión, y da inicio al

cálculo del estado actual. Dicho cálculo lo realiza copiando la cadena respectiva a los bits de estado, en esta cadena se eliminan los bits que no sean cubiertos por el circuito propuesto, tanto del lado izquierdo como del derecho de la cadena leída. Y, una vez aislada la cadena representativa del circuito es obteniendo el número binario correspondiente a la cadena leída del archivo.

El cálculo del estado siguiente, se realiza de manera similar, copiando la cadena correspondiente a los bits del estado siguiente, eliminando los no considerados por el circuito, tanto a la derecha como a la izquierda de la cadena. Convirtiendo los asteriscos en ceros lógicos para todos los casos. Y se obtiene el número binario de la cadena representativa.

Ahora entra en un "loop" llamado de instrucciones, en el cual permanece siempre que exista una instrucción para resolver y el renglón no este resuelto.

En forma anidada se encuentra otro ciclo, el cual prueba con distintas entradas. En primer lugar los valores de la lista son puestos en false, es decir, en cero lógico. Después los valores se llenan de acuerdo al valor binario del estado, tanto en las salidas como en las entradas.

Ahora Ejecuta la instrucción y para ello requiere de varios pasos.

De la estructura del algoritmo de implementación de las intrucciones, se obtiene el destino, que es el caracter encontrado a la izquierda de la asignación "<". En caso de no ser una salida, lo único que queda por alternativa es una "@", y como esta no se encuentra en la lista de valores, es introducida en ella.

Lo que realiza ahora es evaluar el modelo de la implementación, entendiendo por modelo a cada variables de la operación booleana, separada por el símbolo de asignación "<", "and (*)" u "or (+)". Dentro de la evaluación del modelo se realiza una sustitución de las variables por su valor lógico. Dicha sustitución la efectua comparando de la lista de valores, el campo del nombre con la variable, si es igual, entonces se cambia el modelo por el valor del campo correspondiente en la lista.

Una vez conocido el valor de todas las variables, se efectuan las operaciones booleanas indicadas, "and" y "or". Y el resultado es sometido a una operación "and" con la máscara del circuito para inhibir los bits que no interesen del resultado, posteriormente con la máscara de asteiscos para de esta manera obtener el estado siguiente.

Lo que resta por hacer es tomar el código de operación de la instrucción que permitó llegar al estado siguiente dado el actual.

Ahora se toma el nombre de la intrucción se justifica a 10, tomando el código de operación, incluyendo las salidas en caso de que existan en el circuito propuesto, y se colocan en el archivo de salida. Repitiendo el proceso para todos los renglones de la tabla. Si se da el caso de no encontrar alguna instrucción del circuito que lleve del estado actual al siguiente se dará la indicación, recomendando que sea utilizado otro circuito.

Cuando sean cubiertos todos los renglones de la tabla se cierra el archivo de salida y se indica la culminación del proceso por medio de una ventana.

Es importante mencionar que los cálculos de los estados siguientes, es realizado para todas las instrucciones del circuito o los circuitos integrados propuesto, si hay más de uno lo indica, de esta manera se puede implementar el que implique un menor número de compuertas.

A continuación se muestra la tabla resuelta para un contador binario en BCD con tre circuitos integrados secuenciales flip-flop tipo "D".

```
TamRen=068
BitEdo=004
NumEnt=000
NumSal=004
NuClos=004
  FF7474
  BitsCO=003
  FF7474
  BitsCO=003
  FF7474
  BitsCO=003
```

FF7474
BitsCO=003

(continua en la próxima pagina)

	EEEE	SSSS	HHHH		C P L O RDN		C P L O RDN		C P L O RDN		C P L O RDN
	3210	3210	0123								
A	0000	0001	.0000	PON	111	NOP	1=1	NOP	1=1	NOP	1=1
B	0001	0010	.0001	QUITA	0=1	PON	111	NOP	1=1	NOP	1=1
C	0010	0011	.0010	PON	111	NOP	1=1	NOP	1=1	NOP	1=1
D	0011	0100	.0011	QUITA	0=1	QUITA	0=1	PON	111	NOP	1=1
E	0100	0101	.0100	PON	111	NOP	1=1	NOP	1=1	NOP	1=1
F	0101	0110	.0101	QUITA	0=1	PON	111	NOP	1=1	NOP	1=1
G	0110	0111	.0110	PON	111	NOP	1=1	NOP	1=1	NOP	1=1
I	0111	1000	.0111	QUITA	0=1	QUITA	0=1	QUITA	0=1	PON	111
J	1000	1001	.1000	PON	111	NOP	1=1	NOP	1=1	NOP	1=1
K	1001	0000	.1001	QUITA	0=1	NOP	1=1	NOP	1=1	QUITA	0=1

4.7. MODULO SIMPLIFICA

La simplificación de la tabla resuelta obtenida en el módulo anterior, se realiza en esta parte del sistema de cómputo. Para cada salida del sistema digital y cada terminal de control del circuito integrado secuencial, es asignada una función booleana que involucra a las salidas del circuito integrado (estado actual), y a las entradas del sistema digital.

El primer bloque despliega una ventana en el monitor, que indica que se llevará a cabo la simplificación de la tabla que fue resuelta en el módulo anterior, solicitando el nombre de dicha tabla o proceso, e indicando entradas, salidas, circuitos, renglones, funciones a simplificar, y función que esta simplificando así como los pasos del proceso que esta realizando, dicho formato de la pantalla se muestra en la figura 4.11. Las funciones a las que se hace mención son las que se pueden implementar con los circuitos indicados en módulos anteriores.

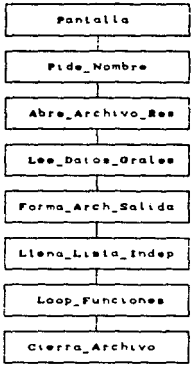


Figura 5.11. Diagrama de bloques del módulo SIMPLIFICA

Como segundo bloque se tiene que pide el nombre del proceso, esto se refiere a la tabla, ya resuelta, que se ha de simplificar. Este nombre es solicitado como entrada al módulo en el renglón correspondiente de acuerdo a la figura 5.12.

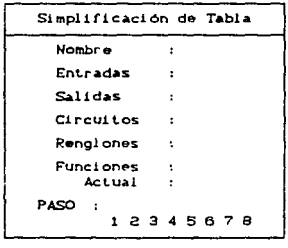


Figura 4.12. Pantalla que muestra el módulo SIMPLIFICA

Una vez introducido el nombre del proceso, o tabla a simplificar, Abre el archivo cuyo nombre es el indicado y la extensión es .RES, es decir, la tabla resuelta que corresponde al diagrama de estados inicial, o sistema a resolver.

El siguiente bloque indica la lactura de los datos generales, dichos datos se encuentran en el encabezado del archivo que genera el módulo anterior, es decir el *.RES y son: numero de entradas, de salidas, de circuitos, de renglones, número de bits utilizados por el código de operación del circuito ó circuitos indicados en módulos anteriores y el número de bits utilizados para los estados.

Abrir el archivo de salida es el siguiente bloque indicado en el algoritmo. Aquí se genera el archivo de salida con la extensión .SIM, y graba el número de circuitos, de entradas, de salidas, así como el nombre de cada uno de los mencionados y el numero de bits utilizados para los estados, así como el de los del código de operación.

El siguiente bloque indica que se llena una lista independiente, dicha lista contiene los nombres de las variables independientes, siguiendo la regla de anteponer al nombre una "e" para los bits de estado y una "c" para las condiciones. Cuando se habla de condiciones se debe comprender que son las entradas al sistema. Por variables independientes se entiende que son estados y condiciones, información que toma del encabezado del archivo.

El siguiente bloque es Loop_Funciones, el cual esta dividido en ocho pasos que a su vez se dividen en varios bloques, a continuación se describen los pasos y posteriormente cada uno de estos.

Antes de entrar a los ocho pasos de este bloque principal del módulo, se realiza lo siguiente. Calcula el numero de funciones a simplificar, esto lo realiza sumando el numero de salidas a el de los bits del código de operación de los circuitos indicados en módulos anteriores. Una vez calculado lo escribe en la pantalla

desplegada que se indicó en el primer bloque del algoritmo, y en el archivo abierto con la extensión .SIM. Después actualiza la variable que se refiere al circuito actual, es decir, al que corresponden los bits que se esten simplificando, y la columna en que se encuentra la función que es simplificada.

Seguido esto, para cada función simplifica por medio de los ocho pasos ya mencionados, escribiendo en la pantalla, en el renglón correspondiente a actual (la función que se simplifica en ese momento), el número de tal función.

Antes de entrar a la descripción del funcionamiento de los ocho pasos, es pertinente hacer notar que el algoritmo es basado en el método del tabulado o mejor conocido como Quine-McCluskey, ya descrito en el capitulo dos.

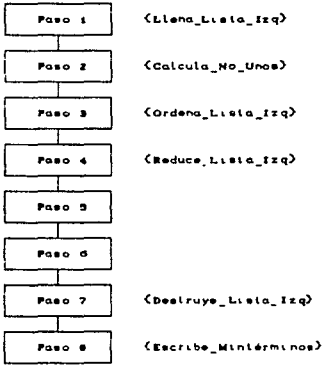


Figura 4.13. Pasos del bloque Loop_Funciones

En el primer paso se realiza el procedimiento Llena_Lista_Izq. Este procedimiento lee del archivo de entrada (*.PES), una función lógica y genera un renglón dinámico por cada

mintérmino correspondiente a uno o asterisco. Dicho renglón dinámico contiene los siguientes campos: Mintérmino, es el número de mintérmino de la función. Status, es de tipo booleano e indica si es seleccionado o no. Guiones, bits no considerados. Número de unos en el mintérmino. Siguiente, es la dirección de memoria del siguiente renglón dinámico. Y por último la dirrección de memoria del renglón anterior. Recordando que es conocida la dirección de memoria del renglón. En este paso unicamente se ocupa el campo de mintérminos.

minterm	status	guiones	no_unos	ste	ant

Figura 4.14. Campos del renglón dinámico generado.

El segundo paso calcula el número de unos existente en el campo mintérminos de cada renglón de la lista ligada, guardando el resultado en el campo de número de unos.

El tercer paso ordena la lista ligada de acuerdo al número de uos calculado en cada mintérmino, esta ordenación se realiza de menor a mayor.

Para una mejor comprensión de estos tres primeros pasos se muestra el siguiente ejemplo.

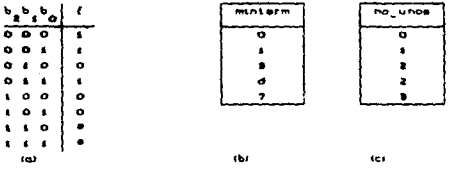


Figura 4.15. (a) Tabla de verdad, (b) Tabla de mintérminos, (c) Número de unos de los mintérminos

Como se puede observar de la función de la tabla de verdad, se obtiene una lista con los mintérminos correspondientes a uno y asterisco, y se calcula el número de unos de cada mintérmino.

Seguido esto se ordena la lista ligada segun el campo de número de unos, de menor a mayor.

En el cuarto paso como se puede observar en la figura 4.13. se realiza el procedimiento Reduce_Lista, el cual hace traslados sucesivos de la tabla izquierda a la derecha y viceversa hasta llegar a la maxima agrupacion posible, en el trayecto guarda los mintérminos aislados, primeros implicados de la lista fuente a la lista destino.

La tabla derecha es la que se generó en los tres pasos anteriores, es decir, una lista con los mintérminos cuyo valor de la función es uno o asterisco, ordenada de acuerdo al número de unos. Ahora con esta lista es dividida en bloques, de acuerdo la cantidad de unos que contenga el número binario, es decir, todos los que tengan cero forman el primer bloque, los de uno el segundo y asi sucesivamente. Si se observa el ejemplo, el cero forma el primer bloque, el uno el segundo, el tres y el seis el tercero, y el cuarto es formado por el siete.

Encuentra limites (superior e inferior) de los dos primeros bloques. Obligando al campo de estatus a un valor de falso, es decir, de no seleccionado. La selección de la que se hace mención es debido a que al comparar los miembros de un bloque con los del adyacente y existe una diferencia de un solo bit ó variable, el mintérmino es seleccionado, y si dicha diferencia es mayor no es seleccionado.

Cada pareja de mintérminos seleccionados es colocada en la lista izquierda, ordenada igualmente por número de unos y de menor a mayor.

Despues de esto entra en un "loop" de traslados a la lista contraria de datos, es decir, si la actual lista es la derecha se trasladan a la izquierda y viceversa. Dicho loop lo mantiene hasta que no exista una pareja de bloques, esto es que se quede un primer bloque sin un segundo, lo cual indica que no hay más mintérminos que comparar.

En el "loop" mencionado se realiza la comparación de los mintérminos que difieran en un bit y pertenzcan a bloques adyacentes. En caso de que exista adyacencia lógica entre el mintérmino del primer bloque y el del segundo, una función del

programa devuelve el único bit que cambia entre uno y otro. En caso de no ser adyacentes devuelve un cero. Adyacentes se refiere precisamente en que cambien sólo en un bit y ésta es la posición del guión, recordando que hay un campo en el renglón dinámico generado anteriormente.

Agrega al final de la lista contraria a la que actualmente maneja, el mintérmino encontrado. Actualizando el guión proporcionado en el procedimiento anterior. Antes de guardar en la lista revisa si este mintérmino ya existe en ella, de ser así, no se introduce. Además marca, colocando el status en verdadero lo que indica que ya fueron utilizados, los mintérminos que sí encontraron pareja, es decir, que difieren en sólo un bit de otro.

Aquellos mintérminos que no tuvieron algun adyacente, después del "loop", son aislados y trasladados a la lista de resultados y las correpondientes posiciones de los guiones.

Una vez que fueron comparados todos los elementos de los dos bloques, destruye el primer bloque de la lista ligada, esto es para ahora realizar la misma comparación entre los dos nuevos primeros bloques. Y da inicio a la misma secuencia anterior, es decir, da paso a la continuación del "loop", hasta terminar con todos los bloques. Por tanto destruye la lista en curso y libera la memoria, recordando que se ha trabajado en memoria dinámica.

El paso cinco es, a su vez, dividido en tres bloques los cuales son mostrados en la figura 4.16.

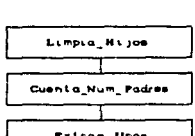


Figura 4.16. Diagrama de bloques del paso cinco

La finalidad de este paso es eliminar los primeros implicados

redundantes (método de Quine_McCluskey). Para cumplir este objetivo se utilizarán dos listas, una que guarda los primeros implicados, llamada lista de padres, esto equivale a un grupo tomado en un mapa de Karnaugh. Y otra que guarda el número de los mintérminos, equivalente a los elementos del mapa, llamada lista de hijos.

En primer lugar se debe encontrar que mintérminos se incuyen en un solo conjunto, determinando así que el conjunto es obligatorio.

El programa sigue la secuencia dada por la figura 4.16.

Limpia_Hijos, limpia los campos de hijo seleccionado que es nombrado "marca", el campo de número de padres (conjuntos que contienen al mintérmino) llamado "padres", y finalmente para cada padre encontrado guarda su nombre, de tal modo que se pueda conocer el padre obligado.

minterm	marca	padres	ult_padre	ste

Figura 4.17. Estructura del renglón de la lista de hijos

minterm	guiones	status	no_unos	ste	ant

Figura 4.18. Estructura del renglón de la lista de padres

Cuenta_No_Padres, es un ciclo de comprobación de hijos y padres. Para cada iteración se verifica si el hijo en turno pertenece al padre actual, en su caso actualiza el contador de padres, en el campo correspondiente del hijo, y el de número de hijos en el de número de unos del de padres. En el campo de ult_padre, el apuntador al último padre que seleccionó al hijo en cuestión.

Extrae_unos, barre la lista de hijos poniendo atención en aquellos con solo un padre. Al ser encontrado dicho hijo, el padre es trasladado a una nueva lista de padres indispensables. Una vez realizado el traslado el padre debe borrarse así como todos sus hijos.

Por su parte el paso seis también se divide en varios procedimientos.

Como se puede observar, hasta el paso cinco.se obtuvieron aquellos padres con sólo un hijo. En este paso se obtienen aquellos padres con el mayor número de hijos. Para ello son utilizadas las listas de padres e hijos. El primer bloque indica que actualiza los campos y después barre la lista de hijos y ocupa el campo de padres colocando el número de los primeros implicados que seleccionan al mintérmino, es decir, número de padres que seleccionan al hijo. En el campo de último padre el apuntador al último padre que seleccionó al hijo, actualizando de la lista de padres el campo número de unos con la cuenta de hijos que se encuentren, seleccionados por dicho padre.

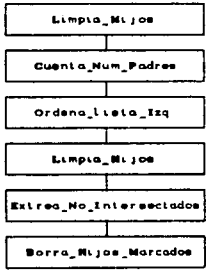


Figura 4.19. Diagrama de bloques del paso seis

El tercer bloque indica la ordenación de la lista izquierda, exactamente igual que el paso tres, es decir, por número de unos y de menor a mayor.

Seguido esto, de nuevo inicializa la lista de hijos. La primer inicialización de la lista de hijos se realizó para ordenarla. Ahora, para realizar los siguientes pasos es necesario limpiar los campos nuevamente.

El siguiente bloque indica que se realiza el procedimiento Extrae_No_Intersectados. Este procedimiento barre las dos listas, el barrido principal es en la lista de hijos, siendo anidado a esta el barrido de la de padres.

En la lista de hijos, el campo de padres indica el número del padre que lo selecciona, y de esta manera podemos saber en el padre correspondiente, en el campo de número de unos, la cantidad de hijos que selecciona.

Así se termina de barrer la lista de hijos, tomando al padre que contenga un mayor número de hijos, y trasladandolo a la lista derecha. Borrando tanto al padre como a todos los hijos en la lista izquierda. Y de nuevo realiza el barrido de la lista de hijos restantes para el siguiente padre. De esta manera se cuida de no trasladar un padre de hijos ya incluidos.

Borra_Hijos_Marcados, es el último bloque de este paso, el cual elimina de la lista de hijos todos los renglones que tengan verdadero el campo marca, es decir, los mintérminos ya utilizados.

Este paso en resumen lleva a cabo un ciclo de revisión de padres e hijos, para seleccionar los padres que posean un mayor número de hijos, eliminando los padres con hijos redundantes, y la selección de padres sea la óptima.

El paso número siete indica la destrucción de la lista izquierda, debido a que la última lista generada es del lado derecho, liberando así la memoria ocupada por la lista izquierda.

En el paso número ocho, se ejecuta el procedimiento Escribe_Mintérminos, el cual, barre la lista ligada resultante de la simplificacion y escribe cada uno de los mintérminos en un renglon del archivo de salida. Para el nombre de la función se utiliza la convencion de anteponer una "s" a los nombres de salidas, y una "oiii" donde "iii" es una cantidad de tres cifras que representa al numero de circuito, y "o" significa que la terminal forma parte del código de operación.

Para terminar la función de este módulo, sólo queda por mencionar que cierra los archivos utilizados, es decir el de

entrada, que es el de la extensión .RES, y el de salida que lleva el mismo nombre que el anterior, pero la extensión es .SIM. Este archivo *.SIM, ya contiene la simplificación del sistema o diagrama de estados descrito en el archivo *.CRE, en forma de operaciónes booleanas.

A continuación se muestra un ejemplo del archivo que entrega este módulo, para un contador binario en BCD resuelto con cuatro flip_flops tipo "D".

```
BitEdo=004
NuClos=004
  FF7474
  FF7474
  FF7474
  FF7474
NumEnt=000
NumSal=004
        H0
        H1
        H2
        H3
NuFuns=016
s           H0=eE3;
s           H1=eE2;
s           H2=eE1;
s           H3=eE0;
o001       CLR=eE3^eE2+
                eE3^eE1+
                JeE0;
o001         D=1;
o001       PON=1;
o002       CLR=eE3+
                JeE0+
                JeE1;
o002         D=1;
o002       PON=1;
o003       CLR=eE3+
                JeE0+
                JeE1+
                JeE2;
o003         D=1;
o003       PON=1;
o004       CLR=eE2+
                eE1+
                JeE0+
                JeE3;
o004         D=1;
o004       PON=1;
```

4.8. MODULO IMPLEMENTA

Para implementar las funciones booleanas calaculadas en el módulo anterior, son utilizadas las estructuras combinacionales de tres operaciones básicas del álgebra de Bool, AND, OR y NOT. Y de esta manera proporcionar al diseñador una alternativa de implementación del sistema digital propuesto.

Siguiendo con el formato para los anteriores módulos el diagrama de bloques es mostrado a continuación.

En el primer bloque se puede apreciar que indica "pantalla", en este bloque se despliega el formato principal del módulo y a continuación de da una explicación gráfica del mismo.

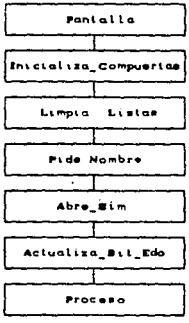


Figura 4.20. Diagrama de estados del módulo IMPLEMENTA.

El bloque Inicializa_Compuertas, como su nombre lo indica, carga los valores inicializales a las estructuras de especificaciones de cada tipo de compuertas NOT, OR y AND.

Estos valores son cargados en una estructura de la compuerta respectiva, dicho estructura es la misma para la tres compuertas lógicas usadas en este módulo la cual es mostrada en la figura 4.22.

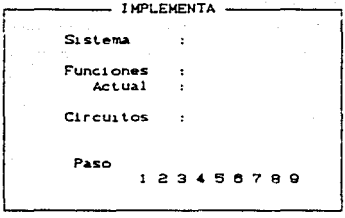


Figura 4.21. Pantalla del proceso del módulo IMPLEMENTA

Primero limpia la estructura de la compuerta, en el campo nombre coloca el número comercial de acuerdo a la tecnología TTL (7404 = NOT, 7408 = AND, 7432 = OR), en el campo número de compuertas introduce el número de compuertas por cada circuito integrado, en num_ents el valor respectivo por compuerta, habilita los bits correspondientes a las terminales de polarización 8192 para la terminal número 14, y 64 para la número 7, el campo ents es una lista que guarada en cada elemento el número de una terminal de salidas, lo mismo para el campo sals, pero con terminales de salida, por supuesto.

cto_nom	ult_cmp	ter_vcc	ter_gnd	ents	sals	num_cmp	num_ent

cto_nom	nombre del circuito
ult_cmp	ultima compuerta usada
ter_vcc	encendido el bit correspondiente a la terminal de vcc
ter_gnd	encendido el bit correspondiente a la terminal de gnd
ents	terminales de entrada
sals	terminales de salida
num_cmp	número de compuertas por cada circuito integrado
num_ent	número de entradas por cada compuerta

Figura 4.22. Campos de la estructura de compuertas

Limpia listas es el bloque siguiente, lo unico que realiza es llenar de espacios en blanco las listas de entradas, circuitos, terminales y una adicional llamada mapa. Y esto se lleva a cabo para evitar que exista "basura" en el momento de ser utilizadas en

el proceso de implementación.

La lista de entradas tiene dos campos en los cules almacena el nombre de la entrada y el nodo al cual pertenece. Un nodo es un punto de unión de varias terminales, los primeros diez estan reservados para polarización y reloj. La lista de terminales tiene tres campos el de número de circuito, de terminal y de nodo. Y por último el mapa guarda el nombre de la función (entrada, control o salida) y el número de la última terminal asignada.

En el bloque pide nombre, lo único que realiza es esperar a que el nombre del proceso, tabla simplificada, sea capturado o seleccionado del catálogo. Colocandolo en el renglón correspondiente de la pantalla principal del módulo.

Abre_Sim, se encarga de intentar abrir el archivo cuya extensión es .SIM, para extraer la información que necesita este módulo, en caso de no encontrar el archivo o sitiene algún problema para abrirlo, indicara la falla correspondiente.

El bloque Actualiza_Bit_Edo, carga el primer renglon del archivo *.SIM y extrae la cantidad de bits para los estados.

Inmediatamente después de actualizar el número de bits de los estados se realiza un procedimiento llamado carga lista de circuito, que considerando que el apuntador al archivo *.SIM se encuentra en el renglón que indica el núero de elementos de una lista, actualiza la variable proporcionada (TOPE) con dicha cantidad, además lee los renglones sucesivos para actualizar la lista cuya dirección se proporciona.

El bloque final del diagrama de bloques de la figura 4.20. lamado proceso, es dividido en nueve pasos descritos a continuación.

Paso 1. Se lleva a cabo la generación de una estructura dinámica para almacenar la implementacion de todas las funciones.

nom	imp	ste

Figura 4.23. Estructura funciones

En el campo "nom" se almacena el nombre de la función, es importante entender por función a toda aquella a la cual se le asigna su respectiva booleana incluyendo algún valor del mismo tipo. El campo "ste" es un apuntador a la siguiente estructura y el campo implementación (imp) es una estructura dinámica que contiene los siguientes campos:

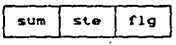


Figura 4.24. Estructura de los campos imp y ste de funciones

En el campo "sum" se almacena el sumando o función booleana asignada a su respectivo nombre. El campo siguiente (ste) es un apuntador a la siguiente estructura y por último el campo bandera "flag" que guarda un valor booleano.

Todos estos datos leidos del archivo del módulo anterior, de acuerdo al formato presentado por ese. Y una vez extraidos, cierra el archivo *.SIM.

Paso 2. Se realiza el procedimiento "Loop_Circuitos", el cual abre cada uno de los circuitos secuenciales de la lista y cada terminal la coloca en la lista correspondiente, además actualiza la lista de mapa, la cual posee el nombre y número de la terminal para cada circuito.

Primero limpia la lista en la cual se guarda la descripción del circuito secuencial, y la lista que lleva el control de distribución del circuto integrado, esta lista guarda el nombre del circuito, el número de bloques que posee y el último bloque utilizado.

Carga el circuito a memoria, la lista que guarda la descripción del circuito es ocupada. De acuerdo a los datos leidos en esta lista, se ocupa la que lleva el control (nombre, número bloques y último bloque utilizado).

Limpia la lista de salidas, esta almacena el nombre de la salida del circuito y la terminal que ocupa. Actualiza la lista ter (nombre de la terminal, numero y tipo), el tipo se refiere a

entrada, salida, control, función, vcc, gnd ó clk.

Almacena la terminal en la estructura correspondiente; primero en el mapa después en la que le corresponde, salidas o funciones. Además, son colocadas en la lista terminales, con el número de circuito, el de terminal (pin) y el nodo al que son conectadas las terminales.

Si el circuito integrado cuenta con alguna función, se carga y es almacenada el la lista respectiva, pero representada por medio de un estado, por ejemplo, si fue asignado el estado E0 a la salida Q de un flip-flop, la función NO_Q será representada por JE0, y así es almcenada en la lista. Por último, actualiza la lista de circuitos, dicha actualización es de acuerdo al número de bloques que posee cada circuito integrado, si se encuentran dos circuitos iguales y su tipo es doble, son resumidos a uno solo.

Paso 3. Sustituye las secciones que puedan implementarse con las terminales de función de los circuitos secuenciales. Primero pone en false el campo "flg" de todos los sumandos de la función apuntada. Verifica si existe algun sumando en el bloque apuntado que contenga a "sumando". Actualiza la variable "sumando" buscando en la estructura de implemetación de la estructura función un "+" o ";". Elimina todos los sumandos de un bloque que se absorvan en una función, el último no se elimina, debido a que se utiliza para agregar el nombre de la función como un nuevo sumando.

Paso 4. Si la función será utilizada asigna un nodo a ella. Barre la estructura apuntada a la primer función en busca de sumandos con terminos negados, cada uno de los encontrados se reemplaza por un macro, siendo nuevo para la primera vez y reutilizado las siguientes veces para una misma variable.

Paso 5. Barre la estructura de sumandos, operador booleano "+" (OR), y para cada pareja que se repita se genera un nuevo macro que la sustituya en cada una de sus apariciones. Después cada pareja unida por "^" (operador booleano AND).es sustituida por un macro.

Paso 6. Barre la estructura de sumandos y para cada pareja que se repita se genera un nuevo macro que la sustituya. Obtiene un macro por cada dos elementos de cada lista de implementaciones.

Paso 7. Actualiza las listas de: circuitos, incluyendo los circuitos (combinacionales) necesarios para resolver todos los macros, terminales, las de las compuertas utilizadas y por último la lista de entradas, esta es formada por la lista de entradas del sistema conm su respectivo nodo.

Las compuertas son asignadas de acuerdo al operador booleano que satisface las necesidades de implementación (recordando que sólo están programadas AND, OR y NOT).

Paso 8. Se realiza un barrido a la estructura de funciones de tal modo que cada una de ellas quede asignada al nodo correspondiente de la terminal de control. Si es un código de operación, lo busca en el mapa y actualiza el nodo, si es una salida, actualiza la estructura de salidas en su campo nodo. Cada vez que encuentre un par de nodos comúnes, predominará el menor, de esta manera se eliminan los nodos no utilizables.

Paso 9. Se graba en el archivo cuyo nombre es el que se ha traido desde su generación como archivo de texto, en forma de descripción de diagrama de estado o carta ASM, pero con la extensión .IMP, y la presentación de un archivo de este tipo es mostrada a continuación.

CIRCUITOS	
Num	Nombre

CONEXIONES			
Clo	Pin	Clo	Pin

Figura 4.25.a. Presentación de los resultados obtenidos por el módulo IMPLEMENTA.

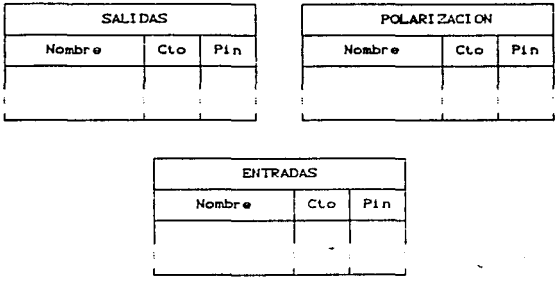


Figura 4.25.b. Presentación de los resultados obtenidos por el módulo IMPLEMENTA.

4.9. MODULO SIMULA

Es necesario recalcar que este módulo se dedica a simular circuitos integados, no simula la solución del sistema.

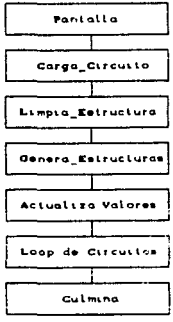


Figura 4.26. Diagrama de bloques del módulo SIMULA

El primer bloque realiza el despliegue de la pantalla en la cual se presenta la simulación del circuito integrado.

El bloque carga circuito, espera la captura o selección del circuito integrado a simular, y lo carga en memoria, en caso de no existir tal circuito, indicará el respectivo mensaje.

Limpia_Estructura es un procedimiento que filtra el archivo, es decir, elimina los renglones en los cuales no se haya escrito algo y elimina los comentarios (aquellos escritos entre corchetes { }), de tal manera que sólo sea analizada la parte coherente de la descripción del circuito.

El siguiente bloque "Genera_Estructuras", primero extrae el tipo de circuito, el número de terminales y limpia las listas de entradas, salidas y terminlaes de control.

Una vez realizado esto, extrae número de terminal (pin), nombre, clasificación y el bloque al que pertenece. De acuerdo a esta información, y recordando que los bloques en un circuito integerado son exactamente iguales, actualiza las listas antes mencionadas para aquellas terminales que pertenezcan al primer bloque.

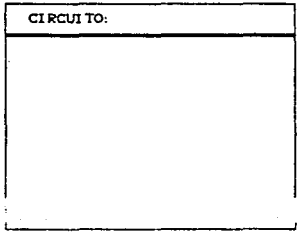


Figura 4.27. Pantalla principal del proceso

Después extrae el numero de instrucciones y llena la lista respectiva con el nombre de la instrucción, el código de operación y la implementación.

El bloque "Actualiza Valores", inicializa la lista de valores, esta lista contiene, las salidas con su respectivo valor binario, una vez ocupadas todas las salidas, en los siguientes lugares de la lista se ubican las entradas tambien con sus respectivos valores binarios. Aunque en esta parte del proceso como su nombre lo indica son inicializados y el valor es cero.

El bloque más importante es el llamado Loop de Circuitos, ya que es en si quien obtiene los resultados, estado siguiente a partir del estado actual. El estado actual depende el circuito integrado, por ejemplo si es un flip-flop, un solo bit, el estado actual es la variante de ese bit (0 y 1), si es un contador será la variante de todos sus bits (si es de 4 bits 0-15), salvo casos particulares, como contadores en BCD.

Ahora entra en un "loop" llamado de instrucciones, en el cual permanece siempre que exista una instrucción para resolver.

En forma anidada se encuentra otro ciclo, el cual prueba con distintas entradas. En primer lugar los valores de la lista son puestos en false, es decir, en cero lógico. Después los valores se llenan de acuerdo al valor binario del estado. Las entradas siempre son consideradas como cero lógico.

Ahora ejecuta la instrucción y para ello requiere de varios pasos.

De la estructura del algoritmo de implementación de las intrucciones, se obtiene el destino, que es el caracter encontrado a la izquierda de la asignación "<". En caso de no ser una salida, lo unico que queda por alternativa es una "@i", y como esta no se encuentra en la lista de valores, es introducida en ella.

Lo que realiza ahora es evaluar el modelo de la implementación, entendiendo por modelo a cada variables de la operación booleana, separada por el símbolo de asignación "<", "and (*)" u "or (+)". Dentro de la evaluación del modelo se realiza una sustitución de las variables por su valor lógico. Dicha sustitución la efectua comparando de la lista de valores, el campo del nombre con lá variable, si es igual, entonces se cambia el modelo por el valor del campo correspondiente en la lista.

Una vez conocido el valor de todas las variables, se efectuan

las operaciones booleanas indicadas, "and" y "or". Y se imprimen en la pantalla el estado actual y el siguiente.

Por último el bloque culmina, lo que realiza es colocar un mensaje en la parte inferio de la pantalla de simulación.

4.10. MODULO PARAMETROS

Este módulo se encarga de las caracteristicas fisicas del equipo, en cuanto al tipo de monitor, La trayectoria y la unidad de disco de la cual debe leer los datos del sistema, tales como archivos de descripción, y los más importantes que son los archivos M.CTO, que son los circuitos integrados, además de que en el disco y trayectoria especificados serán almacenados todos los archivos generados por el sistema de cómputo. Por último indica un lugar de rompimiento, esto es para saber si el sistema se desea trabajar en forma manual, es decir, llevar paso a paso el proceso y seleccionando detenidamente los archivos. Y la otra en forma automática o semiatomática, dependiendo del punto de ruptura. Para que el sistema de computo se lleve en forma manual, el punto de ruptura se coloca en el módulo CREA, en cambio será automático si el punto de ruptura se coloca en el último módulo, es decir en IMPLEMENTA. Se maneja la palabra automático, debido a que una vez creado el archivo de descripción del diagrama de estados o carta ASM, se van ejecutando todos los pasos que componen al sistema de cómputo, sin solicitud de nombre. Si el punto de ruptura es colocado en cualquiera de los otros módulos, se considera como semiautomático, debido a que al llegar al módulo marcado solicita el nombre del proceso.

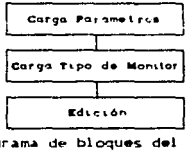


Figura 4.28. Diagrama de bloques del módulo PARAMETROS

El primer bloque indica carga de parámetros, estos parámetros son cargados de la última edición a éste módulo, el cual guarda, los tres parámetros antes mencionados.

El segundo marca que se carga el tipo de monitor, porque en realidad lo que se guarda es una variable booleana que representa el tipo de monitor si es TRUE el monitor es cargado en color, si es FALSE es monocromático.

El bloque pantalla, despliegua la pantalla en la cual se pueden llevar a cabo los cambios a los parámetros mencionados.

CONFIGURACION DEL SISTEMA	
Monitor	
Trayectoria	
Ruptura	

Figura 4.29. Pantalla del módulo PARAMETROS

El bloque edición, como su nombre lo indica es el que permite hacer algún cambio a los tres o a alguno de los tres parámetros. Después de llevar a cabo dicho cambio, se debe almacenar la opción marcada, y este almacenamiento lo realiza en el archivo "parametr.par", Del cual son cargados los parámetros por todos los módulos del sistema de cómputo.

5. APLICACIONES PRACTICAS

De acuerdo a lo tratado en el presente trabajo, es obligado enunciar algunas aplicaciones prácticas para observar con mayor claridad el funcionamiento del sistema de cómputo desarrollado.

Tales aplicaciones son de tipo académico, que permiten una mejor comprensión del sistema, y sin embargo no dejan de ser realmente aplicables.

Control de una cepilladora.

Imaginando la operación de maquinación en la cual una mesa es desplazada hacia un lado y hacia el otro por un motor reversible. Esto sucede en una operación de cepillado en la cual el buril permanece estacionario y la pieza a trabajar se encuentra montada en una mesa oscilatoria. La figura 5.1. muestra las posiciones mencionadas. Cuando el motor gira en una dirección, el conjunto cremallera-piñon mueven la mesa hacia la derecha; cuando el motor gira en la otra dirección, el conjunto mueve la mesa hacia la izquierda. Cuando la mesa se ha movido totalmente a la derecha, acciona la señal que lo indica (TOPDER), el cual indica al circuito de control que invierta el sentido de giro al motor y la mesa se mueve hacia la izquierda; de igual manera opera para el lado opuesto de la mesa, (aunque ahora la señal se llama TOPIZ), y el sentido de desplazamiento es a la derecha. La secuencia se repite tantas veces como sea necesario para terminar la operación.

lo cual es indicado por medio de una entrada al circuito de control (TERMINA).

A continuación se muestra la carta ASM del circuito de control, y posteriormente su interpretación al seudolenguanje del sistema de cómputo.

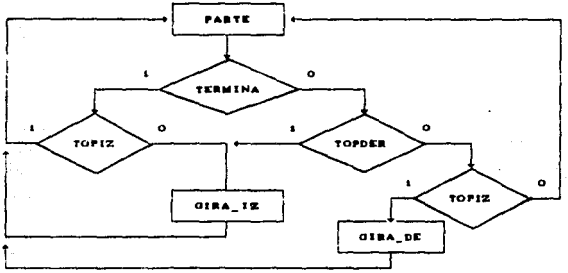


Figura 5.1. Carta ASM del control de una cepilladora

```
PARTE ESTADO;
    SI TERMINA^TOPIZ BRINCA PARTE;
    SI TERMINA^]TOPIZ BRINCA GIZQ;
    SI ]TERMINA^TOPDER BPINCA GIZQ;
    SI ]TERMINA^]TOPDEP^TOPIZ BRINCA GDER;
    SI ]TERMINA^]TOPDEP^]TOPIZ BRINCA PARTE;
GIZQ ESTADO GIRA_IZ;
    BPINCA PARTE;
GDEP ESTADO GIRA_DE;
    BRINCA PARTE;
```

La carta ASM es interpretada por el sistema de cómputo de la manera mostrada en el listado. El siguiente paso es el análisis sintáctico y semántico de la descripción mostrada, dicho análisis entrega un archivo que indica el número de estados, entradas, salidas, la situación del archivo, tal situación se refiere a si el archivo se completo, es decir, que no encontró algún error en el análisis, o, si debido a la presencia de tal error el archivo no se pudo completar el archivo para el ejemplo es el siguiente:

```
Estados.....3
Entradas....3
Salidas.....2
Situacion...COMPLETO

PARTE-ESTADO-;
SI-TERMINA-^-TOPIZ-BRINCA-PARTE-;
SI-TERMINA-^-)-TOPIZ-BRINCA-GIZQ-;
SI-)-TERMINA-^-TOPDER-BRINCA-GIZQ-;
SI-)-TERMINA-^-)-TOPDER-^-TOPIZ-BRINCA-GDER-;
SI-)-TERMINA-^-)-TOPDER-^-)-TOPIZ-BRINCA-PARTE-;
GIZQ-ESTADO-GIRA_IZ-;
BRINCA-PARTE-;
GDER-ESTADO-GIRA_DE-;
BRINCA-PARTE-;
```

El módulo GENERA toma este archivo y forma la tabla característica del sistema digital (circuito de control). El encabezado indica el tamaño del renglón, los bits utilizados por los estados, la cantidad de renglones, el número de entradas y de salidas. Los estados que no son utilizados se ocupan con asteriscos.

```
TamRen=025
BitEdo=002
NumRen=032
NumEnt=003      T    GG
NumSal=002     TE    II
              TOR    RR
              OPM    AA
              PDI    __
           EE IEN SS DI
           10 ZRA 10 EZ

      PARTE 00 000 00 00
            00 001 01 00
            00 010 01 00
            00 011 01 00
            00 100 10 00
            00 101 00 00
            00 110 01 00
            00 111 00 00
       GIZQ 01 000 00 01
            01 001 00 01
            01 010 00 01
            01 011 00 01
            01 100 00 01
            01 101 00 01
            01 110 00 01
            01 111 00 01
       GDER 10 000 00 10
            10 001 00 10
            10 010 00 10
```

(Continúa...)

```
10 011 00 10
10 100 00 10
10 101 00 10
10 110 00 10
10 111 00 10
11 000 ## ##
11 001 ## ##
11 010 ## ##
11 011 ## ##
11 100 ## ##
11 101 ## ##
11 110 ## ##
11 111 ## ##
```

Para resolver la tabla se indica con que circuitos se desea dar solución. En el ejemplo se utilizaron flip-flops tipo JK en especifico elcircuito integrado 7473. Además se observa que en las columnas finales es representado el o los circuitos por medio de sus terminales de control, las cuales forman el código de operación.

```
TamRen=043
BitEdo=002
NumEnt=003
NumSal=002
NuCtos=002
  FF7473
  BitsCO=002
  FF7473
  BitsCO=002
```

```
              T     GG
             TE     II
            TOR     RR
            OPM     AA
            PDI
         EE IEN SS DI
         10 ZRA 10 EZ        KJ        KJ
PARTE    00 000 00 00 2:13  #0 2:13   #0
         00 001 01 00 2:24  #1 2:13   #0
         00 010 01 00 2:24  #1 2:13   #0
         00 011 01 00 2:24  #1 2:13   #0
         00 100 10 00 2:13  #0 2:24   #1
         00 101 00 00 2:13  #0 2:13   #0
         00 110 01 00 2:24  #1 2:13   #0
         00 111 00 00 2:13  #0 2:13   #0
GIZQ     01 000 00 01 2:34  1# 2:13   #0    (Continúa...)
```

```
          01 001 00 01 2:34  1* 2:13  *0
          01 010 00 01 2:34  1* 2:13  *0
          01 011 00 01 2:34  1* 2:13  *0
          01 100 00 01 2:34  1* 2:13  *0
          01 101 00 01 2:34  1* 2:13  *0
          01 110 00 01 2:34  1* 2:13  *0
          01 111 00 01 2:34  1* 2:13  *0
    GDER  10 000 00 10 2:13  *0 2:34  1*
          10 001 00 10 2:13  *0 2:34  1*
          10 010 00 10 2:13  *0 2:34  1*
          10 011 00 10 2:13  *0 2:34  1*
          10 100 00 10 2:13  *0 2:34  1*
          10 101 00 10 2:13  *0 2:34  1*
          10 110 00 10 2:13  *0 2:34  1*
          10 111 00 10 2:13  *0 2:34  1*
          11 000 ** ** 4:123 ** 4:123 **
          11 001 ** ** 4:123 ** 4:123 **
          11 010 ** ** 4:123 ** 4:123 **
          11 011 ** ** 4:123 ** 4:123 **
          11 100 ** ** 4:123 ** 4:123 **
          11 101 ** ** 4:123 ** 4:123 **
          11 110 ** ** 4:123 ** 4:123 **
          11 111 ** ** 4:123 ** 4:123 **
```

Una vez resuelta la tabla, se simplifica a una presentación por medio de funciones booleanas, en cada una de sus entradas, tanto del circuito de control (sistema digital) como del circuito integrado. En el encabezado se agregan algunos datos: numero de circuitos y el nombre de estos, el nombre de cada una de las entradas y salidas del sistema digital. Si la función booleana es una salida se antepone una "s", si es una terminal de control se antepone una "o" con un numero de tres digitos que representa el de circuito.

```
BitEdo=002
NuCtos=002
  FF7473
  FF7473
NumEnt=003
    TOPIZ
   TOPDER
  TERMINA
NumSal=002
   GIRA_DE
   GIRA_IZ
NuFuns=006
s      GIRA_DE=eE1;
s      GIRA_IZ=eE0;
o001          K=1;
o001          J=1#E1^cTOPDER^1cTERMINA+
                1eE1^1cTOPIZ^cTERMINA,          Continua...
```

estas colisiones, llenando la tabla de antemano con asteriscos.

El diagrama de estados es interpretado y representado de la siguiente manera para ser introducido al sistema de computo.

```
IMPOSIBLES S1,S3;
IMPOSIBLES S1,S2;
IMPOSIBLES S2,S3;


PISO3 ESTADO;
   SI B2 BRINCA BAJA;
   SI B1 BRINCA BAJA;
BAJA ESTADO BAJAR;
  SI S2^B2 BRINCA PISO2;
  SI S1    BRINCA PISO1;PISO2 ESTADO;
  SI B3^)B1 BRINCA SUBE;
  SI B1 BRINCA BAJA;
SUBE ESTADO SUBIR;
  SI B2^S2 BRINCA PISO2;
  SI S3    BRINCA PISO3;
PISO1 ESTADO;
  SI B2 BRINCA SUBE;
  SI B3 BRINCA SUBE;
```

Las instrucciones imposibles, indican que tales condiciones no se pueden cumplir en la operación del sistema y por tanto asume que los renglones de la tabla que las contengan deben ser resueltos con condiciones de "no importa" (asteriscos).

El archivo que entrega el módulo ANALIZA es el siguiente:

```
 Estados.....5
 Entradas....8
 Salidas.....2
 Situacion...COMPLETO

IMPOSIBLES-S1-,-S3-;
IMPOSIBLES-S1-,-S2-;
IMPOSIBLES-S2-,-S3-;
PISO3-ESTADO-;
SI-B2-BRINCA-BAJA-;
SI-B1-BRINCA-BAJA-;
BAJA-ESTADO-BAJAR-;
SI-S2-^-B2-BRINCA-PISO2-;
SI-S1-BRINCA-PISO1-;
PISO2-ESTADO-;
SI-B3-^-)-B1-BRINCA-SUBE-;
SI-B1-BRINCA-BAJA-;
SUBE-ESTADO-SUBIR-;
SI-B2-^-S2-BRINCA-PISO2-;
SI-S3-BRINCA-PISO3-;
PISO1-ESTADO-;                                  (Continua...)
```

```
SI-B2-BRINCA-SUBE-;
SI-B3-BRINCA-SUBE-;
```

La tabla generada para este archivo es mostrada a continuación, como es una tabla muy grande (512 renglones), es presentada en varias páginas:

```
TamRen=030
BitEdo=003
NumRen=512
NumEnt=006
NumSal=002
                                  BS
                                  AU
                                  JB
              EEE BBBSSSS SSS AI
              210 123123 210 RR

         PIS03 000 000000 000 00
               000 000001 000 00
               000 000010 000 00
               000 000011 *** **
               000 000100 000 00
               000 000101 *** **
               000 000110 *** **
               000 000111 *** **
               000 001000 000 00
               000 001001 000 00
               000 001010 000 00
               000 001011 *** **
               000 001100 000 00
               000 001101 *** **
               000 001110 *** **
               000 001111 *** **
               000 010000 001 00
               000 010001 001 00
               000 010010 001 00
               000 010011 *** **
               000 010100 001 00
               000 010101 *** **
               000 010110 *** **
               000 010111 *** **
               000 011000 001 00
               000 011001 001 00
               000 011010 001 00
               000 011011 *** **
               000 011100 001 00
               000 011101 *** **
               000 011110 *** **
               000 011111 *** **
               000 100000 001 00
               000 100001 001 00
               000 100010 001 00
               000 100011 *** **                    (Continúa...)
```

	000	100100	001	00
	000	100101	***	**
	000	100110	***	**
	000	100111	***	**
	000	101000	001	00
	000	101001	001	00
	000	101010	001	00
	000	101011	***	**
	000	101100	001	00
	000	101101	***	**
	000	101110	***	**
	000	101111	***	**
	000	110000	001	00
	000	110001	001	00
	000	110010	001	00
	000	110011	***	**
	000	110100	001	00
	000	110101	***	**
	000	110110	***	**
	000	110111	***	**
	000	111000	001	00
	000	111001	001	00
	000	111010	001	00
	000	111011	***	**
	000	111100	001	00
	000	111101	***	**
	000	111110	***	**
	000	111111	***	**
BAJA	001	000000	001	10
	001	000001	001	10
	001	000010	001	10
	001	000011	***	**
	001	000100	100	10
	001	000101	***	**
	001	000110	***	**
	001	000111	***	**
	001	001000	001	10
	001	001001	001	10
	001	001010	001	10
	001	001011	***	**
	001	001100	100	10
	001	001101	***	**
	001	001110	***	**
	001	001111	***	**
	001	010000	001	10
	001	010001	001	10
	001	010010	010	10
	001	010011	***	**
	001	010100	100	10
	001	010101	***	**
	001	010110	***	**
	001	010111	***	**
	001	011000	001	10
	001	011001	001	10
	001	011010	010	10
	001	011011	***	**

(Continua. . .)

```
          001 011100 100 10
          001 011101 *** **
          001 011110 *** **
          001 011111 *** **
          001 100000 001 10
          001 100001 001 10
          001 100010 001 10
          001 100011 *** **
          001 100100 100 10
          001 100101 *** **
          001 100110 *** **
          001 100111 *** **
          001 101000 001 10
          001 101001 001 10
          001 101010 001 10
          001 101011 *** **
          001 101100 100 10
          001 101101 *** **
          001 101110 *** **
          001 101111 *** **
          001 110000 001 10
          001 110001 001 10
          001 110010 010 10
          001 110011 *** **
          001 110100 100 10
          001 110101 *** **
          001 110110 *** **
          001 110111 *** **
          001 111000 001 10
          001 111001 001 10
          001 111010 010 10
          001 111011 *** **
          001 111100 100 10
          001 111101 *** **
          001 111110 *** **
          001 111111 *** **
PIS02     010 000000 010 00
          010 000001 010 00
          010 000010 010 00
          010 000011 *** **
          010 000100 010 00
          010 000101 *** **
          010 000110 *** **
          010 000111 *** **
          010 001000 011 00
          010 001001 011 00
          010 001010 011 00
          010 001011 *** **
          010 001100 011 00
          010 001101 *** **
          010 001110 *** **
          010 001111 *** **
          010 010000 010 00
          010 010001 010 00
          010 010010 010 00
          010 010011 *** **
```

(Continua...)

```
      010 010100 010 00
      010 010101 ### ##
      010 010110 ### ##
      010 010111 ### ##
      010 011000 011 00
      010 011001 011 00
      010 011010 011 00
      010 011011 ### ##
      010 011100 011. 00
      010 011101 ### ##
      010 011110 ### ##
      010 011111 ### ##
      010 100000 001 00
      010 100001 001 00
      010 100010 001 00
      010 100011 ### ##
      010 100100 001 00
      010 100101 ### ##
      010 100110 ### ##
      010 100111 ### ##
      010 101000 001 00
      010 101001 001 00
      010 101010 001 00
      010 101011 ### ##
      010 101100 001 00
      010 101101 ### ##
      010 101110 ### ##
      010 101111 ### ##
      010 110000 001 00
      010 110001 001 00
      010 110010 001 00
      010 110011 ### ##
      010 110100 001 00
      010 110101 ### ##
      010 110110 ### ##
      010 110111 ### ##
      010 111000 001 00
      010 111001 001 00
      010 111010 001 00
      010 111011 ### ##
      010 111100 001 00
      010 111101 ### ##
      010 111110 ### ##
      010 111111 ### ##
SUBE  011 000000 011 01
      011 000001 000 01
      011 000010 011 01
      011 000011 ### ##
      011 000100 011 01
      011 000101 ### ##
      011 000110 ### ##
      011 000111 ### ##
      011 001000 011 01
      011 001001 000 01
      011 001010 011 01
      011 001011 ### ##
```

(Continúa...)

```
        011 001100 011 01
        011 001101 *** **
        011 001110 *** **
        011 001111 *** **
        011 010000 011 01
        011 010001 000 01
        011 010010 010 01
        011 010011 *** **
        011 010100 011 01
        011 010101 *** **
        011 010110 *** **
        011 010111 *** **
        011 011000 011 01
        011 011001 000 01
        011 011010 010 01
        011 011011 *** **
        011 011100 011 01
        011 011101 *** **
        011 011110 *** **
        011 011111 *** **
        011 100000 011 01
        011 100001 000 01
        011 100010 011 01
        011 100011 *** **
        011 100100 011 01
        011 100101 *** **
        011 100110 *** **
        011 100111 *** **
        011 101000 011 01
        011 101001 000 01
        011 101010 011 01
        011 101011 *** **
        011 101100 011 01
        011 101101 *** **
        011 101110 *** **
        011 101111 *** **
        011 110000 011 01
        011 110001 000 01
        011 110010 010 01
        011 110011 *** **
        011 110100 011 01
        011 110101 *** **
        011 110110 *** **
        011 110111 *** **
        011 111000 011 01
        011 111001 000 01
        011 111010 010 01
        011 111011 *** **
        011 111100 011 01
        011 111101 *** **
        011 111110 *** **
        011 111111 *** **
PISO1   100 000000 100 00
        100 000001 100 00
        100 000010 100 00
        100 000011 *** **
```

(Continua...)

```
100 000100 100 00
100 000101 *** **
100 000110 *** **
100 000111 *** **
100 001000 011 00
100 001001 011 00
100 001010 011 00
100 001011 *** **
100 001100 011 00
100 001101 *** **
100 001110 *** **
100 001111 *** **
100 010000 011 00
100 010001 011 00
100 010010 011 00
100 010011 *** **
100 010100 011 00
100 010101 *** **
100 010110 *** **
100 010111 *** **
100 011000 011 00
100 011001 011 00
100 011010 011 00
100 011011 *** **
100 011100 011 00
100 011101 *** **
100 011110 *** **
100 011111 *** **
100 100000 100 00
100 100001 100 00
100 100010 100 00
100 100011 *** **
100 100100 100 00
100 100101 *** **
100 100110 *** **
100 100111 *** **
100 101000 011 00
100 101001 011 00
100 101010 011 00
100 101011 *** **
100 101100 011 00
100 101101 *** **
100 101110 *** **
100 101111 *** **
100 110000 011 00
100 110001 011 00
100 110010 011 00
100 110011 *** **
100 110100 011 00
100 110101 *** **
100 110110 *** **
100 110111 *** **
100 111000 011 00
100 111001 011 00
100 111010 011 00
100 111011 *** **
```

(Continua...)

Ahora es mostrada la tabla ya resuelta con un circuito integrado de registro de corrimiento universal (74194):

```
TamRen=046
BitEdo=003
NumEnt=006
NumSal=002
NuCtos=001
  RC74194
  BitsCO=009
```

```
                                                R
                                                E
                                                S
                                BS              E
                                AU              T
                                JB              TDD
          EEE BBBSSS SSS AI                     _SSSSPPPP
          210 123123 210 RR                     MRLO13210
PISO3     000 000000 000 00 S:123 ****O*****
          000 000001 000 00 S:123 ****O*****
          000 000010 000 00 S:123 ****O*****
          000 000011 *** ** E:123 **********
          000 000100 000 00 S:123 ****O*****
          000 000101 *** ** E:123 **********
          000 000110 *** ** E:123 **********
          000 000111 *** ** E:123 **********
          000 001000 000 00 S:123 ****O*****
          000 001001 000 00 S:123 ****O*****
          000 001010 000 00 S:123 ****O*****
          000 001011 *** ** E:123 **********
          000 001100 000 00 S:123 ****O*****
          000 001101 *** ** E:123 **********
          000 001110 *** ** E:123 **********
          000 001111 *** ** E:123 **********
          000 010000 001 00 CORR_ 1*101*****
          000 010001 001 00 CORR_ 1*101*****
          000 010010 001 00 CORR_ 1*101*****
          000 010011 *** ** E:123 **********
          000 010100 001 00 CORR_ 1*101*****
          000 010101 *** ** E:123 **********
          000 010110 *** ** E:123 **********
          000 010111 *** ** E:123 **********
          000 011000 001 00 CORR_ 1*101*****
          000 011001 001 00 CORR_ 1*101*****
          000 011010 001 00 CORR_ 1*101*****
          000 011011 *** ** E:123 **********
          000 011100 001 00 CORR_ 1*101*****
          000 011101 *** ** E:123 **********
          000 011110 *** ** E:123 **********
          000 011111 *** ** E:123 **********
          000 100000 001 00 CORR_ 1*101*****
          000 100001 001 00 CORR_ 1*101***** (continúa...)
```

```
       000 100011 ***  **  E:123 **********
       000 100100 001  00  CORR_ 1*101*****
       000 100101 ***  **  E:123 **********
       000 100110 ***  **  E:123 **********
       000 100111 ***  **  E:123 **********
       000 101000 001  00  CORR_ 1*101*****
       000 101001 001  00  CORR_ 1*101*****
       000 101010 001  00  CORR_ 1*101*****
       000 101011 ***  **  E:123 **********
       000 101100 001  00  CORR_ 1*101*****
       000 101101 ***  **  E:123 **********
       000 101110 ***  **  E:123 **********
       000 101111 ***  **  E:123 **********
       000 110000 001  00  CORR_ 1*101*****
       000 110001 001  00  CORR_ 1*101*****
       000 110010 001  00  CORR_ 1*101*****
       000 110011 ***  **  E:123 **********
       000 110100 001  00  CORR_ 1*101*****
       000 110101 ***  **  E:123 **********
       000 110110 ***  **  E:123 **********
       000 110111 ***  **  E:123 **********
       000 111000 001  00  CORR_ 1*101*****
       000 111001 001  00  CORR_ 1*101*****
       000 111010 001  00  CORR_ 1*101*****
       000 111011 ***  **  E:123 **********
       000 111100 001  00  CORR_ 1*101*****
       000 111101 ***  **  E:123 **********
       000 111110 ***  **  E:123 **********
       000 111111 ***  **  E:123 **********
BAJA   001 000000 001  10  NOP    1**00******
       001 000001 001  10  NOP    1**00******
       001 000010 001  10  NOP    1**00******
       001 000011 ***  **  E:123 **********
       001 000100 100  10  CARGA  1**11001**
       001 000101 ***  **  E:123 **********
       001 000110 ***  **  E:123 **********
       001 000111 ***  **  E:123 **********
       001 001000 001  10  NOP    1**00******
       001 001001 001  10  NOP    1**00******
       001 001010 001  10  NOP    1**00******
       001 001011 ***  **  E:123 **********
       001 001100 100  10  CARGA  1**11001**
       001 001101 ***  **  E:123 **********
       001 001110 ***  **  E:123 **********
       001 001111 ***  **  E:123 **********
       001 010000 001  10  NOP    1**00******
       001 010001 001  10  NOP    1**00******
       001 010010 010  10  CORR_  1*001******
       001 010011 ***  **  E:123 **********
       001 010100 100  10  CARGA  1**11001**
       001 010101 ***  **  E:123 **********
       001 010110 ***  **  E:123 **********
       001 010111 ***  **  E:123 **********
       001 011000 001  10  NOP    1**00******
       001 011001 001  10  NOP    1**00******
       001 011010 010  10  CORR_  1*001****** (Continua...)
```

```
001 011011 *** ** E:123 **********
001 011100 100 10 CARGA 1**11001*
001 011101 *** ** E:123 **********
001 011110 *** ** E:123 **********
001 011111 *** ** E:123 **********
001 100000 001 10 NOP   1**00*****
001 100001 001 10 NOP   1**00*****
001 100010 001 10 NOP   1**00*****
001 100011 *** ** E:123 **********
001 100100 100 10 CARGA 1**11001*
001 100101 *** ** E:123 **********
001 100110 *** ** E:123 **********
001 100111 *** ** E:123 **********
001 101000 001 10 NOP   1**00*****
001 101001 001 10 NOP   1**00*****
001 101010 001 10 NOP   1**00*****
001 101011 *** ** E:123 **********
001 101100 100 10 CARGA 1**11001*
001 101101 *** ** E:123 **********
001 101110 *** ** E:123 **********
001 101111 *** ** E:123 **********
001 110000 001 10 NOP   1**00*****
001 110001 001 10 NOP   1**00*****
001 110010 010 10 CORP_ 1*001*****
001 110011 *** ** E:123 **********
001 110100 100 10 CARGA 1**11001*
001 110101 *** ** E:123 **********
001 110110 *** ** E:123 **********
001 110111 *** ** E:123 **********
001 111000 001 10 NOP   1**00*****
001 111001 001 10 NOP   1**00*****
001 111010 010 10 CORP_ 1*001*****
001 111011 *** ** E:123 **********
001 111100 100 10 CARGA 1**11001*
001 111101 *** ** E:123 **********
001 111110 *** ** E:123 **********
001 111111 *** ** E:123 **********
PIS02 010 000000 010 00 NOP   1**00*****
010 000001 010 00 NOP   1**00*****
010 000010 010 00 NOP   1**00*****
010 000011 *** ** E:123 **********
010 000100 010 00 NOP   1**00*****
010 000101 *** ** E:123 **********
010 000110 *** ** E:123 **********
010 000111 *** ** E:123 **********
010 001000 011 00 CARGA 1**11110*
010 001001 011 00 CARGA 1**11110*
010 001010 011 00 CARGA 1**11110*
010 001011 *** ** E:123 **********
010 001100 011 00 CARGA 1**11110*
010 001101 *** ** E:123 **********
010 001110 *** ** E:123 **********
010 001111 *** ** E:123 **********
010 010000 010 00 NOP   1**00*****
010 010001 010 00 NOP   1**00*****
010 010010 010 00 NOP   1**00*****
```

```
      010 010011 *** ** E:123 **********
      010 010100 010 00 NOP   1**00*****
      010 010101 *** ** E:123 **********
      010 010110 *** ** E:123 **********
      010 010111 *** ** E:123 **********
      010 011000 011 00 CARGA 1**11110*
      010 011001 011 00 CARGA 1**11110*
      010 011010 011 00 CARGA 1**11110*
      010 011011 *** ** E:123 **********
      010 011100 011 00 CARGA 1**11110*
      010 011101 *** ** E:123 **********
      010 011110 *** ** E:123 **********
      010 011111 *** ** E:123 **********
      010 100000 001 00 2:56  1**10*****
      010 100001 001 00 2:56  1**10*****
      010 100010 001 00 2:56  1**10*****
      010 100011 *** ** E:123 **********
      010 100100 001 00 2:56  1**10*****
      010 100101 *** ** E:123 **********
      010 100110 *** ** E:123 **********
      010 100111 *** ** E:123 **********
      010 101000 001 00 2:56  1**10*****
      010 101001 001 00 2:56  1**10*****
      010 101010 001 00 2:56  1**10*****
      010 101011 *** ** E:123 **********
      010 101100 001 00 2:56  1**10*****
      010 101101 *** ** E:123 **********
      010 101110 *** ** E:123 **********
      010 101111 *** ** E:123 **********
      010 110000 001 00 2:56  1**10*****
      010 110001 001 00 2:56  1**10*****
      010 110010 001 00 2:56  1**10*****
      010 110011 *** ** E:123 **********
      010 110100 001 00 2:56  1**10*****
      010 110101 *** ** E:123 **********
      010 110110 *** ** E:123 **********
      010 110111 *** ** E:123 **********
      010 111000 001 00 2:56  1**10*****
      010 111001 001 00 2:56  1**10*****
      010 111010 001 00 2:56  1**10*****
      010 111011 *** ** E:123 **********
      010 111100 001 00 2:56  1**10*****
      010 111101 *** ** E:123 **********
      010 111110 *** ** E:123 **********
      010 111111 *** ** E:123 **********
SUBE  011 000000 011 01 NOP   1**00*****
      011 000001 000 01 CLEAR 0*********
      011 000010 011 01 NOP   1**00*****
      011 000011 *** ** E:123 **********
      011 000100 011 01 NOP   1**00*****
      011 000101 *** ** E:123 **********
      011 000110 *** ** E:123 **********
      011 000111 *** ** E:123 **********
      011 001000 011 01 NOP   1**00*****
      011 001001 000 01 CLEAR 0*********
      011 001010 011 01 NOP   1**00***** (Continua...)
```

```
011  001011  ***  **  E:123  **********
011  001100  011  01  NOP    1**00******
011  001101  ***  **  E:123  **********
011  001110  ***  **  E:123  **********
011  001111  ***  **  E:123  **********
011  010000  011  01  NOP    1**00******
011  010001  000  01  CLEAR  0*********
011  010010  010  01  CARGA  1**11010***
011  010011  ***  **  E:123  **********
011  010100  011  01  NOP    1**00******
011  010101  ***  **  E:123  **********
011  010110  ***  **  E:123  **********
011  010111  ***  **  E:123  **********
011  011000  011  01  NOP    1**00******
011  011001  000  01  CLEAR  0*********
011  011010  010  01  CARGA  1**11010***
011  011011  ***  **  E:123  **********
011  011100  011  01  NOP    1**00******
011  011101  ***  **  E:123  **********
011  011110  ***  **  E:123  **********
011  011111  ***  **  E:123  **********
011  100000  011  01  NOP    1**00******
011  100001  000  01  CLEAR  0*********
011  100010  011  01  NOP    1**00******
011  100011  ***  **  E:123  **********
011  100100  011  01  NOP    1**00******
011  100101  ***  **  E:123  **********
011  100110  ***  **  E:123  **********
011  100111  ***  **  E:123  **********
011  101000  011  01  NOP    1**00******
011  101001  000  01  CLEAR  0*********
011  101010  011  01  NOP    1**00******
011  101011  ***  **  E:123  **********
011  101100  011  01  NOP    1**00******
011  101101  ***  **  E:123  **********
011  101110  ***  **  E:123  **********
011  101111  ***  **  E:123  **********
011  110000  011  01  NOP    1**00******
011  110001  000  01  CLEAR  0*********
011  110010  010  01  CARGA  1**11010***
011  110011  ***  **  E:123  **********
011  110100  011  01  NOP    1**00******
011  110101  ***  **  E:123  **********
011  110110  ***  **  E:123  **********
011  110111  ***  **  E:123  **********
011  111000  011  01  NOP    1**00******
011  111001  000  01  CLEAR  0*********
011  111010  010  01  CARGA  1**11010***
011  111011  ***  **  E:123  **********
011  111100  011  01  NOP    1**00******
011  111101  ***  **  E:123  **********
011  111110  ***  **  E:123  **********
011  111111  ***  **  E:123  **********
100  000000  100  00  NOP    1**00******
100  000001  100  00  NOP    1**00******
100  000010  100  00  NOP    1**00******
```

```
100 000011 *** ** E:123 **********
100 000100 100 00 NOP    1**00*****
100 000101 *** ** E:123 **********
100 000110 *** ** E:123 **********
100 000111 *** ** E:123 **********
100 001000 011 00 CARGA 1**11110*
100 001001 011 00 CARGA 1**11110*
100 001010 011 00 CARGA 1**11110*
100 001011 *** ** E:123 **********
100 001100 011 00 CARGA 1**11110*
100 001101 *** ** E:123 **********
100 001110 *** ** E:123 **********
100 001111 *** ** E:123 **********
100 010000 011 00 CARGA 1**11110*
100 010001 011 00 CARGA 1**11110*
100 010010 011 00 CARGA 1**11110*
100 010011 *** ** E:123 **********
100 010100 011 00 CARGA 1**11110*
100 010101 *** ** E:123 **********
100 010110 *** ** E:123 **********
100 010111 *** ** E:123 **********
100 011000 011 00 CARGA 1**11110*
100 011001 011 00 CARGA 1**11110*
100 011010 011 00 CARGA 1**11110*
100 011011 *** ** E:123 **********
100 011100 011 00 CARGA 1**11110*
100 011101 *** ** E:123 **********
100 011110 *** ** E:123 **********
100 011111 *** ** E:123 **********
100 100000 100 00 NOP    1**00*****
100 100001 100 00 NOP    1**00*****
100 100010 100 00 NOP    1**00*****
100 100011 *** ** E:123 **********
100 100100 100 00 NOP    1**00*****
100 100101 *** ** E:123 **********
100 100110 *** ** E:123 **********
100 100111 *** ** E:123 **********
100 101000 011 00 CARGA 1**11110*
100 101001 011 00 CARGA 1**11110*
100 101010 011 00 CARGA 1**11110*
100 101011 *** ** E:123 **********
100 101100 011 00 CARGA 1**11110*
100 101101 *** ** E:123 **********
100 101110 *** ** E:123 **********
100 101111 *** ** E:123 **********
100 110000 011 00 CARGA 1**11110*
100 110001 011 00 CARGA 1**11110*
100 110010 011 00 CARGA 1**11110*
100 110011 *** ** E:123 **********
100 110100 011 00 CARGA 1**11110*
100 110101 *** ** E:123 **********
100 110110 *** ** E:123 **********
100 110111 *** ** E:123 **********
100 111000 011 00 CARGA 1**11110*
100 111001 011 00 CARGA 1**11110*
100 111010 011 00 CARGA 1**11110* (continua...)
```

```
100 111011 ### ## E:123 **********
100 111100 011 00 CARGA 1##1111#
100 111101 ### ## E:123 **********
100 111110 ### ## E:123 **********
100 111111 ### ## E:123 **********
101 000000 ### ## E:123 **********
101 000001 ### ## E:123 **********
101 000010 ### ## E:123 **********
101 000011 ### ## E:123 **********
101 000100 ### ## E:123 **********
101 000101 ### ## E:123 **********
101 000110 ### ## E:123 **********
101 000111 ### ## E:123 **********
101 001000 ### ## E:123 **********
101 001001 ### ## E:123 **********
101 001010 ### ## E:123 **********
101 001011 ### ## E:123 **********
101 001100 ### ## E:123 **********
101 001101 ### ## E:123 **********
101 001110 ### ## E:123 **********
101 001111 ### ## E:123 **********
101 010000 ### ## E:123 **********
101 010001 ### ## E:123 **********
101 010010 ### ## E:123 **********
101 010011 ### ## E:123 **********
101 010100 ### ## E:123 **********
101 010101 ### ## E:123 **********
101 010110 ### ## E:123 **********
101 010111 ### ## E:123 **********
101 011000 ### ## E:123 **********
101 011001 ### ## E:123 **********
101 011010 ### ## E:123 **********
101 011011 ### ## E:123 **********
101 011100 ### ## E:123 **********
101 011101 ### ## E:123 **********
101 011110 ### ## E:123 **********
101 011111 ### ## E:123 **********
101 100000 ### ## E:123 **********
101 100001 ### ## E:123 **********
101 100010 ### ## E:123 **********
101 100011 ### ## E:123 **********
101 100100 ### ## E:123 **********
101 100101 ### ## E:123 **********
101 100110 ### ## E:123 **********
101 100111 ### ## E:123 **********
101 101000 ### ## E:123 **********
101 101001 ### ## E:123 **********
101 101010 ### ## E:123 **********
101 101011 ### ## E:123 **********
101 101100 ### ## E:123 **********
```

Los siguientes renglones fueron eliminados del listado por la misma razón que los de la tabla generada.

La simplificación en funciones booleanas es la siguiente:

```
BitEdo=003
NuCtos=001
  RC74194
NumEnt=006
        B1
        B2
        B3
        S1
        S2
        S3
NumSal=002
     BAJAR
     SUBIR
NuFuns=011
s         BAJAR=]eE1^eE0;
s         SUBIR=eE1^eE0;
o001    PESET_M=]cS3+
                ]eE0+
                ]eE1;
o001        DSR=0;
o001        DSL=]eE0;
o001         S0=eE2^cB3+
                eE2^cB2+
                eE1^eE0^cB2^cS2+
                eE1^]eE0^cB1+
                eE1^]eE0^cB3+
                ]eE1^eE0^cS1;
o001         S1=eE2^cB3+
                eE0^cB2^cS2+
                eE1^]eE0^]cB1^cB3+
                ]eE1^eE0^cS1+
                ]eE2^]eE1^]eE0^cB1+
                ]eE1^]eE0^cB2;
o001         P3=]eE0;
o001         P2=]cS1+
                ]eE0;
o001         P1=]eE2^]eE1;
o001         P0=0;
```

Y por ultimo, se muestra la propuesta de circuito de control con compuertas básicas.

CIRCUITOS	
Num	Nombre
1	RC74194
2	7404
3	7408
4	7408
5	7408
6	7408
7	7408
8	7432
9	7432
10	7432

ENTRADAS		
Nombre	Cto	Pin
S3	2	3
B1	6	9
S1	4	9
B3	4	4
B2	7	1
S2	3	12

POLARIZACION		
Nombre	Cto	Pin
VCC	10	14
TIERRA	10	7
RELOJ	1	11

SALIDAS		
Nombre	Cto	Pin
BAJAR	4	10
SUBIR	5	4

CONEXIONES			
Cto	Pin	Cto	Pin
1	2	1	3
1	3	1	8
1	16	2	14
1	8	2	7
1	13	2	1
1	12	2	5
1	7	2	6
1	14	2	11
2	14	3	14
2	7	3	7
2	5	3	1
2	2	3	2
3	1	3	5
2	1	3	5
2	11	3	10
3	14	4	14
3	7	4	7
2	6	4	1
3	4	4	2
3	9	4	4
4	3	4	5
2	13	4	9
3	3	4	10
3	2	4	12
2	10	4	13
1	4	4	11
4	14	5	14
4	7	5	7
3	13	5	1
3	10	5	2
3	6	5	4
3	11	5	5
2	9	5	9
4	5	5	10
3	4	5	12
5	5	5	13
5	14	6	14
5	7	6	7
2	8	6	1
4	6	6	2
4	1	6	4
4	11	6	5
5	9	6	9
6	6	6	10
6	4	6	12
4	12	6	13
6	14	7	14
6	7	7	7

(Continúa...)

CONEXIONES			
Cto	Pin	Cto	Pin
5	1	7	1
6	11	7	2
7	14	8	14
7	7	8	7
4	8	8	1
3	8	8	2
6	12	8	4
2	4	8	5
6	13	8	9
8	6	8	10
1	1	8	8
5	3	8	12
8	3	8	13
8	14	9	14
8	7	9	7
5	6	9	1
8	11	9	2
5	8	9	4
9	3	9	5
6	2	9	9
9	6	9	10
1	9	9	8
5	11	9	12
8	13	9	13
9	14	10	14
9	7	10	7
6	3	10	1
9	11	10	2
6	8	10	4
10	3	10	5
7	3	10	9
10	6	10	10
1	10	10	8
8	4	10	12
2	12	10	13
1	5	10	11

6 CONCLUSIONES

El objetivo del trabajo expuesto en esta tesis, fue desarrollar un sistema de cómputo para ayudar en el diseño de circuitos digitales. Una de las principales metas que se fijaron es tener un sistema versátil, que nos permita hacer modificaciones a parametros de del circuito a diseñar, de tal manera que se comparen las diferentes soluciones obtenidas, de acuerdo a las modificaciones realizadas, y de acuerdo al criterio y experiencia del diseñador decidir la implementación más idonea. Después de realizar un analisis se concluyó que para poder tener una variación en las condiciones de diseño, estas se tienen que realizar desde la descripción del circuito a diseñar, de tal manera que se puedan seguir los cambios experimentados en la dinámica del circuito. Por esta razón el sistema de cómputo se realizó en forma modular; ya que esto nos permite una mayor flexibilidad y posibilita la realización de cambios dentro del módulo en que se esté trabajando.

Para la utilización de este sistema son necesarios conocimientos de diseño lógico, con el fin de tener una visión o se interpreten de manera conveniente los resultados proporcionados por cada uno de los módulos, y el usuario pueda compararlos para cada uno de los cambios realizados en condiciones o parámetros del diseño. Este sistema puede ser utilizado por personas que se

encuentren estudiando un curso de diseño lógico, ya que este le permitirá formarse un criterio amplio en cuanto al diseño de de circuitos digitales y podrán observar como afecta un cambio en las condiciones o parámetros de un circuito en la implementación final.

Por la forma modular en la que se realizó este sistema, es factible incluirle otros más, de tal manera que su versatilidad se vea aumentada. Se puede pensar en un módulo para simular el circuito que se desea implementar, que es presentado por el módulo IMPLEMENTA, para conocer su comportamiento dinámico. O en otro módulo que gráfique el circuito impreso de la implementación final.

Se puede decir que el sistema desarrollado, cuyo nombre es "SISTEMA DISDAC" (Diseño de Sistemas Digitales Asistido por Computadora), cumple con el objetivo propuesto y nos permite tomarlo como ayuda digna de tomarse en cuenta para el diseño de circuitos digitales.

BIBLIOGRAFIA

Duncan, Ray
IBM ROM BIOS
Microsoft

Fletcher, William I.
An Engineering Approach to Digital Design
Prentice-Hall

Green, David
Modern Logic Design
Addison-Wesley

Hayes, John P.
Diseño de Sistemas Digitales y Microprocesadores
McGraw-Hill

Joyanes Aguilar, Luis
Programación en Turbo Pascal (Versiones 4.0,5.0,5.5)
McGraw-Hill

Maloney, Timothy J.
Electrónica Industrial
Prentice-Hall

Mandado, E.
Sistemas Electronicos Digitales
Marcombo

Mano. M. Morris
Lógica Digital y Diseño de Computadores
Prentice-Hall

MOTOROLA
TTL Data Handbook

Peatman. Jhon B.
The Design of Digital Systems
International Student Edition

Tocci, Ronald
Sistemas Digitales Principios y Aplicaciones
Prentice-Hall