UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS FÍSICAS
Departamento de Informática y Automática

TESIS DOCTORAL

Estudio sistemático de circuitos lógicos aritméticos : multiplicadores e integradores digitales

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

María Antonia Canto Díez

Madrid, 2015

© María Antonia Canto Díez, 1983

María Antonia Canto Díez

* 5309860 50X *
UNIVERSIDAD COMPLUTENSE

X-33-0[illegible]-1

ESTUDIO SISTEMATICO DE CIRCUITOS LOGICOS ARITMETICOS: MULTIPLICADORES E INTEGRADORES DIGITALES

Departamento de Informática y Automática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid
1983

BIBLIOTECA

Colección Tesis Doctorales. Nº 68/83

© Mª Antonia Canto Díez
Edita e imprime la Editorial de la Universidad
Complutense de Madrid. Servicio de Reprografía
Noviciado, 3 Madrid-8
Madrid, 1983
Xerox 9200 XB 480
Depósito Legal: M-4968-1983

Autor: Mª ANTONIA CANTO DIEZ

ESTUDIO SISTEMATICO DE CIRCUITOS LOGICOS ARITMETICOS: MULTIPLICADORES E INTEGRADORES DIGITALES

Director: D. Sebastián Dormido Bencomo,
Doctor en Ciencias Físicas, Prof. Agregado de
Física Industrial (Automática)

UNIVERSIDAD COMPLUTENSE

Facultad de Ciencias Físicas

Departamento de Informática y
Automática.

Madrid, 1980.

Deseamos hacer constar nuestro más sincero agradecimeinto al Prof. Dr. D. Sebastián Dormido Bencomo, por su constante dirección y colaboración en el desarrollo de este trabajo.

Al Prof. Dr. D. Mariano Mellado Rodríguez, por sus interesantes sugerencias sobre algunas de las partes de este trabajo, así como por su ayuda en la presentación del mismo, deseamos expresarle nuestra más sincera gratitud.

Estamos también agradecidos a nuestros compañeros, especialmente al Prof. Dr. D. Antonio Vaquero Sánchez que, con sus comentarios, ha contribuido a aclarar algunas de las ideas aquí expuestas. Así como a nuestros profesores, en las distintas etapas de nuestros estudios, que tanto han influido en nuestra formación científica y humana.

Finalmente, damos las gracias a todas aquellas personas que han hecho posible que este trabajo llegara a feliz término.

I N D I C E

pág.

pág.

CAPITULO III. SINTESIS DE CONTADORES PARALELOS GENERALIZADOS

CAPITULO IV. SINTESIS DE MULTIPLICADORES BINARIOS

CAPITULO V. INTEGRADORES DIGITALES DE RESOLUCIÓN EXTENDIDA (IDRE) CON TRANSMISION DE LAS DIFERENCIAS DE SEGUNDO ORDEN

pág.

PROPOSITO Y DESARROLLO DE LA PRESENTE MEMORIA

El objeto del trabajo planteado en la presente memoria ha sido doble. De una parte, se ha abordado de forma general, el problema de la síntesis óptima de funciones de conmutación utilizando un tipo particular de módulos lógicos universales: el multiplexor. De otra, y dando asi título a la memoria, se efectua un estudio sistemático de circuitos lógicos de tipo aritmético.

Dentro del campo del diseño lógico, las propiedades estructurales de las operaciones aritméticas, han recibido una considerable atención. En la actualidad, y debido al auge incesante de los circuitos integrados LSI (Large Scale Integration), los procedimientos convencionales utilizados tienen que ser revisados y planteados desde una óptica diferente, tomando en cuenta las posibilidades que la tecnología nos ofrece.

Para su desarrollo se ha dividido en cinco capítulos organizados de la siguiente forma:

En el primer capítulo, se efectua un análisis de las estructuras de circuitos combinacionales mediante la utilización de módulos lógicos universales (multiplexores). Para la síntesis de cualquier función de conmutación se definen dos tipos de estructuras denominadas arborescentes y generalizadas, y se estudian sus propiedades fundamentales.

En el capítulo segundo, se abordan los problemas de síntesis de circuitos combinacionales realizados con multiplexores. El objetivo ha sido el de dar métodos sistemáticos que nos permitan dada cualquier función de conmutación f(X), pasar a una estructura que la implemente con un coste mínimo (menor número de módulos). En particular se hace uso extensivo del concepto de descomposición funcional, y se pone de manifiesto su importancia en la síntesis de estructuras generalizadas.

El capítulo tercero, está dedicado a estudiar los contadores paralelos generalizados como bloque fundamental en la síntesis de circuitos aritméticos. Este tipo de unidad tiene entre otras, como aplicaciones inmediatas, los sumadores de entrada múlti-

ple, multiplicadores tipo paralelo, procesadores asociativos etc. Se desarrolla un algoritmo de reducción que permite obtener cualquier contador paralelo generalizado que se desee, utilizando únicamente un solo tipo de módulo. Este hecho, es de indudable interés en la síntesis de circuitos aritméticos VLSI.

En el capítulo cuarto, se presenta la síntesis de multiplicadores binarios. Se analizan,con especial interes, los multiplicadores tipo paralelo y sus formas de realización. Básicamente existen dos categorias de multiplicadores paralelos, que son: a) de tipo iterativo, y b) basados en esquemas de reducción de la matriz de productos parciales. Dentro del primero, se propone un nuevo multiplicador iterativo fundamentad en una modificación del esquema de Guild. Para los de la clase b), se generaliza el algoritmo de Stenzel y col., utilizándose de forma extensiva los contadores paralelos generalizados vistos en el capítulo anterior. Asimismo, se proponen nuevos tipos de estructuras para multiplicadores,cuya elección (compromiso velocidad coste) estará obviament en función de la aplicación en concreto a la que se destine el multiplicador.

Finalmente, en el capítulo quinto, se presenta un estudio sobre integradores digitales incrementales de resolución extendida basados en la transmisión de las diferencias de segundo orden. El algoritmo de integración escogido ha sido el de Adams-Bashfort de segundo orden. Se realizan prototipos experimentales, utilizando circuitos integrados MSI, cuando la integración es con respecto al tiempo, y permitiendose adema la inclusión de un coeficiente potenciométrico en el integrador. Tambien se generaliza el integrador digital en el caso de una integral de Stieltjes, presentandose resultado obtenidos por simulación en una calculadora digital.

PRIMERA PARTE

CAPITULO I

ANALISIS DE LAS ESTRUCTURAS DE CIRCUITOS COMBINACIONALES MEDIANTE LA UTILIZACION DE MODULOS LOGICOS UNIVERSALES (MULTIPLEXORES)

1.1.- INTRODUCCION

Uno de los objetivos de la teoría de la conmutación es desarrollar métodos que minimicen parámetros seleccionados en las realizaciones de circuitos lógicos.

Los métodos de los mapas de Karnaugh y la técnica de Quine Mc Cluskey ([1]) determinan la forma de una función de conmutación con el menor número de implicantes primos y de literales. Mientras que estos dos procedimientos ilustran cómo la teoría de la conmutación se puede aplicar para minimizar circuitos lógicos, no consideran, sin embargo, las limitaciones prácticas de fan-in o fan-out. Posteriormente, se desarrollaron algoritmos y procedimientos que minimizan circuitos lógicos sujetos a varias ligaduras. Un método para minimizar circuitos TANT, basado en una extensión de los mapas de Karnaugh, ha sido derivado por Koh ([2]). Un gran esfuerzo de la investigación de circuitos lógicos ha estado centrado en minimizar circuitos NAND/NOR con ligaduras sobre el fan-in ([3])([4])([5])([6]). Los algoritmos mencionados, minimizan circuitos lógicos con un tipo de puertas fijado a priori, Muroga e Ibaraki ([7]), han desarrollado un método general de minimización basado en la programación lineal entera.

Otra área de interés ha sido la de conseguir circuitos lógicos que puedan realizar cualquier función de n variables por cambiar únicamente sus entradas. La mayor parte de la investigación en estos circuitos, que se denominan módulos lógicos universales (MLU), se ha dirigido hacia la minimización del número de conexiones externas (de entrada y salida). Aunque este problema es verdaderamente importante (el costo de un circuito integrado guarda una fuerte interrelación con

el número de conexiones externas), nuestro interés está centrado en presentar métodos generales para el uso óptimo, de una determinada clase de módulos lógicos universales, como componentes básicos para implementar cualquier función de conmutación.

Estos dispositivos que se denominan MLU de tipo multiplexor de orden p, (MLUM(p)) pueden realizar cualquier función de conmutación de $p+1$ variables y se obtienen comercialmente como circuitos integrados denominados multiplexores.

La forma de estos módulos lógicos universales fue deducida por Yau y Tang ([9]), utilizando el teorema de expansión de Shannon ([8]), al expandir una función de p variables respecto a $p-1$ de éstas.

En este capítulo, después de una introducción al problema y una revisión de los trabajos relacionados con los módulos lógicos universales (MLU), se analizan las propiedades fundamentales de las realizaciones con MLU de funciones arbitrarias.

Para la síntesis de cualquier función de conmutación se definen dos tipos de estructuras denominadas arborescentes y generalizadas y se estudian sus propiedades fundamentales. En primer lugar y teniendo en cuenta que nuestro objetivo final va encaminado a dar métodos sistemáticos y a ser posible óptimos (en cuanto al número de MLU necesitados para la síntesis de cualquier función), damos un teorema fundamental válido para ambos tipos de estructuras y que demuestra que únicamente son necesarias estudiar las clases de equivalencia n-p-n para tener definidas las estructuras óptimas.

Esto representa una simplificación drástica en el problema de síntesis, puesto que únicamente nos debemos centrar en la función representativa de la clase de equivalencia. Asimismo, se da un procedimiento sistemático que permite obtener la síntesis de una función cualquiera (bien mediante una estructura arborescente o una estructura generalizada), conociendo la de su función representativa en

la clase de equivalencia n-p-n a la que pertenece.

Es importante advertir desde el principio que el costo (entendido como el número de MLU necesitados en la síntesis) de una estructura arborescente óptima, no es, en general, coincidente con el de una estructura generalizada también óptima.

Con el fin de caracterizar completamente las estructuras arborescentes, se examinan propiedades funcionales individuales, tales como las simetrías parcial y total y se determinan sus efectos, demostrándose que, cada variable simétrica en una función, reduce el número de MLU necesitados en el árbol.

Asimismo, se analizan las propiedades de las estructuras generalizadas y se demuestra que, en general, son preferibles a las estructuras arborescentes al requerir menor número de MLU para su síntesis. Se efectúa una comparación entre ambos tipos de estructuras. Para el caso que denominamos disjunto se hace una comparación directa entre las cotas superiores para cada tipo de estructuras. Sin embargo, no ha sido posible hacer lo mismo para el caso no disjunto.

Finalmente, se caracteriza el número de estructuras generalizadas que existen para un número dado de MLU.

1.2.- MODULOS LOGICOS UNIVERSALES

Como un módulo lógico universal de n variables (MLU(n)) puede realizar cualquier función de conmutación de $n+1$ variables o menos, un MLU(n) ($n \geq 1$) es funcionalmente completo, ya que puede realizar las funciones de dos variables V y 0 y la función de complementar una variable.

Teóricamente, MLU(n) se pueden usar como elementos lógicos en circuitos de conmutación que realizan cualquier función. Este tipo de uso de los MLU(n) está, sin embargo, limitado tanto por la complejidad del módulo como por la dificultad de utilizar el módulo como un elemento lógico.

No obstante, los MLU presentan algunas ventajas sobre las puertas convencionales tipos NAND o NOR, cuando se usan como dispositivos lógicos. En primer lugar, todas las funciones de $n+1$ variables, o menos, pueden realizarse con un MLU(n) (incluyendo, por tanto, las funciones NAND y NOR). Además, muy presumiblemente, deberían de necesitarse un menor número de módulos para realizar una función con MLU(n) que con NAND. Por tanto, incluso aunque un MLU(n) puede ser relativamente complejo, desde un punto de vista interno, un circuito lógico que le utilizase como componente básica debería ser menos complejo que si se construyese mediante puertas convencionales.

Como ya se ha dicho, la mayor parte de los trabajos sobre MLU se han centrado en minimizar el número de conexiones externas (I/O pins), sin preocuparse excesivamente del problema práctico de cómo usar una realización concreta de un MLU en el diseño de sistemas lógicos.

Como un sistema digital típico, en general, utiliza funciones de diferentes dimensiones, caben dos alternativas, o bien el MLU es lo suficientemente grande para poder manejar la función de mayor tamaño, o por el contrario se dispone de algún método para interconexionar MLU. El primer punto de vista no siempre resulta admisible si existe un gran rango en el número de variables por función. El último método es, pues, la mejor forma de utilizar los MLU si los procedimientos para realizar e interconectarles son cómodos.

Desgraciadamente muchos MLU(n) propuestos no realizan de una manera fácil la implementación de funciones lógicas. Por ejemplo, Preparata ([10]) propone un MLU que requiere, por parte del usuario, la determinación de un conjunto de funciones, que él denomina "funciones separatrices" y que son realizadas internamente por el módulo para ser realimentadas a sus entradas. Si las entradas originales al módulo no son variables simples, las funciones separatrices se hacen relativamente complejas respecto a sus entradas. Así pues, este tipo de MLU y muchos otros,

no parecen ser apropiados para utilizar en circuitos lógicos que requieran más de 1 módulo.

Los MLU propuestos por Yau y Tang [9], que hemos denominado MLUM (módulos lógicos universales de tipo multiplexor) presentan la indudable ventaja de que su uso como bloque fundamental para realizar cualquier función de conmutación es directo.

El primer paso para deducir la forma de un MLUM(p) es utilizar la fórmula de expansión de Shannon, expandiendo una función de $p+1$ variables, respecto de p de estas variables

$$f(x_p, x_{p-1}, \ldots, x_0) = \bar{x}_p\, \bar{x}_{p-1} \ldots \bar{x}_1\, f(0,0,\ldots,x_0) + \ldots + x_p\, x_{p-1} \ldots x_1\, f(1,1,\ldots,1,x_0) \tag{1.1}$$

donde las variables x_i, $i=1,2,\ldots,p$ se denominan <u>variables de expansión</u> y las funciones $f(j_p, j_{p-1}, \ldots, j_1, x_0)$, $j_i=0$ ó 1, se conocen como <u>funciones residuos</u>. Efectuemos las siguientes sustituciones. Sea:

$$c_i = x_{i+1} \qquad (i = 0, 1, \ldots, p-1)$$

c_i se llama la <u>entrada de control</u> i-ésima.

$$d_r = f(j_p, j_{p-1}, \ldots, j_1, x_0)$$

donde:

$$(r)_{10} = (j_p, j_{p-1}, \ldots, j_1)_2$$

d_r se llama la <u>entrada de datos</u> r-ésima.

$$f_{p+1} = f(x_p, x_{p-1}, \ldots, x_1, x_0)$$

f_{p+1} es la función de salida del MLUM(p)

Entonces:

$$f_{p+1} = \bar{c}_{p-1}\,\bar{c}_{p-2},\ldots,\bar{c}_0 d_0 + \bar{c}_{p-1}\,\bar{c}_{p-2},\ldots,c_0 d_1 + \ldots + c_{p-1}\,c_{p-2},\ldots,c_0 d_{2p-1} \quad (1.2)$$

Una implementación NAND de (1.2) para $p=2$ y su representación en diagrama de bloques se da en la Figura 1.1. Este circuito y su diagrama son equivalentes a los de un multiplexor de 2 entradas. Para usar este multiplexor como un MLU de 3 variables, cualquier función de este número de variables se expande respecto a dos cualesquiera de ellas

$$f(x_2\ x_1\ x_0) = \bar{x}_2\,\bar{x}_1\,f(0,0,x_0) + \bar{x}_2\,x_1\,f(0,1,x_0) + x_2\,\bar{x}_1\,f(1,0,x_0) + x_2\,x_1\,f(1,1,x_0) \quad (1.3)$$

donde x_1 y x_2 son las variables de expasión y $f(i_2,\ i_1,\ x_0)$ son las funciones residuos $i_2,\ i_1 \in [0,1]$.

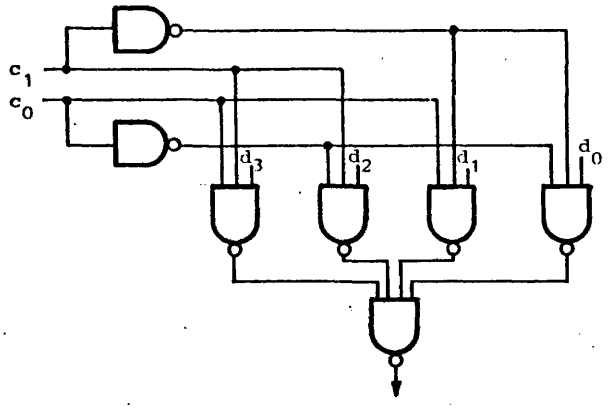
Entonces, las variables de expansión se utilizan como las entradas de control y las funciones residuos como las entradas de datos, tal como se muestra en el ejemplo de la Figura 1.2.

MLUM(p) se pueden utilizar fácilmente para realizar cualquier función de n variables ($n > p + 1$), al expandir repetidamente respecto a p variables. Un ejemplo de como una función arbitraria de 5 variables se podría realizar con MLUM(2) se da en la Figura 1.3. Este circuito es un MLU de 5 variables.

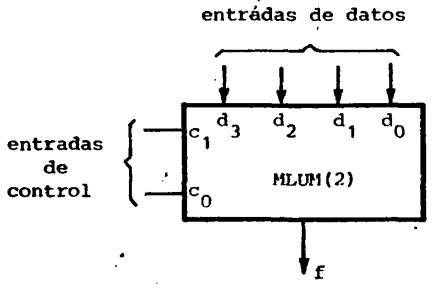
1.3.- REVISION DE LOS TRABAJOS RELACIONADOS CON MLU

La mayor parte de la investigación llevada a cabo en el área del diseño con MLU ha estado centrada en la construcción de módulos con un número mínimo de conexiones externas. Vamos a considerar en esta sección los avances efectuados en esta línea de trabajo, siguiendo su propia evolución histórica.

El trabajo inicial de Forslund y Waxman ([11]) intentaba minimizar al mismo tiempo el número de conexiones externas y el número de puertas en un MLU. Al



a) Diagrama de circuito (realización con puertas NAND)



b) Diagrama de bloques

Figura 1.1.- Módulo lógico universal tipo multiplexor de 2 entradas (MLUM(2))

$$f(x_2,x_1,x_0) = \bar{x}_2\bar{x}_1x_0 + \bar{x}_2x_1 + x_2\bar{x}_1\bar{x}_0$$

$$f(x_2,x_1,x_0) = \bar{x}_2\bar{x}_1(x_0) + \bar{x}_2x_1(1) + x_2\bar{x}_1(\bar{x}_0) + x_2x_1(0)$$

$$c_0 = x_1 \quad , \quad c_1 = x_2$$

$$d_0 = x_0,\ d_1 = 1,\ d_2 = \bar{x}_0,\ d_3 = 0$$

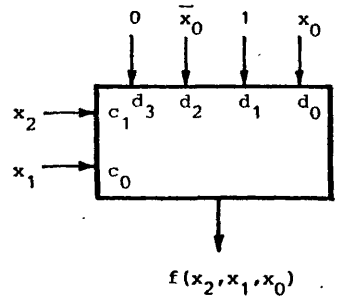


Figura 1.2.- Ejemplo de uso funcional de un MLUM(2)

$$f_j = f(i_4, i_3, i_2, i_1, x_0)$$
$$(j)_{10} = (i_4\ i_3\ i_2\ i_1)_2$$

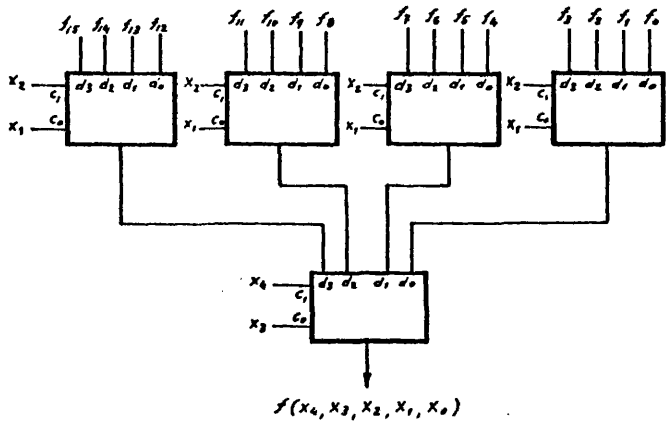


Figura 1.3.- Síntesis de un MLUM(4) con 4 MLUM(2)

permitir permutaciones y complementaciones de las variables y clasificar las 256 funciones booleanas de 3 variables en 10 clases de equivalencia.

Las 10 funciones representando las clases de equivalencia se realizan en un único circuito lógico de 3 variables. El MLU(3) resultante tiene 5 entradas y 2 salidas. Los autores, a continuación, intentan sintetizar un MLU(4), tratando de encontrar funciones de más variables que puedan descomponerse en las 208 clases de equivalencia de 4 variables que ellos determinan. Como el número de funciones de 6 y 7 variables es elevado (2^{26} y 2^{27}), utilizan un proceso aleatorio para seleccionar funciones. La mejor función que encuentran es una de 7 variables que es descomponible en 205 clases.

King ([12]) da un método basado en la definición de un conjunto de operadores, mediante el cual se puede realizar un conjunto de funciones en la forma de un circuito lógico de uso general. Una función se llama una función sucesor de f_i si las operaciones sobre cada minterm produce un minterm de f_i. Si se puede encontrar un conjunto de funciones sucesoras disjuntas, la unión de sus minterms constituirá el circuito lógico de uso general. King unifica sus ideas de manera que se puedan utilizar en un método sistemático que repetidamente aplica operadores hasta que todos los sucesores son disjuntos. El método se puede utilizar para realizar un MLU, sin embargo, no se garantiza que el número de conexiones externas sea mínimo.

Yau y Orsic ([13]) presentan un método que origina un MLU(7) con un mínimo de conexiones externas. Para un MLU(n), definen una matriz de minterms de dim n x s. Plantean un teorema, el cual dice que si esta matriz de minterms P transforma el espacio vectorial n dimensional V_n en V_δ, de tal manera que los δ puntos están sobre la esfera de radio unidad en V_δ, entonces cualquiera de las 2^s funciones se pueden obtener tomando el producto lógico de los minterms y una función lógica lineal. Los autores definen particiones primarias de todos los 2^n minterms,

los cuales satisfacen el teorema y minimizan el número de conexiones de entrada necesitadas para las funciones lógicas lineales.

Se dan las particiones requeridas para 3 variables y se demuestra que las particiones para más variables se pueden deducir de las de 3 variables.

Preparata y Muller ([14]) presentan un método, el cual también particiona todos los minterms de n variables en orden a realizar un MLU(n) con un número mínimo de conexiones externas y donde las variables de entrada pueden aparecer, tanto en forma no complementada como complementada. Estas particiones o bloques tienen la propiedad de que para cada subconjunto propio no vacío del bloque, existe al menos una variable, la cual aparece únicamente en forma complementada o no complementada.

Los bloques se deducen de un conjunto de operaciones matriciales, las cuales, aplicadas repetidamente, transforman un matriz de minterms en una nueva matriz, la cual no tiene filas comunes. Una vez que se encuentra la partición, el conjunto de minterms en una partición se multiplica por una variable de control para dar la función MLU. Este método resulta en una realización de MLU con un número relativamente bajo de conexiones externas cuando n es pequeño y se aproxima al límite teórico mínimo en cuanto al número de conexiones cuando n aumenta.

Un enfoque diferente al problema de minimizar el número de conexiones externas en un MLU ha sido utilizar funciones de más de una variable como entradas al MLU. Patt ([15]) desarrolló un módulo con dos conjuntos de entradas: unas que están constituidas por variables sin complementar se utilizan para realizar la "función separatriz" que se realimentan al otro conjunto de entradas; las otras las constituyen las funciones separatrices y funciones de variables simples se usan para implementar la función MLU. Preparata ([16]) extiende las ideas de Patt de forma sistemática. En este método, los bloques de minterms se constituyen de forma similar al propuesto en el trabajo previo de Preparata y Muller. Una vez

que todos los minterms han sido incluidos, las funciones se construyen de las características de cada bloque de manera que puede seleccionarse cualquier subconjunto de minterms.

En el trabajo de Osman y Weiss ([17]) el problema de minimizar las conexiones externas se enfoca mediante la construcción de unos módulos bases universales, los cuales cuando se dan en conjunción con un conjunto de funciones externas pueden realizar cualquier función de un tamaño dado.

En dos trabajos, Yau y Tang ([9])([18]), deducen un MLU básico al expansionar una función booleana genérica de n variables respecto a $n-1$ de estas variables. Este MLU se denomina MLU tipo multiplexor (MLUM).

Asimismo utilizan estos MLUM como elementos en dos nuevos tipos de MLU con menor número de conexiones externas. Un Q-MLU minimiza dicho número al imponer que se efectúen determinadas conexiones internas para realizar una determinada función. Las conexiones internas se obtienen como salida de un circuito combinacional de salida múltiple que realiza cualquier función de k variables (ellos toman por simplicidad $k=2$). Por tanto, un Q-MLU es un circuito lógico universal de $k+n$ variables con $n+1$ conexiones externas y 2^k conexiones internas. El otro nuevo tipo de MLU utiliza un registro de desplazamiento para suministrar las entradas a otro MLU. La función que el MLU tipo serie realiza viene especificada por la carga secuencial que se le dé al registro de desplazamiento. Estas MLU tienen un número de conexiones externas muy reducido aunque aumente su complejidad interna.

Yau y Tang apuntan en sus trabajos que estos MLU se pueden usar para realizar cualquier función booleana.

1.4.- ESTRUCTURAS ARBORESCENTES Y ESTRUCTURAS GENERALIZADAS

Las estructuras arborescentes emanan de la aplicación iterada del teo-

rema de expansión de Shannon.

Para sintetizar una función booleana f de n variables mediante MLUM(p) se aplica el teorema de expasión de Shannon, resultando:

$$f = \sum_{i=0}^{k} (\dot{x}_0, \dot{x}_1, \ldots, \dot{x}_{p-1})_i \; g_i(x_p, \ldots, x_{n-1}) \qquad (1.4)$$

donde $k = 2^p - 1$, $\dot{x}_j$ puede ser $\bar{x}_j$ o x_j, $(\dot{x}_0, \dot{x}_1, \ldots, \dot{x}_{p-1})_i$ representa el minterm i-ésimo en las variables $x_0, x_1, \ldots, x_{p-1}$ que son p variables cualesquiera de las n variables de f y las funciones $g_i(x_p, \ldots, x_{n-1})$ son los residuos de f. A continuación cada función g_i se realiza siguiendo el mismo procedimiento, hasta que quedan reducidas, o a constantes o variables simples. El resultado es una estructura en árbol de MLUM(p) que realiza la función f.

Definición 1.1.- Una estructura de MULM(p) se denomina arborescente si se cumplen las dos condiciones siguientes:

a) Las entradas de control a cualquier MLUM(p) del circuito son una variable simple.

b) Las salidas de los MLUM(p) del nivel j se conectan a las entradas de datos de algún MLUM(p) del nivel $j-1$. (Ver Figura 1.4).

Ejemplo 1.1.- Utilizando MLUM(2) se desea sintetizar la siguiente función de 5 variables

$$f(x_4, x_3, x_2, x_1, x_0) = \sum(0,1,3,4,5,7,8,9,10,12,13,14,15,17,21,22,23,24,28,29,30,31) \qquad (1.5)$$

Si efectuamos la síntesis expansionando en el primer nivel las variables x_4, x_3 y x_2, x_1 en el segundo, resulta lo siguiente:

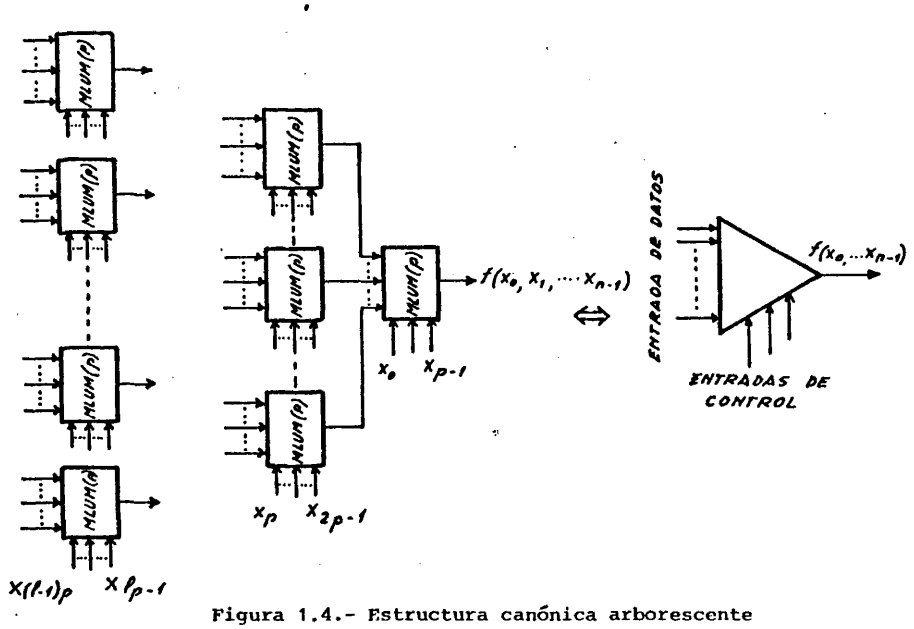


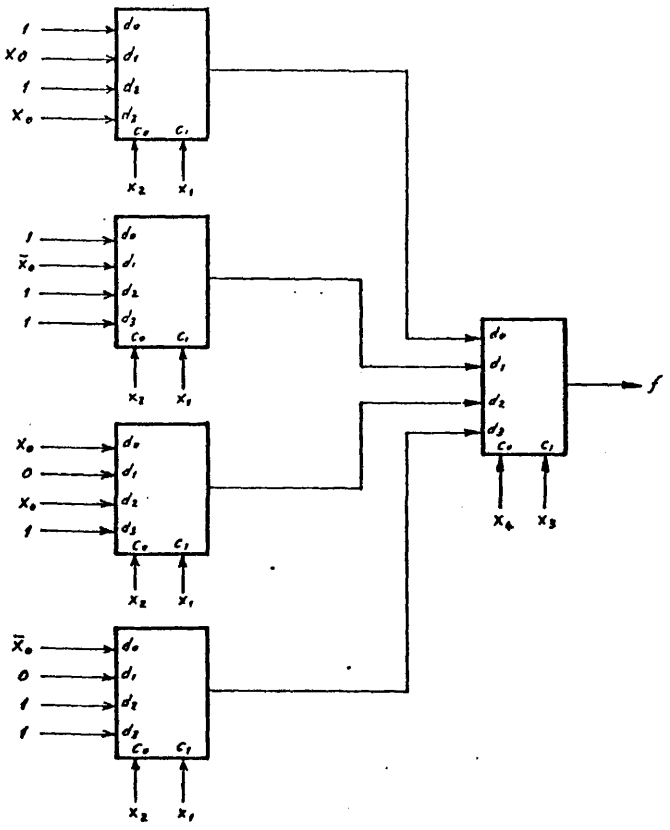
Figura 1.4.- Estructura canónica arborescente

$$
\begin{aligned}
f(x_4,x_3,x_2,x_1,x_0) = {} & \bar{x}_4\bar{x}_3\left[\bar{x}_2\bar{x}_1(1)+\bar{x}_2x_1(x_0)+x_2\bar{x}_1(1)+x_2x_1(x_0)\right] + \\
& \bar{x}_4x_3\left[\bar{x}_2\bar{x}_1(1)+\bar{x}_2x_1(\bar{x}_0)+x_2\bar{x}_1(1)+x_2x_1(1)\right] + \\
& x_4\bar{x}_3\left[\bar{x}_2\bar{x}_1(x_0)+\bar{x}_2x_1(0)+x_2\bar{x}_1(x_0)+x_2x_1(1)\right] + \\
& x_4x_3\left[\bar{x}_2\bar{x}_1(\bar{x}_0)+\bar{x}_2x_1(0)+x_2\bar{x}_1(1)+x_2x_1(1)\right]
\end{aligned}
\qquad (1.6)
$$

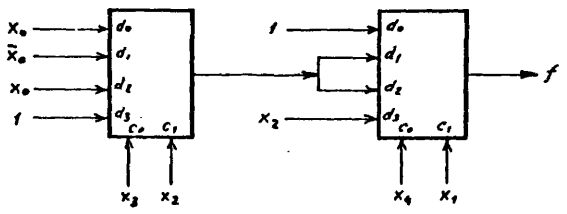
Correspondiéndole la estructura arborescente de la figura 1.5a.

Sin embargo, si expansionamos en el primer nivel respecto a x_4, x_1, y en el segundo respecto a x_3, x_2, se deduce:

$$
\begin{aligned}
f(x_4,x_3,x_2,x_1,x_0) = {} & \bar{x}_4,\bar{x}_1(1) + \\
& \bar{x}_4x_1\left[\bar{x}_3\bar{x}_2(x_0)+\bar{x}_3x_2(\bar{x}_0)+x_3\bar{x}_2(x_0)+x_3x_2(1)\right] + \\
& x_4\bar{x}_1\left[\bar{x}_3\bar{x}_2(x_0)+\bar{x}_3x_2(\bar{x}_0)+x_3\bar{x}_2(x_0)+x_3x_2(1)\right] + \\
& x_4x_1(x_2)
\end{aligned}
\qquad (1.7)
$$



a) Circuito original del ejemplo 1-1.



b) Circuito modificado

Figura 1.5

La estructura arborescente, en este caso, se simplifica como se indica en la Figura 1.5b.

Este ejemplo pone de manifiesto que para realizar una función de conmutación utilizando MLUM, dependiendo del orden en que se expansionen las variables en los distintos niveles, la síntesis resulta más o menos económica.

Se plantea así un problema de minimización de MLUM en estructuras arborescentes.

El problema de minimización es el de decidir el orden en el cual las variables deberían ser expansionadas para dar una estructura en árbol mínima.

Definición 1.2.- Diremos que una estructura arborescente es mínima si no existe ninguna otra estructura arborescente que implemente a la función y que requiera un menor número de MLUM.

En el próximo capítulo se darán procedimientos sistemáticos para la síntesis de estructuras arborescentes óptimas o cuasióptimas, ahora más bien estamos interesados en analizar las propiedades funcionales de dichas estructuras.

En principio, se podría pensar en un procedimiento exhaustivo que tuviese en cuenta todas las posibles permutaciones de las variables de control; sin embargo, esto daría lugar a un proceso considerablemente largo, incluso realizado mediante ordenador. En efecto, para una función de n variables que se expande completamente mediante MLUM(p), el número de expansiones se puede calcular fácilmente. La función se puede expandir respecto a cualquier combinación de las n variables tomadas de p en p. De igual manera, cada una de las 2^p funciones residuos respecto a cualquier combinación de las n-p variables restantes tomadas de p en p. Para el nivel de expansión i-ésimo, las 2^{ip} funciones residuos se pueden expandir respecto a cualquier combinación de las $n-ip$ variables restantes tomadas de p en p.

Por tanto, para una expansión de ℓ niveles, el número de posibles expan-

siones viene dado por:

$$m = \prod_{i=0}^{\ell-1} \binom{n-ip}{p} 2^{ip} \qquad (1.8)$$

A priori, se tienen las siguientes cotas superiores para realizar una función f de n variables con MLUM(p).

Como en cada nivel se eliminan p variables (las p entradas de control al MLUM(p)) y como las entradas de datos al último nivel pueden ser constantes lógicas o una variable simple, se deduce fácilmente que el número de niveles en la estructura arborescente es:

$$\ell = \left\lceil \frac{n-1}{p} \right\rceil \qquad (1.9)$$

donde $\lceil x \rceil$ se define como el menor entero mayor o igual a x.

Si no se puede eliminar ningún MLUM(p) en la estructura arborescente, el número de módulos necesitados viene dado como la suma de los términos de una progresión geométrica de razón $k = 2^p$, el número de término ℓ (número de niveles) y primer término igual a 1.

$$N_{max} = 1 + k + k^2 + \ldots + k^{\ell-1} = \frac{k^\ell - 1}{k-1} \qquad \text{con} \quad k = 2^p \qquad (1.10)$$

Si p no divide exactamente a $n-1$, el primer nivel tendrá sin usar alguna de sus entradas de control o posiblemente se escoja un MLUM(p_1); $p_1 < p$ en el primer nivel. p_1 viene dado por la expresión: $p_1 = n-1-p\,(\ell-1)$.

En este caso, el valor de N_{max} se calcula de la siguiente expresión:

$$N_{max} = 1 + 2^{p_1}(1 + k + \ldots + k^{\ell-2}) = 1 + 2^{p_1}\,\frac{k^{\ell-1}-1}{k-1} \qquad (1.11)$$

Obviamente, las estructuras arborescentes de MLUM(p) no representan la

forma más general de interconectar módulos lógicos universales para implementar una función.

Definición 1.3.- Una estructura de MLUM(p) se denomina generalizada si se permite que las entradas de control a cualquier MLUM(p) del circuito sean funciones multivariables y no variables simples, como era el caso de la estructura arborescente (Ver Figura 1.6).

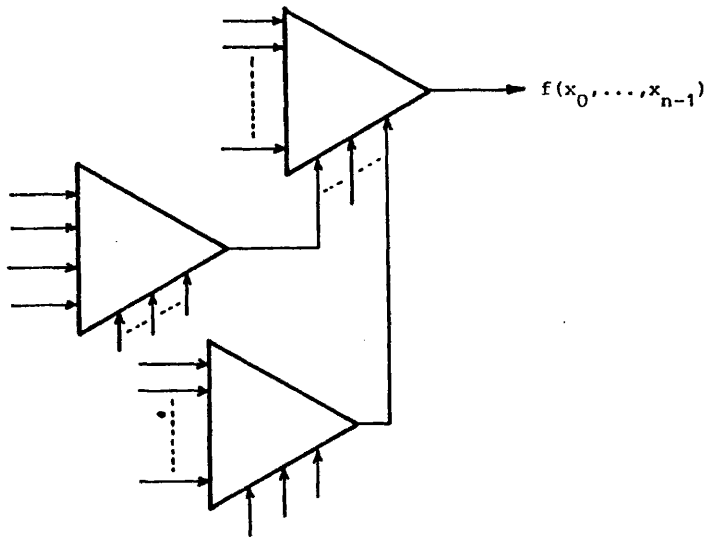


Figura 1.6.- Estructura generalizada de MLUM(p)

Definición 1.4.- Diremos que una estructura generalizada es mínima si no existe ninguna otra estructura generalizada que implemente a la función y que requiera un menor número de MLUM.

La estructura generalizada de MLUM(p) presenta una mayor complejidad en su tratamiento, ya que las entradas de control a un MLUM(p) cualquiera pueden ser

a su vez, una estructura arborescente. Es decir, que una estructura generalizada de MLUM(p) corresponde a una estructura arborescente de estructuras arborescentes.

Ejemplo 1.2.- Para algunas funciones se puede obtener una reducción sustancial en el número de MLUM(p) necesitados, si se utiliza una estructura generalizada. Por ejemplo, únicamente 2 MLUM(1) se requieren para realizar la función de la Fig. 1.7b) con una estructura generalizada, mientras que, al menos 3 MLUM(1) hacen falta en la estructura arborescente mínima que se muestra en la Figura 1.7a.

Las estructuras generalizadas no han sido estudiadas ni a nivel de sus propiedades ni de los métodos de síntesis.

En particular, nuestro objetivo en este capítulo es el de dar las propiedades generales de ambos tipos de estructuras y las interrelaciones que existen entre ellas.

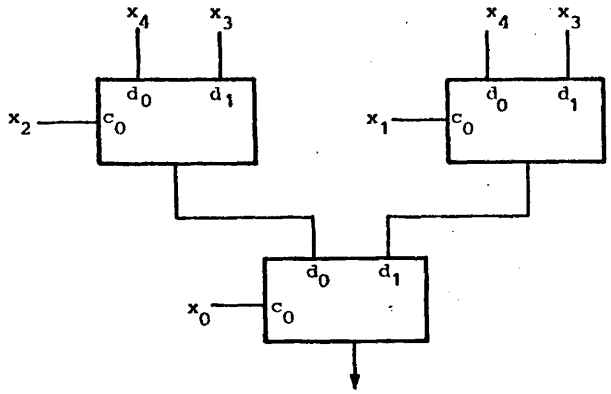
1.4.1.- Relaciones de equivalencia en las estructuras arborescentes y generalizadas.-

La síntesis directa de todos los circuitos óptimos con MLUM(p) para cada una de las 2^{2^n} funciones de n variables, requeriría una cantidad enorme de tiempo de cálculo. Existe, sin embargo, un medio por el cual el problema de síntesis se puede reducir drásticamente.

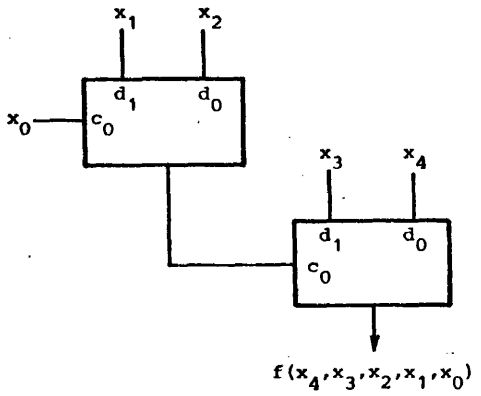
Efectuando simples transformaciones, las funciones se pueden agrupar en clases. Este concepto es útil a la hora de preparar un catálogo de circuitos para cualquier función de un número determinado de variables.

Sean dos funciones de conmutación $f_1(x_{n-1}, x_{n-2},\ldots,x_0)$ y $f_2(x_{n-1}, x_{n-2}, \ldots,x_0)$ tales que f_2 se puede hacer idéntica con f_1 permutando o complementando alguna de sus variables. Por ejemplo, consideremos $f_1(x_2,x_1,x_0) = x_2\bar{x}_1 + \bar{x}_2x_1x_0$ y $f_2(x_2,x_1,x_0) = x_2x_0 + \bar{x}_2\bar{x}_0x_1$. Si en f_2 se permutan x_1 y x_0 y a continuación se

$$f(x_4,x_3,x_2,x_1,x_0) = x_3x_1x_0 + x_3x_2\bar{x}_0 + x_4\bar{x}_1x_0 + x_4\bar{x}_2\bar{x}_0 + x_4\bar{x}_2\bar{x}_1$$



a) Estructura arborescente mínima



b) Estructura generalizada

Figura 1.7.- Comparación estructuras arborescentes - Estructuras generalizadas

complementa x_1, la función que resulta es δ_1. Sin embargo, la función $\delta_3 = x_2 + x_1x_0$ no puede hacerse coincidir con δ_1 independientemente de que variables se complementen o permuten. Hemos llegado así al concepto de clase de equivalencia n-p, basado en la permutación y complementación de variables. Así pues, δ_1 y δ_2 pertenecen a la misma clase de equivalencia n-p y δ_1 y δ_3 no. Si además introducimos la operación de complementar la propia función, ampliamos el espectro de las funciones que pertenecen a la misma clase de equivalencia. Por ejemplo:

$$\delta_4 = x_2\bar{x}_0 + \bar{x}_2x_0 + \bar{x}_2\bar{x}_1 + \bar{x}_1\bar{x}_0 = \overline{(x_2x_0 + \bar{x}_2\bar{x}_0x_1)} = \bar{\delta}_2$$

se deduce de δ_2 y en consecuencia de δ_1. Esto nos lleva al concepto de clase de equivalencia n-p-n y que formalizamos en la definición siguiente:

Definición 1.5.- δ_1 es equivalente n-p-n a δ_2, denotado por $\delta_1 \overset{\text{n-p-n}}{\sim} \delta_2$, si δ_2 se puede obtener de δ_1 por cualquier combinación de las tres operaciones siguientes:

1) Negación (complementación) de una o más variables de δ_1
2) Permutación de variables de δ_1
3) Negación (complementación) de δ_1

De esta forma, las 2^{2^n} funciones de n variables se pueden particionar en clases de equivalencia n-p-n.

Así, para $n=3$ existen 14 clases de equivalencia n-p-n y para $n=4$ el número se eleva a 222.

El contar el número de funciones que pertenece a cada clase y el de clases de equivalencia que existen para cualquier n no es un problema trivial.

Una clase de equivalencia tiene la propiedad importante de que una estructura óptima (arborescente o generalizada) de MLUM(p) para una función determinada perteneciente a la clase puede ser, de forma directa y fácil, modificada,

pero siempre con el mismo número de MLUM(p) en un circuito también óptimo para cualquier otra función de la misma clase. Así pues, determinando todas las estructuras óptimas para una función (llamada función representativa de la clase de equivalencia) escogida de cada una de las clases de equivalencia n-p-n, resulta implícitamente en una determinación de todos los circuitos óptimos para las 2^{2^n} funciones de conmutación de n variables o menos.

El número máximo de funciones, $v(n)$, que puede contener una clase de equivalencia n-p-n viene dado por:

$$v(n) = 2^{n+1} \cdot n! \qquad (1.12)$$

Como diferentes combinaciones de negaciones y permutaciones a menudo producen la misma función, a partir de una dada, muchas clases de equivalencia contienen un número de elementos menor que $v(n)$. En la tabla 1.1 están representadas las funciones representativas de las 14 clases de equivalencia que existen para funciones de tres variables, indicándose, además, el número de funciones que pertenecen a cada una de las clases. Se observa que ninguna clase contiene el máximo número de elementos $v(3) = 96$. Se ha utilizado una notación de dos dígitos hexadecimales para representar la función. Así por ejemplo:

$$f_1 = (\underbrace{0\ 0\ 0\ 1}_{1}\ \ \underbrace{1\ 0\ 1\ 1}_{B})$$

La función representativa de la clase de equivalencia n-p-n se escogió como aquélla que tenga el menor valor hexadecimal entre los miembros de la clase respectiva.

Si se desea obtener una función f de tres variables, que no es una función representativa, se procederá de la forma siguiente:

1) Efectuar las 96 combinaciones posibles mediante los pasos a)-c) para

número de clase de equivalencia	función representativa de la clase de equivalencia	representación hexadecimal	número de funciones en la clase de equivalencia
1	$f_1(x_2,x_1,x_0) = 0$	0 0	2
2	$f_2(x_2,x_1,x_0) = x_2x_1x_0$	0 1	16
3	$f_3(x_2,x_1,x_0) = x_2x_1$	0 3	24
4	$f_4(x_2,x_1,x_0) = x_2\ (x_1 \oplus x_0)$	0 6	24
5	$f_5(x_2,x_1,x_0) = x_2\ (x_1 + x_0)$	0 7	48
6	$f_6(x_2,x_1,x_0) = x_2$	0 F	6
7	$f_7(x_2,x_1,x_0) = x_2x_1\bar{x}_0 + \bar{x}_2x_1x_0 + x_2\bar{x}_1x_0$	1 6	16
8	$f_8(x_2,x_1,x_0) = x_1x_0 + x_2\ (x_1 + x_0)$	1 7	8
9	$f_9(x_2,x_1,x_0) = x_2\bar{x}_1\bar{x}_0 + \bar{x}_2x_1x_0$	1 8	8
10	$f_{10}(x_2,x_1,x_0) = x_1x_0 + x_2\bar{x}_1\bar{x}_0$	1 9	48
11	$f_{11}(x_2,x_1,x_0) = x_1x_0 + x_2\bar{x}_0$	1 B	24
12	$f_{12}(x_2,x_1,x_0) = (x_1x_0 + x_2)(\bar{x}_2 + \bar{x}_1 + \bar{x}_0)$	1 E	24
13	$f_{13}(x_2,x_1,x_0) = x_2 \oplus x_1$	3 C	6
14	$f_{14}(x_2,x_1,x_0) = x_2 \oplus x_1 \oplus x_0$	6 9	2

Número de clases de equivalencia n-p-n para funciones de hasta 6 variables

n	0	1	2	3	4	5	6
nº de clases de equivalencia n-p-n	1	2	4	14	222	616126	200.253.952.527.184

TABLA 1.1.- Clases de equivalencia n-p-n para n = 3 variables

f, registrando todas las combinaciones (puede existir más de una) que dan la función f' con el valor hexadecimal más pequeño.

a) Complementar ninguna, una o más variables de f (8 posibilidades).

b) Efectuar cualquier permutación de las variables de f (6 posibilidades).

c) Complementar o dejar sin complementar la función resultante (2 posibilidades).

2) La función f' es la función representativa para la clase de equivalencia n-p-n de la cual f es un miembro. Encontrar el circuito óptimo de f'.

3) Convertir el circuito óptimo s' que realiza f' en un circuito óptimo s que realiza f, tomando en cuenta los pasos 1a)-1c) que transforman f en f' y efectuando las acciones siguientes:

a) Permutar las variables en s' de forma inversa a como fueron permutadas en el paso 1-b) (por ejemplo, si x_i fue permutada con x_j en 1-b), toda aparición de x_j y $\overline{x}_j$ en s' debería reemplazarse por x_i y $\overline{x}_i$ respectivamente).

b) Complementar las mismas variables en el circuito resultante que las que fueron complementadas en el paso 1-a) si aparecen como entradas de datos, si no (es decir, aparecen como entradas de control) dejarlas invariantes pero permutar adecuadamente las funciones residuos correspondientes.

c) Si la función fue complementada en el paso 1-c), complementar todas las entradas de datos que no se utilicen para generar entradas de control si la estructura es generalizada.

Ejemplo 1.3.- Se desea sintetizar la función $f(x_2, x_1, x_0)$ con representación hexadecimal D4.

Efectuando el paso 1) del proceso descrito, se obtiene que la función $f' = 17$ corresponde a la función representativa de la clase de equivalencia n-p-n. f' se deduce de f, complementando x_0 (paso 1-a), permutando x_2 y x_1 (paso 1-b) y complementando la función resultante (paso 1-c).

$$f'(x_2, x_1, x_0) = \overline{f}(x_1, x_2, \overline{x}_0) \qquad (1.13)$$

El paso 2) nos indica que el circuito óptimo para f' corresponde a f_8. La Figura 1.8a, ilustra la conversión de este circuito en un circuito óptimo para f. En primer lugar, de acuerdo al paso 3-a), se intercambian las variables x_2 y x_1 (cuando únicamente intervienen dos variables, una permutación y su inversa son idénticas), obteniéndose el circuito de la Figura 1.8b. A continuación, el paso 3-b) nos indica que se debe complementar la variable x_0 (como aparece como entrada de control se deja invariable y se permutan las entradas de datos correspondientes al MLUM(1), resultando el circuito de la Figura 1.8c. Finalmente, de acuerdo al paso 3-c), se complementan las entradas de datos al MLUM(1) que no se utilice para generar ninguna entrada de control, Figura 1.8d.

Ejemplo 1.4.- Las funciones AND, NAND, OR y NOR, pertenecen a una misma clase de equivalencia n-p-n y por lo tanto su síntesis se deduce fácilmente a partir de una de ellas como se observa en la Figura 1.9.

1.4.2.- Propiedades funcionales especiales

El teorema de expansión de Shannon se puede utilizar para detectar distintas propiedades de las funciones de conmutación. Por ejemplo, usando expansiones de variables simples se pueden encontrar variables simétricas y funciones autoduales. Es deseable en el diseño de circuitos digitales utilizar propiedades de la función como ayuda en el propio proceso de diseño. Así, la simetría se ha usado

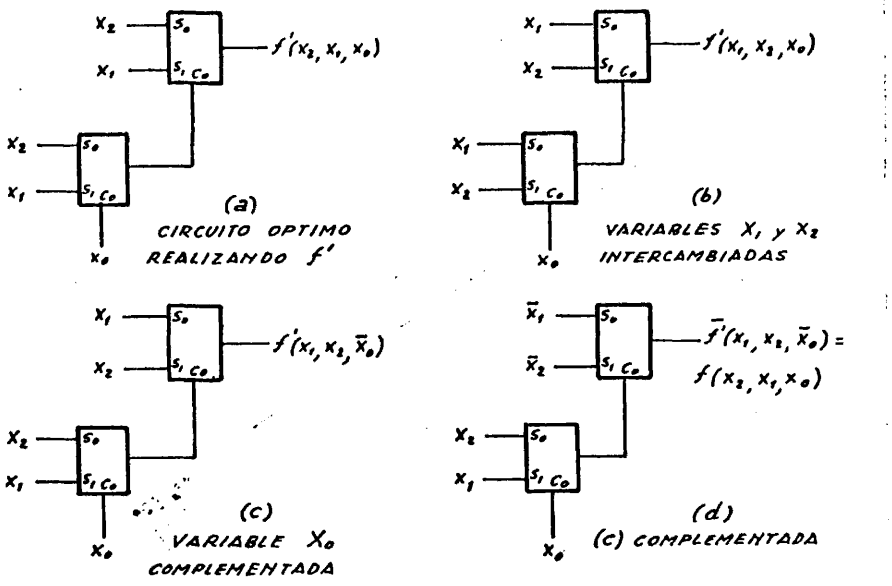


Figura 1.8.- Conversión del circuito óptimo para f' (hexadedimal 17) en el circuito óptimo para f (hexadecimal D4)

extensivamente en la simplificación de circuitos de relé ([19]).

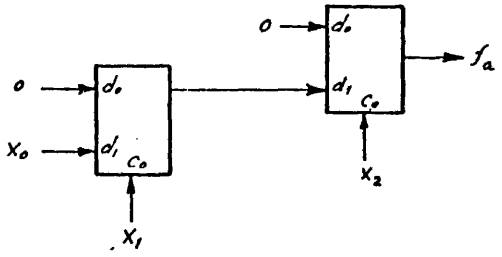
Como (1.4) indica que todas las entradas de datos en una estructura arborescente de MLUM(p) deben ser funciones residuos de la función de salida, es lógico buscar propiedades funcionales específicas que nos conduzcan o bien a una reducción en el número de funciones residuos distintas que se necesitan o a una simplificación en la secuencia de expansionar las variables.

Definición 1.6.- Una función de conmutación $f(x_{n-1},\ldots,x_j,\ldots,x_k,\ldots,x_0)$ es simétrica con respecto a $x_k^{i_k}$ y $x_j^{i_j}$ si

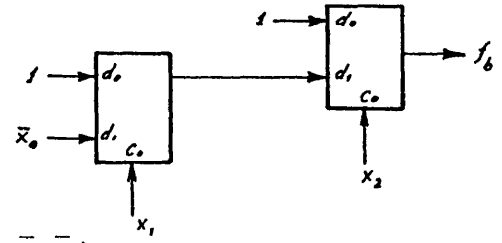
$$f(x_{n-1},\ldots,x_k^{i_k},\ldots,x_j^{i_j},\ldots,x_0) = f(x_{n-1},\ldots,x_j^{i_j},\ldots,x_k^{i_k},\ldots,x_0)$$

donde $i_k, i_j = 0$ ó 1, $x_k^0 = x_k$ y $x_k^1 = \bar{x}_k$

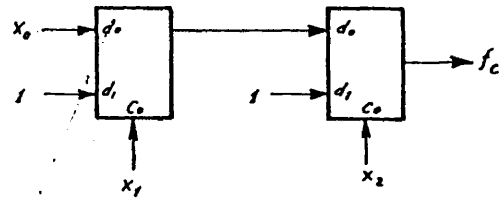
AND → $f_a = x_2x_1x_0$



NAND → $f_b = \overline{x_2x_1x_0} = \overline{f}_a$



OR → $f_c = x_2 + x_1 + x_0 = f_b(\overline{x}_2, \overline{x}_1, \overline{x}_0)$



NOR → $f_d = \overline{x_2 + x_1 + x_0} = \overline{f}_c$

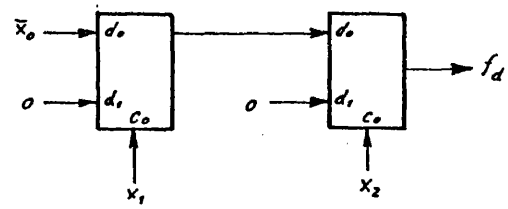


Figura 1.9.- Relación entre los circuitos AND, NAND, OR y NOR y paso de unas a otras

Esta definición nos dice que si $x_k^{i_k}$ y $x_j^{i_j}$ se pueden intercambiar en la función sin modificar su relación de entrada-salida, entonces son simétricas.

Un método de detectar simetría parcial es expandir la función respecto al par de variables en cuestión. Si

$$f(x_{n-1},\ldots,i_k,\ldots,i_j,\ldots,x_0) = f(x_{n-1},\ldots,i_j,\ldots,i_k,\ldots,x_0)$$

entonces f es parcialmente simétrica con respecto a $x_k^{i_k}$ y $x_j^{i_j}$. El hecho de que una función sea simétrica con respecto al menos a dos variables permite reducir el número de MLUM(p) necesitados en el circuito. Si las dos variables simétricas se utilizan como entradas de control en el MLUM(p) $(p \geq 2)$ de salida o del primer nivel (para MLUM(1) las variables simétricas deberían usarse en los dos primeros niveles) entonces al menos 2^{p-1} de las entradas de datos a dicho MLUM(p) son idénticas (para MLUM(1) la mitad de las entradas de datos del segundo nivel son idénticas). Estas entradas de datos deben de coincidir porque 2 de las 4 funciones residuos resultantes de la expansión respecto al par de variables simétricas deben ser iguales. Como las entradas de datos idénticas se pueden manejar, implementando solamente una vez dicha función y permitiendo transmitirla (no hay restricción de fan-out) a los puntos apropiados, no hay por qué realizarlas individualmente.

Como el número de MLUM(p) necesitados para realizar una función de n variables, según (1.10) es a lo más:

$$N_{max} = \sum_{i=0}^{\ell-1} 2^{p.i} \qquad \text{donde} \quad \ell = \left\lceil \frac{n-1}{p} \right\rceil$$

Entonces, la realización de esa función con MLUM(p) $(p \geq 2)$ requiere a lo más:

$$N_{max} = \sum_{i=0}^{\ell-1} 2^{p.i} - \frac{1}{4}\left(\sum_{i=0}^{\ell-1} 2^{p.i} - 1\right) = \frac{3k^{\ell} + k - 4}{4(k-1)} \text{ MLUM(p)} \qquad (1.14)$$

con $k = 2^p$

La cantidad

$$\frac{1}{4}\left(\sum_{i=0}^{\ell-1} k^i - 1\right)$$

se deduce del hecho de que 1/4 de los MLUM(p) restantes no se necesitan porque tienen entradas de datos idénticas. Como un MLUM(p) se utiliza en la etapa de salida, hay que restar un MLUM(p) de esta cantidad. (Ver Figura 1.10 para el caso $p=2$).

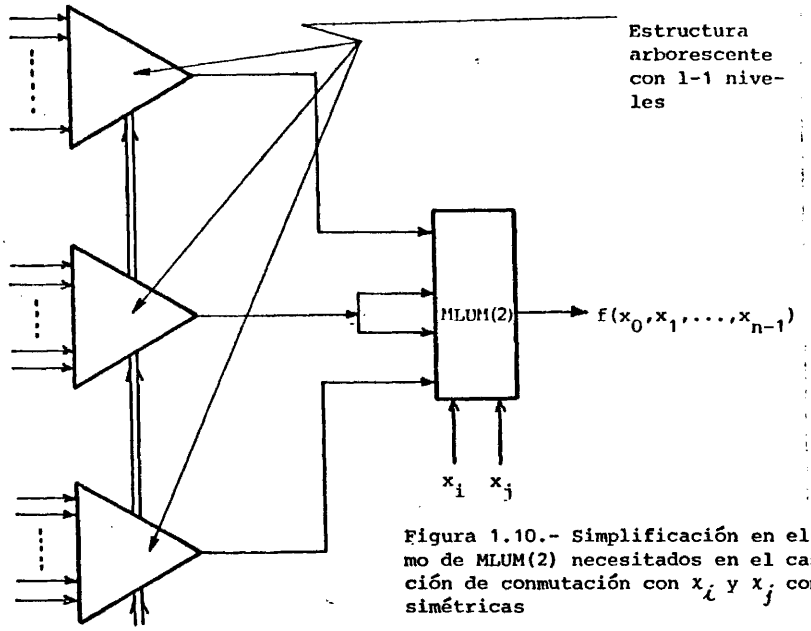


Figura 1.10.- Simplificación en el número máximo de MLUM(2) necesitados en el caso de una función de conmutación con x_i y x_j como variables simétricas

Si $p=1$, entonces 2 niveles y 3 MLUM(1) se requieren para efectuar la expansión respecto a las dos variables simétricas y, por tanto, la cota superior sobre el número de MLUM(1) resulta:

$$N_{max} = \sum_{i=0}^{\ell-1} 2^i - \frac{1}{4}\left(\sum_{i=0}^{\ell-1} 2^i - 3\right) = \frac{3k^\ell - k - 2}{4(k-1)} \quad \text{MLUM(1)} \qquad (1.15)$$

con $k=2$

Para cada variable simétrica adicional, más funciones residuos se hacen idénticas y, en consecuencia, se reduce el número de MLUM(p) necesitados para realizar la función.

Lema 1.1.- Para toda función simétrica f con respecto a p variables, si se expande respecto a estas variables, únicamente existen $p+1$ funciones residuos diferentes.

Demostración: Para probar este Lema, consideremos la p-tupla de dígitos binarios $(i_{p-1}, i_{p-2},\ldots,i_0)$, $i_j \in [0,1]$. Las 2^p p-tuplas distintas se pueden particionar en $p+1$ clases de equivalencia distintas, cada una con un número diferente de 0 y 1 bajo la operación de permutación.

Consideremos la función residuo $f(x_{n-1},\ldots,x_p, i_{p-1},\ldots, i_j,\ldots,i_k,\ldots,i_0)$ respecto a la expansión de sus p variables simétricas $x_0, x_1,\ldots, x_j,\ldots,x_k,\ldots,x_{p-1}$. Si se permutan i_j e i_k se obtiene $f(x_{n-1},\ldots, x_p, i_{p-1},\ldots, i_k,\ldots, i_j,\ldots, i_0)$

Si estas dos funciones residuos no son iguales, entonces $x_j^{i_j}$ y $x_k^{i_k}$ no pueden ser simétricas (en contra de las hipótesis). Esta permutación se puede efectuar para dos variables cualesquiera del conjunto de p variables simétricas y debe de resultar siempre en funciones residuos iguales. Esto implica que hay solamente una función residuo distinta para cualquier conjunto de funciones residuos con M unos en el conjunto de variables de expansión. Como esto corresponde a la partición efectuada, existirán únicamente $p+1$ funciones residuos para cualquier expansión respecto a p variables simétricas.

Definición 1.7.- Una función de conmutación $f(x_{n-1}, x_{n-2},\ldots,x_0)$ se dice que es totalmente simétrica respecto a sus variables $x_{n-1},\ldots,x_0$, si permanece invariable para cualquier permutación de las mismas.

Por ejemplo la función $f(x_2,x_1,x_0) = x_1x_0 + x_2x_1 + x_2x_0$ es totalmente

simétrica. Como consecuencia de este Lema, se deduce el siguiente teorema.

Teorema 1.1.- Una realización raborescente mediante MLUM(p) de una función totalmente simétrica requiere a lo más:

$$\sum_{i=0}^{\ell-1} (p.i+1) = \ell + \frac{\ell(\ell-1).p}{2} \quad MLUM(p) \qquad (1.16)$$

Siendo ℓ el número de niveles en la realización, tal como refleja (1.9) y su árbol mínimo es independiente del orden en el cual las variables se utilizan como entradas de control.

Demostración.- Se necesita un MLUM(p) para la función de salida. A continuación se requieren $p+1$ MLUM(p) para la expansión respecto a p variables, y $2p+1$ MLUM(p) cuando se expansionan $2p$ variables.

Este proceso continúa hasta que la función se expande respecto a $n-1$ variables, existiendo $(n-1)+1$ entradas de datos diferentes. Debe observarse que, debido a la simetría total, el orden en el cual se utilizan en el proceso de expansión las variables no afecta nada a las propiedades de las funciones residuos excepto intercambiar las variables no expandidas.

Definición 1.8.- La función dual de una función de conmutación $f(x_{n-1}, x_{n-2}, \ldots, x_0)$ se define como $\bar{f}(\bar{x}_{n-1}, \bar{x}_{n-2}, \ldots, \bar{x}_0)$. Representamos la función dual por $f^d(x_{n-1}, x_{n-2}, \ldots, x_0)$. En particular si $f(x_{n-1}, x_{n-2}, \ldots, x_0) = f^d(x_{n-1}, x_{n-2}, \ldots, x_0)$ entonces se dirá que la función $f(x_{n-1}, \ldots, x_0)$ es auto-dual.

Hemos visto ya que el árbol mínimo de MLUM(p) de una función dual, se puede determinar fácilmente de la realización mínima de la función original, ya que pertenecen a la misma clase de equivalencia n-p-n. La clase de las funciones auto-duales, pueden utilizar este hecho para simplificar el procedimiento de su diseño.

Toda función auto-dual se puede escribir como:

$$f^{a.d}(x_{n-1}, \ldots, x_i, \ldots, x_0) = x_i\, f(x_{n-1}, \ldots, x_{i+1}, x_{i-1}, \ldots, x_0) + \bar{x}_i\, f^d(x_{n-1}, \ldots, x_{i+1}, x_{i-1}, \ldots, x_0) \qquad (1.17)$$

De (1.17) se puede ver que si se tiene una realización óptima para $f(x_{n-1},\ldots,x_{i+1},x_{i-1},\ldots,x_0)$, entonces $f^{a.d}(x_{n-1},\ldots,x_0)$ se puede realizar económicamente utilizando el mismo orden de expansión de f para implementar f^d. Aunque esto no garantiza una realización mínima para $f^{a.d}$ se obtiene un ahorro sustancial en el tiempo de expansión utilizando esta propiedad.

Ejemplo 1.5.- Se desea sintetizar la función $f(x_3,x_2,x_1,x_0) = \sum (3,4,6,7,10,13,14,15)$. Si en el primer nivel se expande respecto a la variable x_3 resulta:

$$f(x_3,x_2,x_1,x_0) = \bar{x}_3\, g_1(x_2,x_1,x_0) + x_3\, g_2(x_2,x_1,x_0)$$

donde:

$$g_1(x_2,x_1,x_0) = x_1\, x_0 + x_2\, \bar{x}_0$$

$$g_2(x_2,x_1,x_0) = x_2\, x_0 + x_1\, \bar{x}_0 = \bar{g}_1(\bar{x}_2,\bar{x}_1,\bar{x}_0)$$

Así pues g_2 y g_1 son funciones duales, cuya síntesis requiere un único MLUM(1) y, por lo tanto, el coste de implementar f es de 3 MLUM(1), tal como refleja la Figura 1.11.

1.4.3.- Propiedades generales de las estructuras generalizadas

Sin pérdida de generalidad, en este apartado se van a comparar las estructuras generalizadas y arborescentes realizadas con MLUM(1). Como ya se sabe, toda entrada de datos a un MLUM(1) en una estructura arborescente es una función residuo.

Para un MLUM(1) que tenga como entrada de control la función g_1, la salida se puede escribir:

$$f(X) = g_1(X')\, h_1(X'') + \bar{g}_1(X')\, h_2(X''') \qquad (1.18)$$

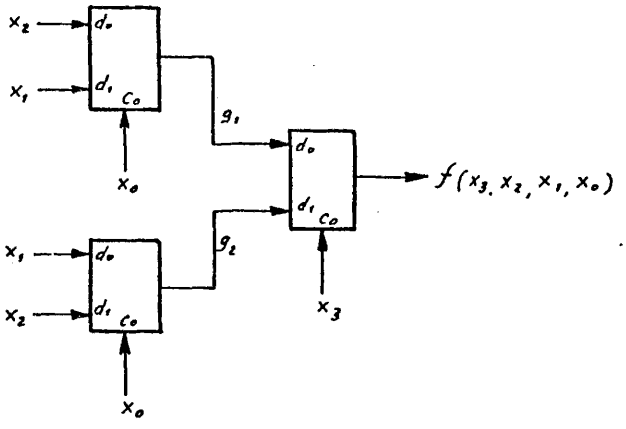


Figura 1.11.- Síntesis de la función autodual $f(x_3,x_2,x_1,x_0) = \sum(3,4,6,7,10,13,14,15)$

donde $X = (x_{n-1},x_{n-2},\ldots,x_0)$, X', X'' y $X''' \subseteq X$ y g_1, h_1 y h_2 son funciones arbitrarias. Ver Figura 1.12.

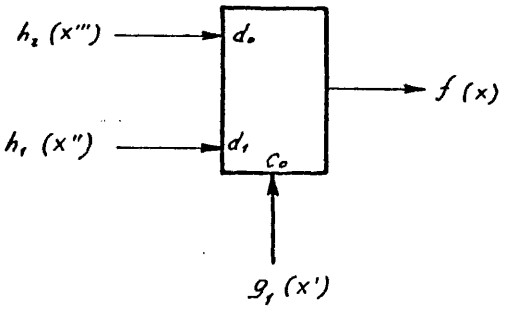


Figura 1.12.- $f(X) = g_1(X')\ h_1(X'') + \bar{g}_1\ (X')\ h_2\ (X''')$

Como únicamente se va a utilizar g_1 como entrada de control, vamos a considerar dos casos en función de la relación que existe entre los argumentos de g_1 y aquéllos de h_1 y h_2.

En el primero, el conjunto X' será disjunto de X'' y X''', es decir:

$$X' \cap (X'' \cup X''') = \emptyset$$

mientras que en el segundo no es disjunto

$$X' \cap (X'' \cup X''') \neq \emptyset$$

Primer caso: $X' \cap (X'' \cup X''') = \emptyset$ y $X' \cup X'' \cup X''' = X$

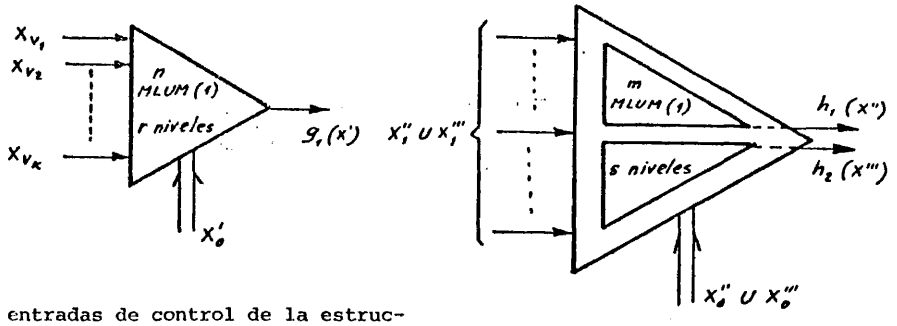
Teorema 1.2.- Sea la ecuación (1.18) con subfunciones disjuntas. Si se conoce que:

a) $g_1(X')$ se puede realizar óptimamente en una estructura arborescente con n MLUM(1), con un retardo de r niveles y que existen k variables simples utilizadas como entradas de datos.

b) Para las subfunciones h_1 y h_2, su combinación puede realizarse óptimamente en una estructura arborescente con m MLUM(1) y con un retardo de s niveles. Entonces $f(X)$ se puede realizar en una estructura arborescente de MLUM(1) con una cota superior de $n + m + k$ MLUM(1) y con un retardo máximo de $r + s + 1$ niveles (Figura 1.13).

Sin embargo, la estructura generalizada que utiliza $g_1(X')$ como entrada de control (Figura 1.14) requiere a lo más únicamente $n + m + 1$ MLUM(1) para realizar $f(X)$ y con un retardo máximo de $1 + max\ (r,s)$ niveles, donde $max(r,s)$ representa el valor máximo de r o s.

Demostración.- Cuando $f(X)$ se expande respecto a una variable en X', podemos escribir:

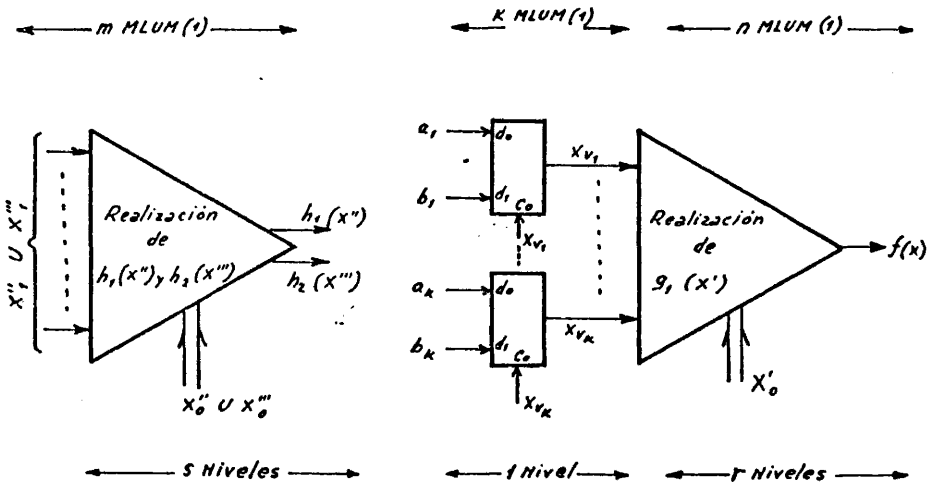


X'_0 - entradas de control de la estructura arborescente óptima de $g_1(X')$

X'_1 - $(x_{v_1} \ldots x_{v_k})$ → entradas de datos

$X' = X'_0 \cup X'_1$

$X'' \cup X''' = (X''_0 \cup X'_0{}'') \cup (X''_1 \cup X'_1{}'')$



$$a_i,\ b_i \in [0,1,h_1(X''),\ h_2(X'''')] \qquad i = 1,\ldots,k$$

Figura 1.13.- Realización de $f(X)$ mediante una estructura arborescente, conocidas las estructuras arborescentes de $g_1(X')$, $h_1(X'')$ y $h_2(X'''')$.

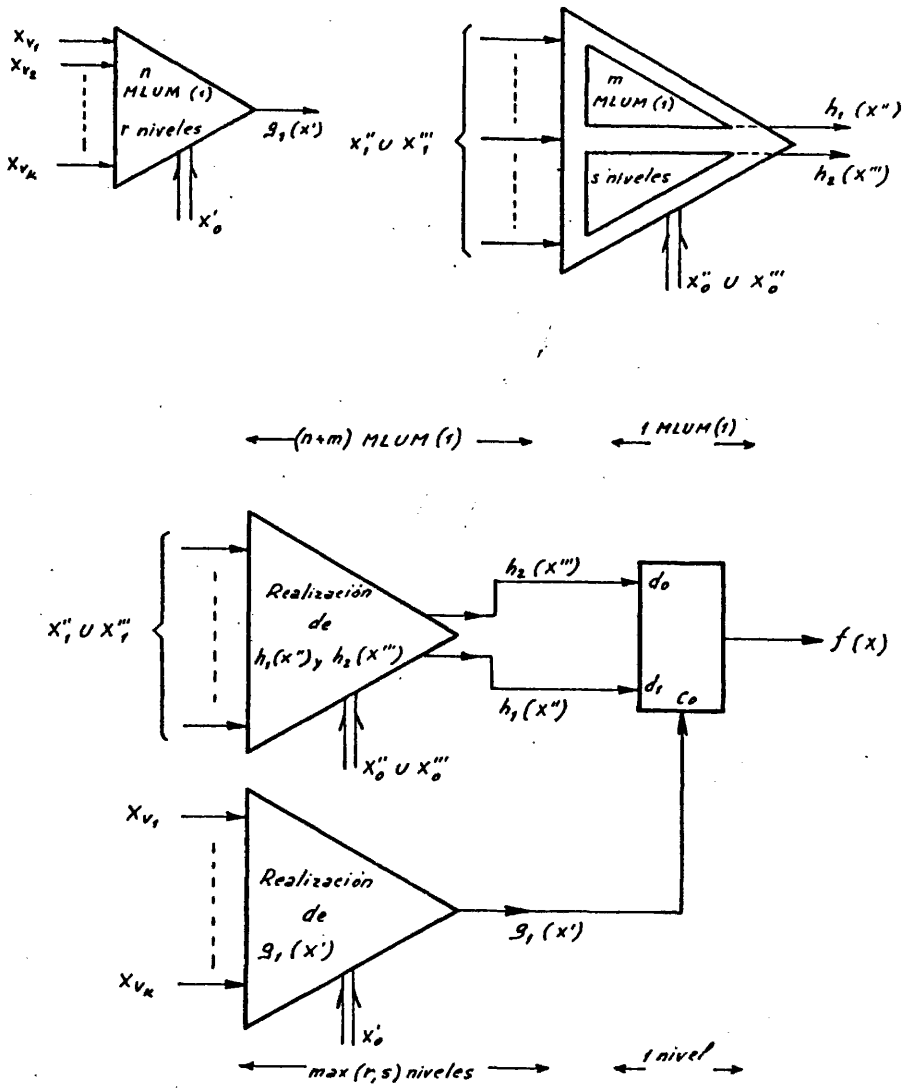


Figura 1.14.- Realización de $f(X)$ mediante una estructura generalizada, conocidas las estructuras arborescentes óptimas de $g_1(X')$, $h_1(X'')$ y $h_2(X''')$

$$f(X) = \bar{x}_i[g_1(X'_{i*})h_1(X'') + \bar{g}_1(X'_{i*})h_2(X''')] + x_i[g_1(X'_{i*})h_1(X'') + \bar{g}_1(X'_{i*})h_2(X''')] \quad (1.19)$$

donde X'_{i*} es igual al conjunto X' con 0 ó 1 sustituido para $x_i \in X'$. Si la expansión en (1.19) se repite para las variables en X', hasta que cada función residuo de $g_1(X')$ (es decir $g_i(X'_{i\ j\ \ldots\ n})$) sea una constante lógica o una función de variable simple, existen cuatro posibilidades para las cantidades entre corchetes ([ ]) en la expansión.

$g_1(X'_{i\ j\ \ldots\ n})$ puede ser igual a 0, 1, x_v o $\bar{x}_v$ donde $x_v \in X'_{i\ j\ \ldots\ n}$.

Consideremos los siguientes términos para cada uno de estos cuatro casos:

$$g_1(X'_{i\ j\ \ldots\ n})\ h_1(X'') + \bar{g}_1(X'_{i\ j\ \ldots\ n})\ h_2(X''') \quad (1.20)$$

En una realización arborescente de $f(X)$, la función (1.20) representa una entrada de datos a un MLUM(1). Si $g_1(X'_{i\ j\ \ldots\ n}) = 0$ entonces esa entrada de datos se puede reemplazar por $h_2(X''')$. Si $g_1(X'_{i\ j\ \ldots\ n}) = 1$, la entrada de datos sería $h_1(X'')$. Por tanto,ningún MLUM(1) adicional se necesita antes de usar X'' y X''' en la secuencia de expansión. Si $g_1(X'_{i\ j\ \ldots\ n})$ es igual a x_v o $\bar{x}_v$, entonces la entrada de datos al MLUM(1) es: $x_v\ h_1(X'') + \bar{x}_v\ h_2(X''')$ o $\bar{x}_v\ h_1(X'') + x_v\ h_2(X''')$ respectivamente. Por tanto, un MLUM(1) adicional se necesita para la expansión respecto a x_v antes de utilizar las variables X'' y X''' como variables de control. Como la secuencia de expansión mínima para $g_1(X'')$ requería n MLUM(1) con k variables simples utilizadas como entradas de datos, $n + k$ MLUM(1) se utilizan en expansionar $f(X)$ respecto a X'. Como $h_1(X'')$ y $h_2(X''')$ requieren m MLUM(1) más, entonces $n + m + k$ MLUM(1) son suficientes para realizar $f(X)$.

Debe de observarse que este número no es necesariamente el mínimo para $f(X)$. Algunas expansiones, utilizando alternativamente X', X'' y X''', pueden dar mejores resultados. Sin embargo, el método reseñado da una cota superior.

Si se emplea $g_1(X')$ como entrada de control, tal como muestra la Figura 1.14, se requiere únicamente $n + m + 1$ MLUM(1) para generar $f(X)$. Esta cantidad se deduce fácilmente porque n MLUM(1) implementan $g_1(X')$, mientras que m MLUM(1) realizan $h_1(X'')$ y $h_2(X''')$. Un MLUM(1) más se necesita para combinar las tres subfunciones en $f(X)$. Como $g_1(X')$ es disjunto de $h_1(X'')$ y $h_2(X''')$, el valor $(n + m + 1)$ es el mínimo para realizar la estructura generalizada de $f(X)$, supuesto que no exista ninguna descomposición adicional para ninguna de las subfunciones.

Si se utilizan, en primer lugar, como variables de expansión X', entonces $f(X)$ presenta un retardo de $(\lambda + \delta + 1)$ módulos en su realización arborescente, ya que $g_1(X')$ requiere λ módulos y un nivel más es necesario para combinar las funciones residuos de variable simple de $g_1(X')$ con $h_1(X'')$ y $h_2(X''')$. No obstante, cuando se usa $g_1(X')$ como entrada de control en la estructura generalizada, hay $1 + \max(\lambda, \delta)$ niveles de retardo. Esto es obvio puesto que un nivel se precisa para combinar g_1, h_1 y h_2 en f, mientras que λ niveles son necesarios para g_1 y δ niveles para h_1 y h_2. La Figura (1.14) ilustra lo dicho en este teorema.

Ejemplo 1.6.- Se desea sintetizar la función:

$$f(x_5, x_4, x_3, x_2 x_1, x_0) = \sum (8,9,10,11,12,14,15,18,19,22,23,26,27,29,30,31,42,43,45,46,47,50,51,54,55,56,57,58,59,60,62,63)$$

$f(X)$ admite una descomposición del tipo (1.18) con:

$$X = (x_5, x_4, x_3, x_2, x_1, x_0); \quad X' = (x_5, x_2, x_0); \quad X'' \cup X''' = (x_4, x_3, x_1)$$

$$g_1(X') = x_5 \bar{x}_2 + (\bar{x}_5 x_0 + x_5 \bar{x}_0) x_2$$

$$h_1(X'') = x_4 x_3 \bar{x}_1 + \bar{x}_4 x_3 x_1 + x_4 x_1$$

$$h_2(X''') = x_3 \bar{x}_4 + x_1 x_4$$

Las estructuras arborescentes óptimas de g_1, h_1 y h_2 se dan en la Figura 1.15a, b, c. Siguiendo el procedimiento reseñado, en la Figura 1.15d, se obtiene una realización de $f(X)$ mediante una estructura arborescente.

$$f(X) = x_5\bar{x}_2 \cdot h_1(X'') + \bar{x}_5 x_2 x_0\ h_1(X'') + x_5 x_2 \bar{x}_0\ h_1(X'') + \bar{x}_5\bar{x}_2\ h_2(X''') + \bar{x}_5 x_2 \bar{x}_0\ h_2(X''') + x_5 x_2 x_0\ h_2(X''')$$

La estructura generalizada que corresponde a la descomposición (1.18), en este caso, se da en la Figura 1.15e. Se observa que existe un ahorro de 2 módulos respecto a la síntesis mediante una estructura arborescente. En el próximo capítulo, cuando hayamos desarrollado nuevos procedimientos de síntesis, efectuaremos una implementación aún más económica que la de la Figura 1.15e.

Segundo caso.- $X' \cap (X'' \cup X''') \neq \emptyset$ y $X' \cup X'' \cup X''' = X$. En este caso es difícil cuantificar la posible reducción frente a las estructuras arborescentes.

Teorema 1.3.- Supuesto que $X' \cap (X'' \cup X''') = X^0 \subseteq X$ y bajo las mismas hipótesis del Teorema (1.2) respecto al número de módulos y al de niveles en la realización de los árboles mínimos de $g_1(X')$, $h_1(X'')$ y $h_2(X''')$ entonces $f(X)$ se puede realizar en una estructura arborescente de MLUM(1) con un número de módulos siempre menor que $n + m + k$ y un retardo menor que $\lambda + \delta + 1$ niveles.

Demostración.- Supongamos que $x_i \in X^0$ y efectuemos una expansión de $f(X)$ respecto a x_i

$$f(X) = \bar{x}_i\left[g_1(X'_{i*})h_1(X''_{i*}) + \bar{g}_1(X'_{i*})h_2(X'''_{i*})\right] + x_i\left[g_1(X'_i)h_1(X'') + \bar{g}_1(X'_i)h_2(X'''_i)\right] \qquad (1.21)$$

donde X'_{i*}, X''_{i*} y X'''_{i*} significa que $x_i \in X' \cap (X'' \cup X''')$ ha sido reemplazada por 0 ó 1.

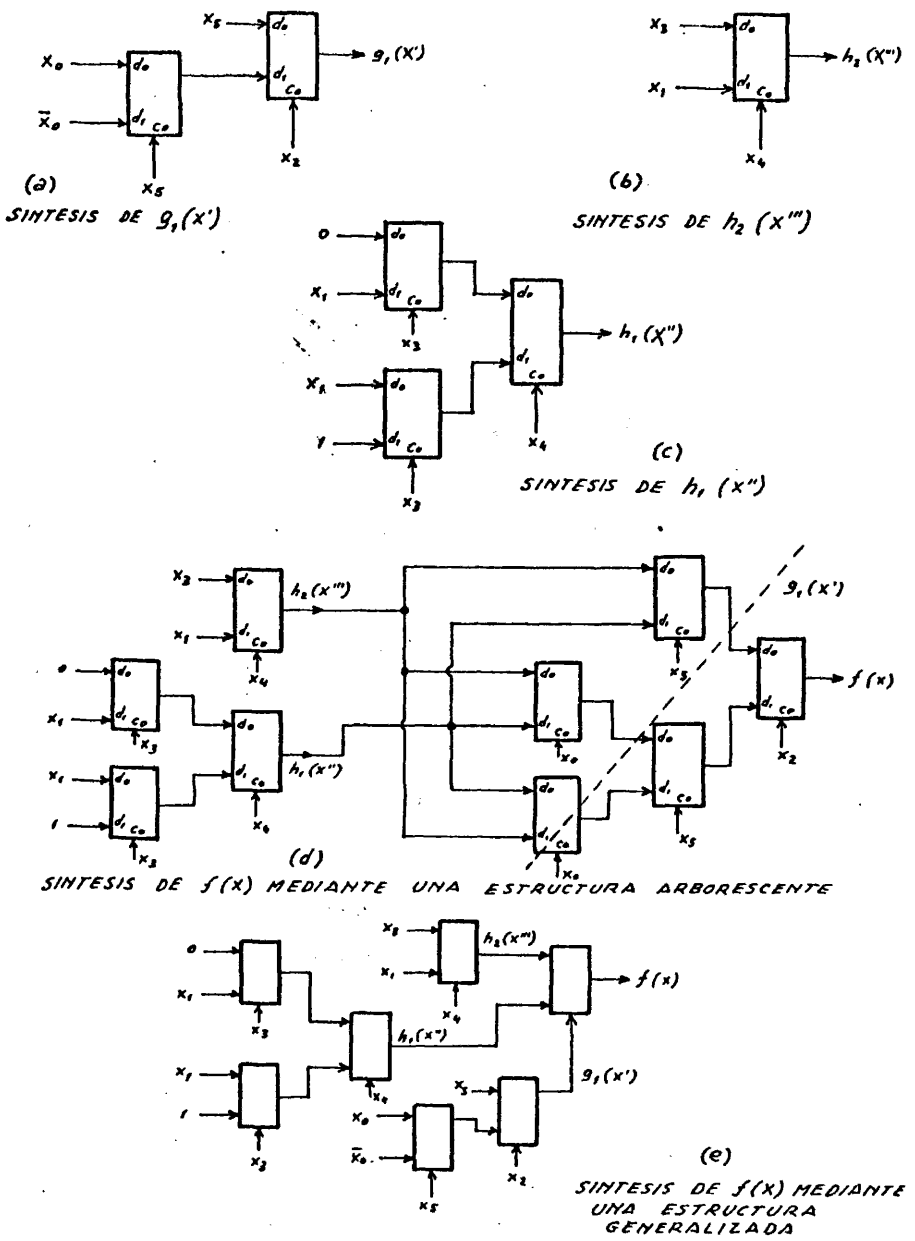


Figura 1.15.- Síntesis de la función

$$f(x_5,x_4,x_3,x_2,x_1,x_0)=\sum\{8,9,10,11,12,14,15,18,19,22,23,26,27,29,30,31,42,43,45,46,47,50,51,54,55,56,57,58,59,60,62,63\}$$

Una vez que todas las variables en el conjunto X' han sido expandidas, únicamente quedan funciones residuos de $h_1(X'')$ y $h_2(X'')$. Por tanto, mientras que m MLUM(1) se necesitan para realizar $h_1(X'')$ y $h_2(X'')$, menos de m módulos son necesarios para implementar las funciones residuos restantes. Esto implica que el número de MLUM(1) necesitados en la estructura arborescente de $f(X)$ para el caso no disjunto es menor que $n + m + k$ MLUM(1). También, por consiguiente, el retardo será inferior a $r + s + 1$ niveles de MLUM(1).

Para la estructura generalizada existe la probabilidad de que las realizaciones arborescentes mínimas de g_1 y la combinación de h_1 y h_2 tengan entradas de datos idénticas. Si $g_1(X')$, $h_1(X'')$ y $h_2(X'')$ se expanden en un orden apropiado, sus funciones residuos pueden ser funciones de X^0 únicamente. Por tanto, alguna de estas funciones residuos y sus correspondientes entradas de datos podrían ser iguales. Si esto ocurre, entonces la realización de f mediante una estructura generalizada en forma no disjunta requiere menos de $n + m + 1$ MLUM(1) y un retardo menor que $1 + max(r, s)$ niveles.

1.4.4.- Sobre el número de estructuras generalizadas

Un problema de tipo combinatorial muy importante que se puede plantear es el de conocer el número de estructuras diferentes que existen para un número dado de módulos lógicos universales. Es decir con q MLUM(p) ¿cuántas estructuras se pueden formar?.

Sea $N_p(q)$ este número. Un MLUM(p) tiene $(p + 2^p)$ terminales de entrada (p entradas de control y 2^p entradas de datos).

El problema, pues, se reduce a determinar el número de árboles $(p + 2^p)$-arios que se pueden tener con q nodos (Ver Knuth ([20])).

Así pues:

$$N_p(q) = \binom{1+(2^p+p)q}{q} \frac{1}{1+(2^p+p)q} = \binom{(2^p+p)q}{q} \frac{1}{(2^p+p-1)q+1} \qquad (1.22)$$

La Figura 1.16 nos muestra, en una tabla de doble entrada el valor de $N_p(q)$ para distintos valores de los parámetros p y q. Asimismo, en la Figura 1.17 están representados para los casos de $p=1$ con $q=1,2,3,4$ las diferentes estructuras que se pueden conseguir.

p ↓ \ q →	1	2	3	4	5
1	1	3	12	55	273
2	1	6	51	506	5481
3	1	11	176	3311	68211
4	1	20	590	20540	784245

Figura 1.16.- $N(p,q)$= número de estructuras diferentes con q módulos del tipo MLUM(p)

Del examen de estas estructuras se observa que, una vez fijado p y q, el retardo que representamos por r, de la estructura es variable.

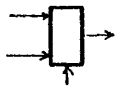
Definición 1.9.- Se denomina retardo de una estructura el número máximo de MLUM (p) que es necesario atravesar para ir desde la entrada a la salida.

Teorema 1.4.- Dados p y q, el número máximo de retardos que puede presentar una estructura es $r_{max} = q$ y el número mínimo es $r_{min} = \lceil log_P[P-1]q+1] \rceil$ donde $P = 2^p + p$ y $\lceil x \rceil$ representa el menor entero mayor que x.

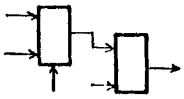
Demostración.- El valor de r_{max} se deduce trivialmente, si se tiene en cuenta que

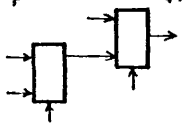
MLUM (1)

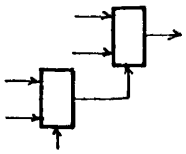
$q = 1 \Rightarrow N(q) = 1$



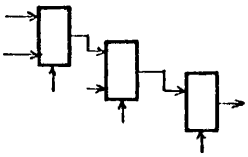
$q = 2 \Rightarrow N(q) = 3$

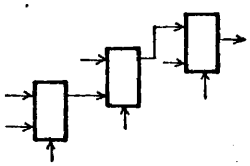


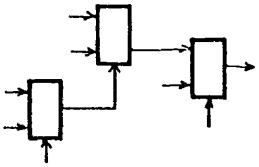


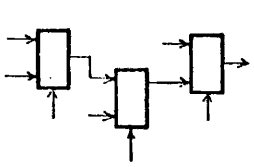


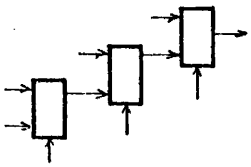
$q = 3 \Rightarrow N(q) = 12$

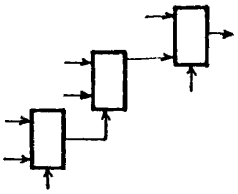


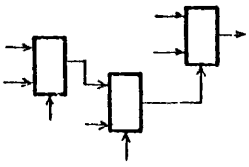


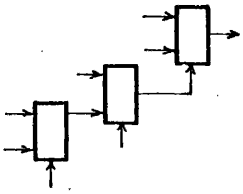


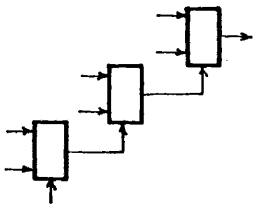


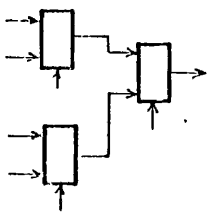


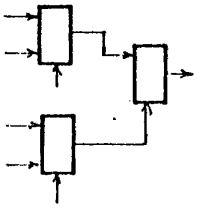


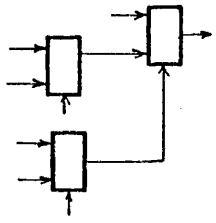




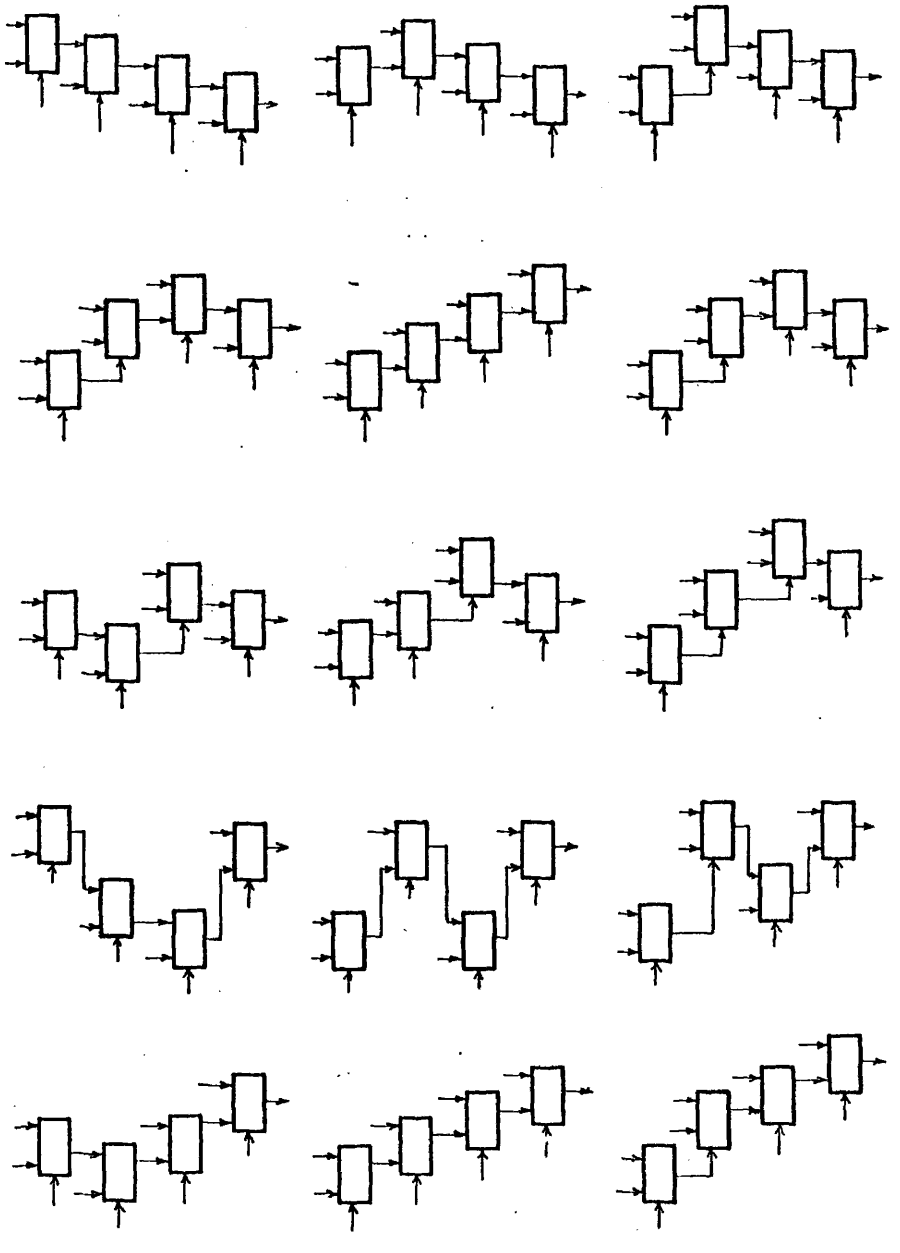


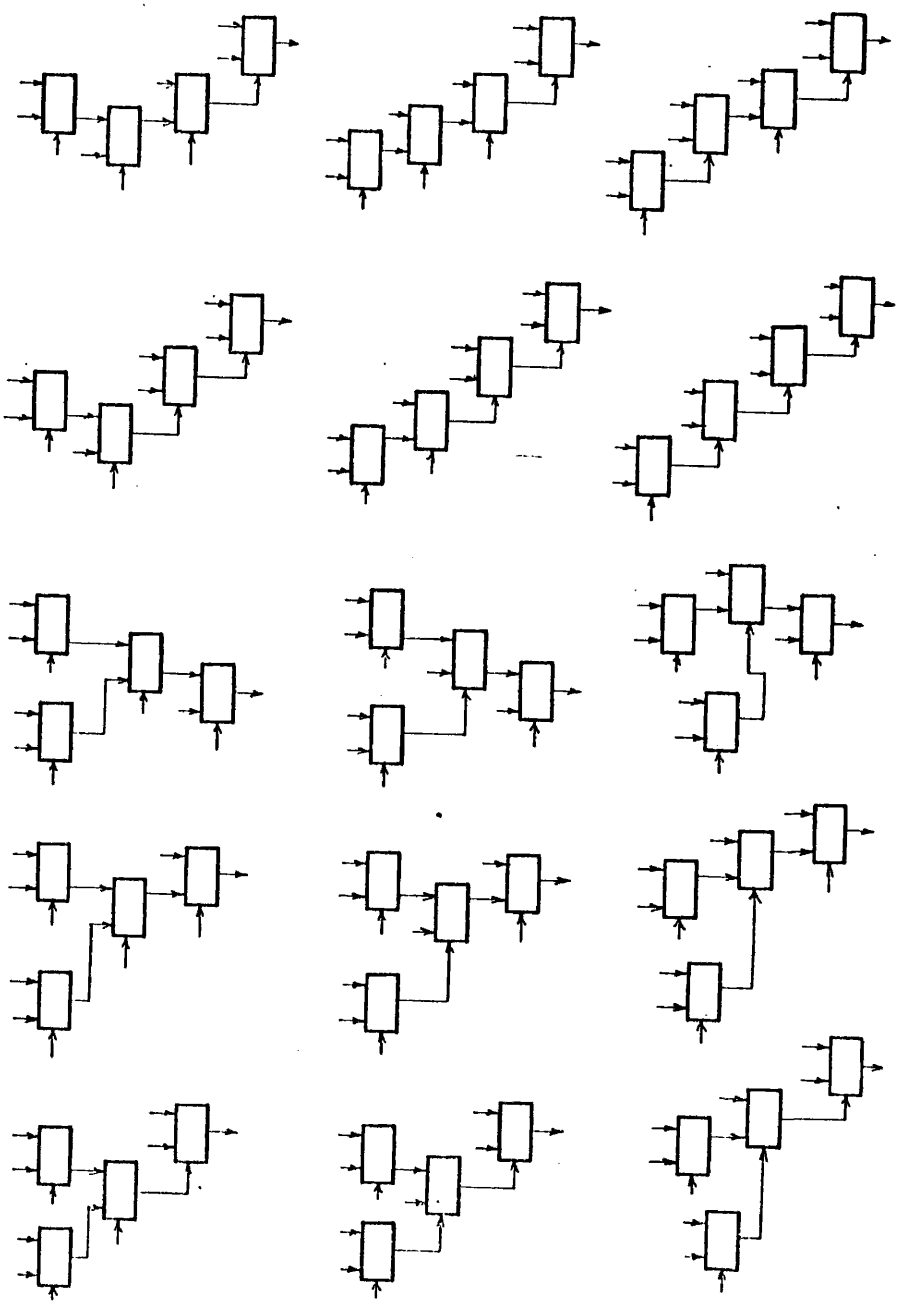


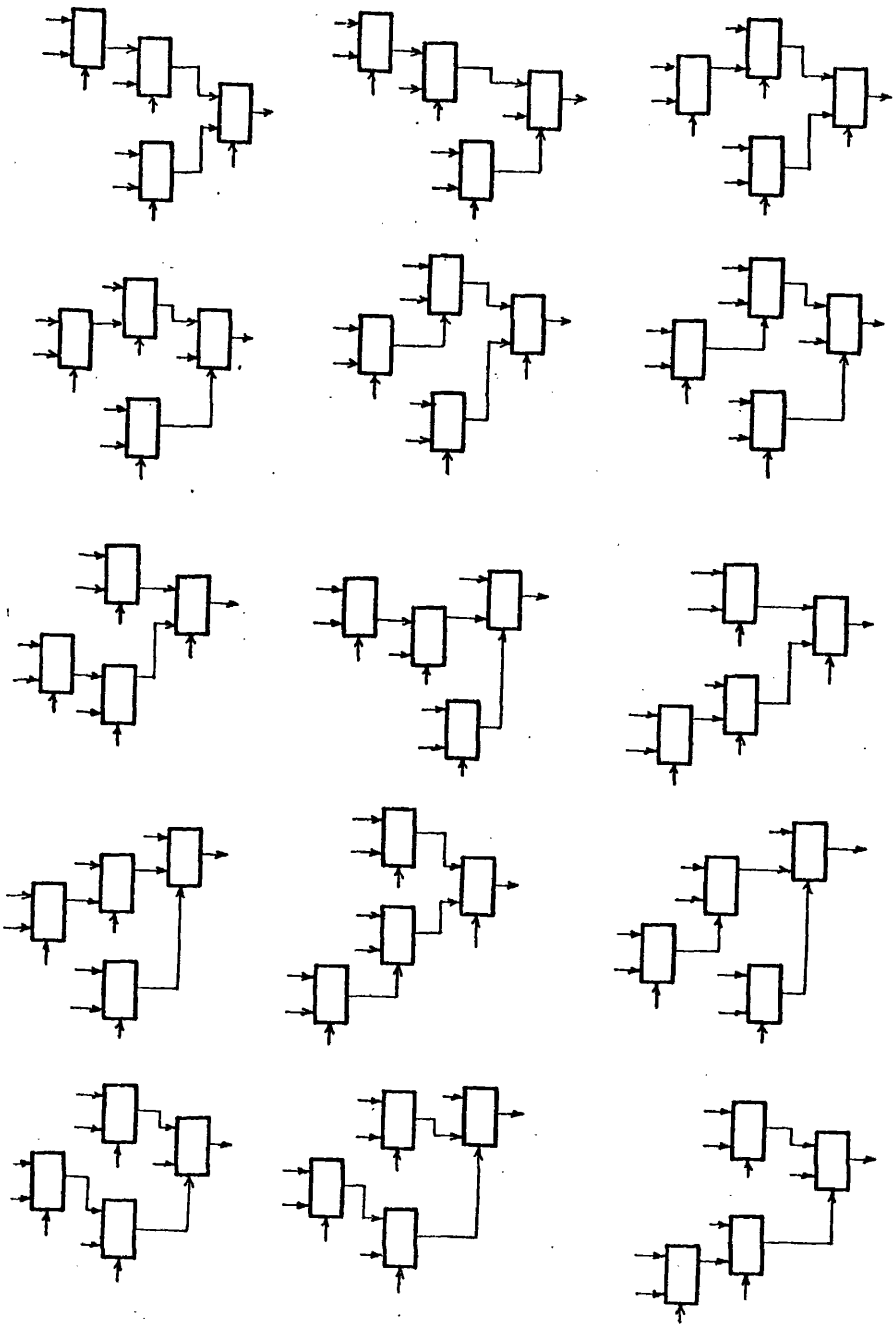




$q = 4 \Rightarrow N(q) = 55$







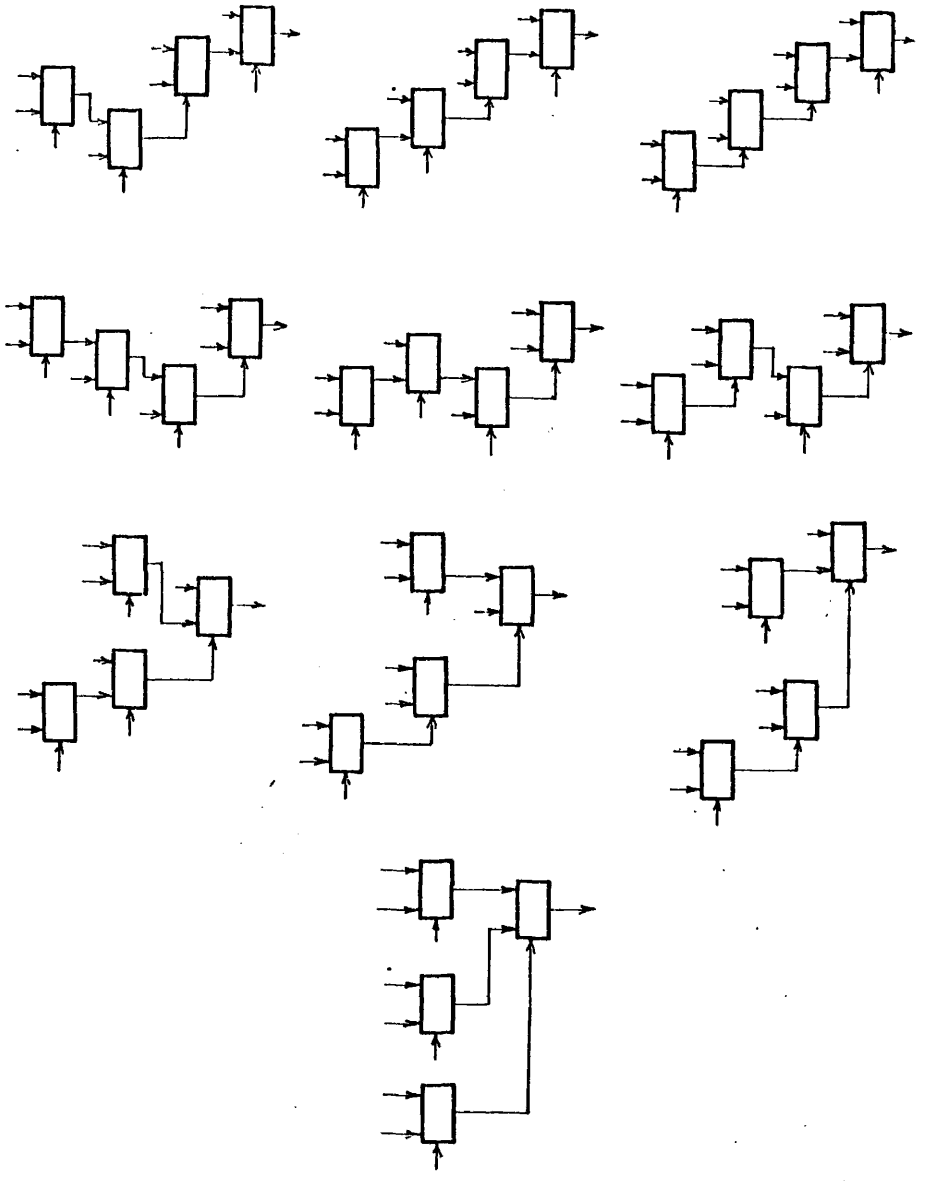


Figura 1.17.- Catálogo de todas las estructuras posibles con MLUM(1) para $q = 1,2,3,4$.

con q elementos, el máximo número de retardos se conseguirán disponiendo los q MLUM(p) de manera que se utiliza uno solo por nivel.

El número de estructuras generalizadas que presentan un retardo máximo viene dado por P^q (variaciones con repetición de P elementos tomados de q en q).

Para deducir el valor de r_{min} conviene darse cuenta que el número de niveles será mínimo cuando en todos los niveles anteriores (se empieza a contar niveles desde el terminal de salida de la estructura), no existe ningún terminal de entrada libre.

Con q módulos se establece la siguiente acotación:

$$1 + P +\ldots+ P^{r_{min}-2} < q \leq 1 + P +\ldots+ P^{r_{min}-1} = \frac{P^{r_{min}}-1}{P-1} \qquad (1.23)$$

De (1.23) se deduce $(P-1)\ q + 1 \leq P^{r_{min}}$

De donde tomando logaritmo en base P

$$r_{min} = \lceil \log_P (P-1)\ q + 1 \rceil \qquad (1.24)$$

Tomando como base el número de estructuras generalizadas efectuamos la siguiente conjetura.

CONJETURA: Dada una estructura generalizada cualquiera, le corresponde al menos una clase de equivalencia n-p-n para la cual es óptima al requerir el menor número posible de MLUM(p) para sintetizar su función representativa.

El problema, pues, se reduciría a encontrar un procedimiento sistemático que nos permitiese encontrar la estructura óptima (o las estructuras óptimas) que corresponden a una clase de equivalencia n-p-n dada.

Esto nos llevaría, por ejemplo, al hecho de que para sintetizar cualquier función de 4 variables con MLUM(1), únicamente se necesitarían 5 módulos ($N_1(5)=273$,

y el número de clases de equivalencia n-p-n es de 222), en lugar de los siete previsibles mediante la estructura arborescente cuando en ésta no se puede eliminar ningún módulo.

En el capítulo siguiente, en el cual abordamos procedimientos de síntesis, veremos cómo aplicar estas ideas de manera que se consigan mejores resultados que los que cabría esperar utilizando únicamente estructuras arborescentes.

Conviene darse cuenta de la complejidad del problema, máxime si tenemos presente que en nuestra exposición, sobre el número de estructuras diferentes, no se ha incluido la posibilidad de que la salida de un MLUM(p) en particular pueda conectarse a más de una entrada (sea de datos o de control) de niveles anteriores.

REFERENCIAS

(1) Mc CLUSKEY, E.J., "Introduction to the theory of switching circuits", Mc Graw-Hill, 1965.

(2) KOH, K.S., "A minimization technique for TANT Networks", IEEE Trans. on Computers, Vol. C-20, January 1971, pp. 105-107.

(3) IBARAKI, T. and MUROGA, A., "Synthesis of networks with a minimum number of negative gates", IEEE Trans. on Computers, Vol. C-20, January 1971, pp.49-58.

(4) DIETMEYER, D.L. and SU, Y.H., "Logic design automation of fan-in limited NAND networks", IEEE Trans. on Computers, Vol. C-18, January 1969, pp. 11-22.

(5) SU, S.Y. and NAM, C.W., "Computer aided Synthesis of multiple output multi-level NAND Networks with fan in and fan out constraints", IEEE Trans. on Computers, Vol. 20, Dec. 1971, pp. 1445-1455.

(6) DAVIDSON, E.S., "An algorithm for NAND decomposition under network constraints", IEEE Trans. on Computers, Vol. C-18, Dec. 1969, pp. 1098-1109.

(7) MUROGA, S. and IBARAKI, T., "Design of optimal switching networks by integer programming", IEEE Trans. on Computers, Vol. C-21, June 1972, pp. 573-582.

(8) SHANNON, C.E., "The synthesis of two terminal switching circuits", Bell System Technical Journal, Vol. 28, Jan. 1949, pp. 59-98.

(9) YAU, S.S. and TANG, C.K., "Universal logic circuits and their modular realizations", AFIPS, Conf. Proc. Vol. 32, 1968, pp. 297-305.

(10) PREPARATA, F.P., "Universal logic modules of a new type", IEEE Trans. on Computers, Vol. C-20, April 1971, pp. 418-423.

(11) FORSLUND, D.C. and WAXMAN, R., "The universal logic block (ULB) and its application to logic design", Seventh annual symposium on Switching and automata Theory, Oct. 1966, pp. 236-250.

(12) KING, W.F., "The synthesis of multipurpose logic devices", Seventh annual symposium on switching and automata theory, Oct. 1966, pp. 227-235.

(13) YAU, S.S. and ORSIC, M., "Synthesis of Universal Logic modules", Proc. of the Third annual Princeton Conference on Information Science and Systems, 1969, pp. 498-501.

(14) PREPARATA, F.P. and MULLER, D.E., "Generation near-optimal universal boolean functions", Journal of Computer and Systems Sciences, Vol. 4, April 1970, pp. 93-102.

(15) PATT, Y.N., "Universal logic modules with still fewer external interconnections", Proc. Fourth Hawaii International Conference on System Science, Jan 1971, pp. 696-697.

(16) PREPARATA, F.P., "On the design of universal boolean functions", IEEE Trans. on Computers, Vol. C-20, April 1971, pp. 418-423.

(17) OSMAN, M.Y. and WEISS, C.D., "Universal base functions and modules for realizing arbitrary switching functions", IEEE Trans. on Computers, Sept. 1972, pp. 985-995.

(18) YAU, S.S. and TANG, C.K., "Universal logic modules and their applications", IEEE Trans. on Computers, Vol. C-19, Feb. 1970, pp. 141-149.

(19) CALDWELL, S.H., "Switching circuits and logical design", John Wiley, 1958.

(20) KNUTH, D.E., "The art of computer programming", Vol. 1 Addison-Wesley 1969.

CAPITULO II

SINTESIS DE CIRCUITOS COMBINACIONALES MEDIANTE LA UTILIZACION DE MODULOS LOGICOS UNIVERSALES (MULTIPLEXORES)

2.1.- INTRODUCCION

En el capítulo anterior se han estudiado las propiedades funcionales de las estructuras formadas por MLUM(p), pero sin concentrar nuestros esfuerzos en los procedimientos de síntesis que abordamos en este capítulo.

Nuestro objetivo ahora, es el de dar métodos sistemáticos que nos permitan dada cualquier función de conmutación $f(X)$, pasar a una estructura que la implemente con un coste mínimo (menor número de MLUM(p)).

En primer lugar y basado en una modificación del método de Quine-McCluskey, se obtienen condiciones necesarias y suficientes para la síntesis de una función de conmutación de n variables mediante un único MLUM(p) $(n > p + 1)$.

A continuación se analiza el procedimiento de síntesis para estructuras arborescentes propuesto por Voith ([1]). Un análisis crítico del mismo nos va a permitir plantear de manera formal la síntesis óptima de estructuras arborescentes de MLUM(p), como un problema de programación lineal entera 0-1. Para su resolución se emplea una variante del método de enumeración implícita de Glover ([2]). Asimismo se efectúa una generalización del método de Voith de manera que puedan tomarse en cuenta la existencia de términos inoperantes, en la función a sintetizar.

Partiendo del concepto de descomposición funcional de una función booleana se desarrolla un procedimiento muy directo y de naturaleza cuasióptima para la síntesis con MLUM(p). Sus características más sobresalientes son las siguientes:

1ª.- Permite la obtención, tanto de estructuras arborescentes como generalizadas. En el segundo caso puede dar lugar a una disminución

en el número de MLUM(p) necesitados cuando se le compara con la estructura arborescente óptima.

2ª.- La aplicación del método es muy rápida en tiempo de cálculo, cuando se implementan los algoritmos a que da lugar. De hecho, el método lo que hace es una optimación por niveles, eligiendo como variables del primer nivel aquéllas que dan lugar a una reducción máxima en el número de MLUM(p) necesitados en ese nivel, e iterando el procedimiento a los niveles sucesivos.

Utilizando este método se han conseguido los siguientes resultados:

1°) Dar las estructuras óptimas (arborescentes o generalizadas) para todas las clases de equivalencia n-p-n, de una función de conmutación de 3 variables sintetizadas con MLUM(1).

2°) La síntesis de cualquier función de 4 variables con MLUM(1) no requiere nunca utilizar la cota máxima de 7 módulos que sería imprescindible si únicamente se tomase en cuenta las estructuras arborescentes.

En particular, se observa que las descomposiciones no disjuntas juegan un papel fundamental en este tipo de problemas, tal como se desprende de los ejemplos que se presentan. Sin embargo, el problema en este punto continúa abierto y no se ha podido encontrar en general un método (que no sea el simplemente exhaustivo) que permita determinar la síntesis óptima con estructuras generalizadas para cualquier función de conmutación dada.

Finalmente, se amplían los resultados obtenidos a la síntesis de funciones múltiples, cuestión ésta que tampoco se había tratado previamente. Este punto nos permitirá introducir el concepto de descomposición múltiple simple disjunta como una generalización de la descomposición simple disjunta y que, como se verá, juega un papel importante en la síntesis de funciones múltiples.

2.2.- CONDICIONES PARA LA SINTESIS DE UNA FUNCION DE CONMUTACION MEDIANTE UN UNICO MLUM(p) ([3])

Es un hecho bien conocido ([4])([5])([6]) que un MLUM(p) se puede utilizar para implementar cualquier función de conmutación de n variables ($n > p$), eliminando p variables y teniendo que sintetizar 2^p funciones residuos de n-p variables.

Obviamente, en el caso en que $n=p+1$, la síntesis requiere únicamente 1 MLUM(p). Para este caso, recientemente ([7]) se ha desarrollado un procedimiento algorítmico para determinar las p variables que deben utilizarse como entradas de control, de manera que un máximo de las 2^p entradas de datos puedan ser las constantes 1 ó 0. Si $n > p + 1$, es posible utilizar las cartas de descomposición de Ashenhurst ([8]) en orden a determinar si existe, al menos, un conjunto de p entradas o variables de control, tales que las 2^p funciones residuos se puedan sintetizar directamente sin utilizar circuitería lógica adicional.

El objetivo de este apartado es demostrar que una solución, si existe, se puede calcular algorítmicamente aplicando, de forma combinada, el método de Quine-McCluskey, y un teorema que da condiciones necesarias y suficientes para obtener una solución sin necesidad de hacer un uso exhaustivo de todas las posibles cartas de descomposición de Ashenhurst formadas por n variables con p entradas de control al MLUM(p).

En lo que sigue, un implicante primo se representa por $p^i = \alpha^i\ (\beta^i)$ donde α^i es el conjunto de todos los minterms que forman p^i y el conjunto de números β^i indica los pesos de las variables que se eliminan en la formación de p^i.

Por ejemplo, para $n=5$ $\quad p=0,1,8,9\ (1,8)$

$$x_4 \to 16,\ x_3 \to 8,\ x_2 \to 4,\ x_1 \to 2,\ x_0 \to 1 \longrightarrow p = \bar{x}_4\bar{x}_2\bar{x}_1$$

Sea

$$\Lambda = \{1,2,\ldots,2^{n-1}\} \qquad (2.1)$$

y $c(a)$ = cardinal del conjunto a

Teorema 2.1.- Una condición necesaria para que una función de conmutación de n variables se pueda sintetizar mediante un único MLUM(p) es que el número de minterms de la función debe ser un múltiplo de 2^{n-p-1}.

Demostración.- Como se observa en la Tabla 2-1, cada una de las 2^p entradas de datos, puede recibir a su entrada los valores que se muestran y a los cuales corresponden el número de minterms que se indican:

valor en la entrada de datos	n°de minterms
1	2^{n-p}
0	0
variable complementada o no	2^{n-p-1}

TABLA 2-1.- Cálculo del número de minterms

De la Tabla 2-1, el Teorema es evidente.

Teorema 2.2.- Una función de conmutación f de n variables se puede sintentizar mediante un único MLUM(p) si se verifican las siguientes condiciones:

1) Existe un recubrimiento de la función

$$R(f) = \{p^1, \ldots, p^k\} \qquad \text{donde} \quad p^i = \alpha^i \ (\beta^i)$$

2) Existe $\Lambda_{n-p} \subset \Lambda$ .. con $c(\Lambda_{n-p}) = n-p$ tal que

a) $c(\Lambda_{n-p} \cap \beta^i) \geq n-p-1 \qquad i = 1,2,\ldots,k$

b) Si $\alpha^i \cap \alpha^j \neq \emptyset \qquad c(\Lambda_{n-p} \cap (\beta^i \cap \beta^j)) \geq n-p-1$

$$\forall\ i,j=1\ldots k,\ i \neq j$$

y los pesos de las variables que se deben utilizar como entradas de control son

$$\Lambda_p = \Lambda - \Lambda_{n-p} \qquad (2.2)$$

Demostración.- Con un único MLUM(p), las entradas de datos (funciones residuos), dependen de n-p variables. La condición a) garantiza que todos los implicantes primos que forman $R(\delta)$ no dependen del conjunto dado Λ_{n-p} en al menos $n-p-1$ variables.

En el caso de que la intersección de implicantes primos no sea vacia, la condición b) garantiza que debería eliminarse el mismo conjunto de $n-p-1$ variables.

Algoritmo para la determinación de Λ_p

El procedimiento es como sigue:

A-1.- Expresar la función que se desea implementar en su forma canónica de minterm y aplicar el método de minimización de Quine-McCluskey en su forma decimal.

Ejemplo 2.1.- Sea la función:

$$\delta(x_4, x_3, x_2, x_1, x_0) = \sum(0,1,4,7,8,9,12,15,18,19,21,23,24,26,28,29) \qquad (2.3)$$

El conjunto de implicantes primos de (2.3): $PI=\{E,Q\}$, donde:

E = conjunto de extremales de δ

$Q = PI-E$ = conjunto de implicantes primos que no son extremales.

Resulta:

$$E = \begin{cases} \Pi_1 = 0,1,8,9 & (1,8) \\ \Pi_2 = 0,4,8,12 & (4,8) \\ \Pi_3 = 7,15 & (8) \end{cases} \qquad Q = \begin{cases} \Pi_4 = 8,12,24,28 & (4,16) \\ \Pi_5 = 18,19 & (1) \\ \Pi_6 = 18,26 & (8) \\ \Pi_7 = 24,26 & (2) \\ \Pi_8 = 7,23 & (16) \\ \Pi_9 = 19,23 & (4) \\ \Pi_{10} = 21,23 & (2) \\ \Pi_{11} = 21,29 & (8) \\ \Pi_{12} = 28,29 & (1) \end{cases}$$

A-2.- Calcular $p = n\text{-}1\text{-}min\{c(\beta^i)\}$ $\forall\ \beta^i \in E$

En nuestro ejemplo $n=5$, $min\{c(\beta^i)\} = 1$ $p=3$

A-3.- Los términos extremales deben pertenecer necesariamente a $R(f)$, calcular

$\Omega = \{\Lambda^1_{n-p}, \ldots, \Lambda^\ell_{n-p}\}$ $\forall$ $\beta^i \in E$ de manera tal que las condiciones a) y b) del teorema se verifiquen.

En nuestro caso $\Omega = \{\Lambda^1_2, \Lambda^2_2\}$ donde $\Lambda^1_2 = \{1,8\}$ y $\Lambda^2_2 = \{4,8\}$

Pueden suceder los siguientes casos:

A-3.1.- $c(\Omega) = 0$ ir a A-4.

A-3.2.- $c(\Omega) = 1$

Entonces escoger del conjunto Q aquellos implicantes primos que satisfacen las condiciones definidas en el Teorema 2.2, con respecto a Λ^1_{n-p} y verificar si la función se recubre.

Si esto es cierto, calcular Λ_p de (2.2), si no ir a A-4.

A-3.3.- $c(\Omega) = \ell > 1$

Calcular

$$\Lambda' = \bigcap_{k=1}^{\ell} \Lambda^k_{n-p}$$

Seleccionar del conjunto Q aquellos implicantes primos que satisfacen $\beta^i = \Lambda'$ y eliminar aquellos que cumplen $\beta^i \cap \Lambda^k_{n-p} = \emptyset$ $k=1,\ldots,\ell$.

Una solución óptima se puede, entonces, definir como aquélla que presenta una menor carga lógica a las entradas de datos del MLUM(p) (mayor número de constantes lógicas). Si no, construir una tabla de implicantes primos con los minterms que quedan por cubrir y los implicantes primos de Q no considerados aún.

Para cada Λ^k_{n-p} $k = 1,\ldots,\ell$ seleccionar de la tabla construida, los implicantes primos que satisfacen las condiciones definidas por el Teorema 2.2, y verificar si se obtiene un recubrimiento para la función, si esto es cierto, determinar Λ^k_p.

Si las condiciones no se cumplen para ningún valor de k ir a A-4.

En nuestro ejemplo $c(\Omega) = 2$

$$\Lambda' = \{1,8\} \cap \{4,8\} = \{8\}$$

Por tanto, escogemos los implicantes primos Π_6 y Π_{11} y eliminamos Π_7, Π_8 y Π_{10} de Q. Como no se consigue aún el recubrimiento, se forma la siguiente tabla de implicantes primos

β	Π	19	23	24	28
4,16	Π_4			X	X
1	Π_5	X			
4	Π_9	X	X		
1	Π_{12}				X

Un recubrimiento no se obtiene con $\Lambda^1_2 = \{1,8\}$, sin embargo, con $\Lambda^2_2 = \{4,8\}$ los implicantes primos Π_4 y Π_9 efectúan un recubrimiento

de la función verificando al mismo tiempo las condiciones del Teorema 2.2.

Así pues, $\Lambda_3 = \{1,2,4,8,16\} - \{4,8\} = \{1,2,16\}$

Es decir, la función f se puede sintetizar con un único MLUM(3) y las entradas de control son x_4, x_1 y x_0.

A-4.- Si $p = n-2$ entonces el problema no tiene solución. Parar.

Si no $p = p + 1$ ir a A-3.

Entradas de datos al MLUM(p)

Los conjuntos Λ_p y Λ_{n-p} definen una partición simple disjunta del conjunto Λ de n variables.

La carta de descomposición de la partición dada $\Lambda_p|\Lambda_{n-p}$ viene representada por la matriz $C\,D_n(\Lambda_p|\Lambda_{n-p})$ de 2^p filas y 2^{n-p} columnas. Las entradas de la carta son los números decimales de los minterms para la combinación correspondiente y las filas corresponden a las funciones residuos. Para representar una función f en la carta marcamos con un círculo los minterms correspondientes a la función.

En nuestro ejemplo resulta la siguiente carta de descomposición. Ver Figura 2-1.

$x_4 x_1 x_0$ \ $x_3 x_2$	00	01	10	11
000	(0)	(4)	(8)	(12)
001	(1)	5	(9)	13
010	2	6	10	14
011	3	(7)	11	(15)
100	16	20	(24)	(28)
101	17	(21)	25	(29)
110	(18)	22	(26)	30
111	(19)	(23)	27	31

funciones residuos

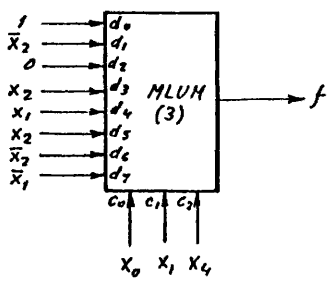


Figura 2.1.- Carta de descomposición y síntesis con un único MLUM(3) de la función $f(x_4,x_3,x_2,x_1,x_0) = \sum(0,1,4,7,8,9,12,15,18,19,21,23,24,26,28,29)$

2.3.- SINTESIS OPTIMA DE ESTRUCTURAS ARBORESCENTES DE MLUM(p): METODO DE VOITH

Los métodos de síntesis de estructuras arborescentes de MLUM(p) están basados en el siguiente teorema, cuya demostración es trivial.

Teorema 2.3.- Para cualquier función $f(x_{n-1},\ldots,x_0)$ las entradas de datos a cada uno de los MLUM(p) en una estructura arborescente que realiza la función, son funciones residuos de $f(x_{n-1},\ldots,x_0)$.

Este teorema indica que el árbol de MLUM(p) óptimo de una función se puede calcular si se determina la expansión que da lugar al menor número de funciones residuos multivariables.

En lo que sigue y sin que suponga pérdida de generalidad, consideraremos la síntesis con MLUM(1). La extensión al caso de síntesis con MLUM(p) se realiza de forma directa.

Para comprender el método de Voith son necesarias las siguientes definiciones:

Definición 2.1.- Un implicante MLUM(p) de una función f es un término producto cuyo residuo es una función trivial.

Si se utiliza un implicante MLUM(p) para recubrir una función, entonces a cada literal del término producto le corresponde una entrada de control de un determinado MLUM(p) de la estructura arborescente y el residuo es una entrada de datos.

Estos implicantes MLUM(p) son extensiones de los implicantes ordinarios de una función, que son términos productos cuyo residuo g_i tiene valor 1. Debe observarse que no existen implicantes primos MLUM(p) en correspondencia con los implicantes primos de una función. Esto se pone fácilmente de manifiesto con el siguiente ejemplo, sea una función que tiene los siguientes implicantes MLUM(1):

$x_0, x_3x_2, x_3x_2\bar{x}_0$ y $x_3x_2x_0$ entre otros. El término producto x_3x_2 recubre a los términos productos $x_3x_2\bar{x}_0$ y $x_3x_2x_0$. Resulta obvio que x_3x_2 es un implicante primo, mientras que $x_3x_2\bar{x}_0$ y $x_3x_2x_0$ no lo son. Sin embargo, si se escoge para recubrir la función el implicante -MLUM(1) x_0, el MLUM(1) de salida (primer nivel) tendrá a x_0 como su entrada de control, lo cual elimina la posibilidad de elegir x_3x_2 para recubrir a la función, puesto que esto exigiría que la variable de control del primer nivel fuera x_3 o x_2. No obstante, la elección de x_0 no impide que podamos escoger a $x_3x_2\bar{x}_0$, por ejemplo, para recubrir a la función.

Esto plantea el problema de que, una vez determinados los implicantes -MLUM(p) hay que escoger un subconjunto de ellos para recubrir la función con la condición de que sean compatibles, es decir, no planteen conflictos a la hora de generar la estructura arborescente.

Definición 2.2.- Un conjunto de implicantes -MLUM(1) es compatible sí y sólo si los términos productos tienen por lo menos una variable x en común y los dos subconjuntos generados por la siguiente regla son, o conjuntos compatibles o conjuntos vacíos.

Regla de generación.- Disponer los términos productos que contienen x en forma no complementada en un subconjunto, así como los que contienen a x en forma complementada. Eliminar la variable x de todos los términos productos.

Ejemplo 2.2.- Determinar si el conjunto de implicantes -MLUM(1) siguiente:

$C_0 = \{I_1 = x_5x_2\bar{x}_1,\ I_2 = \bar{x}_5x_4x_0,\ I_3 = \bar{x}_5\bar{x}_4x_3x_2,\ I_4 = x_5x_4x_1\}$ es compatible.

Todos los términos productos tienen una variable en común, x_5. Así pues, se deberá tomar x_5 como la entrada de control al MLUM(1) del primer nivel. La Figura 2-2

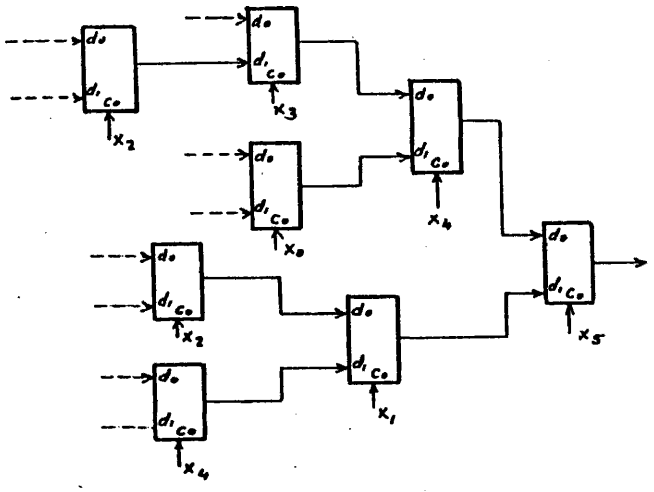


Figura 2.2.- Compatibilidad de los implicantes -MLUM(1) del Ejemplo 2.2.

ilustra este hecho. Esta elección particiona el conjunto C_0 en dos subconjuntos C_1 y C_1' basado sobre si la variable x_5 está complementada o no. Los dos subconjuntos son:

$C_1 = \{x_5x_2\bar{x}_1,\ x_5x_4x_1\}$ y $C_1' = \{\bar{x}_5x_4x_0,\ \bar{x}_5\bar{x}_4x_3x_2\}$

Si eliminamos la variable x_5 de ambos conjuntos, se obtienen los conjuntos siguientes:

$$\{x_2\bar{x}_1,\ x_4x_1\} \quad y \quad \{x_4x_0,\ \bar{x}_4x_3x_2\}$$

Cada uno de estos conjuntos deberá tener una variable común para evitar conflictos en el segundo nivel de la estructura arborescente. Prosiguiendo el particionamiento se observa que no se producen conflictos y, por lo tanto, el conjunto C está formado por implicantes -MLUM(1) compatibles.

Todo queda reducido a la determinación de los siguientes algoritmos:

a) Algoritmo para la obtención de todos los implicantes MLUM(1)

b) Algoritmo de compatibilidad de implicantes -MLUM(1) y cálculo de los conjuntos máximos compatibles.

Definición 2.3.- Un conjunto de implicantes MLUM(1) compatibles entre sí dos a dos, se dice que es máximo compatible si no se puede determinar un conjunto de implicantes -MLUM(1) compatible que lo contenga como subconjunto propio.

Veamos un poco en detalle el desarrollo de ambos:

- Algoritmo de cálculo de los implicantes -MLUM(1) de una función

Para identificar un implicante MLUM(1) es necesario identificar el término producto que le corresponde así como su función residuo trivial.

El término producto se puede representar de la misma forma que los implicantes normales de una función cuando se aplica el método de Quine-McCluskey. Si se utiliza la notación decimal, el término producto viene representado por dos número enteros $D(M)$. Donde M se calcula como la suma de las potencias de dos de las variables que no aparecen en el término producto y D es el minterm de mayor valor contenido en el término producto.

Por ejemplo, si a las variables x_4, x_3, x_2, x_1 y x_0 de una función se le asocian, respectivamente, los pesos 2^4, 2^3, 2^2, 2^1 y 2^0, entonces el término producto $x_2\overline{x}_1x_0$ vendrá representado por $D(M) = 29(24)$.

La función implicada por el implicante -MLUM(1) se identificará añadiendo a la representación del término producto un número de función F. Como para n variables existen $2n+2$ funciones triviales, la asignación se efectúa de la forma siguiente:

Función trivial	F
0	0
1	1
x_i $(i=0,\ldots,n-1)$	$i+2$
$\overline{x}_i$ $(i=0,\ldots,n-1)$	$n+i+2$

Con estos convenios, un implicante -MLUM(1) vendrá representado por el triplete $\mathcal{D}(M,F)$. Así, si al término producto $x_2\overline{x}_1x_0$ le corresponde como función residuo $\overline{x}_4$, lo que podemos representar por $x_2\overline{x}_1x_0(\overline{x}_4)$, su notación decimal correspondiente será 29(24,11).

Regla de combinación de implicantes MLUM(1)

Dos implicantes -MLUM(1), $I_2 = \mathcal{D}_2(M_2,F_2)$ e $I_3 = \mathcal{D}_3(M_3,F_3)$ se pueden combinar para formar un nuevo implicante MLUM(1) $I_1 = \mathcal{D}_1(M_1,F_1)$ si se verifican las condiciones siguientes:

1) $\mathcal{D}_3 - \mathcal{D}_2 = 2^i$
2) $M_3 = M_2$
3) índice (I_3) = Indice (I_2) + 1, donde índice (I_i) = número de variables sin complementar que forman I_i
4) $F_3 = F_2$ ó F_2 y $F_3 \in \{0,1\}$

Si se cumplen estas condiciones

$$\mathcal{D}_1 = \mathcal{D}_3$$
$$M_1 = M_3 + 2^i$$

y F_1 se asigna de acuerdo con la tabla siguiente:

$F_3 = F_2$	$F_1 = F_2$
$F_2 = 0, \quad F_3 = 1$	$F_1 = i+2$
$F_2 = 1, \quad F_3 = 0$	$F_1 = n+i+2$

Como en el caso de la formación de los implicantes primos de una función todos los implicantes MLUM(1) que corresponden a términos productos en los que faltan i variables se forman a partir de aquéllos en los que faltan $i-1$ variables. Estas consideraciones nos llevan al siguiente algoritmo

Algoritmo a: Determinación de implicantes -MLUM(1):

A1) Inicialización:

Comenzar con todos los minterms de la función a los cuales corresponderán una representación de la forma $D_i(0,k)$, $k \in \{0,1\}$ dependiendo de si el minterm está o no presente en la función.

A2) $j=0$ (la variable j indica el número de variables que faltan en el implicante MLUM(1)).

A3) $j=j+1$

A4) Formar todos los implicantes MLUM(1) en los que faltan j-variables mediante las reglas de combinación de implicantes -MLUM(1) dadas previamente.

A5) Si $j=n$ Parar

A6) Si no existe ningún implicante MLUM(1) en el que falten j variables. Parar.

A7) Ir a A3.

Ejemplo 2.3.- Determinar los implicantes -MLUM(1) de la función

$$f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$$

El algoritmo se ilustra en la Tabla 2-2. Debe observarse que los implicantes MLUM(1) que son útiles para reducir el número de MLUM(1) necesarios en una estructura arborescente son aquéllos que están situados en la columna 3 y posteriores. Los implicantes en columna 2 no representan ningún ahorro en módulos, ya que darían lugar a simplificación en el nivel n del árbol y únicamente se necesitan $n-1$ niveles para una función de n variables sintetizada con MLUM(1).

Algoritmo b: Compatibilidad de implicantes -MLUM(1)

Sea C_i un conjunto de implicantes -MLUM(1) que son compatibles e I_t un implicante -MLUM(1) que se desea comprobar si puede añadirse al conjunto C_i. El procedimiento que se sigue es el siguiente:

A1) Si C_i e I_t tienen variables comunes, entonces ir a A2, si no I_t no es compatible con C_i. Parar.

A2) Sea C_{i+1} el subconjunto de C_i tal que las variables comunes tienen el mismo valor literal que I_t.

A3) Eliminar las variables comunes de C_{i+1} e I_t y hacer $I_{t+1}=I_t$.

A4) $i = i + 1$, $t = t+1$

A5) Si C_i o I_t está vacío, entonces I_t es compatible con C_i si no ir a A1

El algoritmo comienza con los conjuntos C_i formados por un solo elemento $i = 1,\ldots,M$ siendo M el número de implicantes -MLUM(1) que se ha determinado en el algoritmo previo y que dan lugar a una reducción en el número de módulos de la estructura arborescente (es decir, están situados en la columna 3 y siguientes). A partir de éstos se determinan todas las parejas de implicantes MLUM(1) que son compatibles por un examen exhaustivo de todas las posibles combinaciones. El proceso continúa hasta que no se puede expansionar más los conjuntos de impli-

Columna 1	Columna 2	Columna 3	Columna 4
15(0,1)	15(1,2)	15(5,2)	13(13,5)
14(0,0)	15(2,1)	15(6,1)	
13(0,1)	15(4,1)	14(6,7)	
11(0,1)	15(8,1)	13(5,1)	
7(0,1)	14(2,7)	13(9,5)	
12(0,1)	14(4,0)	13(12,5)	
10(0,0)	14(8,9)	11(10,5)	
9(0,1)	13(1,1)	7(3,3)	
6(0,1)	13(4,1)	7(5,4)	
5(0,0)	13(8,5)	12(12,5)	
3(0,0)	11(1,2)	9(9,5)	
8(0,1)	11(2,1)	5(5,0)	
4(0,0)	11(8,5)	3(3,0)	
2(0,0)	7(1,1)		
1(0,0)	7(2,3)		
0(0,0)	7(4,4)		
	12(4,1)		
	12(8,5)		
	10(2,7)		
	10(8,0)		
	9(1,1)		
	9(8,5)		
	6(2,3)		
	6(4,4)		
	5(1,0)		
	5(4,0)		
	3(1,0)		
	3(2,0)		
	8(8,5)		
	4(4,0)		
	2(2,0)		
	1(1,0)		

TABLA 2.2.- Implicantes -MLUM(1) de la función $f(x_3,x_2,x_1,x_0)=\sum(7,8,9,11,12,13,15)$ del Ejemplo 2.3.

cantes MLUM(1) que son compatibles entre sí. Esto recuerda mucho a la determinación de los estados compatibles en una máquina secuencial especificada de forma incompleta. Ahora la relación de compatibilidad viene expresada por el algoritmo b.

Ejemplo 2.4.- Para el ejemplo anterior se da en la tabla 2-3 todos los posibles conjuntos de implicantes MLUM(1) compatibles.

Los conjuntos máximos compatibles son aquéllos marcados con un asterisco. De estos conjuntos se escoge aquel que da lugar a un mayor ahorro en el número de MLUM(1) necesitados en su implementación. La determinación de la estructura arborescente se efectúa por un procedimiento análogo al del ejemplo 2.2.

En nuestro caso, este conjunto resulta ser $(x_3x_1(x_0), \bar{x}_3x_1(x_2), \bar{x}_1(x_3))$ al que le corresponde la estructura óptima arborescente de la figura 2.3.

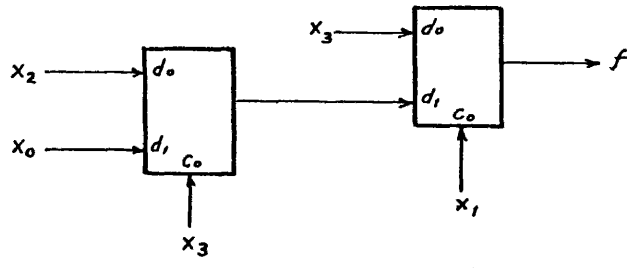


Figura 2.3.- Estructura arborescente óptima para la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$ del Ejemplo 2.4.

Columna 1	Columna 2	Columna 3	Columna 4
$x_3x_1(x_0)$	$x_3x_1(x_0), x_3\overline{x}_1(1)$	$x_3x_1(x_0), x_3\overline{x}_1(1), \overline{x}_3\overline{x}_2(\emptyset)$	$x_3x_1(x_0), x_3\overline{x}_1(1), \overline{x}_3\overline{x}_2(\emptyset), \overline{x}_3x_2(x_1)$*
$x_3x_0(1)$	$x_3x_1(x_0), x_2\overline{x}_1(x_3)$	$x_3x_1(x_0), x_3\overline{x}_1(1), \overline{x}_3x_1(x_2)$	$x_3x_1(x_0), x_3\overline{x}_1(1), \overline{x}_3x_1(x_2)\overline{x}_3\overline{x}_1(\emptyset)$*
$x_3\overline{x}_0(\overline{x}_1)$	$x_3x_1(x_0), \overline{x}_1x_0(x_3)$	$x_3x_1(x_0), x_3\overline{x}_1(1), \overline{x}_3x_2(x_1)$	$x_3x_1(x_0), x_2\overline{x}_1(x_3), \overline{x}_3x_1(x_2), \overline{x}_2\overline{x}_1(x_3)$*
$x_3\overline{x}_1(1)$	$x_3x_1(x_0), \overline{x}_3x_2(x_1)$	$x_3x_1(x_0), x_3x_1(1), \overline{x}_3\overline{x}_1(\emptyset)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_2(x_1), \overline{x}_3\overline{x}_2(\emptyset)$*
$x_2\overline{x}_1(x_3)$	$x_3x_1(x_0), \overline{x}_3x_1(x_2)$	$x_3x_1(x_0), x_2\overline{x}_1(x_3), \overline{x}_3x_1(x_2)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_1(x_2), \overline{x}_3\overline{x}_1(\emptyset)$*
$\overline{x}_1x_0(x_3)$	$x_3x_1(x_0), \overline{x}_2\overline{x}_1(x_3)$	$x_3x_1(x_0), x_2\overline{x}_1(x_3), \overline{x}_2\overline{x}_1(x_3)$	
$\overline{x}_2x_0(x_3)$	$x_3x_1(x_0), \overline{x}_3\overline{x}_1(\emptyset)$	$x_3x_1(x_0), \overline{x}_1x_0(x_3), \overline{x}_3x_1(x_2)$	
$\overline{x}_3x_2(x_1)$	$x_3x_1(x_0), \overline{x}_3\overline{x}_2(\emptyset)$	$x_3x_1(x_0), \overline{x}_3x_2(x_1), \overline{x}_3\overline{x}_2(\emptyset)$	
$\overline{x}_3x_1(x_2)$	$x_3x_1(x_0), \overline{x}_1(x_3)$	$x_3x_1(x_0), \overline{x}_3x_1(x_2), \overline{x}_2\overline{x}_1(x_3)$	
$\overline{x}_2\overline{x}_0(x_3)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1)$	$x_3x_1(x_0), \overline{x}_3x_1(x_2), \overline{x}_3\overline{x}_1(\emptyset)$	
$\overline{x}_2\overline{x}_1(x_3)$	$x_3x_0(1), \overline{x}_3x_2(x_1)$	$x_3x_1(x_0), \overline{x}_3x_1(x_2), \overline{x}_1(x_3)$*	
$\overline{x}_3\overline{x}_1(\emptyset)$	$x_3x_0(1), \overline{x}_3x_1(x_2)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_2(x_1)$	
$\overline{x}_3\overline{x}_2(\emptyset)$	$x_3x_0(1), \overline{x}_2\overline{x}_0(x_3)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_1(x_2)$	
$\overline{x}_1(x_3)$	$x_3x_0(1), \overline{x}_3\overline{x}_1(\emptyset)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3\overline{x}_1(\emptyset)$	
	$x_3x_0(1), \overline{x}_3\overline{x}_2(\emptyset)$	$x_3x_0(1), x_3\overline{x}_0(\overline{x}_1), \overline{x}_3\overline{x}_2(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_1x_0(x_3)$	$x_3x_0(1), \overline{x}_3x_2(x_1), \overline{x}_3\overline{x}_2(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_2x_0(x_3)$	$x_3x_0(1), \overline{x}_3x_1(x_2), \overline{x}_3\overline{x}_1(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_2(x_1)$	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_2(x_1), \overline{x}_3\overline{x}_2(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_1(x_2)$	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3x_1(x_2), \overline{x}_3\overline{x}_1(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3\overline{x}_1(\emptyset)$	$x_3\overline{x}_1(1), \overline{x}_3x_2(x_1), \overline{x}_3\overline{x}_2(\emptyset)$	
	$x_3\overline{x}_0(\overline{x}_1), \overline{x}_3\overline{x}_2(\emptyset)$	$x_3\overline{x}_1(1), \overline{x}_3x_1(x_2), \overline{x}_3\overline{x}_1(\emptyset)$	
	$x_3\overline{x}_1(1), \overline{x}_3x_2(x_1)$	$x_2\overline{x}_1(x_3), \overline{x}_2x_0(x_3), \overline{x}_2\overline{x}_0(x_3)$	
	$x_3\overline{x}_1(1), \overline{x}_3x_1(x_2)$	$\overline{x}_2x_0(x_3), \overline{x}_3x_2(x_1), \overline{x}_2\overline{x}_0(x_3)$	

Columna 1	Columna 2	Columna 3	Columna 4
	$x_3\overline{x}_1(1), \overline{x}_3\overline{x}_1(\emptyset)$		
	$x_3\overline{x}_1(1), \overline{x}_3\overline{x}_2(\emptyset)$		
	$x_2\overline{x}_1(x_3), \overline{x}_2x_0(x_3)$		
	$x_2\overline{x}_1(x_3), \overline{x}_3x_1(x_2)$		
	$x_2\overline{x}_1(x_3), \overline{x}_2\overline{x}_0(x_3)$		
	$x_2\overline{x}_1(x_3), \overline{x}_2\overline{x}_1(x_3)$		
	$x_2\overline{x}_1(x_3), \overline{x}_3\overline{x}_2(\emptyset)$		
	$\overline{x}_1x_0(x_3), \overline{x}_3x_1(x_2)$		
	$\overline{x}_1x_0(x_3), \overline{x}_2\overline{x}_0(x_3)$		
	$\overline{x}_2x_0(x_3), \overline{x}_3x_2(x_1)$		
	$\overline{x}_2x_0(x_3), \overline{x}_2\overline{x}_0(x_3)$		
	$\overline{x}_3x_2(x_1), \overline{x}_2\overline{x}_0(x_3)$		
	$\overline{x}_3x_2(x_1), \overline{x}_2\overline{x}_1(x_3)$		

TABLA 2.3.- Conjuntos compatibles para la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$ del Ejemplo 2.4.

2.4.- GENERALIZACION DEL METODO DE VOITH

El método de Voith presentado en el apartado anterior presenta algunas limitaciones que hacen que su utilidad quede, algunas veces, disminuida.

En particular merecen destacarse las siguientes:

a) No permite manejar términos inoperantes en la síntesis de funciones.

b) La existencia de funciones residuos idénticas no se introduce en el proceso.

c) No contempla la síntesis mediante estructuras arborescentes de funciones múltiples.

En esta sección abordaremos la generalización del método de Voith respecto al punto a), es decir, introduciendo la posibilidad de manejar términos inoperantes.

En primer lugar, se presenta un método de determinar implicantes -MILUM(1) que resulta de una aplicación directa de las cartas de descomposición.

2.4.1.- Determinación de implicantes MLUM(1) mediante el uso de cartas de descomposición ([8])([9])

Sea un conjunto dado $X = \{x_{n-1}, x_{n-2}, \ldots, x_0\}$ de n variables. Por una partición del conjunto X, se entiende un par ordenado de conjuntos

$$Y = \{y_1, \ldots, y_s\} \quad , \quad Z = \{z_1, \ldots, z_t\}$$

de variables de X y en las cuales se permiten, en general, repeticiones pero satisfacen $Y \cup Z = X$. Es decir, $\forall\, x_k \in X$, pertenece, al menos, a uno de los dos subconjuntos Y y/o Z.

Representamos esta partición de X por:

$$Y|Z = y_1, \ldots, y_s | z_1, \ldots, z_t$$

El número entero δ en la partición se denomina grado primario de $Y|Z$ y el número entero t el grado secundario

Como $Y \cup Z = X$, se debe verificar $\delta + t \geq n$. El entero no negativo $h = \delta + t - n$ se denomina grado de repetición de la partición $Y|Z$.

Si $h=0$, entonces Y y Z deben ser disjuntos y las variables en Y y en Z son todas distintas. En este caso, la partición $Y|Z$ se llama disjunta. Existen 2^n particiones disjuntas de X.

La carta de descomposición de la partición $Y|Z$ viene dada por la matriz $C\ \mathcal{D}_n\ (Y|Z)$ de 2^δ filas y 2^t columnas definida como sigue: La filas corresponde a los 2^δ enteros binarios $y = y_1 y_2, \ldots, y_\delta$ dados en orden creciente y las columnas a los 2^t enteros binarios $z = z_1, \ldots, z_t$ también en orden creciente. Como $Y|Z$ es una partición de X, cada punto $x = x_1 x_2, \ldots, x_n$ de B^n(*) determina unívocamente los enteros binarios $y = y_1, \ldots, y_\delta$ y $z = z_1, \ldots, z_t$. En este caso, definimos el elemento en la fila y-ésima y en la columna z-ésima como el número decimal entero que representa a x. Así, se obtienen los elementos en 2^n posiciones de la matriz $C\ \mathcal{D}_n\ (Y|Z)$. En cada una de las restantes posiciones $2^{\delta+t}-2^n$ de la matriz se coloca el signo *, que representa una condición de término inoperante. En el caso de una partición disjunta no existen términos inoperantes.

El teorema siguiente es obvio de la definición dada para las cartas de descomposición.

Teorema 2-4.- Para una partición arbitraria $Y|Z$ de $X = \{x_{n-1}, \ldots, x_0\}$ la $C\ \mathcal{D}_n(Z|Y)$ es la transpuesta de $C\ \mathcal{D}_n\ (Y|Z)$, esto es $C\ \mathcal{D}_n(Z|Y)$ se puede obtener intercambiando las filas y las columnas de $C\ \mathcal{D}_n\ (Y|Z)$.

(*) $B^n = \underbrace{B \times B \ldots \times B}_{\text{n veces}}$; $B = \{0,1\}$

La ventaja que presentan las cartas de descomposición de una partición disjunta es que permiten aplicar, de forma rápida, el método de expansión de Shannon respecto a cualquier conjunto de variables que se desee, y de esta forma se pueden determinar todas aquellas funciones residuos que son triviales y dan lugar por tanto, a un implicante MLUM(1).

Veamos, pues, cuál es el número de cartas de descomposición necesitadas para n variables.

Para una función de n variables, buscar todas sus posibles particiones es buscar todas las posibles combinaciones de n elementos en el conjunto Y o en el conjunto Z. Estas serían, pues, 2^n. Sin embargo, de éstas podemos sacar algunos casos triviales como las particiones $y_1y_2,\ldots,y_n|\emptyset$ y $\emptyset|z_1z_2,\ldots,z_n$ por carecer de sentido. De la misma forma, las particiones cuyo conjunto de variables Z está compuesta por una sola variable x_i, $0 \leq i \leq n-1$, son triviales puesto que equivaldría a aplicar sobre la función, una expansión de Shannon respecto a $n-1$ variables, y por lo tanto le corresponderían funciones residuos de una variable (a estas cartas de descomposición corresponden los implicantes -MLUM(1) de la columna 2 del método de Voith, que como sabemos ya, no producen ahorro de módulos en la estructura arborescente). Así pues, habrá que descontar n particiones. En total se tienen que analizar 2^n-n-2 particiones. Ahora bien, teniendo en cuenta el Teorema 2-4, una partición cualquiera y su dual (cambio de filas por columnas y viceversa) se pueden estudiar sobre la misma carta de descomposición. De esta manera, el problema de determinar todas las posibles descomposiciones de una función de n variables se reduciría a analizar el conjunto $2^{n-1}-1$ cartas de descomposición.

Ejemplo 2.5.- En la figura 2-4 se muestran las cartas para $n = 4$ (7 tablas). Su uso es muy sencillo, puesto que en cada uno de los mapas de Marquand están expre-

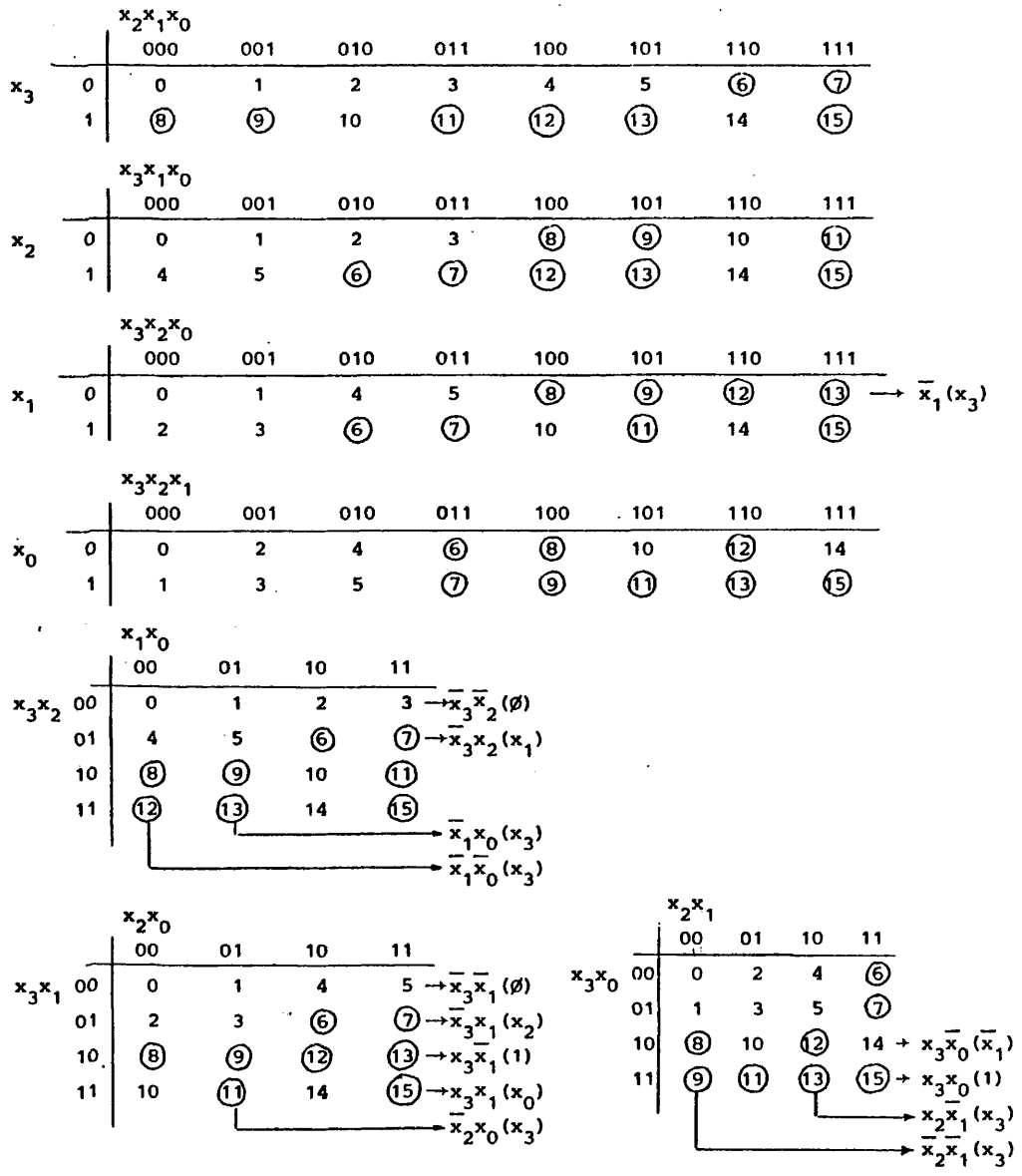


Figura 2.4.- Determinación del conjunto de implicantes -MLUM(1), mediante el caso de las cartas de descomposición, de la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$

sados todos los minterms en su representación decimal; no hay más que envolver con un círculo cada minterm que intervenga en la función a estudiar (se ha representado la función tratada en el ejemplo anterior). La determinación de los implicantes MLUM(1) (de la columna 2 y posteriores) se hace una vez representada la función en las cartas de descomposición, observando si las funciones residuos correspondientes a una determina expansión de Shannon son triviales o no. En la figura 2-4 se han representado, asimismo, las implicantes -MLUM(1), al lado de cada fila o columna de las distintas cartas de descomposición que dan lugar a funciones residuos triviales.

2.4.2.- Determinación de implicantes -MLUM(1) cuando existen términos inoperantes

2.4.2.a).- Método de las cartas de descomposición

Como se sabe, la existencia de términos inoperantes permite una mayor flexibilidad a la hora de efectuar la síntesis de una función. Esta afirmación sigue siendo válida en el caso particular que nos ocupa de obtener síntesis óptimas de estructuras arborescentes de MLUM(1).

En princípio,la dificultad que parece presentarse,es que la existencia de un determinado implicante -MLUM(1) dependerá de las asignaciones que se le den a los términos inoperantes. Dichas asignaciones deberán de ir explicitadas al lado del implicante -MLUM(1), ya que las relaciones de compatibilidad entre implicantes -MLUM(1) van a depender adicionalmente de los valores que tomen los términos inoperantes.

Afortunadamente no se hace necesario tener en cuenta que valores toman los términos inoperantes, ya que si dos implicantes MLUM(1) son combinables, los minterm que componen cada uno de los términos productos deben ser disjuntos, y en consecuencia no pueden existir incompatbilidades en la asignación de valores lógicos (0 ó 1) a los términos inoperantes.

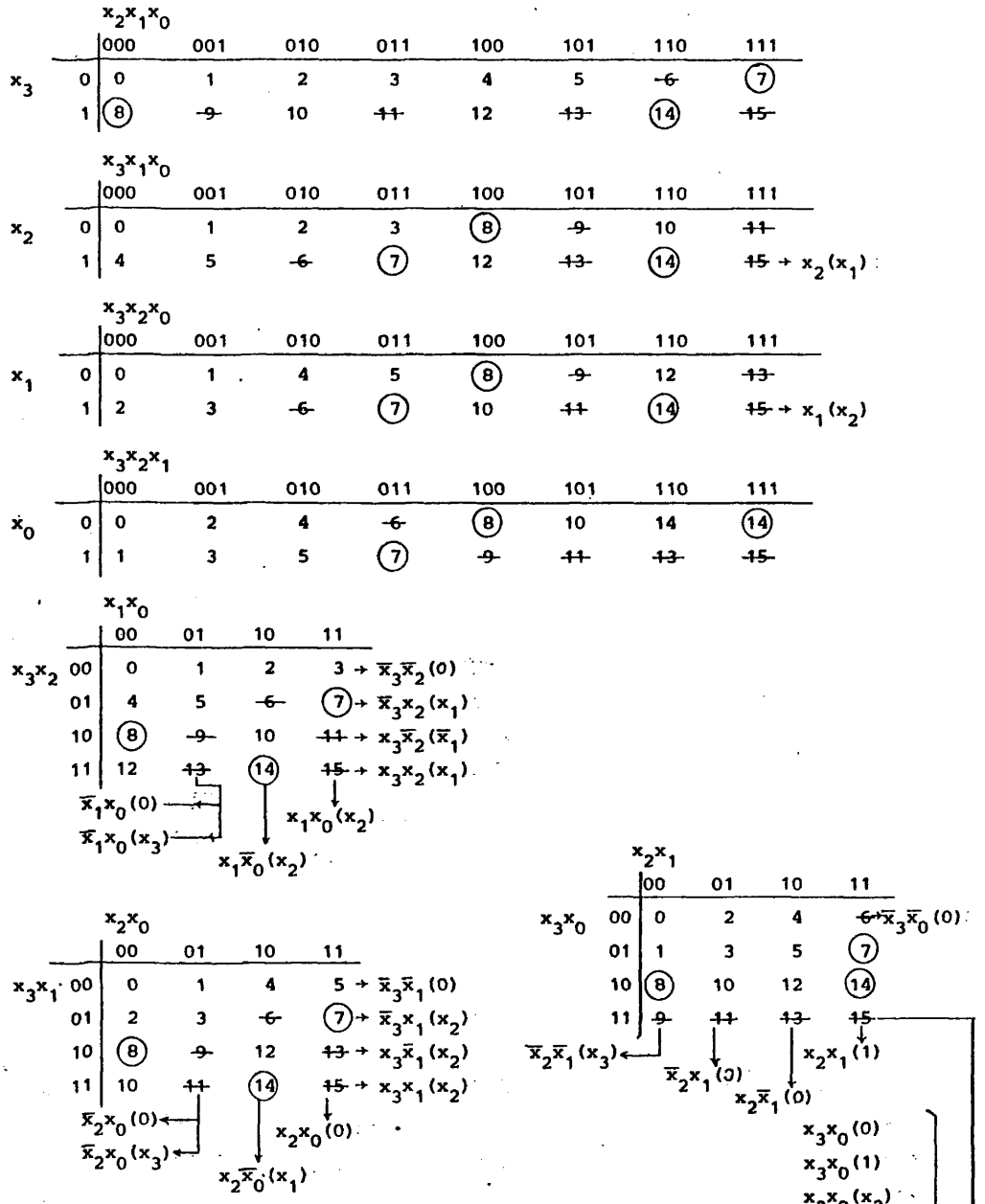


Figura 2.5.- Determinación del conjunto de implicantes -MLUM(1), mediante el uso de las cartas de descomposición, de la función $f(x_3,x_2,x_1,x_0) = \sum(7,8,14) + k\,(6,9,11,13,15)$

Análogamente, los implicantes MLUM(1) que forman un conjunto compatible, deberan ser disjuntos entre sí.

Las cartas de descomposición se pueden utilizar, asimismo, en el caso de existir términos inoperantes. En la representación de la función, sobre las cartas de descomposición, los términos inoperantes se especifican por una línea horizontal sobre el minterm.

La elección de los términos inoperantes se efectúa de manera que se fuerza (si ello es posible) a la función residuo correspondiente a ser trivial. Puede existir más de una asignación que verifique esta condición y todas ellas deberán de especificarse.

Con el fin de poner de manifiesto el procedimiento analicemos con detalle el siguiente ejemplo:

Ejemplo 2-6.- Determinar el conjunto de implicantes -MLUM(1) para la función especificada de forma incompleta siguiente:

$$f(x_3,x_2,x_1,x_0) = \sum(7,8,14) + k(6,9,11,13,15)$$

En la figura 2-5 está representada la función sobre las cartas de descomposición.

Se observa, que a determinados implicantes MLUM(1), les corresponden más de una asignación diferente como función residuo, dependiendo de los valores que tomen los términos inoperantes. Así, por ejemplo, al término producto $\bar{x}_2x_0$ se le asocian las funciones residuos triviales 0 (minterms 9 y 11 toman el valor 0) ó x_3(minterms 9 y 11 toman el valor 1).

La especificación de las diferentes funciones residuos triviales que resultan de las distintas asignaciones de los términos inoperantes son necesarias,pues, dependiendo de las funciones que resulten, pueden dar lugar a una combinación posterior de implicantes MLUM(1), lo que redundaria en definitiva, en poder utilizar un menor número de módulos para sintetizar la función.

A pesar de su simplicidad es obvio pensar que en cuanto el número de variables crece, el análisis de las cartas de descomposición se convierte en un examen engorroso, así para 6 variables habrá que examinar 31 mapas en ambos sentidos.

Por este motivo sería conveniente extender el procedimiento tabular de Voith tomando en cuenta la presencia de términos inoperantes.

2.4.2.b) Método tabular

Unicamente hay que generalizar la regla de combinación de implicantes -MLUM(1).

Dos implicantes -MLUM(1), $I_2 = D_2(M_2,F_2)$ e $I_3 = D_3(M_3,F_3)$ se pueden combinar para formar un nuevo implicante -MLUM(1), $I_1 = D_1(M_1,F_1)$ si se verifican las condiciones siguientes:

1) $D_3 - D_2 = 2^i$
2) $M_3 = M_2$
3) índice (I_3) = índice (I_2) + 1

   donde índice (I_i) = número de variables sin complementar que forman I_i
4) $F_3 = F_2$ ó F_2 y $F_3 \in \{0,1,\emptyset\}$

Se observa que con respecto al caso anterior, se amplia la condición 4). Si se cumplen estas condiciones:

$$D_1 = D_3$$
$$M_1 = M_3 + 2^i$$

y F_1 se asigna de acuerdo con la tabla siguiente:

1	$F_3 = F_2$	$F_1 = F_2$
2	$F_2 = 0$, $F_3 = 1$	$F_1 = i + 2$
3	$F_2 = 1$, $F_3 = 0$	$F_1 = n + i + 2$
4	$F_2 = 0$, $F_3 = \emptyset$	a) $F_1 = F_2$ b) $F_1 = i + 2$
5	$F_2 = 1$, $F_3 = \emptyset$	c) $F_1 = F_2$ d) $F_1 = n + i + 2$
6	$F_2 = \emptyset$ $F_3 = 0$	e) $F_1 = F_3$ f $F_1 = n + i + 2$
7	$F_2 = \emptyset$ $F_3 = 1$	g) $F_1 = F_3$ h) $F_1 = i + 2$

el algoritmo a) de Voith de combinación de implicantes MLUM(1),con esta regla generalizada, queda modificado tal como pone de manifiesto el algoritmo c) que se expresa a continuación.

En la tabla anterior, la aparición de los términos inoperantes se observa en las filas 4,5,6 y 7, con dos posibilidades permitidas para la formación de F_1. Es precisamente esta cualidad, la que posibilita una mayor flexibilidad cuando se utilizan términos inoperantes.

Algoritmo C.- Determinación de implicantes -MLUM(1) cuando existen términos inoperantes

A1) Inicialización:

Comenzar con todos los minterms de la función a los cuales corresponderán una representación de la forma $\mathcal{D}_i(0,k)$, $k \in \{0,1,\emptyset\}$ dependiendo de si el minterm está o no presente o es inoperante.

A2) $j=0$ (la variable j indica el número de variables que faltan en el implicante MLUM(1).

A3) $j=j+1$

A4) Formar todos los implicantes -MLUM(1) en los que faltan j-variables mediante las reglas de combinación de implicantes -MLUM(1) dadas en este apartado.

A5) Si $j = n$ Parar.

A6) Si no existe ningún implicante -MLUM(1) en el que falten j variables. Parar.

A7) ir a A3)

Al finalizar el proceso de determinación de implicantes MLUM(1), aquellos que tienen la forma $\mathcal{D}(M,\emptyset)$ les corresponden $2+2(n-j)$ funciones residuos triviales, siendo j el número de variables que componen el término producto.

Sin embargo, estos implicantes MLUM(1), se pueden eliminar a la hora de formar los conjuntos de implicantes compatibles, ya que estan formados únicamente por minterms de la función que son inoperantes, y por lo tanto su introducción se hace superflua. No obstante, en algunos casos, como por ejemplo si se analizan funciones residuos idénticas o si se sintetizan funciones múltiples, la aparición de estos términos puede redundar en una síntesis con menor coste.

El procedimiento descrito, retiene las características del método original de Voith. Para poner esto de manifiesto se expone a continuación un ejemplo de

aplicación.

Ejemplo 2-7.- Determinar los implicantes -MLUM(1) de la función especificada de forma incompleta siguiente:

$$f(x_3,x_2,x_1,x_0) = \sum(7,8,14) + k\,(6,9,11,13,15)$$

corresponde a la misma función tratada en el método de las cartas de descomposición. El algoritmo se ilustra en la tabla 2-4.

Una vez determinado el conjunto de implicantes MLUM(1), se aplica el algoritmo b) de compatibilidad de implicantes MLUM(1) visto anteriormente al tratar el método de Voith. Como en este algoritmo únicamente interviene el término producto y no se toman en cuenta las funciones residuos, en aquellos implicantes MLUM(1) a los que corresponda mas de una función residuo trivial,únicamente se tomará una de ellas. Si es posible, aquella que presente una menor carga lógica al circuito (para minimizar al máximo los problemas de fan-in que se puedan presentar).

Así, por ejemplo, en la columna tres de la Tabla 2.4, se observan los dos implicantes MLUM(1) siguientes:

$$13(12,0) \rightarrow \bar{x}_1 x_0 (0)$$

$$13(12,5) \rightarrow \bar{x}_1 x_0 (x_3)$$

de estos tomaremos $\bar{x}_1 x_0 (0)$ en el algoritmo de compatibilidad.Esta eliminación únicamente se podrá llevar a efecto al finalizar el algoritmo de determinación de implicantes MLUM(1) (algoritmo c)), ya que si se hiciese durante la generación de implicantes MLUM(1), estaríamos reduciendo la posibilidad de combinación que permite la existencia de términos inoperantes.

Finalmente, de estos conjuntos de implicantes MLUM(1) compatibles, hay que extraer los conjuntos máximos compatibles.

Columna 1	Columna 2	Columna 3	Columna 4
15(0,Ø)	15(1,1)	15(3,3)	15(11,3)
14(0,1)	15(1,6)	15(5,4)	15(13,4)
13(0,Ø)	15(2,Ø)	15(9,1)	
11(0,Ø)	15(4,Ø)	15(6,Ø)	
7(0,1)	15(8,1)	15(10,3)	
12(0,0)	15(8,9)	15(12,4)	
10(0,0)	14(2,3)	14(10,3)	
9(0,Ø)	14(4,4)	14(12,4)	
6(0,Ø)	14(8,1)	13(5,8)	
5(0,0)	14(8,5)	13(9,0)	
3(0,0)	13(1,0)	13(12,0)	
8(0,1)	13(1,3)	13(12,5)	
4(0,0)	13(4,Ø)	11(3,7)	
2(0,0)	13(8,0)	11(9,0)	
1(0,0)	13(8,5)	11(10,0)	
0(0,0)	11(1,0)	11(10,5)	
	11(1,2)	7(3,3)	
	11(2,Ø)	7(5,4)	
	11(8,0)	9(9,5)	
	11(8,5)	6(6,0)	
	7(1,1)	5(5,0)	
	7(1,2)	3(3,0)	
	7(2,3)		
	7(4,4)		
	12(4,8)		
	12(8,0)		
	10(2,7)		
	10(8,0)		
	9(1,1)		
	9(1,6)		
	9(8,0)		
	9(8,5)		
	6(2,0)		
	6(2,3)		

Columna 1	Columna 2	Columna 3	Columna 4
	6(4,0)		
	6(4,4)		
	5(1,0)		
	5(4,0)		
	3(1,0)		
	3(2,0)		
	8(8,5)		
	4(4,0)		
	2(2,0)		
	1(1,0)		

TABLA 2.4.- Implicantes -MLUM(1) de la función $f(x_3,x_2,x_1,x_0) = \sum(7,8,14) + k\,(6,9,11,13,15)$

Ejemplo 2.8.- Para el ejemplo 2.7 y siguiendo un procedimiento similar al visto en el caso de funciones completamente especificadas, se deduce que los conjuntos máximos compatibles que dan lugar a un mayor ahorro en la implementación de la estructura arborescente son:

$$C_3 = \begin{cases} x_1(x_2) \\ \bar{x}_3\bar{x}_1(0) \\ x_3\bar{x}_1(\bar{x}_2) \end{cases}$$

$$C_3' = \begin{cases} x_2(x_1) \\ \bar{x}_3\bar{x}_2(0) \\ x_3\bar{x}_2(\bar{x}_1) \end{cases}$$

$$C_3'' = \begin{cases} x_1(x_2) \\ \bar{x}_2\bar{x}_1(x_3) \\ x_2\bar{x}_1(0) \end{cases}$$

$$C_3''' = \begin{cases} x_2(x_1) \\ \bar{x}_2x_1(0) \\ \bar{x}_2\bar{x}_1(x_3) \end{cases}$$

a los que corresponden las estructuras óptimas arborescentes de las figuras 2-6a) b),c) y d). La asignación de los términos inoperantes en todos los casos ha sido:

6	9	11	13	15
↓	↓	↓	↓	↓
1	1	0	0	1

En la Figura 2-6e) está representada la estructura arborescente óptima de la función si no se considerasen los términos inoperantes. Se observa que para este ejemplo al considerarlos se reduce el número de módulos necesarios de 5 a 2.

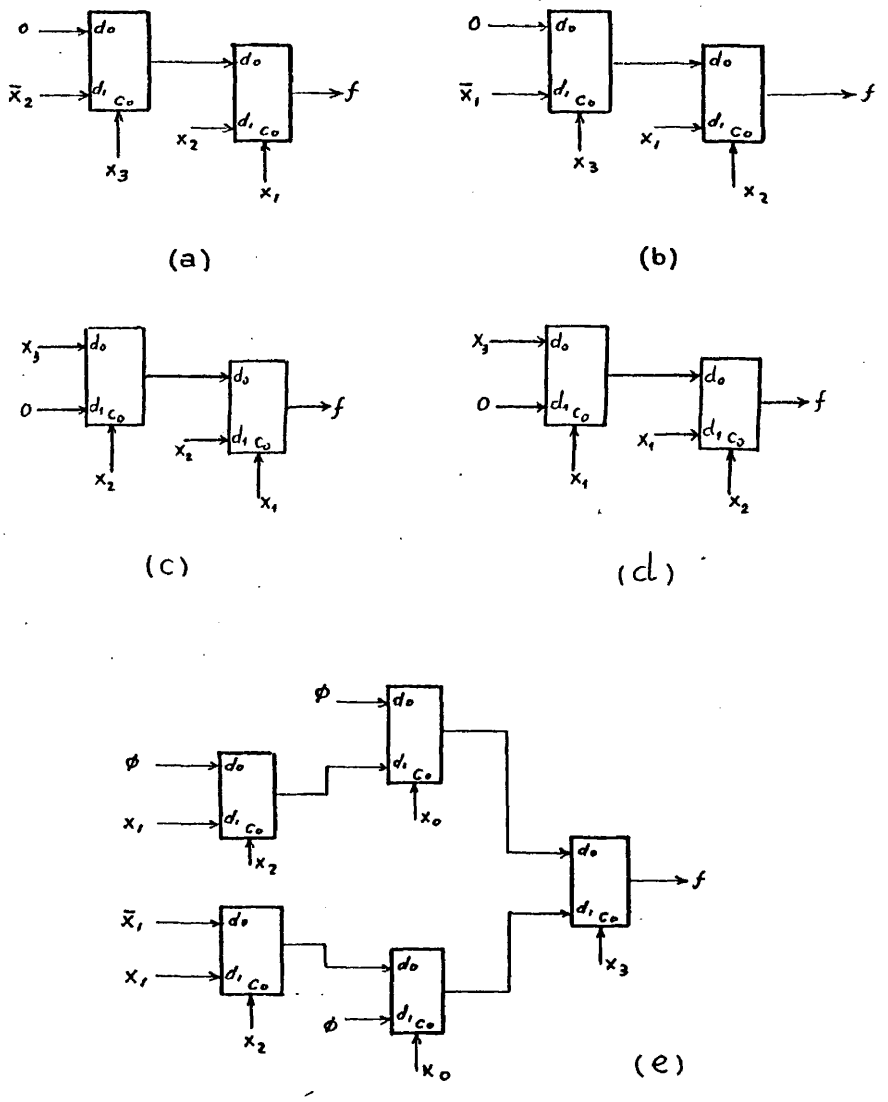


Figura 2.6.- Estructuras óptimas arborescentes

(a), (b), (c), (d) $\} \; f(x_3,x_2,x_1,x_0) = \sum(7,8,14) + (6,9,11,13,15)$

(e) $\rightarrow f(x_3,x_2,x_1,x_0) = \sum(7,8,14)$

2.5.- SINTESIS DE ESTRUCTURAS ARBORESCENTES OPTIMAS POR UN METODO DE ENUMERACION IMPLICITA

En esta sección se demuestra, en primer lugar, que la síntesis de estructuras arborescentes de MLUM(1) se puede reducir a un problema de programación lineal entera 0-1.

Para hacer la sección suficientemente autocontenida se introduce el concepto de enumeración implícita y el esquema clásico de Glover. Esto da pie para formular nuestro algoritmo de minimización.

Finalmente se generaliza el algoritmo al permitir la introducción de funciones residuos idénticas en el proceso. Esto supone eliminar la restricción b) que habíamos detectado en el algoritmo de Voith.

2.5.1.- Formulación del problema de programación lineal entera 0-1

Definición 2.6.- Un implicante -MLUM(1) se dice que es de nivel j si su término producto contiene j literales.

Teorema 2.5.- El número de módulos MLUM(1) que se eliminan en la estructura arborescente al considerar un implicante -MLUM(1) de nivel j es $a = 2^{\ell-j}-1$, donde ℓ corresponde al número máximo de niveles en la estructura ($\ell = n-1$).

Demostración.-

La existencia de un implicante -MLUM(1) de nivel j implica que no hay que considerar el subárbol que emana de su entrada de datos. El número de niveles del subárbol es $\ell-j$. Por lo tanto, el número de módulos que se eliminan será:

$$a = 1 + 2 +\ldots+ 2^{\ell-j-1} = 2^{\ell-j}-1 \qquad (2.3)$$

c.q.d.

El Teorema 2.5 permite asociar con cada implicante -MLUM(1) I_i, un nú-

mero entero a_i que representa el número de MLUM(1) que se ahorraría si se utilizase en la síntesis de la estructura arborescente a I_i.

El problema de síntesis está planteado en los términos de encontrar el conjunto de implicantes -MLUM(1) C que da un ahorro máximo en el número de módulos necesarios. Sea $\{I_1,\ldots,I_M\}$ el conjunto de todos los implicantes -MLUM(1) de la función que se desea sintetizar de nivel $1 \leq k \leq n-2$ (Obviamente no son necesarios considerar los implicantes -MLUM(1) de niveles $n-1$ y n puesto que, como se sabe, éstos son triviales y no dan lugar a ninguna economía en el número de MLUM(1) necesarios).

Con cada implicante -MLUM(1) I_i $\forall\, i \in [1,\ldots,M]$ se asocia una variable booleana X_i de tal forma que I_i estará en el conjunto C sí y solamente si $X_i = 1$.

Esto lleva a definir la siguiente función de coste:

$$J = \sum_{i=1}^{M} a_i X_i \qquad (2.4)$$

que representa el número máximo de módulos que se pueden eliminar. El valor de a_i viene dado por (2.3), según sea el nivel del implicante MLUM(1) que se le asocia.

En (2.4) hay que considerar las ligaduras que caracterizan la estructura arborescente de MLUM(1).

En primer lugar, el conjunto $C = \{I_i | X_i = 1\}$ debe ser un conjunto compatible de acuerdo con la Definición 2.2. En segundo lugar, se deben considerar las relaciones que implican el hecho de que un implicante -MLUM(1) I_k contiene a otro I_j.

Definición 2.7.- Un implicante MLUM(1) I_k contiene (subsume) a I_j si todos los literales de I_k están contenidos en los de I_j. Si además I_j contiene literales que

I_k no tiene, se dirá que I_k contiene (subsume) estrictamente a I_j $(I_j \subseteq I_k)$ y se cumple siempre que $\overline{I}_j + I_k = 1$.

Si $I_j \subseteq I_k$, entonces X_j y X_k no pueden ser simultáneamente iguales a 1. En lo que sigue se designan por V a este tipo de ligaduras.

Como una consecuencia de todo lo dicho, la formulación del problema de síntesis óptima viene plasmado por las siguientes expresiones:

$$\max J = \sum_{i=1}^{M} a_i X_i \qquad (2.5)$$

Sujeto a las siguientes ligaduras:

$$\begin{aligned} &V: X_j + X_k \leq 1 \qquad \forall\, I_j \subseteq I_k \\ &C = \{I_i | X_i = 1\} \text{ es un conjunto compatible} \qquad (2.6) \\ &X_i = 0,1 \quad (i = 1,\ldots,M) \end{aligned}$$

Las ecuaciones (2.5) y (2.6) se pueden transformar en una formulación equivalente pero como un problema de minimización al reemplazar $X_i = 1 - Y_i$

$$\min J' = \sum_{i=1}^{M} a_i Y_i \qquad (2.7)$$

sujeto a las siguientes ligaduras:

$$\begin{aligned} &V: Y_j + Y_k \geq 1 \qquad \forall\, I_j \subseteq I_k \\ &C = \{I_i | Y_i = 0\} \text{ es un conjunto compatible} \qquad (2.8) \\ &Y_i = 0,1 \quad (i = 1,\ldots,M) \end{aligned}$$

El problema de síntesis óptima de estructuras arborescentes queda, pues, reducido al de minimizar (2.7), teniendo en cuenta las ligaduras dadas por (2.8). Esto es un problema de programación lineal entera 0-1.

2.5.2.- Concepto de enumeración implícita

En general, el espacio solución de un problema de programación lineal entera, admite un número finito de posibles puntos admisibles. Un método directo para resolver este tipo de problemas es enumerar exhaustivamente (o explícitamente) todos estos puntos. En este caso, la solución óptima se determina por el (o los) punto(s) que dan el valor óptimo de la función de coste.

La desventaja obvia de los métodos explícitos es que el número de puntos que hay que testear en el espacio solución se hace rápidamente muy grande, de manera que la solución no se puede determinar en un tiempo razonable.

La idea de una enumeración implícita (o parcial) tiene como fin el considerar únicamente una parte (si es posible pequeña) de todos los posibles puntos solución, eliminando los restantes automáticamente como no deseables. Para ilustrar este punto, consideremos el siguiente problema.

Se desea determinar todas las soluciones admisibles para la siguiente inigualdad:

$$3\,x_1 - 8\,x_2 + 5\,x_3 \leq -6 \qquad x_j = (0,1) \quad j = 1,2,3$$

Un razonamiento simple nos hace ver que en cualquier solución admisible $x_2 = 1$. Esto significa que cualquier combinación binaria (x_1,x_2,x_3) que tenga $x_2 = 0$ no puede ser solución al problema. Así pues, las cuatro combinaciones (0,0,0), (1,0,0), (0,0,1) y (1 0 1) se eliminan automáticamente y se dice que se examinan implícitamente.

Dado que $x_2 = 1$ es una condición necesaria para la admisibilidad, la inigualdad es satisfecha por -6-(-8) = 2 unidades. De esta forma, x_1 ó x_2 (o ambos) pueden valer 1 únicamente si su contribución al lado izquierdo de la inigualdad no excede a 2. Como el coeficiente de x_1 es igual a 3, x_1 se deberá fijar

a cero, lo que implica que (1,1,0) y (1,1,1) están examinadas implícitamente como no admisibles.

Dados $x_1 = 0$ y $x_2 = 1$, la combinación (0,1,1) se elimina porque el coeficiente de x_3 es igual a 5. Así pues, la única combinación que queda (0,1,0) es la solución admisible de la inigualdad.

Para poder apreciar el impacto de utilizar la enumeración implícita, la Figura 2.7 da una representación gráfica de las ideas expuestas en el ejemplo anterior. La primera conclusión es que x_2 se debe fijar en el primer nivel, de manera que el subárbol que emana de la rama $x_2 = 0$ (se muestra en línea de trazos) se elimina (fathomed). Para $x_2 = 1$ la inigualdad nos dice que $x_1 = 1$ no puede conducir a una solución admisible y, por tanto $(x_2 = 1, x_1 = 1)$ se suprime. Finalmente, dado $x_2 = 1$ y $x_1 = 0$, la rama $x_3 = 1$ conduce a una solución no admisible, pero la rama $x_3 = 0$ da una solución correcta, y el proceso termina puesto que se han considerado las 2^3 combinaciones posibles.

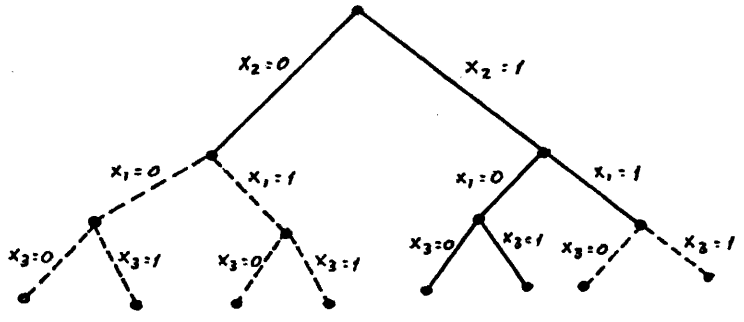


Figura 2.7.- Solución admisible de la inigualdad $3x_1 - 8x_2 + 5x_3 < -6$
$x_j = \{0,1\} \quad j = 1,2,3.$

Del razonamiento expuesto se pueden concluir dos puntos importantes para una implementación satisfactoria del método de enumeración implícita.

a) Necesariamente, el esquema de enumeración que se adopte deberá asegurar que todos los puntos del espacio solución se examinan (implícita o explícitamente) de forma no redundante.

b) Se deben de construir determinados tests de eliminación que permitan excluir como no óptimas tantas combinaciones como sea posible.

En este ejemplo, ha sido fácil mantener un control de las posibles soluciones que se iban enumerando dado que su número era reducido (únicamente 8).

En el caso general, se necesita un método eficiente y flexible que mantenga un control de todas las soluciones que se han considerado (o implícita, o explícitamente) y de las que restan de forma no redundante. Eficacia aquí indica que el esquema no debería imponer grandes restricciones (en cuanto a capacidad de memoria) en su implementación y flexibilidad en el sentido de facilidad en el almacenamiento y búsqueda de la información asociada.

2.5.3.- Esquema de enumeración de Glover ([2])

La enumeración implícita, no considera combinaciones binarias "completas", en su lugar comienza con una (o más) variable(s), estando fijadas en un valor binario, de manera que la "solución" se construye gradualmente al añadir nuevas variables con valores determinados. En el ejemplo visto, la enumeración comienza por fijar $x_2 = 1$, a continuación $x_1 = 0$, etc. En el curso de cada una de estas adiciones, se hace evidente que se pueden eliminar determinadas combinaciones sin ser consideradas de forma explícita. Esta idea es la base del esquema de enumeración de Glover, que se utilizará posteriormente para la síntesis óptima de estructuras arborescentes de MLUM(p).

Las siguientes definiciones se introducen con el fin de facilitar la presentación del esquema. En lo que sigue, la notación $+j[-j]$ se utiliza para in-

dicar que $x_j = 1$ ($x_j = 0$). Por ejemplo, la información $x_5 = 0$, $x_2 = 1$ y $x_3 = 1$ se puede representar en un vector como {-5, +2, +3}.

Definición 2.8a.- Solución parcial (J). Es un conjunto ordenado que asigna valores a un subconjunto J de N. En nuestro ejemplo, $J = \{+2, -1\}$ es una solución parcial, en la cual $x_2 = 1$ y $x_1 = 0$.

Definición 2.8b.- Variables libres (N-J). Cualquier variable que no se le asigna un valor binario en la solución parcial J es libre de tomar el valor cero o uno.

Definición 2.8c.- Compleción de J. Da una asignación completa a todas las variables en N y viene determinada por las asignaciones dadas en J, junto con una especificación binaria para cada una de las variables libres.

En el ejemplo, $J = \{+2, -1\}$ tiene dos compleciones: {+2, -1, -3} y {+2, -1, +3}.

Definición 2.8d.- Solución parcial eliminada. Una solución parcial J se elimina si todas sus compleciones se pueden descartar como no satisfactorias. En el ejemplo, $J = \{-2\}$ y $J = \{+2, +1\}$ son soluciones parciales eliminadas.

Un organigrama del esquema de enumeración de Glover se da en la Figura 2-8.

Ilustremos la mecánica del esquema utilizando el ejemplo de introducción ya expuesto. La Figura 2-9 ilustrará gráficamente los pasos del algoritmo, de manera que se pueda apreciar el efecto del esquema cuando se compara con el árbol de enumeración completa de la Figura 2-7. Para el ejemplo en cuestion, una solución se elimina únicamente si no tiene compleciones admisibles.

Debe tenerse presente que el esquema está adecuado para su implementación en una calculadora digital, así que lo que puedan parecer detalles innecesa-

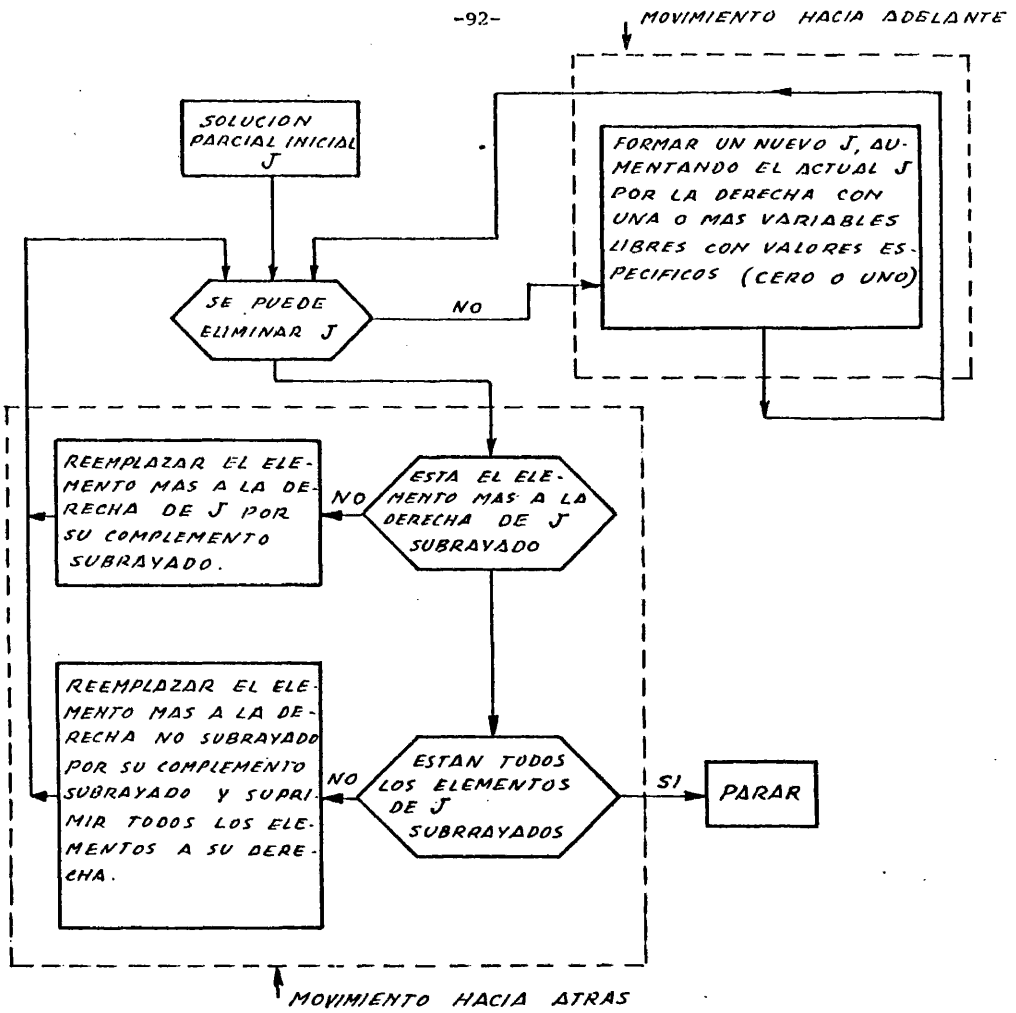


Figura 2-8.- Organigrama del esquema de enumeración de Glover

rios son útiles a la hora de plasmarlo en un programa.

Supongamos que la solución parcial inicial está vacía, esto es, $J_0 = \emptyset$. Las variables x_1 x_2 y x_3 son libres. Como $x_2 = 1$ es un paso prometedor en orden a conseguir una solución admisible, se ejecuta el movimiento hacia adelante (ver el organigrama) para producir $J_1 = \{+2\}$. Sin embargo, como el coeficiente de x_2

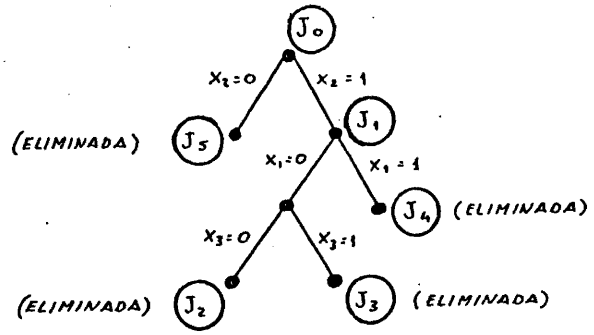


Figura 2- 9.- Algoritmo de Glover aplicado a la inigualdad $3x_1-8x_2+5x_3\leq-6$

$x_j=\{0,1\}$ $j=1,2,3$.

es más pequeño que el miembro del lado derecho en 8-6=2 unidades, para conseguir que la solución sea admisible se deberá verificar $x_1=x_3=0$. Es decir, el movimiento hacia adelante produce J_2 = {+2, -1, -3}, y J_2 se etiqueta como una solución admisible. Como J_2 representa una asignación binaria completa de las tres variables se elimina (obviamente). De acuerdo con el organigrama de la Figura se produce un movimiento de vuelta atrás J_3 = {+2, -1, $\underline{+3}$}. En términos de la Figura 2- 9, J_3 se obtiene al recorrer la rama $x_3 = 0$ hacia arriba y a continuación moviéndose hacia abajo a lo largo de la rama $x_3 = 1$.

El nombre de vuelta atrás ("backtracking") emana de la naturaleza del movimiento que se efectúa en el árbol.

El elemento subrayado nos recuerda que J_2 (en el cual $x_3 = 0$), ha sido eliminado. J_3 (que no es admisible) es una asignación completa y, por lo tanto, se elimina. El movimiento de vuelta atrás da J_4 = {+2, $\underline{+1}$}. Como el coeficiente de la variable libre x_3 es positivo, J_4 no tiene una compleción admisible y,

por lo tanto, se elimina. Esto da $J_5 = \{-\underline{2}\}$ que, asimismo, se elimina también puesto que $x_2 = 1$ es una condición necesaria para la admisibilidad. La enumeración se ha completado al estar subrayados todos los elementos de J_5.

Cabría preguntarse por qué el esquema de Glover no toma en cuenta el hecho de que x_2 debe ser igual a uno en cualquier solución admisible de la inigualdad. Podemos analizar este punto, tomando la secuencia de soluciones parciales siguientes:

$$J_1 = \{+\underline{2}\}$$
$$J_2 = \{+\underline{2}, -1, -3\}$$
$$J_3 = \{+\underline{2}, -1, +\underline{3}\}$$
$$J_4 = \{+\underline{2}, +\underline{1}\}$$

Como J_4 se elimina y todos sus elementos están subrayados, la enumeración está completa y el proceso finaliza. La idea aquí es que $J_1 = \{+\underline{2}\}$ es equivalente a asegurar que la solución parcial $\{-2\}$ ha sido eliminada, esto es, las compleciones en las cuales $x_2 = 0$ no pueden darnos una solución admisible.

El esquema de Glover examina las soluciones parciales sobre la base de una regla tipo pila. Esto se deduce de la forma en cómo son generadas las soluciones parciales. Por ejemplo, en la Figura 2-9, cuando se considera J_1, para su eliminación el esquema (implícitamente) almacena J_5 para un examen posterior. También, cuando se considera J_2, se guardan J_4 y J_3. En este punto, la lista almacenada incluye J_5, J_4 y J_3.

Aunque J_3 ha sido la última en almacenarse es la primera que se toma de la lista para testear su posible eliminación.

Veamos cómo las $8(=2^3)$ soluciones del ejemplo han sido enumeradas (implícita o explícitamente). De acuerdo con la Figura 2.9, las soluciones parciales eliminadas son J_2, J_3, J_4 y J_5. Como en un problema de n variables el número de comple-

ciones de una solución parcial de δ elementos es $2^{n-\delta}$, el número de compleciones que se enumeran en nuestro ejemplo son $2^{3-3}+2^{3-3}+2^{3-2}+2^{3-1}=8$.

2.5.4.- Algoritmo de minimización

Una vez revisado el concepto de enumeración implícita podemos formular el algoritmo siguiente (basado en el esquema de Glover) donde todas las soluciones posibles de (2.7) y (2.8) se examinan de forma implícita.

Sea S el conjunto de variables booleanas a las cuales se le dan las constantes 0 ó 1. Por ejemplo, $S = (1,\overline{2})$ significa $Y_1 = 1$ e $Y_2 = 0$.

A1) Si la solución se hace admisible al hacer ceros todas las variables Y_i, entonces ésta es la solución óptima. Si no sea

$$J_{mín} = \sum_{i=1}^{M} a_i$$

y $S = \emptyset$ (conjunto vacío).

A2) Encontrar V', el conjunto de ligaduras que se violan en V cuando la solución parcial S se completa por hacer cero todas las variables que no están en el conjunto S.

A3) Si V' no está vacío, entonces ir a A5).

A4) Si el conjunto de implicantes correspondientes a las variables $Y_i = 0$ en S y todas las variables que no están en S es compatible, ir a A10).

A5) Encontrar J_p, el valor de J' cuando S se completa por poner a cero todas las variables que no están en S. Fijar el límite de la función de coste $J_{mín}-J_p$.

A6) Almacenar en el conjunto U todas las variables que no están en el conjunto S y cuyo coeficiente es menor que el límite $J_{mín}-J_p$.

A7) Si U está vacío, ir a A13).

A8) Si todas las ligaduras en V' no pueden hacerse admisibles por añadir únicamente las variables de U entonces ir a A13).

A9) Si V' no está vacío, añadir a S la variable en U con la suma de coeficientes más grande e ir a A2).

Si V' está vacío, añadir a S' la variable V_j en U hasta que los implicantes correspondientes a las variables $V_i = 0$ en S se hacen incompatibles con los implicantes correspondientes a las variables en S'. Entonces, añadir a S la variable V_j e ir a A2).

A10) Completar la solución parcial S haciendo cero todas las variables que no están en S. Esta solución se convierte en la mejor solución $V_{mín}$ hasta el momento y el valor de la función de coste en $V_{mín}$ es el menor valor de $J_{mín}$.

A11) Localizar en S el elemento positivo más a la derecha. Reemplazarlo por su complemento (correspondiente a $V_i=0$) y suprimir todos los elementos a su derecha.

A12) Si todos los elementos que se han complementado en S son compatibles, entonces ir a A2. Si no ir a A13).

A13) Si algún elemento no está complementado entonces ir a A11) si no Parar el proceso de búsqueda. La mejor solución obtenida hasta ese momento es la óptima.

NOTA.- Las acciones A7), A8) y A11) son pasos de vuelta atrás en el esquema de enumeración implícita. En A7) y A8) la vuelta atrás tiene lugar en el momento que se observa que no se puede mejorar el valor actual de la función de coste.

Ejemplo 2.9.- Para poner de manifiesto claramente el algoritmo de minimización anterior se resuelve un ejemplo de síntesis.

Sea la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$, es decir, la mis-

ma que se ha utilizado para el método de Voith. Teniendo en cuenta los implicantes -MLUM(1) determinados ya en la tabla 2-2, podemos asociarle a cada uno de ellos el número entero a_i en función del nivel que tenga y la variable booleana y_i, tal como se indica en la Tabla 2.5.

Implicante -MLUM(1)	a_i	variable booleana Y
$I_1=\bar{x}_1(x_3)$	3	Y_1
$I_2=x_3x_1(x_0)$	1	Y_2
$I_3=x_3\bar{x}_0(\bar{x}_1)$	1	Y_3
$I_4=\bar{x}_3x_1(x_2)$	1	Y_4
$I_5=x_3\bar{x}_1(1)$	1	Y_5
$I_6=x_2\bar{x}_1(x_3)$	1	Y_6
$I_7=\bar{x}_1x_0(x_3)$	1	Y_7
$I_8=\bar{x}_2x_0(x_3)$	1	Y_8
$I_9=\bar{x}_3x_2(x_1)$	1	Y_9
$I_{10}=\bar{x}_2\bar{x}_0(x_3)$	1	Y_{10}
$I_{11}=\bar{x}_2\bar{x}_1(x_3)$	1	Y_{11}
$I_{12}=\bar{x}_3\bar{x}_1(\emptyset)$	1	Y_{12}
$I_{13}=\bar{x}_3\bar{x}_2(\emptyset)$	1	Y_{13}
$I_{14}=x_3x_0(1)$	1	Y_{14}

TABLA 2.5.- Asignación del número a_i y de la variable booleana y_i a los implicantes -MLUM(1) de la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$

Las relaciones de compatibilidad entre pares de implicantes -MLUM(1) se dan en la tabla 2.6.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}
I_2	C												
I_3	I	I											
I_4	C	C	C										
I_5	I	C	I	C									
I_6	I	C	I	C	I								
I_7	I	C	C	C	I	I							
I_8	I	I	C	I	I	C	I						
I_9	I	C	C	I	C	I	I	C					
I_{10}	I	I	I	I	I	C	C	C	C				
I_{11}	I	C	I	C	I	C	I	I	C	I			
I_{12}	I	C	C	C	C	I	I	I	I	I	I		
I_{13}	I	C	C	I	C	C	I	I	C	I	I	I	
I_{14}	I	I	C	C	I	I	I	I	C	C	I	C	C

C - compatible
I - incompatible

TABLA 2.6.- Relación de compatibilidad entre los implicantes -MLUM(1) de la función del ejemplo 2.10.

El problema de minimización dado por las ecuaciones (2.7) y (2.8) se formula en este caso de la forma siguiente:

$$\text{mín: } F = 3\,y_1 + \sum_{i=2}^{14} y_i \qquad (2.9)$$

Sujeto a las ligaduras siguientes:

$$V:\ \begin{aligned} &Y_1 + Y_5 \geq 1 \\ &Y_1 + Y_6 \geq 1 \\ &Y_1 + Y_7 \geq 1 \\ &Y_1 + Y_{11} \geq 1 \\ &Y_1 + Y_{12} \geq 1 \end{aligned} \qquad (2.10)$$

$C = \{I_i | Y_i = 0\}$ es un conjunto compatible

$Y_i = 0,1 \ (i = 1,2,\ldots,14)$

El algoritmo se resume como sigue:

Paso 1) $J_{min} = 16$, $S = \emptyset \rightarrow$

" 2) $V' \equiv V$ ir a A5)

" 3) $J_p = 0$, $J_{min} - J_p = 16$

" 4) $U = \{Y_1, Y_2, \ldots, Y_{14}\}$

" 5) $S = \{1\}$ ir a A2)

" 6) $V' \equiv \emptyset$

" 7) $J_p = 3$, $J_{min} - J_p = 13$

" 8) $U = \{Y_2, \ldots, Y_{14}\}$

" 9) $S' = \{2\}$, $S = \{1,3\}$ ir a A2)

Se sigue en este bucle hasta que en un momento intermedio de la ejecución del algoritmo se tiene:

1') $V' \equiv \emptyset$

2') $J_p = 11$, $J_{min} = 5$

3') $U = \{Y_2, Y_4, Y_5, Y_{12}, Y_{14}\}$

4') $S' = \{2,4,5,12\}$, $S = \{1,3,6,7,8,9,10,11,13,14\}$

5') $V' \equiv \emptyset$

6') ir a A10) (se sale del bucle anterior)

7') $Y_{min} = (1,\overline{2},3,\overline{4},\overline{5},6,7,8,9,10,11,\overline{12},13,14)$, $J_{min} = 12$

En este punto se tiene una solución al problema que es la mejor obtenida hasta ese instante. El conjunto de implicantes -MLUM(1) necesitados para dicha solución es $C = \{I_2, I_4, I_5, I_{12}\}$ a los que corresponde la estructura arborescente de la figura 2.10.

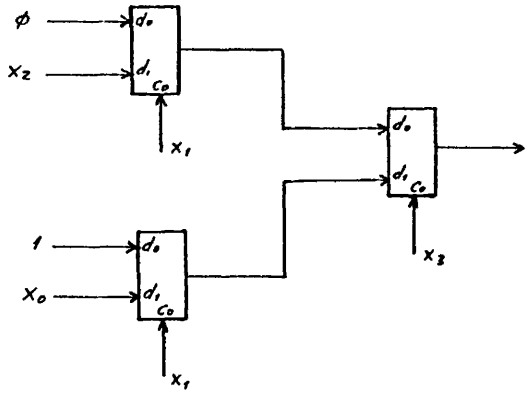


Figura 2.10.- Estructura arborescente subóptima para la función $f(x_3, x_2, x_1, x_0) = \sum(6,7,8,9,11,12,13,15)$

Prosiguiendo de este forma sobre el algoritmo se llega finalmente al siguiente resultado:

1") $V' \equiv \emptyset$

2") $J_p = 10$, J_{min}-J_p=2

3") $U = \{Y_1, Y_2, Y_4, Y_{14}\}$

4") $S' = (1,2,4)$, $S = (3,5,6,7,8,9,10,11,12,13,14)$

5") $V' = \emptyset$

6") ir a A10) (se sale del bucle)

7") $Y_{min} = (\overline{1}, \overline{2}, 3, \overline{4}, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$, $J_{min} = 11$

Esta nueva solución mejora a la obtenida anteriormente. El conjunto de implicantes -MLUM(1) que necesita es: $C = \{I_1, I_2, I_4\}$a los que corresponde la estructura arborescente de la Figura 2.11.

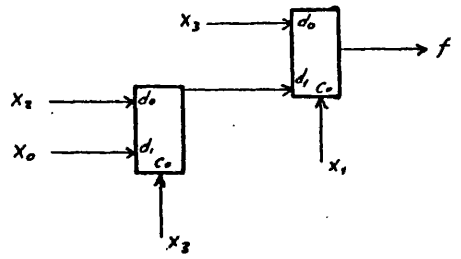


Figura 2.11.- Estructura arborescente óptima para la función $f(x_3,x_2,x_1,x_0) = \sum(6,7,8,9,11,12,13,15)$ (coincide con la Fig. 2.3, obtenida por el método de Voith)

Esta solución fue la obtenida previamente mediante el algoritmo de Voith, así que sabemos que corresponde a la estructura arborescente óptima para este ejemplo.

Si se continuase el algoritmo se vería que no se puede mejorar ya la solución que se tiene en este momento.

De la discusión de este ejemplo se observa que el algoritmo propuesto presenta la ventaja de que en cualquier momento se puede extraer una solución que si no es la óptima (pues el algoritmo no ha terminado) puede ser lo suficientemente aceptable para nuestros objetivos. En otras palabras, después de unas pocas iteraciones se puede disponer de una solución subóptima.

2.6.- GENERALIZACION DEL METODO DE ENUMERACION IMPLICITA MEDIANTE EL USO DE FUNCIONES RESIDUOS IDENTICAS

Los métodos de síntesis óptima de estructuras arborescentes de MLUM(1) que se han desarrollado, únicamente permiten el uso de funciones residuos triviales. Otra simplificación posible ocurre cuando la salida de cualquier módulo en el nivel j de la estructura arborescente, puede ser compartida por más de una

entrada de datos de módulos en el nivel $j-1$, tal como se indica en la Figura 2.12.

Definición 2.9.- Un implicante -MLUM(1) generalizado de una función f es un término producto cuyo residuo no es una función trivial.

En lo que sigue, lo representamos por $I_g = (p,g)$ donde p es el término producto y g la función residuo que dicho término implica.

Definición 2.10.- El conjunto $\mathcal{D} = \{I_{g_1},\ldots,I_{g_q}\}$ representa un conjunto de implicantes -MLUM(1) generalizados con residuo común sí y sólo si:

a) $g_1 \equiv g_2 \equiv \ldots \equiv g_q$

b) todos los términos productos contienen el mismo número de literales (es decir pertenecen al mismo nivel j)

Representemos por q el cardinal del conjunto $\mathcal{D}$, es decir el número de elementos que posee dicho conjunto.

Teorema 2.6.- El número de módulos MLUM(1) que se eliminan en la estructura arborescente al considerar un conjunto de implicantes -MLUM(1) generalizados con residuo común $\mathcal{D}_i$ es $b_i = (q_i-1)\,(2^{\ell-j}-1)$ donde ℓ corresponde al número máximo de niveles en la estructura ($\ell = n-1$).

Demostración.- La existencia de un conjunto de implicantes -MLUM(1) generalizados con residuo común de nivel j implica que no hay que considerar (q_i-1) subárboles.

El número de niveles del subárbol es $\ell-j$. Por lo tanto, el número de módulos que se eliminan será:

$$b_i = (q_i-1)\left[1 + 2 +\ldots+ 2^{\ell-j-1}\right] = (q_i-1)\,(2^{\ell-j}-1) \qquad \text{c.q.d.}$$

(2.11)

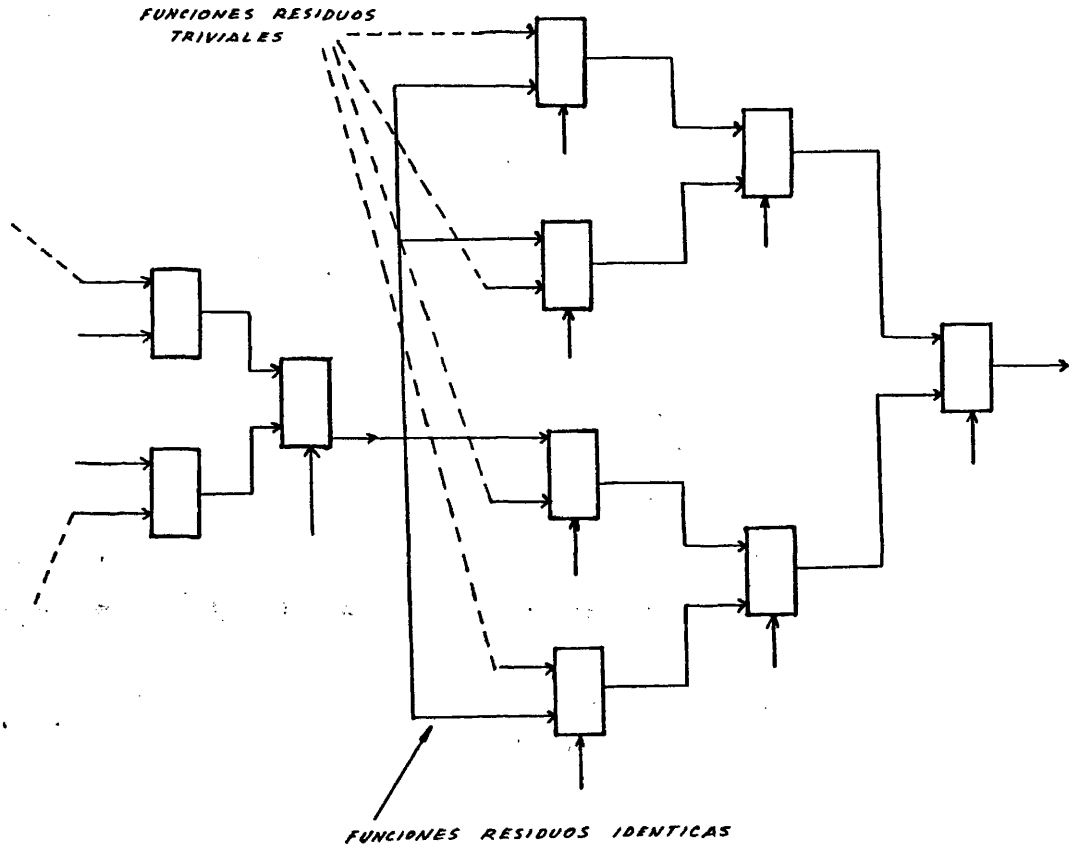


Figura 2.12.- Simplificación en el número de módulos necesitados en la estructura arborescente cuando existen funciones residuos multivariables idénticas.

Hay que determinar, pues, el conjunto $\mathcal{D}$ de implicantes -MLUM(1) generalizados con residuo común en el caso de que existan.

En este caso, el método tabular de Voith es poco apropiado para efectuar el cálculo de dichos conjuntos, sin embargo, el método de las cartas de descomposición resulta directo pues únicamente se trata de determinar sobre las distintas cartas de descomposición si existen o no, filas o columnas idénticas.

Ejemplo 2.10.- Sea la función de 4 variables

$$f(x_3,x_2,x_1,x_0) = \sum(7,11,12,13,14,15)$$

En las cartas de descomposición de la figura 2.13 se observan los dos conjuntos de implicantes MLUM(1) generalizados con residuos comunes siguientes:

$$\mathcal{D}_1 = \{(\bar{x}_3 x_2, x_1 x_0) \ , \ (x_3\bar{x}_2, x_1x_0)\}$$

$$\mathcal{D}_2 = \{(\bar{x}_1\bar{x}_0, x_3x_2) \ , \ (\bar{x}_1x_0, x_3x_2) \ , \ (x_1\bar{x}_0, x_3x_2)\}$$

Sin embargo, el $\mathcal{D}_2$ al tener 3 elementos, da lugar a que todos los subconjuntos que contengan dos elementos deban ser considerados también. Es decir,

$$\mathcal{D}_3 = \{(\bar{x}_1\bar{x}_0, x_3x_2) \ , \ (\bar{x}_1x_0, x_3x_2)\}$$

$$\mathcal{D}_4 = \{(\bar{x}_1\bar{x}_0, x_3x_2) \ , \ (x_1\bar{x}_0, x_3x_2)\}$$

$$\mathcal{D}_5 = \{(\bar{x}_1x_0, x_3x_2) \ , \ (x_1\bar{x}_0, x_3x_2)\}$$

Si el número de funciones residuos idénticas es m, entonces existirán

$$\sum_{i=2}^{m} \binom{m}{i} = 2^m - (m+1)$$

conjuntos de implicantes MLUM(1) generalizados con residuos comunes. En nuestro

ejemplo, los 4 conjuntos D_2, D_3, D_4 y D_5 están relacionados con la misma función residuo, pues $m = 3$.

No obstante, únicamente uno de estos $2^m-(m+1)$ conjuntos puede contribuir activamente en el procedimiento de minimización.

El procedimiento de síntesis óptima de estructuras arborescentes formulado en la sección anterior se puede generalizar fácilmente al caso en que se presentan funciones residuos idénticas. Unicamente habrá que tomar en cuenta las posibles ligaduras adicionales debidas al hecho de que D_k pueda contener (subsumir) a I_j o a D_j.

$$\min J' = \sum_{i=1}^{M} a_i Y_i + \sum_{i=1}^{N} b_i Z_i \qquad (2.12)$$

sujeto a las siguientes ligaduras:

$$\begin{aligned} V: \quad & Y_j + Y_k \geq 1 && \forall\ I_j \subseteq I_k \\ & Y_j + Z_k \geq 1 && \forall\ I_j \subseteq D_k \\ & Z_j + Z_k \geq 1 && \forall\ D_j \subseteq D_k \\ & Z_{i_1} + \ldots + Z_{i_\delta} > \delta - 1 && \end{aligned} \qquad (2.13)$$

$D_{i_1},\ldots,D_{i_\delta}$ están relacionadas con la misma función residuo.

$C \neq \{(I_i, C_j) \mid Y_i=0 \text{ y } Z_j = 0\}$ es un conjunto compatible

$Y_i = 0,1$ y $Z_j = 0,1$ $(i=1,\ldots,M$ y $j=1,\ldots,N)$

donde Y_i es la variable booleana asociada al implicante -MLUM(1) I_i y Z_j es la variable booleana asociada al conjunto de implicantes -MLUM(1) generalizado con residuo común D_j.

Para determinar la solución óptima de las ecuaciones (2.12) y (2.13),

CARTAS DE DESCOMPOSICION PARA 4 VARIABLES

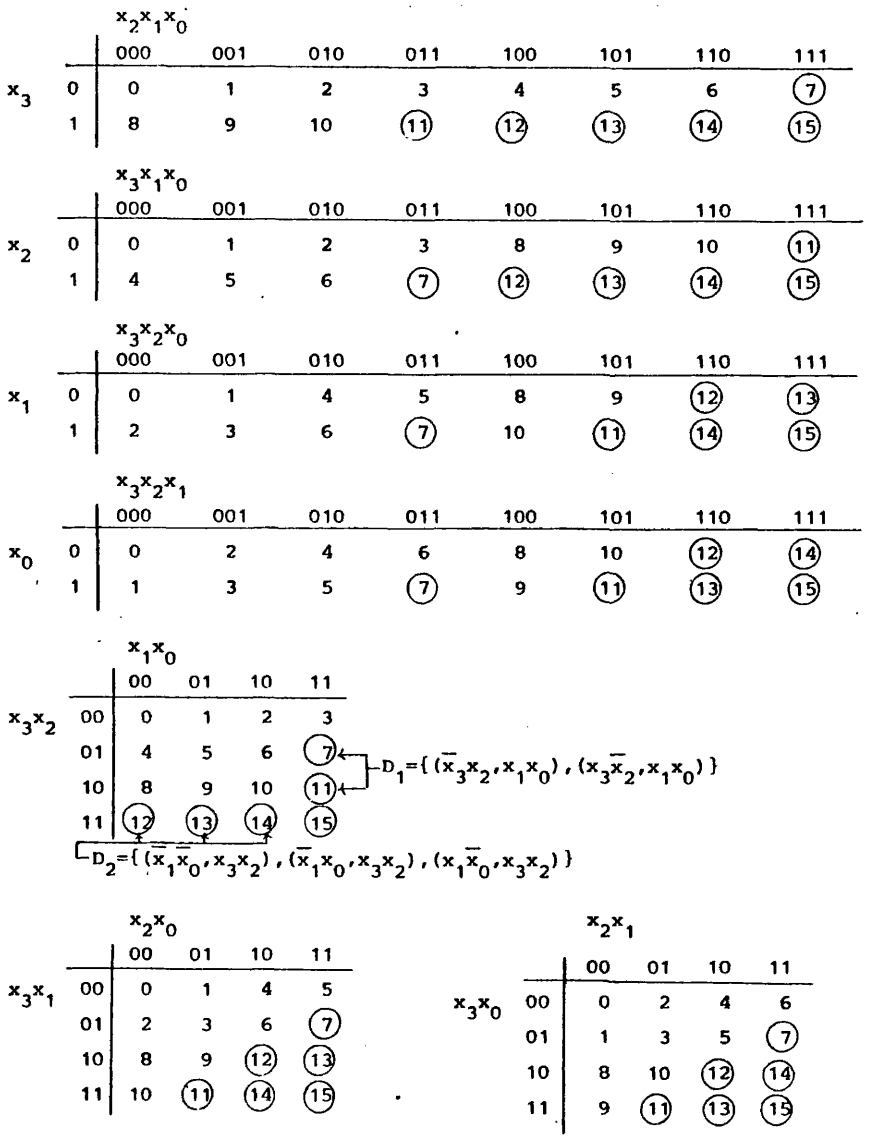


Figura 2.13.- Generalización del método de enumeración implícita mediante el uso de funciones residuos idénticas $f(x_3, x_2, x_1, x_0) = \sum(7, 11, 12, 13, 14, 15)$

se puede aplicar directamente el método de enumeración implícita visto en la sección anterior, simplemente reemplazando $Y_{M+j} = Z_j$.

Ejemplo 2.11.- Si se lleva a cabo el procedimiento reseñado con la función de conmutación del ejemplo anterior se obtiene el siguiente conjunto compatible:

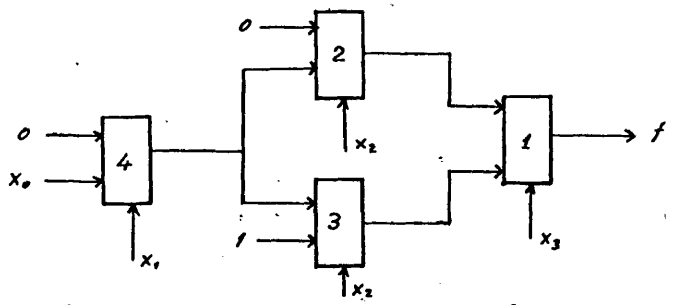


Figura 2.14.- Estructura arborescente óptima de la función $f(x_3,x_2,x_1,x_0) = \sum(7,11,12,13,14,15)$ obtenida mediante el empleo de funciones residuos idénticas por el método de enumeración implícita.

$$C = \{I_1, I_6, D_1\}$$

donde $I_i = \bar{x}_3\bar{x}_2(0)$, $I_6 = x_3x_2(1)$ y D_1 está ya definido.

Al conjunto C le corresponde la estructura arborescente óptima de la Figura 2.14.

Se observa que el módulo número 4 que implementa al conjunto de implicantes -MLUM(1) generalizado con residuo común D_1, tiene su salida conectada a entradas de datos de los módulos números 2 y 3.

2.7.- SINTESIS DE ESTRUCTURAS ARBORESCENTES CUASIOPTIMAS POR UN PROCEDIMIENTO DE MINIMIZACION POR NIVEL

La filosofía del procedimiento de minimización por nivel se basa en elegir las variables de expansión de forma secuencial nivel por nivel, y no de forma

global, tal como se hacía en los métodos de síntesis óptima. El criterio que se fija para la elección de las variables de expansión, es el de escoger aquéllas que dan un mayor número de funciones residuos triviales, en el nivel que estemos considerando. Esto lleva consigo que los subárboles que emanarían de dichas entradas de datos no tienen que considerarse y habría un ahorro en los módulos necesarios para implementar la función. Así pues, las p variables de control de los MLUM(p), se seleccionan por niveles comenzando por el primer nivel y, una vez escogidas, se consideran ya fijas y no aparecen en la elección de los niveles subsiguientes.

Ejemplo 2.12.- Sea la función $f(x_3,x_2,x_1,x_0) = \sum(7,8,10,12,13,14)$ que se desea sintetizar con MLUM(1).

En primer lugar hay que determinar las funciones residuos que se producen cuando la variable del primer nivel es $x_i (i = 0,1,2,3)$, ver Figura 2.15a. Se observa que escogiendo x_0 como variable del primer nivel existe una función residuo trivial

$$f(x_3,x_2,x_1,x_0) = x_0\, g_1(x_3,x_2,x_1) + \bar{x}_0(x_3)$$

$$g_1(x_3,x_2,x_1) = x_2(\bar{x}_3 x_1 + x_3\bar{x}_1)$$

Así pues, la síntesis queda tal como refleja la Figura 2.15b.

Para estructuras arborescentes de dos niveles, el procedimiento es exhaustivo y garantiza la solución óptima, pero cuando la estructura tiene más niveles, aunque el método no garantiza la minimización del número de módulos (en general) la solución que se obtiene es cuasi-óptima. Su principal ventaja estriba en que se reducen considerablemente los tiempos de cálculo si se compara con un método exhaustivo de enumeración completa en el cual se tengan en cuenta todas las posibles permutaciones de las variables de control.

CARTAS DE DESCOMPOSICION PARA 4 VARIABLES

x_3	$x_2x_1x_0$ 000	001	010	011	100	101	110	111
0	0	1	2	3	4	5	6	(7)
1	(8)	9	(10)	11	(12)	(13)	(14)	15

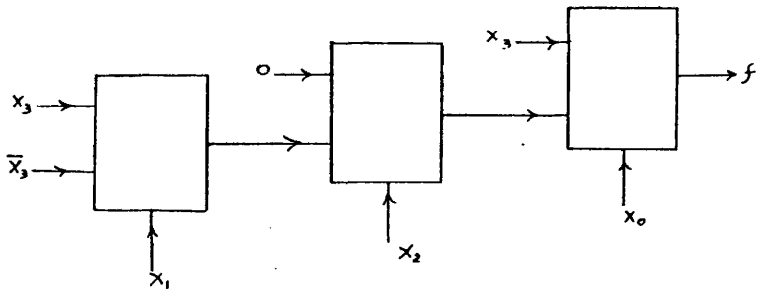
x_2	$x_3x_1x_0$ 000	001	010	011	100	101	110	111
0	0	1	2	3	(8)	9	(10)	11
1	4	5	6	(7)	(12)	(13)	(14)	15

x_1	$x_3x_2x_0$ 000	001	010	011	100	101	110	111
0	0	1	4	5	(8)	9	(12)	(13)
1	2	3	6	(7)	(10)	11	(14)	15

x_0	$x_3x_2x_1$ 000	001	010	011	100	101	110	111
0	0	2	4	6	(8)	(10)	(12)	(14)
1	1	3	5	(7)	9	11	(13)	15

a)

función residuo trivial (se escoge x_0 como variable del primer nivel)



b)

Figura 2.15.- Síntesis de la función $f(x_3,x_2,x_1,x_0) = \sum(7,8,10,12,13,14)$ por el procedimiento de minimización por nivel.

Para demostrar la reducción en los cálculos, sea $q = n-1$ el número total de variables de control en una estructura arborescente que implementa una función arbitraria f de n variables.

Despreciando las permutaciones de variables de control dentro de un nivel dado (ya que no tienen ningún efecto sobre la complejidad de ese nivel) el número de permutaciones utilizando enumeración completa, viene dado por:

$$P_e = n \prod_{i=0}^{\ell-2} \binom{q-ip}{p} = n\,(P_1 \cdot P_2 \cdot \ldots \cdot P_{\ell-1}) \qquad (2\text{-}14)$$

donde P_j es el número de permutaciones necesarias en el nivel j.

Si el número de variables de control es el mismo en todos los niveles, se puede ver fácilmente que:

$$P_e = n \frac{q!}{(p!)^{\ell}} \qquad (2.15)$$

Utilizando la técnica de minimización por nivel que se propone, el número mínimo de permutaciones viene dado por:

$$P_{mín} = n\,P_1 + P_2 + \ldots + P_{\ell-1} \qquad (2.16)$$

Esto representa el caso cuando es posible hacer una elección definida de las variables de control en cada nivel. En el caso de que exista más de una elección posible, se puede seguir uno de los dos criterios siguientes.

a) Llevarlas en paralelo hasta que en un nivel posterior podamos distinguir cuál es la que garantiza una mayor reducción.

b) Escoger aquélla que da lugar a un número mayor de funciones residuos que sean constantes lógicas (lo cual minimiza los problemas de fan-in que se puedan presentar).

Si por el contrario no se consigue ningún ahorro en el primer nivel, no podemos fijar las variables de control de esta etapa. Así, el próximo conjunto de permutaciones se tomará simultáneamente sobre el primer y segundo niveles. Esto aumentará el número de permutaciones a:

$$P = n\,P_1 + n\,P_{12} + P_3 + \ldots + P_{\ell-1} \qquad (2.17)$$

donde:

$$P_{12} = \frac{q!}{(q-2p)!\,2p!} \qquad (2.18)$$

Este procedimiento se adopta en todos los niveles subsiguientes. El caso más desfavorable ocurre si no es posible ninguna minimización (existencia de funciones residuos triviales) o únicamente es posible en el nivel $\ell-1$, lo que resulta en el número máximo de permutaciones siguientes:

$$P_{max} = n\,(P_1 + P_{12} + P_{123} + \ldots + P_{12\ldots\ell-1}) \qquad (2.19)$$

donde:

$$P_{12\ldots\ell_k} = \frac{q!}{(q\ kp)!\,(kp)!} \qquad (2.20)$$

La tabla 2-7 muestra para el caso de utilizar MLUM(2) ($p=2$)que incluso, este número máximo de permutaciones es mucho menor que el requerido por una búsqueda exhaustiva.

Es precisamente esta reducción tan drástica que se observa, la que permite una reducción considerable en los tiempos de cálculo frente a los que se obtienen en el método exhaustivo de enumeración completa que tiene en cuenta todas las posibles permutaciones de las variables de control.

UNIVERSIDAD COMPLUTENSE DE MADRID
BIBLIOTECA

N° de variables n	Enumeración completa P_e	Minimización por nivel $P_{máx}$	$P_{mín}$
5	30	30	30
7	630	210	111
9	22680	1134	273
11	1247400	5610	544
13	97297200	26598	952

TABLA 2-7.- número de permutaciones de las variables de control utilizando MLUM(2)

En el caso de funciones de 4 variables sintetizadas con MLUM(1), el siguiente teorema demuestra que la minimización por nivel es óptima.

Teorema 2-7.- Para n=4, la síntesis de estructuras arborescentes con MLUM(1) por el método de minimización por nivel es óptima.

Demostración

El único caso que en principio puede plantear problema es el de la figura 2.16, es decir, que existe otro orden de expansión de las variables que disminuye el número de módulos necesitados.

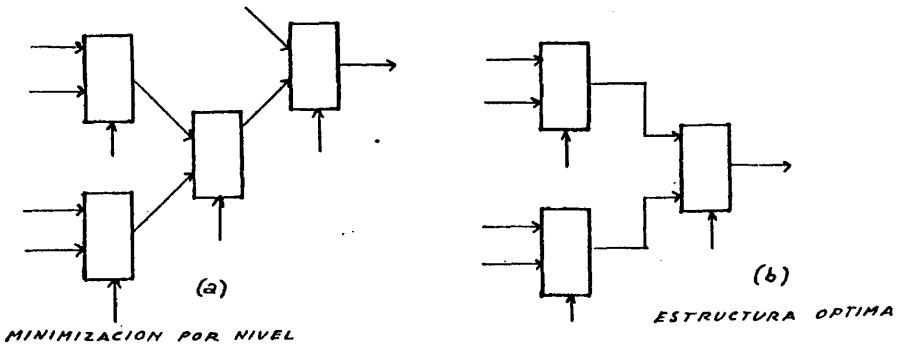


Figura 2.16.- Unico caso en que la minimización por nivel no daría lugar a la estructura óptima.

Vamos a ver que tal posibilidad no se puede presentar nunca. Sin pérdida de generalidad, supongamos que en la minimización por nivel, la variable x_3 se utiliza en el primer nivel

$$f(x_3,x_2,x_1,x_0) = \bar{x}_3\, g_1(x_2,x_1,x_0) + x_3\, g_2\,(x_2,x_1,x_0)$$

donde $g_1(x_2,x_1,x_0)$ es una función residuo trivial $\begin{cases} 0 \\ 1 \\ x_j \\ \bar{x}_j \end{cases}$ $j = 0,1,2$

$g_2(x_2,x_1,x_0)$ es una función de tres variables que requiere para su síntesis 3MLUM(1) (es decir, no admite simplificación en la estructura canónica).

Hay que considerar 4 casos según sea el valor de la función residuo trivial g_1

Caso a.- $g_1(x_2,x_1,x_0) = 0 \Rightarrow f(x_3,x_2,x_1,x_0) = x_3\, g_2(x_2,x_1,x_0)$

Con el fin de intentar reducir el número de módulos expandimos con respecto a cualquier variable que no sea x_3 (por ejemplo, x_2)

$$f(x_3,x_2,x_1,x_0) = \bar{x}_2[x_3 h_1(x_1,x_0)] + x_2\,[x_3 h_2(x_1,x_0)] = \bar{x}_2\hat{h}_1(x_3,x_1,x_0) + x_2\hat{h}_2(x_3,x_1,x_0)$$

donde h_1 y h_2 no son funciones triviales, porque si no g_2 no requeriría 3MLUM(1) para su síntesis, en contra de la hipótesis, de aquí se deduce que $\hat{h}_1$ y $\hat{h}_2$ necesitan al menos 2MLUM(1) cada una para su síntesis y, por lo tanto, el coste se hace así superior al obtenido en la minimización por nivel.

Caso b.- $g_1(x_2,x_1,x_0) = 1 \Rightarrow f(x_3,x_2,x_1,x_0) = x_3\, g_2(x_2,x_1,x_0) + \bar{x}_3$

Este caso se puede reducir fácilmente al caso anterior. Según sabemos ya,conoci-

da la estructura óptima arborescente de una función, se conoce la de su función complementaria sin más que complementar sus entradas de datos.

$$\overline{f}(x_3,x_2,x_1,x_0) = x_3\,\overline{g}_2(x_2,x_1,x_0)$$

que corresponde al caso a.

Caso c.- $g_1(x_2,x_1,x_0) = x_2$ (se toma x_2 sin pérdida de generalidad) $\Rightarrow f(x_3,x_2,x_1,x_0) =$

$$= \overline{x}_3\,x_2 + x_3\,g_2(x_2,x_1,x_0)$$

Se supone que la minimización por nivel tomando como primera variable de expansión x_2 da una estructura similar (en caso de que el coste fuese menor correspondería a la estructura óptima).

Efectuemos la expansión de f en el primer nivel respecto a x_1.

$$f(x_3,x_2,x_1,x_0) = \overline{x}_1\left[\overline{x}_3x_2+x_3h_1(x_2,x_0)\right] + x_1\left[\overline{x}_3x_2+x_3h_2(x_2,x_0)\right] = \overline{x}_1\hat{h}_1(x_3,x_2,x_0) +$$

$$+ x_1\hat{h}_2(x_3,x_2,x_0)$$

donde, como antes, h_1 y h_2 no son funciones triviales. Por un razonamiento análogo al anterior $\hat{h}_1$ y $\hat{h}_2$ requieren como mínimo 2MLUM(1) cada una para su síntesis y se obtiene así un coste superior al obtenido por la minimización por nivel.

Caso d.- $g_1(x_2,x_1,x_0) = \overline{x}_2 \Rightarrow f(x_3,x_2,x_1,x_0) = \overline{x}_3\overline{x}_2 + x_3g_2(x_2,x_1,x_0)$

Bajo las mismas condiciones del caso c), se puede reducir a él sin más que complementar la función tomando como variable de expansión en el primer nivel x_3.

$$\overline{f}(x_3,x_2,x_1,x_0) = \overline{x}_3x_2 + x_3\overline{g}_2(x_2,x_1,x_0)$$

2.8.- ALGUNAS CONSIDERACIONES SOBRE LA SINTESIS DE ESTRUCTURAS GENERALIZADAS:

IMPORTANCIA DE LA DESCOMPOSICION FUNCIONAL

Supongamos que se desea sintetizar la función siguiente:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = x_{n-1} \oplus x_{n-2} \oplus \ldots \oplus x_1 \oplus x_0 \qquad (2.21)$$

con MLUM(p) en una estructura arborescente. Si se aplica cualquiera de los procedimientos dados previamente, se observa que la estructura arborescente óptima requiere exactamente $N_{max} = k^{\ell}-1/k-1$ con $k=2^p$, es decir, no existe posibilidad de eliminar ningún MLUM(p) en la estructura. Parece, pues, natural preguntarse si no existiría una estructura generalizada (ya definida en el capítulo anterior) que nos mejore la síntesis.

Para fijar ideas, estudiemos el caso particular siguiente $n=7$ y $p=1$. La función f se puede definir en este caso iterativamente mediante las siguientes expresiones:

$$f(x_6, x_5, \ldots, x_0) = x_6 \oplus \varphi_5$$

$$\varphi_i = x_i \oplus \varphi_{i-1} \qquad i = 5,4,3,2,1,0 \qquad (2.22)$$

que se pueden sintetizar como indica la estructura generalizada de la Figura 2.17.

En este caso, frente a los 127 MLUM(1) de la estructura arborescente, únicamente son necesarios 6MLUM(1) en la estructura generalizada. En general, la síntesis de (2.22) en una estructura generalizada del tipo descrito con MLUM(p) solamente necesita $\left\lceil \frac{n-1}{p} \right\rceil$ MLUM(p).

Desgraciadamente, y a pesar de nuestros esfuerzos hasta la fecha, no hemos conseguido obtener un procedimiento sistemático de síntesis óptima de estructuras generalizadas. Sin embargo, hemos detectado que la descomposición funcional booleana se puede utilizar como ayuda para mejorar la síntesis producida por las

estructuras arborescentes, ya que conducen, de forma natural, a la aparición de estructuras generalizadas.

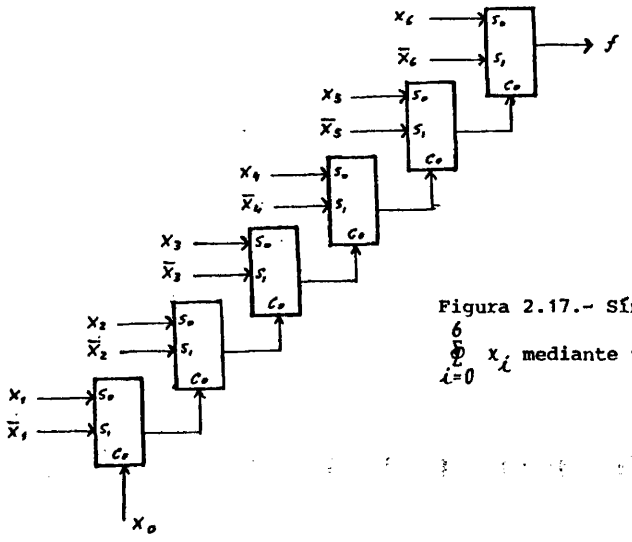


Figura 2.17.- Síntesis de $f(x_0,x_1,\ldots,x_6) = \sum_{i=0}^{6} x_i$ mediante una estructura generalizada

Ilustremos este punto con un ejemplo.

Ejemplo 2.13.- Se desea sintetizar la siguiente función de conmutación con MLUM(1).

$$f(x_4,x_3,x_2,x_1,x_0) = x_3x_2\bar{x}_0 + x_3x_1x_0 + x_4\bar{x}_1x_0 + x_4\bar{x}_2\bar{x}_0 =$$

$$= \sum(11,12,14,15,16,17,18,21,24,25,26,27,28,29,30,31)$$

La estructura arborescente óptima se observa en la Figura 2.18 a). Se necesitan, pues, 3 MLUM(1).

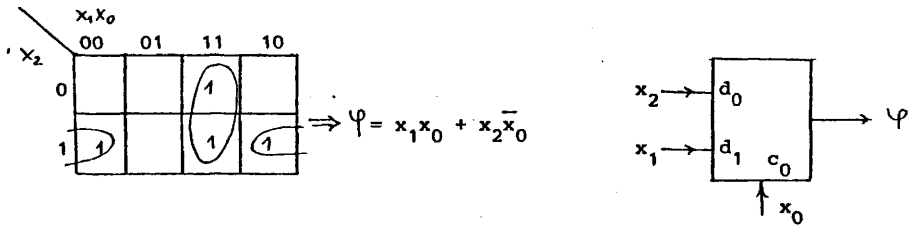
Sin embargo, $f(X)$ tiene una descomposición simple disjunta (d.s.d.):

$f(X) = F(X_1, \varphi(X_2))$ donde $X = \{x_4, x_3, x_2, x_1, x_0\}$; $X_1 = \{x_4, x_3\}$; $X_2 = \{x_2, x_1, x_0\}$

tal como se observa en la carta de descomposición adjunta:

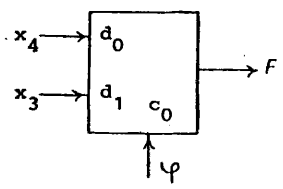
x_4x_3 \ $x_2x_1x_0$	000	001	010	011	100	101	110	111
00	0	1	2	3	4	5	6	7
01	8	9	10	(11)	(12)	13	(14)	(15) → φ
10	(16)	(17)	(18)	19	20	(21)	22	23
11	(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)

Como $\nu=2$ (ν = multiplicidad de columna = número de vectores columnas distinto en la carta de descomposición) admite d.s.d.



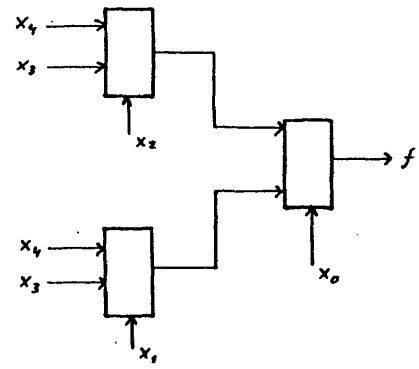
Por su parte F de la carta de descomposición se puede expresar como:

$$F = F(x_4, x_3, \varphi) = \bar{x}_4 x_3 \varphi + x_4 \bar{x}_3 \bar{\varphi} + x_4 x_3 = \bar{\varphi}\, x_4 + \varphi\, x_3$$

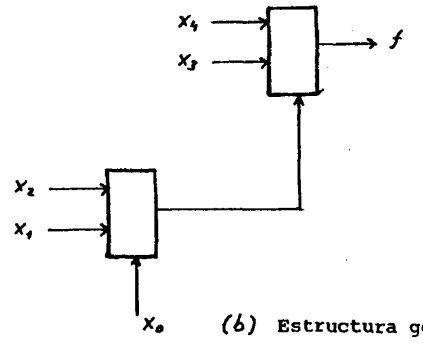


Así pues, la estructura resultante es la de la figura 2-18b) que únicamente requiere 2 MLUM(1).

Con el fin de poner de manifiesto, de una manera sistemática, la impor-



(a) Estructura arborescente óptima



(b) Estructura generalizada

Figura 2.18.- Síntesis del ejemplo 2.13

tancia de la descomposición funcional en la síntesis de estructuras generalizadas, hemos procedido a un análisis minucioso y exhaustivo de su incidencia en el caso de funciones de 3 y 4 variables implementadas con MLUM(1).

Como ya sabemos, únicamente nos tendremos que centrar en las clases de equivalencia n-p-n (14 para las funciones de 3 variables y 222 para las de 4). En el caso de 3 variables, los resultados conseguidos permiten asegurar que el catálogo que se presenta es el óptimo para todas las clases de equivalencia n-p-n y en el caso de 4 variables se demuestra otro hecho verdaderamente interesante y es que nunca son necesarios utilizar el número máximo de MLUM(1) de la estructura canónica (7 para 4 variables).

2.9.- SINTESIS OPTIMA MEDIANTE MLUM(1) DE LAS CLASES DE EQUIVALENCIA n-p-n DE FUNCIONES DE 3 VARIABLES

En el capítulo anterior, en la tabla 1-1 , se daban las clases de equivalencia n-p-n para funciones de 3 variables, así como la función representativa de la clase de equivalencia.

La estructura canónica arborescente de cualquier función de 3 variables realizada con MLUM(1) es la de la Figura 2.19.

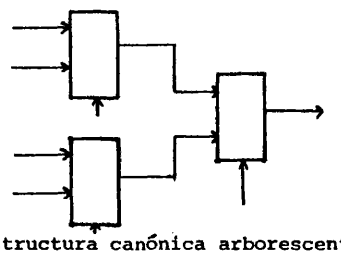


Figura 2.19.- Estructura canónica arborescente de una función de 3 variables

Si se empleasen los procedimientos de síntesis óptima de estructuras arborescentes discutidos previamente, se obtendrían para las distintas clases de equivalencia uno de los 5 tipos siguientes (ver Figura 2.20).

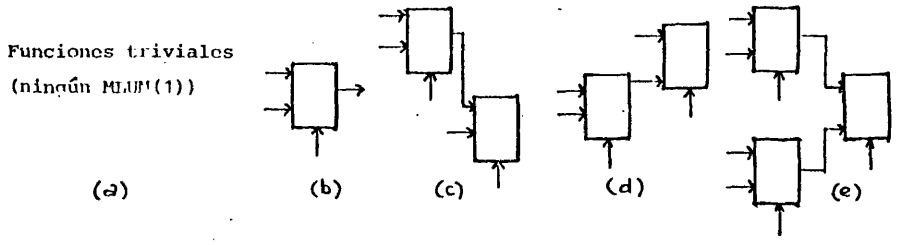


Figura 2.20.- Tipos de estructuras arborescentes en la síntesis de funciones de 3 variables

Es fácil darse cuenta que las estructuras a), b), c) y d) son óptimas de forma absoluta, puesto que no podremos formar ninguna estructura generalizada con menor coste. Sin embargo, no sucede lo mismo con la estructura del tipo e), ya que puede existir una estructura generalizada (ver Figura 2.21) que realice la función y requiere únicamente 2 MLUM(1).

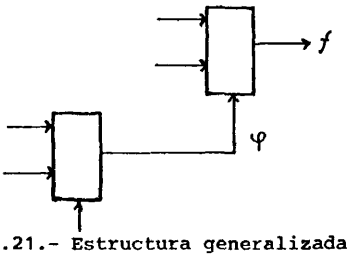


Figura 2.21.- Estructura generalizada con dos MLUM(1)

El problema se queda reducido a analizar aquellas clases de equivalencia n-p-n para las cuales la síntesis óptima de estructuras arborescentes no da lugar a ningún ahorro en el número de módulos.

Desde el punto de vista de la teoría de descomposición funcional booleana, la estructura generalizada de la Figura 2.21 se puede representar por:

$$f(X) = F(X_0, \varphi(X_1)); \quad X_0 \cup X_1 = X$$

con la condición de que la síntesis de las funciones $\varphi(X_1)$ y $F(X_0, \varphi)$ requieran únicamente 1 MLUM(1) cada una.

En principio, son posibles dos situaciones:

a) $X_0 \cap X_1 = \emptyset$ y $card(X_1) = 2 \Rightarrow$ *d.s.d.*

b) $X_0 \cap X_1 \neq \emptyset$ y $card(X_1) = 3 \Rightarrow$ *d.s.n.d.* (descomposición simple no disjunta)

Se trata, pues, de determinar en aquellos casos en que la estructura arborescente es del tipo de la Figura 2.20e) si la función representativa de la clase de equivalencia admite una d.s.d. o una d.s.n.d. que den lugar a la estructura generalizada de la Figura 2.21.

Después de aplicar el procedimiento de síntesis óptima de estructuras arborescentes a las 14 clases de equivalencia n-p-n, se observa que las clases de equivalencia números 7,8,9 y 14 no admiten simplificación.

De estas 4 clases de equivalencias n-p-n en dos de ellas (las números 8 y 14) su función representativa admite, o bien una d.s.n.d. o una d.s.d. del tipo descrito.

Así, la función representativa de la clase de equivalencia n-p-n número 8 es:

$$\delta_8 = \delta_8(x_2, x_1, x_0) = \sum(3,5,6,7) = x_1x_0 + x_2(x_1 + x_0)$$

Después de una serie de manipulaciones podemos expresar δ_8 en la forma siguiente:

$$\delta_8 = F(x_2, x_1, \varphi(x_2, x_1, x_0)) \qquad (2.23)$$

con $\varphi = \overline{x}_0 x_2 + x_0 x_1$

$$F = \overline{\varphi}\, x_2 + \varphi x_1 \qquad (2.24)$$

A las que corresponden las siguientes implementaciones con MLUM(1) (Ver Figura 2.22a).

Que se combinan para dar la estructura generalizada de la Figura 2.22b).

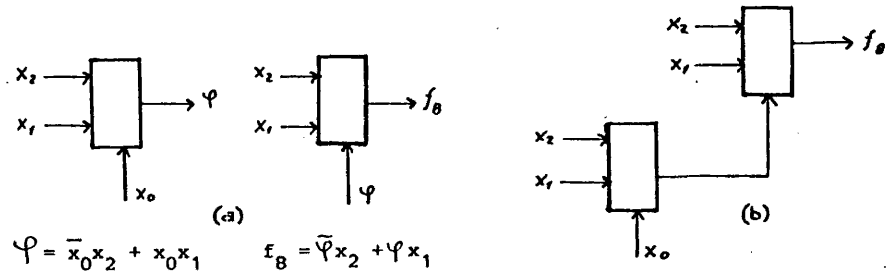


Figura 2.22.- Síntesis de la función $f_8(x_2,x_1,x_0) = \sum(3,5,6,7)$

En este caso, (2.23) corresponde a una d.s.n.d. En el caso de la clase de equivalencia n-p-n número 14, su función representativa resulta ser:

$$f_{14} = f_{14}(x_2,x_1,x_0) = x_2 \oplus x_1 \oplus x_0 \qquad (2.25)$$

Esta función es del tipo que analizamos en la introducción del apartado anterior

$$f_{14} = F(x_2,\varphi(x_1,x_0)) \qquad (2.26)$$

con $\varphi = x_1 \oplus x_0$

$F = x_2 \oplus \varphi$ (2.27)

A las que corresponden la realización siguiente con MLUM(1) (ver Figura 2.23a). Que se combinan para dar la estructura generalizada de la Figura 2.23b).

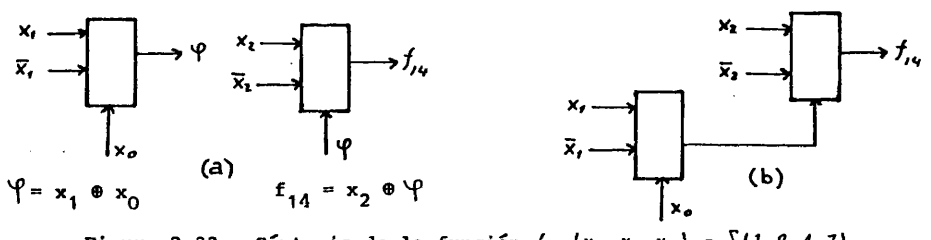


Figura 2.23.- Síntesis de la función $f_{14}(x_2,x_1,x_0) = \sum(1,2,4,7)$

En este caso, (2.26) corresponde a una d.s.d.

En la Figura 2.24 están representadas la síntesis óptima mediante MLUM(1) de las funciones representativas de las 14 clases de equivalencia n-p-n. El número medio de MLUM(1) utilizados para sintetizar cualquier función de 3 variables resulta ser $\overline{n} = 1.82$.

Ejemplo 2.14.- En el Ejemplo 1.6 del capítulo anterior se había obtenido la síntesis de la Figura 1.15e, con un coste de 7 MLUM(1). Ahora estamos en condiciones de efectuar una realización más económica, teniendo en cuenta que las subfunciones $g_1(X')$, $h_1(X'')$ y $h_2(X''')$ son funciones de 3 variables.

$f(x) = \overline{g_1(X')}\ h_2(X''') + g_1(X')\ h_1(X'')$

$g_1(X') = x_5\,\overline{x}_2 + (\overline{x}_5 x_0 + x_5\overline{x}_0)\,x_2 \in \{12\}$ cuyo coste es de 2 MLUM(1)

$h_1(X'') = x_4x_3\overline{x}_1 + \overline{x}_4x_3x_1 + x_4x_1 \in \{8\}$ cuyo coste es de 2 MLUM(1)

$h_2(X''') = x_3\overline{x}_4 + x_1x_4 \in \{11\}$ cuyo coste es de 1 MLUM(1)

Todavía es posible un ahorro adicional, puesto que la síntesis de $h_2(X'')$ se puede extraer de la función $h_1(X'')$, tal como se observa en la Figura 2.25c).

2.10.- SINTESIS CUASI-OPTIMA MEDIANTE MLUM(1) DE LAS CLASES DE EQUIVALENCIA N-P-N DE FUNCIONES DE 4 VARIABLES

El número de clases de equivalencia n-p-n para funciones de 4 variables es 222 y la estructura canónica arborescente de cualquier función de 4 variables, realizada con MLUM(1), es la de la Figura 2.26.

Un análisis, en este caso, siguiendo una línea de razonamiento idéntica a la empleada con funciones de 3 variables resultaría muy problemática y, en

SINTESIS OPTIMA CON MLUM(1)

FUNCION		N°de funciones en la clase de equivalencia n-p-n
f_1	TRIVIAL	2
f_2		16
f_3		24
f_4		24
f_5		48
f_6	TRIVIAL	6
f_7		16
f_8		8
f_9		8
f_{10}		48
f_{11}		24
f_{12}		24
f_{13}		6
f_{14}		2

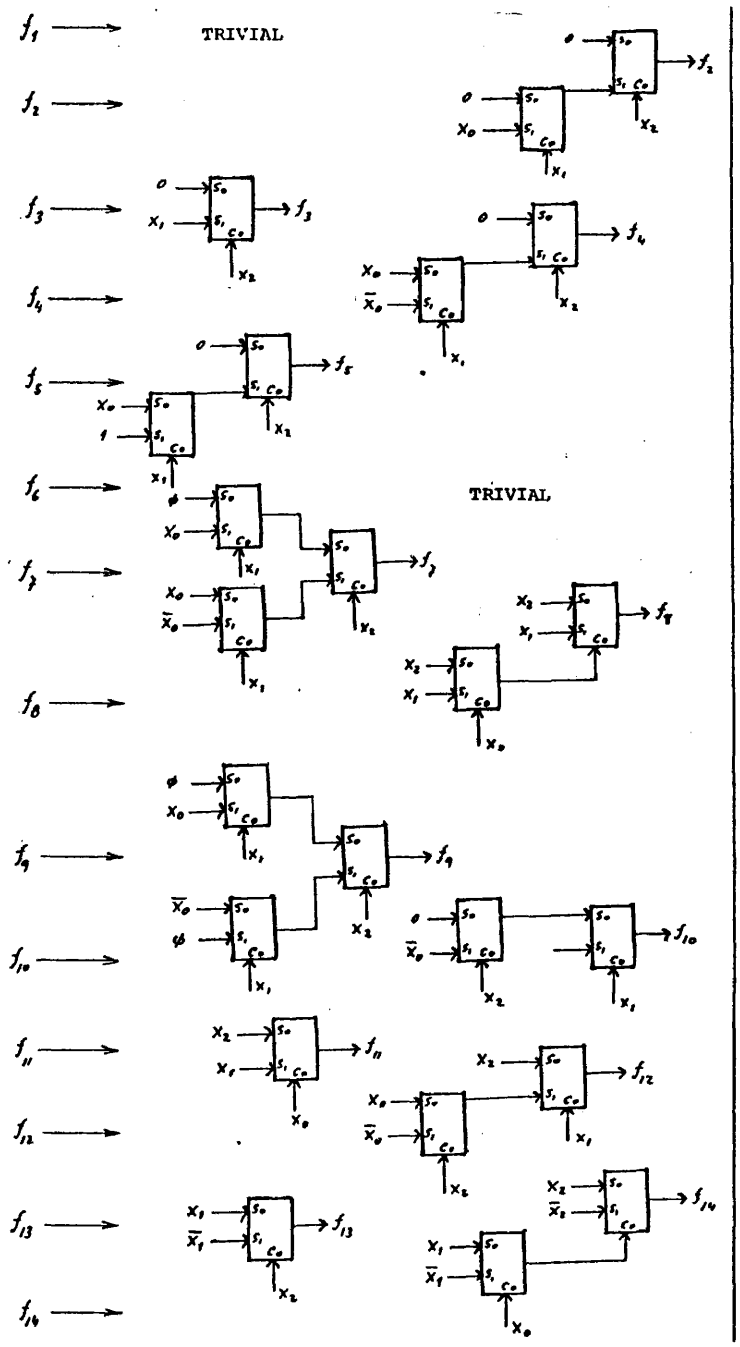


Figura 2.24.- Síntesis óptima mediante MLUM(1) para clases de equivalencia n-p-n

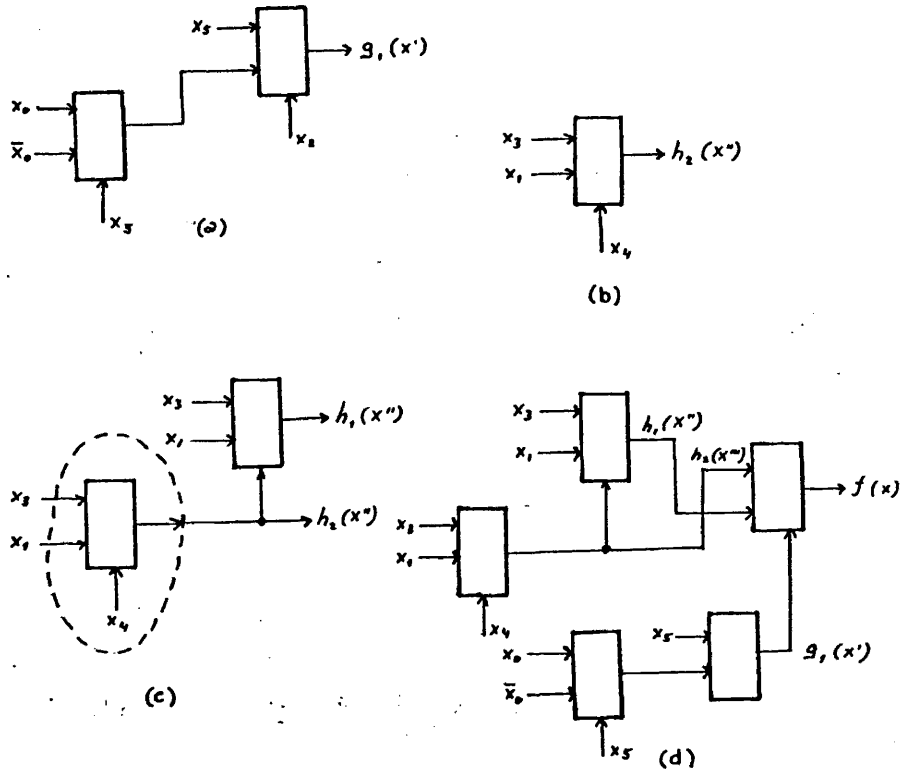


Figura 2.25.- Síntesis de la función $f(X) = \overline{g_1(X')}\ h_2(X''') + g_1(X')\ h_1(X'')$ del Ejemplo 2.14.

principio, poco viable. En efecto, supongamos que la estructura arborescente óptima de la función representativa de una clase de equivalencia requiere 6 módulos MLUM(1). Para conseguir una estructura óptima tendremos que analizar todas las estructuras con 5 módulos o menos, que se puedan tener (número que se determinó en el capítulo anterior) y para cada una de ellas intentar combinatoriamente asignar las variables y constantes lógicas a las entradas libres de la estructura, con el fin de obtener la función representativa que se desea implementar.

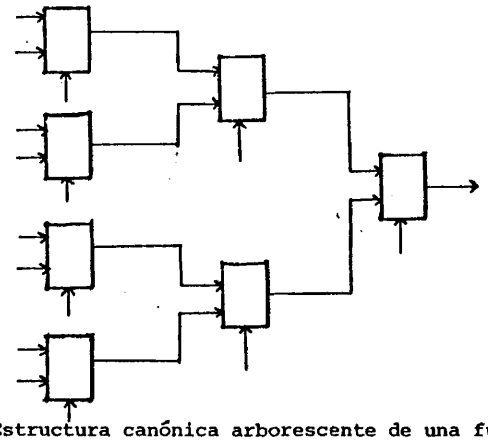


Figura 2.26.- Estructura canónica arborescente de una función de 4 variables.

Aun en el supuesto de poderlo conseguir, tendríamos que ser bastante cautelosos en este caso de decir que la estructura es óptima puesto que las estructuras que, en principio, estamos considerando no toman en cuenta la posibilidad de que determinadas funciones residuos multivariables puedan ser compartidas como entradas de más de un módulo (análogamente a como sucedía con las estructuras arborescentes).

En este sentido, parece más lógica la siguiente línea de acción: Dada una estructura cualesquiera ¿cuántas clases de equivalencia se pueden asignar de forma que no exista ninguna otra estructura con menor número de módulos que la implemente?. Este planteamiento presenta la ventaja de que el procedimiento constructivo se realiza a partir de las estructuras más sencillas. Por ejemplo, una función propia de 4 variables requerirá como mínimo 2 MLUM(1) para su implementación. Parece, pues, natural preguntarse a cuántas clases de equivalencia les corresponde una estructura determinada. En la Figura 2.27 están representadas todas las clases de equivalencia n-p-n de una estructura arborescente con dos MLUM(1). Para estas clases de equivalencia n-p-n al menos, sí podemos garantizar que la estructura es óptima. Otra cuestión que conviene tener en cuenta es que determinadas clases de equivalencia n-p-n pueden ser representadas con el mismo coste

por diferentes estructuras (ver Figura 2.28). Diremos que una clase de equivalencia n-p-n es <u>propia</u> de una estructura si únicamente se puede implementar de forma

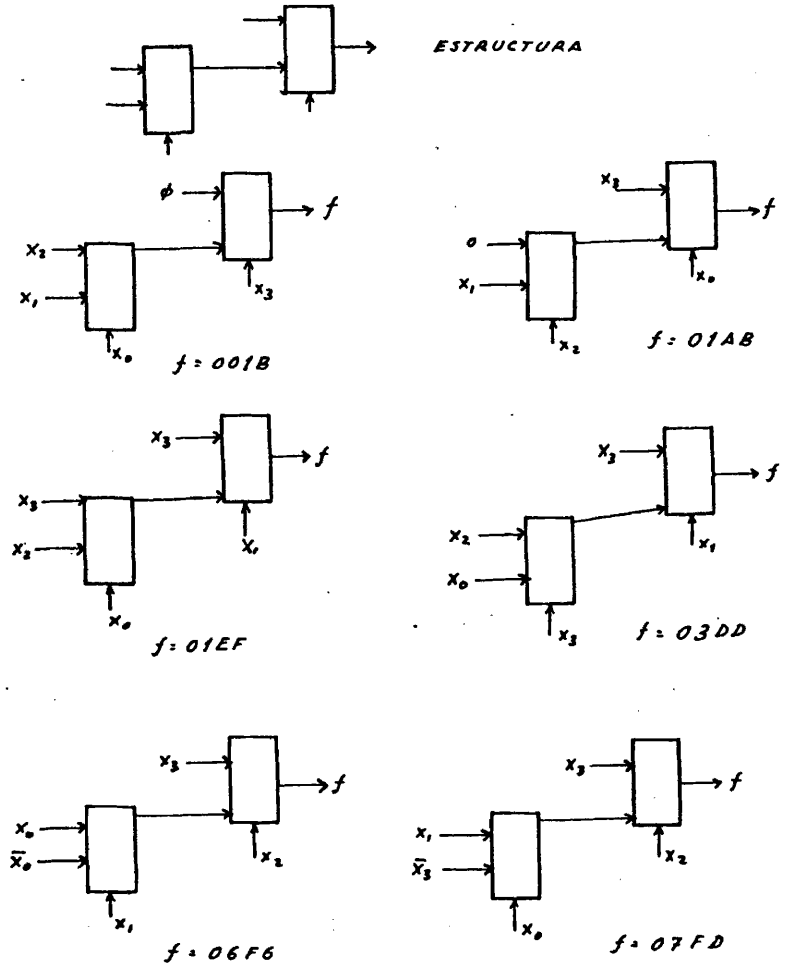


Figura 2.27.- Clases de equivalencia n-p-n que corresponden a una determinada estructura. (La función representativa se representa por la notación hexadecimal introducida en el capítulo anterior).

óptima con dicha estructura. Así, la clase de equivalencia n-p-n que corresponde a la función representativa 03DD es propia de la estructura de la Figura 2.27.

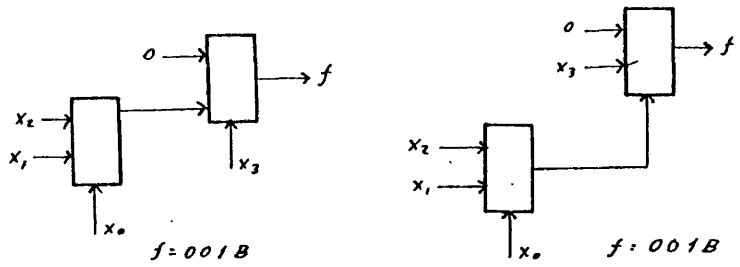


Figura 2.28.- Representación de una misma clase de equivalencia n-p-n por dos estructuras diferentes con el mismo coste.

Hasta la fecha no hemos podido automatizar este proceso de forma satisfactoria. Nuestros intentos se han movido a lo largo de los siguientes puntos:

a) Se ha demostrado que ninguna clase de equivalencia n-p-n de una función de 4 variables requiere los 7 módulos de la estructura canónica.

b) Explotación de la descomposición simple disjunta (d.s.d.) para reducir el número de módulos frente a la estructura arborescente óptima.

c) Explotación de la descomposición simple no disjunta (d.s.n.d.) para reducir el número de módulos frente a la estructura arborescente óptima.

Analicémoslas, pues, con cierto detenimiento.

2.10.1.- Síntesis de las clases de equivalencia n-p-n que no admiten simplificación en su estructura arborescente

Una vez aplicado el método de síntesis óptima de estructuras arborescen-

tes a todas las clases de equivalencia n-p-n de una función de 4 variables, se detectaron 6 clases de equivalencia n-p-n cuyas funciones representativas no admitían simplificación posible y, por lo tanto, en principio, requerían los 7 módulos de las estructura canónica arborescente.

Estas funciones representativas eran las siguientes:

Representación hexadecimal

$$1668 \longrightarrow \delta_a(x_3,x_2,x_1,x_0) = \sum(3,5,6,9,10,12)$$
$$5737 \longrightarrow \delta_b(x_3,x_2,x_1,x_0) = \sum(3,5,6,9,10,12,15)$$
$$166B \longrightarrow \delta_c(x_3,x_2,x_1,x_0) = \sum(3,5,6,9,10,12,14,15)$$
$$1681 \longrightarrow \delta_d(x_3,x_2,x_1,x_0) = \sum(3,5,6,8,15)$$
$$177E \longrightarrow \delta_e(x_3,x_2,x_1,x_0) = \sum(3,5,6,7,9,10,11,12,13,14)$$
$$6996 \longrightarrow \delta_f(x_3,x_2,x_1,x_0) = \sum(1,2,4,7,8,11,13,14)$$

El método de síntesis empleado para ver si es posible reducir el número de módulos es el siguiente: Teniendo en cuenta el Teorema 2.7, se efectúa la expansión en el primer nivel respecto a x_i $(i = 3,2,1,0)$ y se analizan las funciones residuos de 3 variables resultantes, determinándose su síntesis óptima de acuerdo con la Figura 2.24 del apartado anterior. Si alguna de las funciones residuos pertenecen a las clases de equivalencia n-p-n número 8 ó 14, se habrá obtenido un ahorro en el número de módulos necesitados.

Síntesis de δ_a

Independientemente de cuál es la variable que se tome en el primer nivel de expansión, las funciones residuos que resultan son siempre del mismo tipo (esto se puede ver sobre las cartas de descomposición de 4 variables).

Tomemos pues x_3 como variable del primer nivel (ver Figura 2.29a).

Las funciones residuos normalizadas que resultan son:

$$g_1(x_2,x_1,x_0) = \sum(3,5,6)$$
$$g_2(x_2,x_1,x_0) = \sum(1,2,4)$$

Como son funciones de 3 variables, su síntesis óptima ya la conocemos y requieren:

$g_1 \in \{7\} \longrightarrow$ 3 MLUM(1): $g_1 = x_2\, x_1\, \overline{x}_0 + \overline{x}_2\, x_1\, x_0 + x_2\, \overline{x}_1\, x_0$

$g_2 \in \{7\} \longrightarrow$ 3 MLUM(1): $g_2 = \overline{x}_2\, \overline{x}_1\, x_0 + \overline{x}_2\, x_1\, \overline{x}_0 + x_2\, \overline{x}_1\, \overline{x}_0$

Con {7} se indica que pertenecen a la clase de equivalencia n-p-n número 7 de la Figura 2.24. La estructura resultante es la de la Figura 2.29b). Tal como se observa en dicha figura, es posible una simplificación adicional (ver Figura 2.29c). En este caso se necesitan pues 6 MLUM(1), y la reducción ha venido posibilitada por la existencia de dos funciones residuos idénticas. Debe observarse que la estructura resultante sigue siendo arborescente.

Síntesis de f_b

Análogamente al caso de f_a, también ahora, independientemente de cuál es la variable que se tome en el primer nivel de expansión, las funciones residuos que resultan son siempre del mismo tipo. Tomemos pues x_3 como variable del primer nivel (ver Figura 2.30a). Las funciones residuos normalizadas que resultan son:

$$g_1(x_2,x_1,x_0) = \sum(3,5,6)$$
$$g_2(x_2,x_1,x_0) = \sum(1,2,4,7)$$

Como son funciones de 3 variables su síntesis óptima ya la conocemos y requieren:

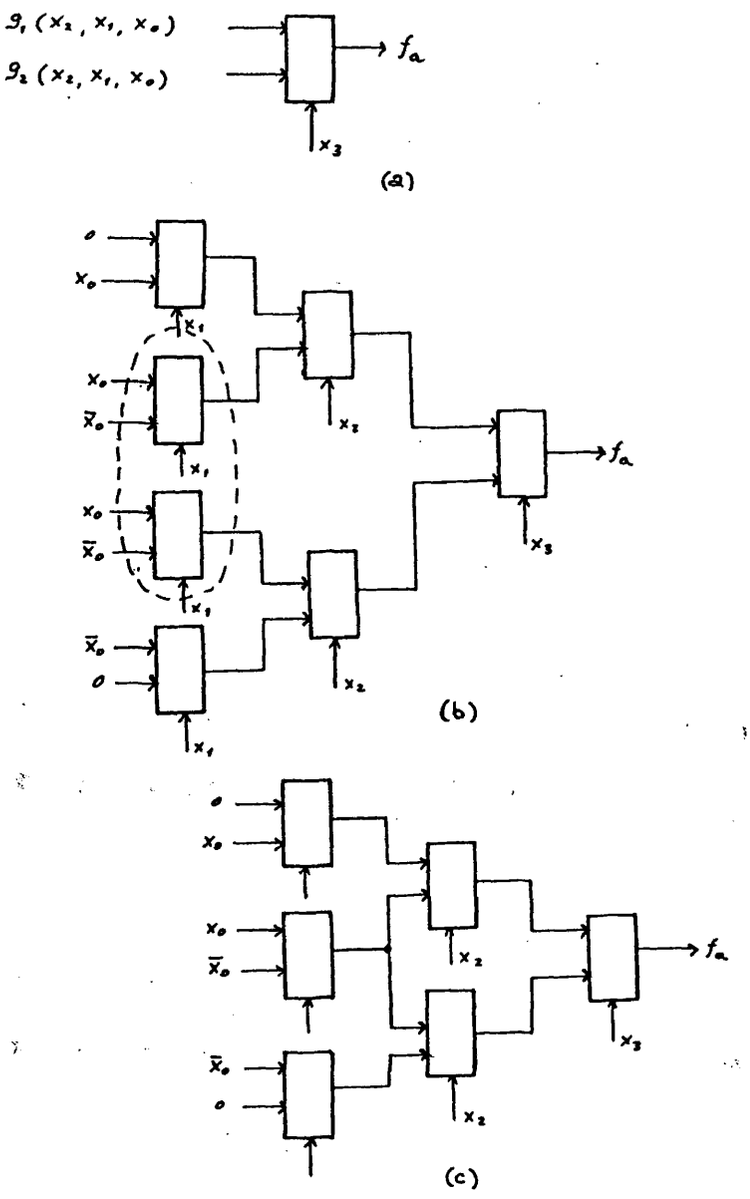
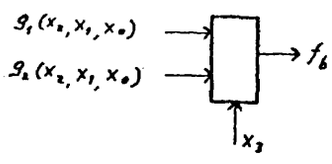
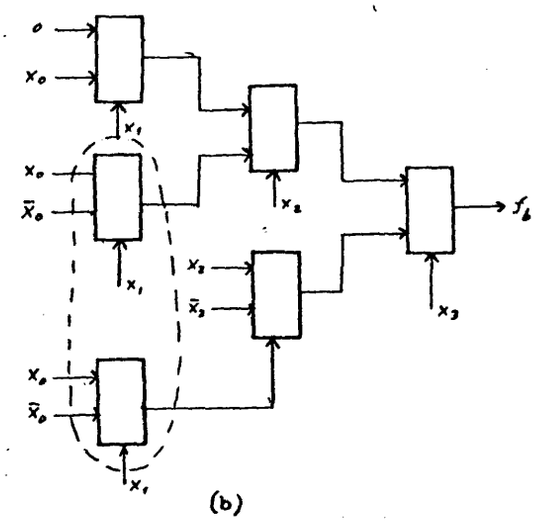


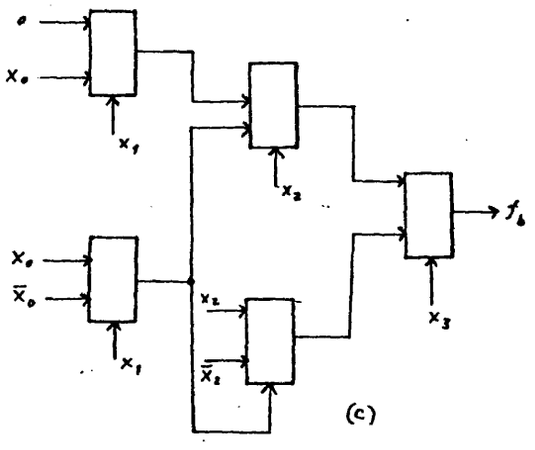
Figura 2.29.- Síntesis de $f_a(x_3, x_2, x_1, x_0) = \sum(3,5,6,9,10,12)$



(a)



(b)



(c)

Figura 2.30.- Síntesis de $f_6(x_3, x_2, x_1, x_0) = \sum(3,5,6,9,10,12,15)$

$g_1 \in \{7\} \longrightarrow 3\ MLUM(1)$: $g_1 = x_2\, x_1\, \bar{x}_0 + \bar{x}_2\, x_1\, x_0 + x_2\, \bar{x}_1\, x_0$

$g_2 \in \{14\} \longrightarrow 2\ MLUM(1)$: $g_2 = x_2 \oplus x_1 \oplus x_0$

La estructura resultante es la de la Figura 2.30b). Tal como se observa en dicha figura, es posible una simplificación adicional (ver Figura 2.30c). En este caso, se necesitan 5 MLUM(1) y la reducción ha venido posibilitada, de una parte, porque en las funciones residuos de 3 variables ha sido factible el ahorro de un módulo y, de otra, por la existencia de dos funciones residuos idénticas. La estructura resultante es del tipo generalizado.

Síntesis de δ_c

Si la variable que se toma en el primer nivel es x_3 ó x_2 ó x_1, las funciones residuos normalizadas que resultan son:

$g_1 \in \{7\} \longrightarrow 3\ MLUM(1)$: $g_1 = \sum(3,5,6)$

$g_2 \in \{7\} \longrightarrow 3\ MLUM(1)$: $g_2 = \sum(1,2,4,6,7)$

Sin embargo, si la variable que se toma en el primer nivel es x_0, las funciones residuos normalizadas que resultan son:

$g_1 \in \{8\} \longrightarrow 2\ MLUM(1)$: $g_1 = \sum(3,5,6,7)$

$g_2 \in \{14\} \longrightarrow 2\ MLUM(1)$: $g_2 = \sum(1,2,4,7)$

La estructura resultante en este caso es la de la Figura 2.31, necesitándose 5 MLUM(1) en su implementación.

Síntesis de δ_d

Si la variable que se toma en el primer nivel es x_2 ó x_1 ó x_0, las fun-

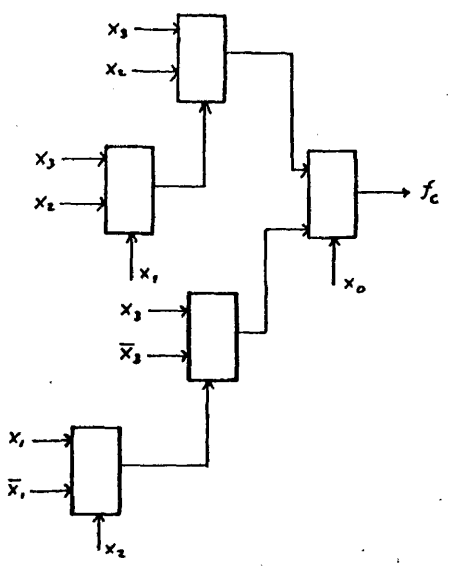


Figura 2.31.- Síntesis de $f_c(x_3,x_2,x_1,x_0) = \sum(3,5,6,9,10,12,14,15)$

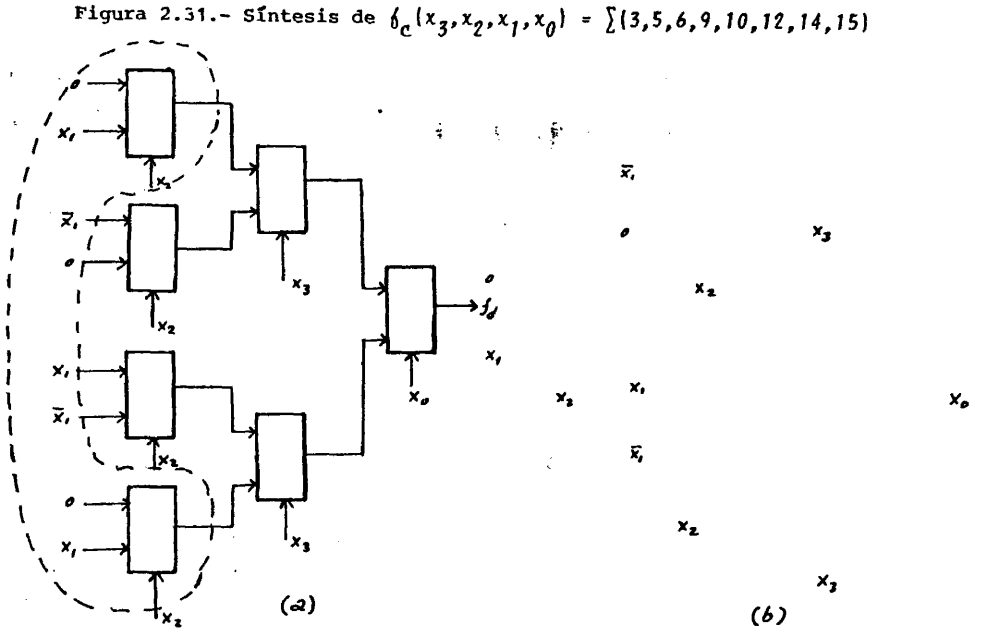


Figura 2.32.- Síntesis de $f_d(x_3,x_2,x_1,x_0) = \sum(3,5,6,8,15)$

ciones residuos normalizadas que resultan son:

$g_1 \in \{9\} \longrightarrow$ 3 MLUM(1): $g_1 = \sum(3,4)$

$g_2 \in \{7\} \longrightarrow$ 3 MLUM(1): $g_2 = \sum(1,2,7)$

Si la variable que se toma es x_3, las funciones residuos normalizadas que resultan son:

$g_1 \in \{7\} \longrightarrow$ 3 MLUM(1): $g_1 = \sum(3,5,6)$

$g_2 \in \{9\} \longrightarrow$ 3 MLUM(1): $g_2 = \sum(0,7)$

Tomemos x_0 en el primer nivel lo que nos da la estructura resultante de la Figura 2.32a. Se observa en dicha figura que es posible una simplificación debido a la existencia de dos funciones residuos idénticas (ver Figura 2.32b). Se necesitan, pues, 6 MLUM(1). El caso de esta función es análogo al de la síntesis de δ_a y da como resultado una estructura arborescente.

Síntesis de δ_e

También en este caso, independientemente de cuál es la variable que se toma en el primer nivel de expansión, las funciones residuos que resultan son siempre las mismas. Tomemos pues x_3 como variable del primer nivel. Las funciones residuos normalizadas que se obtienen son:

$g_1 \in \{8\} \longrightarrow$ 2 MLUM(1): $g_1 = \sum(3,5,6,7)$

$g_2 \in \{9\} \longrightarrow$ 3 MLUM(1): $g_2 = \sum(1,2,3,4,5,6)$

La estructura resultante es la de la Figura 2.33, necesitándose 6 MLUM(1) en su implementación.

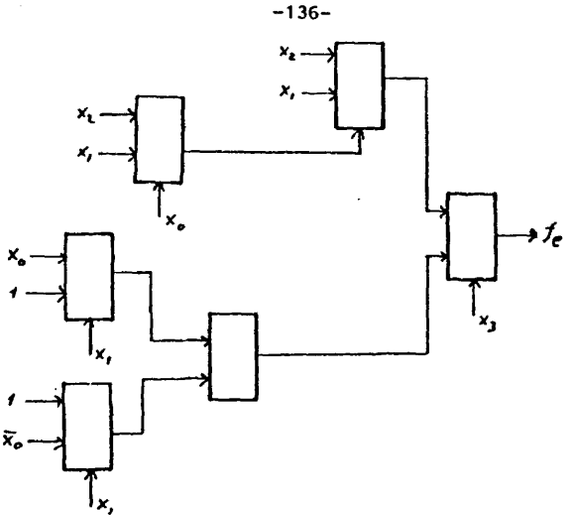


Figura 2.33.- Síntesis de $f_e(x_3,x_2,x_1,x_0) = \sum(3,5,6,7,9,10,11,12,13,14)$

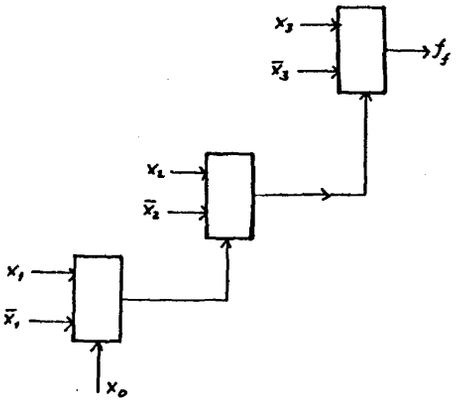


Figura 2.34.- Síntesis de $f_f(x_3,x_2,x_1,x_0) = \sum(1,2,4,7,8,11,13,14)$

Síntesis de δ_f

La función δ_f se puede expresar de la forma siguiente:

$$\delta_f(x_3,x_2,x_1,x_0) = x_3 \oplus x_2 \oplus x_1 \oplus x_0$$

Corresponde, pues, al tipo descrito en el apartado 2.8, al hablar de estructuras generalizadas y su síntesis, como ya se sabe requiere 3 MLUM(1). (Ver Figura 2.34).

La síntesis de estas seis funciones, δ_a, δ_b, δ_c, δ_d, δ_e, δ_f, nos permite enunciar el siguiente Teorema de carácter general.

Teorema 2.8.- La síntesis de cualquier función de conmutación de n variables, con $n \geq 4$, mediante MLUM(1), nunca requerirá el número máximo de módulos dado por la estructura canónica correspondiente.

Demostración

Hemos visto anteriormente que cualquier función de 4 variables nunca necesitará los 7 MLUM(1) de la estructura canónica, ya que las funciones δ_a a δ_f, que en principio parecía que no admitían ningún ahorro, después de un estudio exhaustivo de cada una de ellas se ha demostrado que se pueden implementar con 6 MLUM(1) o menos.

Por tanto, cualquier función de n variables con $n > 4$, tampoco necesitará el número de módulos máximos dados por su estructura canónica, pues, al menos, en las funciones residuos de 4 variables que resultan en la expansión respecto a $n-4$ variables cualesquiera, siempre podemos garantizar el ahorro de algún módulo.

2.10.2.- Aplicación de la d.s.d. a la síntesis de estructuras generalizadas de

funciones de 4 variables con un coste menor que la estructura arborescente óptima correspondiente

La importancia de que la función representativa de una clase de equivalencia n-p-n admita una d.s.d. viene dada por el siguiente Lema.

Lema 2.1.- Si una función de conmutación de 4 variables admite una d.s.d., su implementación con MLUM(1) requiere como máximo 4 módulos.

Demostración.- Las dos únicas posibilidades de que una función de conmutación de 4 variables admite una d.s.d. son:

$$\delta(X) = F(X_1, \varphi(X_2))) \qquad \text{a) } card\ X_1 = 2 \text{ y } card\ X_2 = 2$$

$$X = X_1 \cup X_2 \qquad \text{b) } card\ X_1 = 1 \text{ y } card\ X_2 = 3$$

En ambos casos, el coste máximo de δ (cota superior en el número de módulos necesitados) es la suma del coste de una función de 2 variables (= 1 módulo) y de una función de 3 variables (= 3 módulos). De manera que la cota superior en cuanto al número de módulos necesitados para implementar δ es de 4 módulos.

Como es bien sabido, no toda función de conmutación admite una d.s.d. Se ha llevado a cabo un estudio exhaustivo de las funciones representativas de todas las clases de equivalencia n-p-n de 4 variables y se ha determinado que 60 de ellas admitían algún tipo de d.s.d. En 25 de dichas funciones, la síntesis efectuada mejora a la obtenida empleando la estructura arborescente mínima. La clasificación de estas 25 funciones, según el número de módulos utilizados, se da en la Tabla 2.8.

En las 35 funciones representativas restantes, la síntesis mediante d.s.d. da un coste igual que la estructura arborescente óptima y su clasificación, según el número de módulos utilizados, se da en la Tabla 2.9.-

número de funciones representativas	Módulos que utilizan
4	2
14	3
7	4

TABLA 2.8.- Módulos que utilizan las funciones representativas de las clases de equivalencia n-p-n con d.s.d., que mejora la síntesis de la estructura arborescente óptima.

número de funciones representativas	Módulos que utilizan
1	0 (función trivial)
3	1
9	2
17	3
5	4

TABLA 2.9.- Módulos que utilizan las funciones representativas de las clases de equivalencia n-p-n con d.s.d. que igualan la síntesis de la estructura arborescente óptima.

A la vista de las Tablas 2.8 y 2.9, podemos asegurar que de las 60 clases de equivalencia n-p-n que admiten d.s.d. en 48 de ellas, el procedimiento de síntesis basado en la descomposición funcional nos asegura una estructura óptima. En el caso de las funciones representativas cuya síntesis requiere 4 módulos, se debe ser más cauto en tal afirmación, pues puede existir algún tipo de descomposición más compleja que dé lugar a una estructura generalizada que únicamente utiliza 3 módulos. Sin embargo, la probabilidad de que tal cosa suceda no es muy elevada.

En la Figura 2.35 se han representado, en la columna derecha, la sínte-

sis de las 25 funciones representativas de las clases de equivalencia n-p-n que admitiendo d.s.d. mejoran a su estructura arborescente óptima correspondiente (que se representan en la columna izquierda).

2.10.3.- Aplicación de la d.s.n.d. a la síntesis de estructuras generalizadas de 4 variables con un coste menor que la estructura arborescente óptima correspondiente

En la síntesis óptima mediante MLUM(1) de las clases de equivalencia n-p-n de funciones de 3 variables, hemos visto cómo para la clase de equivalencia número 8, la estructura resultante provenía de una descomposición simple no disjunta (d.s.n.d.) de su función representativa.

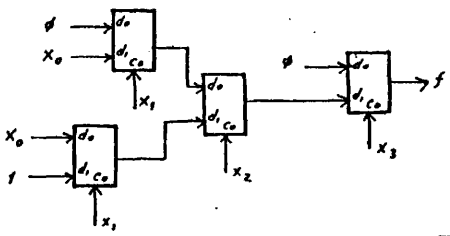
Análogamente a como se hizo en el apartado anterior con la d.s.d., se ha efectuado también un estudio exhaustivo de las funciones representativas de todas las clases de equivalencia n-p-n de 4 variables que no admitieron d.s.d. (160 en total) y se ha determinado que 69 de ellas admitían algún tipo de d.s.n.d. En 32 de dichas funciones, la síntesis efectuada mejora a la obtenida empleando la estructura arborescente óptima. La clasificación de estas 32 funciones, según el número de módulos utilizados se da en la Tabla 2.10.

número de funciones representativas	Módulos que utilizan
1	2
10	3
20	4
1	5

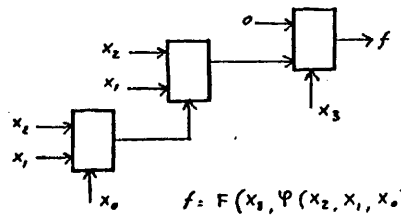
TABLA 2.10.- Módulos que utilizan las funciones representativas de las clases de equivalencia n-p-n con d.s.n.d. que mejoran la síntesis de la estructura arborescente óptima.

D 5 D

ESTRUCTURA ARBORESCENTE OPTIMA

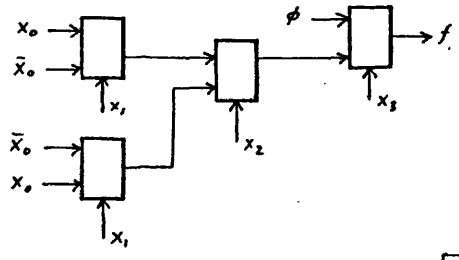


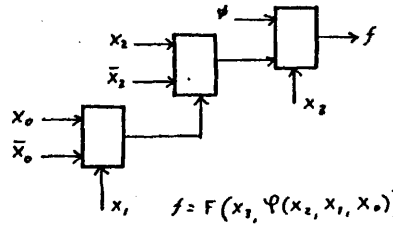
ESTRUCTURA GENERALIZADA



$f = F(x_3, \varphi(x_2, x_1, x_0))$

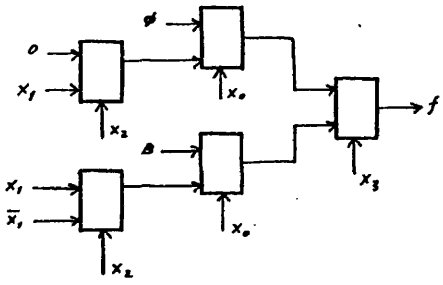
0017

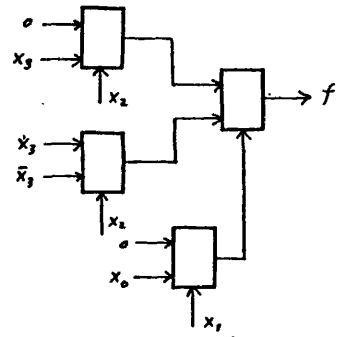




$f = F(x_3, \varphi(x_2, x_1, x_0))$

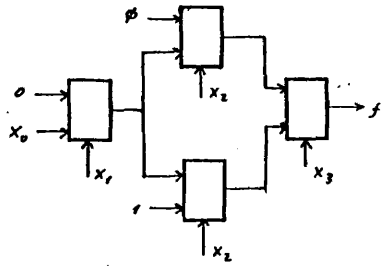
0069

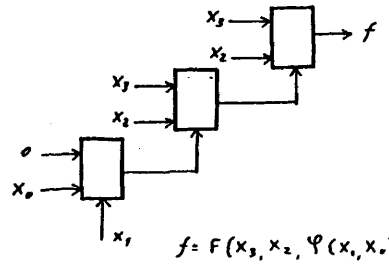




$f = F(x_3, x_2, \varphi(x_1, x_0))$

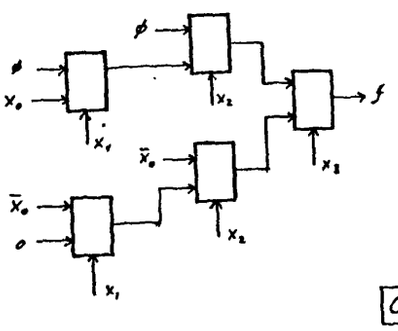
011E



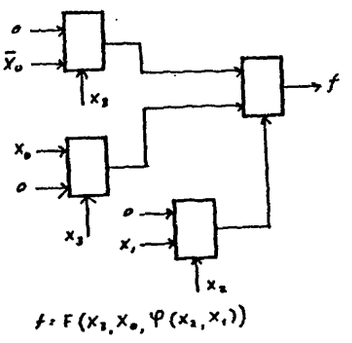


$f = F(x_3, x_2, \varphi(x_1, x_0))$

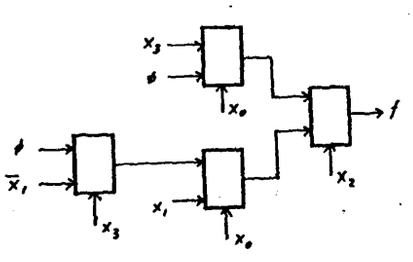
011F



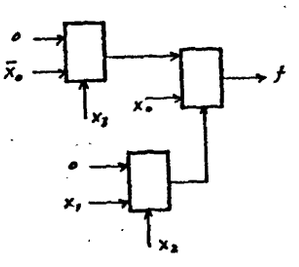
01A8



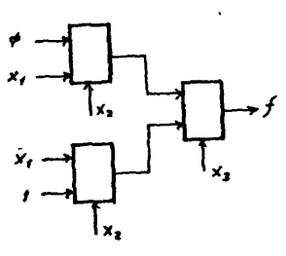
$f = F(x_3, x_0, \varphi(x_2, x_1))$



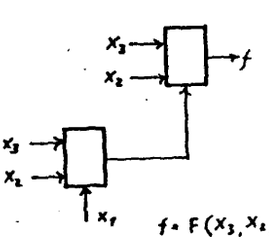
01A9



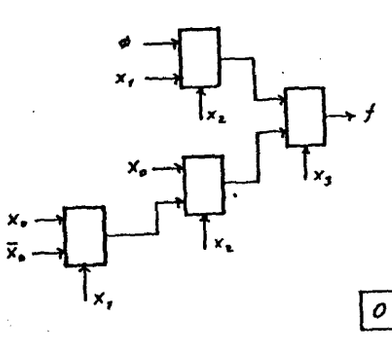
$f = F(x_3, x_0, \varphi(x_2, x_1))$



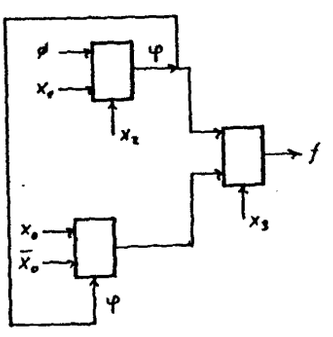
033F



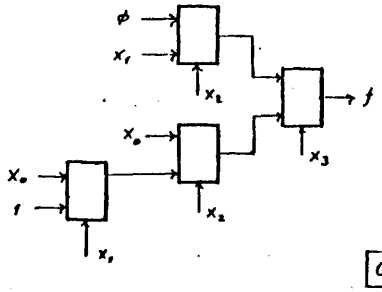
$f = F(x_3, x_2, x_1)$



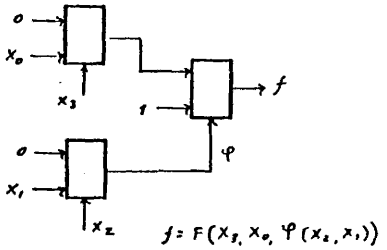
0356



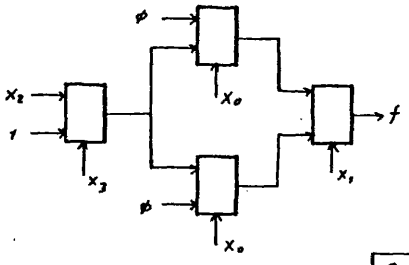
$f = F(x_3, x_0, \varphi(x_2, x_1))$



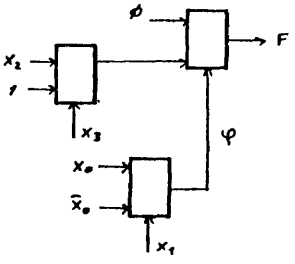
0357



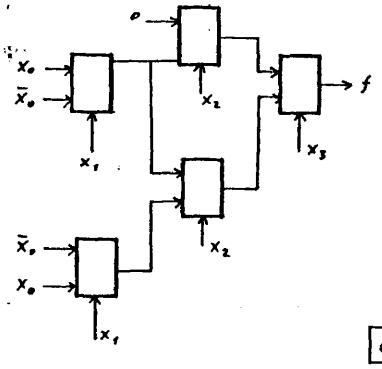
$f = F(x_3, x_0, \varphi(x_2, x_1))$



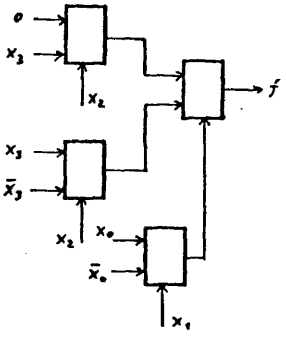
0666



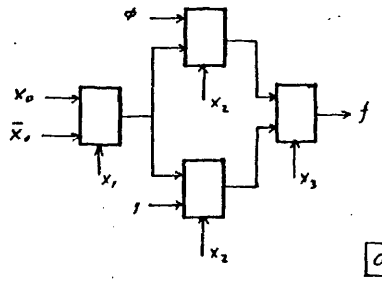
$f = F(x_3, x_2, \varphi(x_1, x_0))$



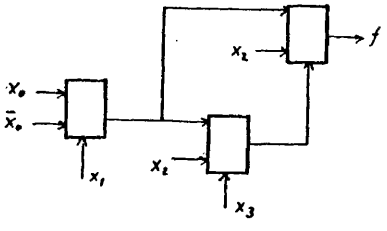
0669



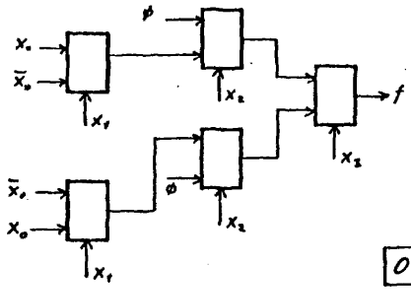
$f = F(x_3, x_2, \varphi(x_1, x_0))$



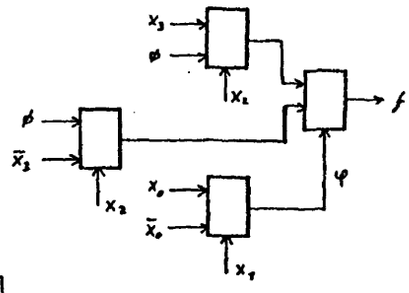
066F



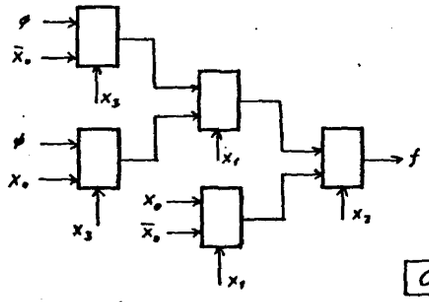
$f = F(x_3, x_2, \varphi(x_1, x_0))$



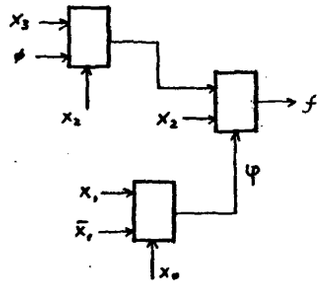
0690



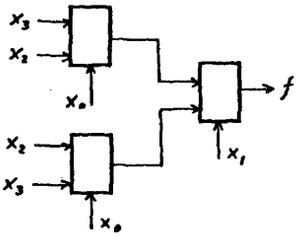
$f = F(x_3, x_2, \varphi(x_1, x_0))$



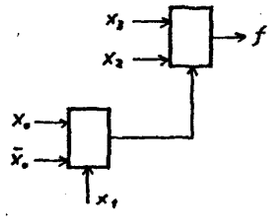
0696



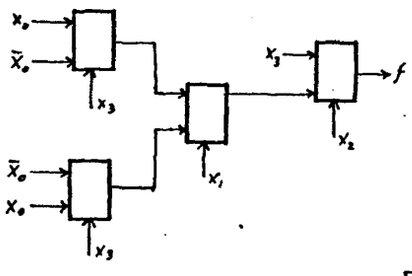
$f = F(x_3, x_2, \varphi(x_1, x_0))$



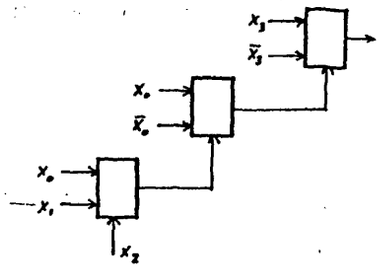
069F



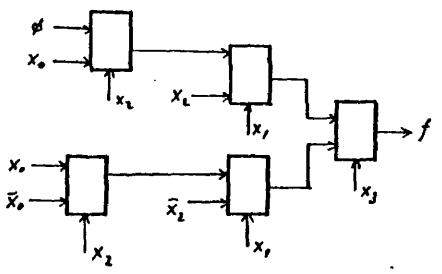
$f = F(x_3, x_2, \varphi(x_1, x_0))$



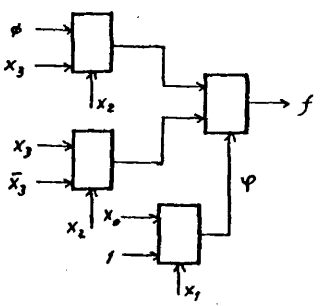
06F9



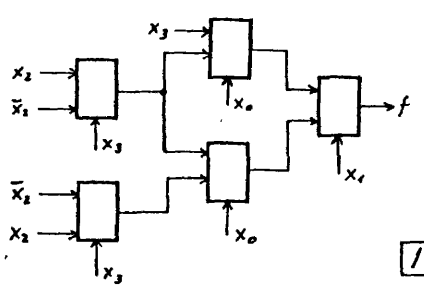
$f = F(x_3, \varphi(x_2, x_1, x_0))$



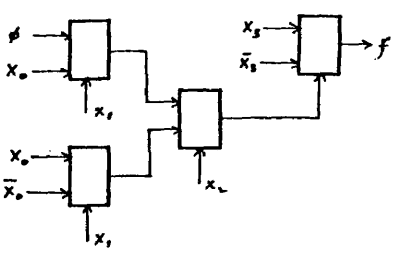
0778



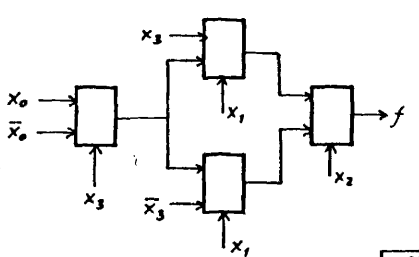
$f = F(x_3, x_2, \varphi(x_1, x_0))$



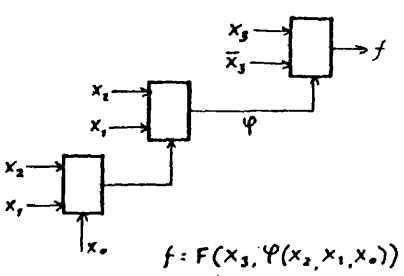
16E9



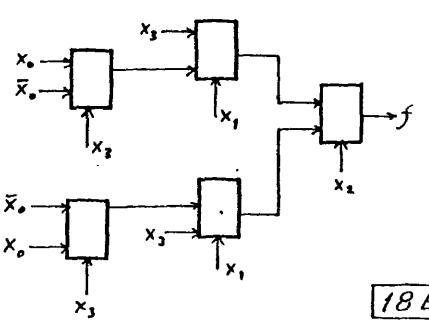
$f = F(x_3, \varphi(x_2, x_1, x_0))$



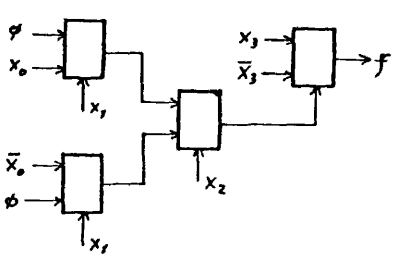
17E8



$f = F(x_3, \varphi(x_2, x_1, x_0))$



18E7



$f = F(x_3, \varphi(x_2, x_1, x_0))$

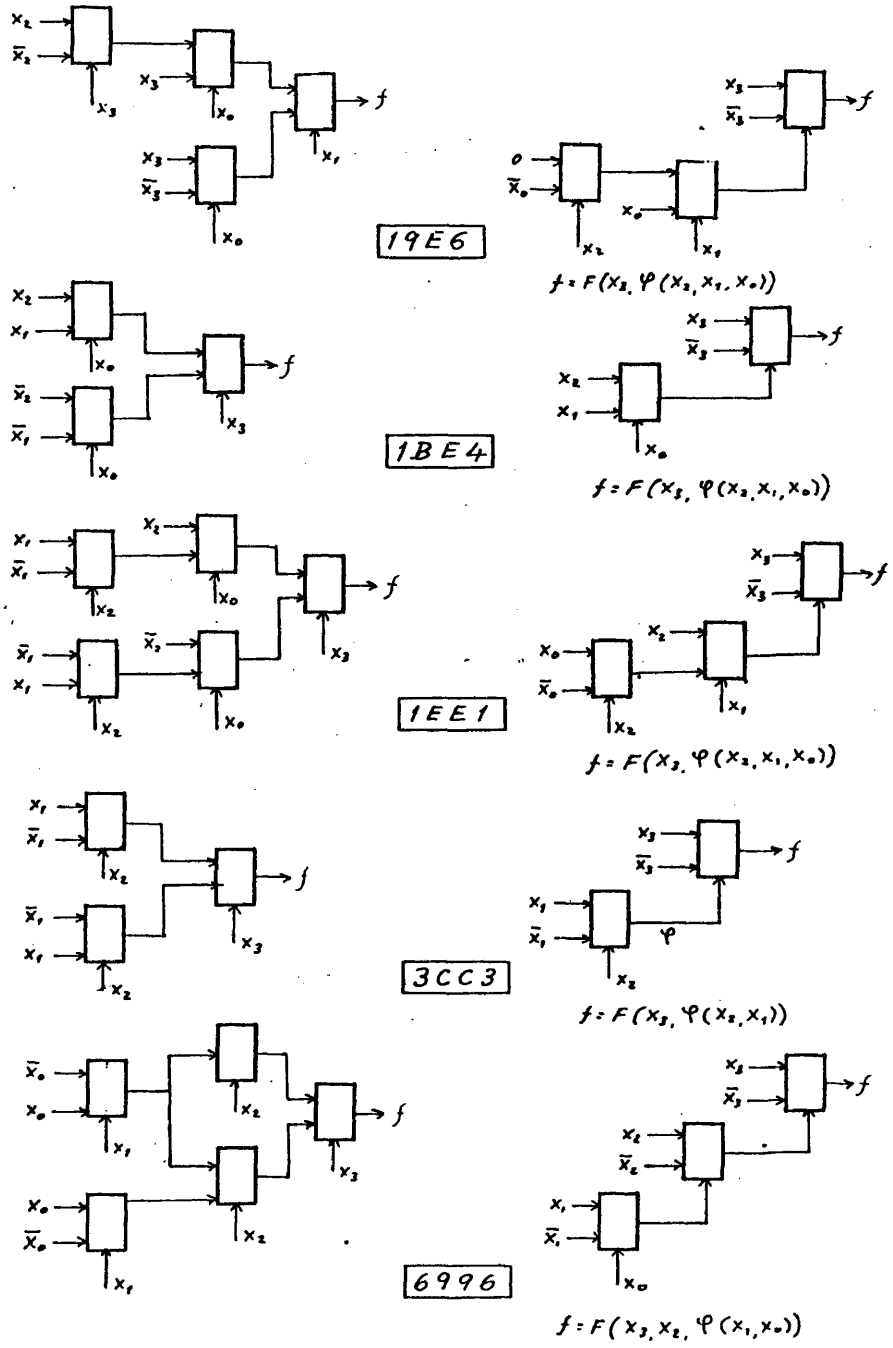


Figura 2.35.- Catálogo de las funciones representativas, de las clases de equivalencia n-p-n, con d.s.d. que mejoran la síntesis de la estructura arborescente óptima

En las 37 funciones representativas restantes, la síntesis mediante d.s.n.d., da un coste igual que la estructura arborescente óptima, y su clasificación, según el número de módulos utilizados, se da en la Tabla 2.11.

número de funciones representativas	Módulos que utilizan
2	2
10	3
20	4
5	5

TABLA 2.11.- Módulos que utilizan las funciones representativas de las clases de equivalencia n-p-n con d.s.n.d. que igualan la síntesis de la estructura arborescente óptima.

Para funciones de 4 variables, la d.s.n.d. corresponde al siguiente esquema de descomposición:

$$f(X) = F[X_1, \varphi(X_2)]$$

con $X_1 \cup X_2 = X$

$card\ X_1 = 2, \qquad card\ X_2 = 3$

$Card\ (X_1 \cap X_2) = 1$

El método utilizado para determinar las d.s.n.d. ha sido el de las cartas de descomposición no disjuntas de Curtis ([9]) (en el momento presente no se conoce un método analítico como el de Deschamps para d.s.d., que se válido para d.s.n.d.).

Veamos en un ejemplo concreto su aplicación.

Ejemplo 2.15.- Sintetizar con MLUM(1) la siguiente función de 4 variables:

$$\delta(x_3, x_2, x_1, x_0) = \sum(7,10,12,13,14,15)$$

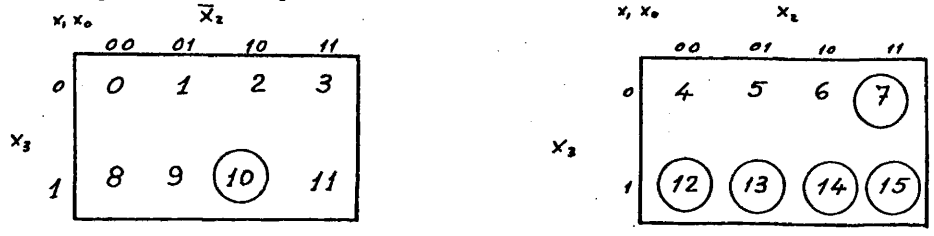
Esta función que no admite d.s.d, sin embargo sí tienen d.s.n.d. como se observa en las cartas de descomposición simples no disjuntas de 4 variables que se da en la Tabla (2.12). En dichas cartas se enlazan con un círculo los minterms de la función. Para que la función admita una d.s.n.d deberá existir al menos una carta tal que en las dos subcartas que le corresponde, la multiplicidad de columnas P en cada una de ellas sea como máximo de dos.

En nuestro caso, las cartas marcadas con (x) corresponden a d.s.n.d. del tipo

a) $\delta(X) = F(x_3, x_2, \varphi, (x_3, x_1, x_0))$

b) $\delta(X) = F(x_3, x_2, \varphi, (x_2, x_1, x_0))$

Tomemos la segunda carta marcada con asterisco a la que corresponde una descomposición del tipo b).



La síntesis parcial de cada subtabla será:

$$\overline{\varphi}_0(x_1, x_0) = x_1 \overline{x}_0 \qquad \varphi_1(x_1, x_0) = x_1 x_0$$

$$F_0(\varphi_0, x_3) = \overline{\varphi}_0 x_3 \qquad F_1(\varphi_1, x_3) = \varphi_1 \overline{x}_3 + x_3$$

De donde resulta para F y φ las siguientes expresiones:

$$F = F(x_3, x_2, \varphi) = F_0 \overline{x}_2 + F_1 x_2 = \overline{\varphi}_0 x_3 \overline{x}_2 + \varphi_1 \overline{x}_3 x_2 + x_3 x_2$$

Haciendo $\varphi_0 = \varphi_1 = \varphi$ se obtiene

$$F = F(x_3, x_2, \varphi) = \bar{\varphi} x_3 \bar{x}_2 + \varphi \bar{x}_3 x_2 + x_3 x_2 = \varphi x_2 + \bar{\varphi} x_3 \tag{2.28}$$

$$\varphi = \varphi(x_2, x_1, x_0) = \varphi_0 \bar{x}_2 + \varphi_1 x_2 = \overline{x_1 \bar{x}_0} \bar{x}_2 + x_1 x_0 x_2 = x_1 x_0 + \bar{x}_1 \bar{x}_2 \tag{2.29}$$

La implementación con MLUM(1) de (2.28) y (2.29) se dan en la Figura 2-36a y 2-36b, que al combinarse dan la estructura generalizada de la Figura 2-36 c, que requiere únicamente 2 MLUM(1), sin embargo, la estructura arborescente mínima para realizar esta función necesita 4 MLUM(1), tal como se observa en la Figura 2.36d.

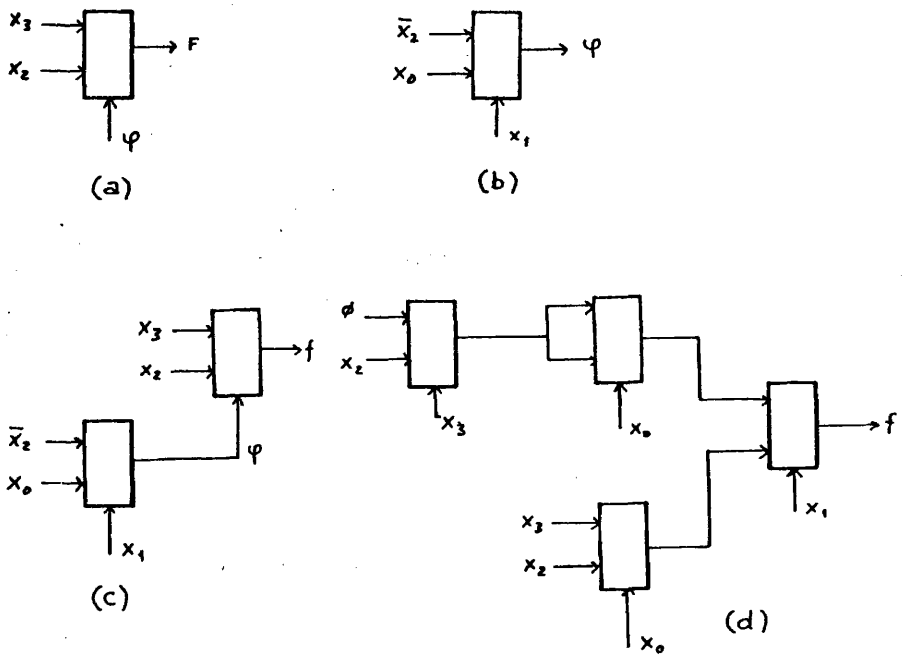


Figura 2.36.- Síntesis de la función $f(x_3, x_2, x_1, x_0) = \sum(7,10,12,13,14,15)$ mediante una estructura generalizada y una estructura arborescente mínima.

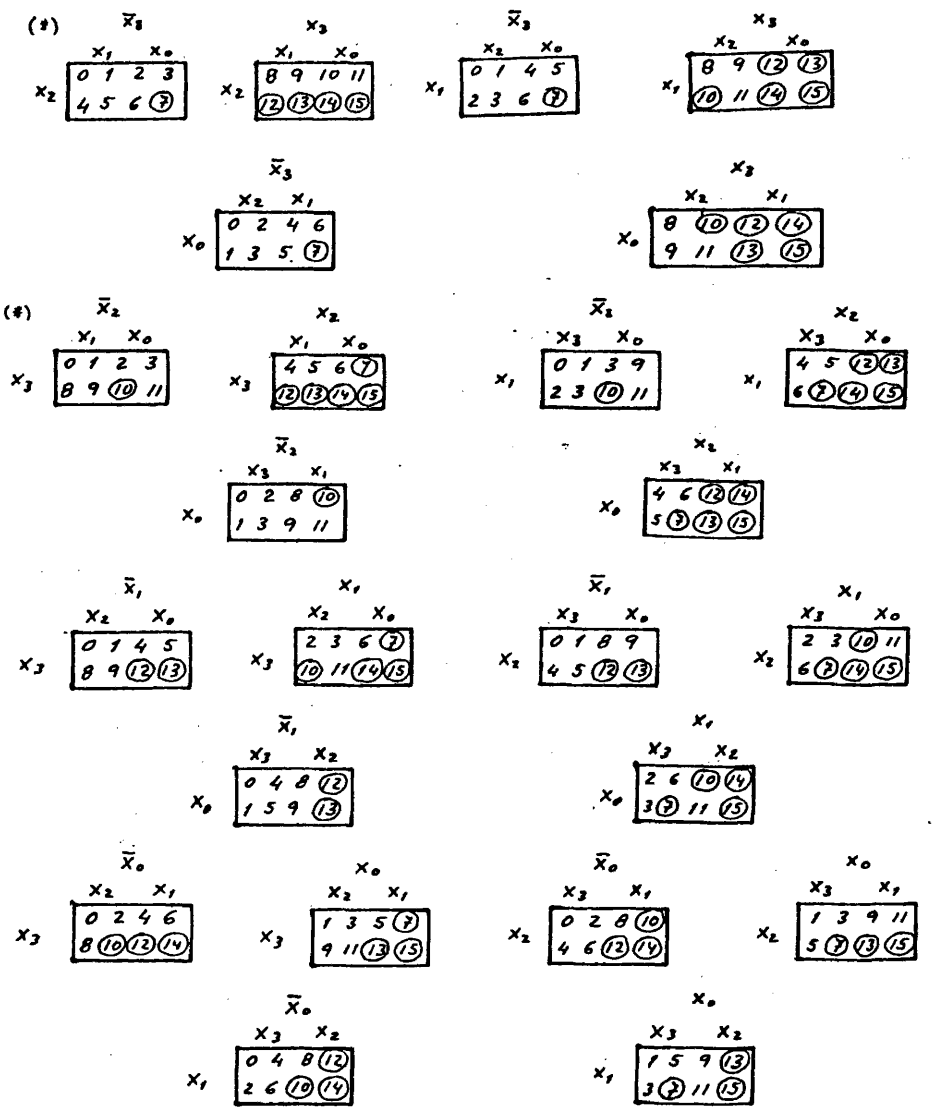


TABLA 2.12.- Cartas de descomposiciones simples no disjuntas de 4 variables función $f(x_3,x_2,x_1,x_0) = \sum(7,10,12,13,14,15)$

Conviene advertir que en la síntesis de $\varphi_0(x_1,x_0)$ se ha escogido a propósito la función complementaria $\overline{\varphi}_0$, pues esto nos daba lugar a una síntesis más económica. En efecto, si se hubiese escogido $\varphi_0(x_1,x_0) = x_1\overline{x}_0$ las expresiones resultantes para F y φ hubiesen sido:

$$F = F(x_3,x_2,\varphi) = x_3x_2 + \varphi(x_3 + x_2) \qquad (2.30)$$

$$\varphi = \varphi(x_2,x_1,x_0) = x_1\,(\overline{x_0 \oplus x_2}) \qquad (2.31)$$

Como funciones de 3 variables, la función F pertenece a la clase de equivalencia número 8 y su coste es de 2 MLUM(1), y la función pertenece a la clase de equivalencia número 4 cuyo coste es también de 2 MLUM(1). Así pues, en este caso, la síntesis de la función f tendría un coste total de 4 MLUM(1) (ver Figura 2.37).

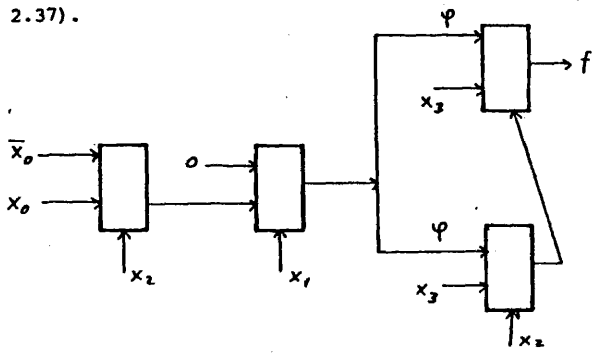


Figura 2.37.- Síntesis de la función $f(x_3,x_2,x_1,x_0) = \sum(7,10,12,13,14,15)$ cuando en la síntesis de las funciones parciales se toma φ_0 en lugar de $\overline{\varphi}_0$.

Este ejemplo pone de manifiesto que, para determinar la síntesis más económica con d.s.n.d. hay que examinar cuidadosamente todas las posibilidades de elección de las funciones parciales φ_0 y φ_1 (4 en total (φ_0,φ_1), $(\varphi_0,\overline{\varphi}_1)$, $(\overline{\varphi}_0,\varphi_1)$, $(\overline{\varphi}_0,\overline{\varphi}_1)$.

En este caso en concreto, la síntesis óptima (además en sentido absoluto) se obtiene con la pareja $(\overline{\varphi}_0,\varphi_1)$.

Siguiendo un procedimiento análogo al indicado en el ejemplo, se ha efectuado un catálogo de las 32 funciones representativas de las clases de equivalencia n-p-n que admitiendo d.s.n.d. mejoran a su estructura arborescente correspondiente. En la Figura 2.38, vienen representadas en la columna de la derecha la estructura generalizada y en la de la izquierda la arborescente óptima.

Hay que hacer notar que en la síntesis de la función representativa $f(x_3,x_2,x_1,x_0) = \sum(7,8,12,14,15)$ (hexadecimal 018B) la estructura que resulta es arborescente aunque con funciones residuos multivariables idéntica como entradas de datos a MLUM(1) en distintos niveles. Hemos creído conveniente incluirla en la figura, pues dicha mejora se detectó estudiando las d.s.n.d. de la función.

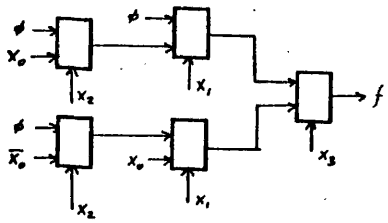
2.11.- LA DESCOMPOSICION COMPLEJA DE FUNCIONES BOOLEANAS EN LA SINTESIS DE ESTRUCTURAS GENERALIZADAS

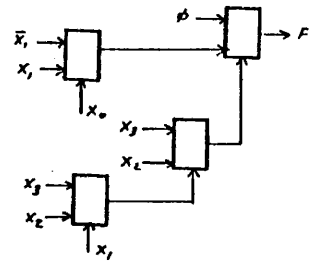
Dada una función booleana $f(X)$, la determinación de todas sus posibles d.s.d. resulta un problema interesante, pues éstas se pueden componer entre sí para dar lugar a un nuevo tipo de descomposición funcional que Curtis ([9]) denominó descomposición compleja.

Recientemente Deschamp ([10]) ha algebrizado el problema al demostrar que el conjunto de los conjuntos ligados (es decir, de las particiones disjuntas $X_1|X_2$ del conjunto X que dan lugar a d.s.d.) tiene estructura de retículo. Esta propiedad se explota por Deschamp para desarrollar un método analítico de tipo no exhaustivo que permite encontrar todos los conjuntos ligados.

Definición 2.11.- Sea X un conjunto de n variables $\{x_{n-1},\ldots,x_0\}$ entradas de una función f, y sean $X_1,X_2,\ldots,X_m$ subconjuntos propios de X que verifican:

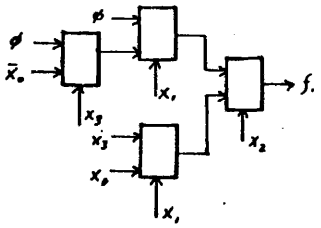
D5ND

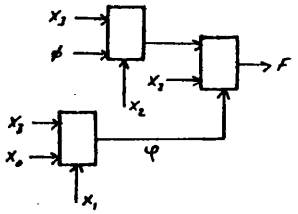




0119

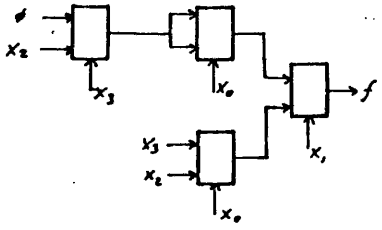
$f = F(x_1, x_0, \varphi(x_3, x_2, x_1))$

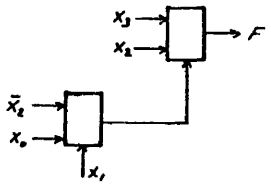




012D

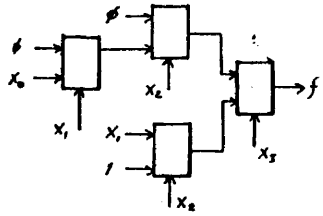
$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

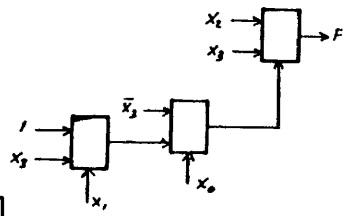




012F

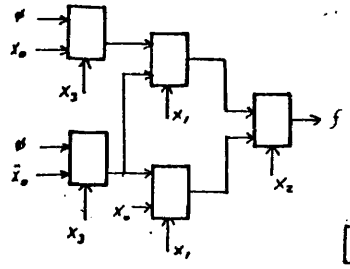
$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$

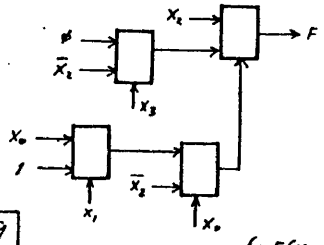




013F

$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

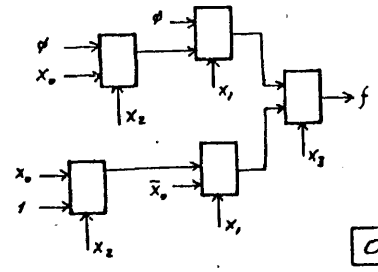


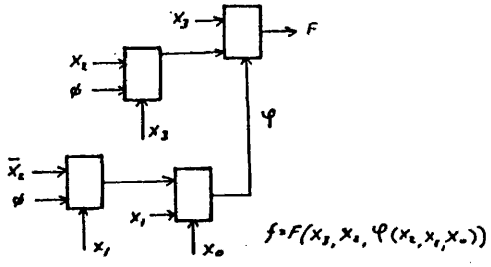


0169

$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$

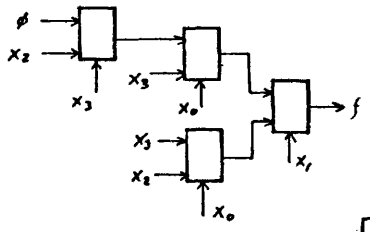
DSND

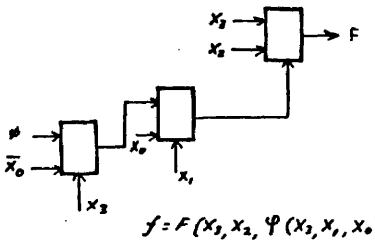




$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$

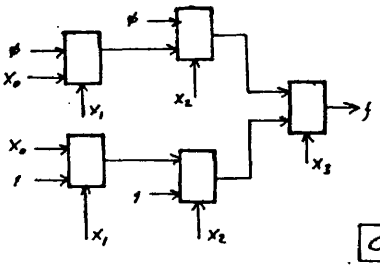
016E

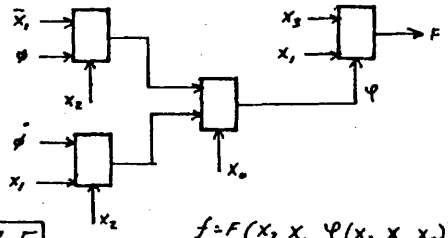




$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

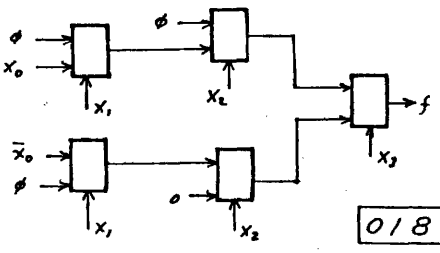
016F

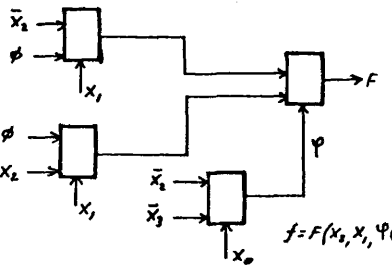




$f = F(x_3, x_1, \varphi(x_2, x_1, x_0))$

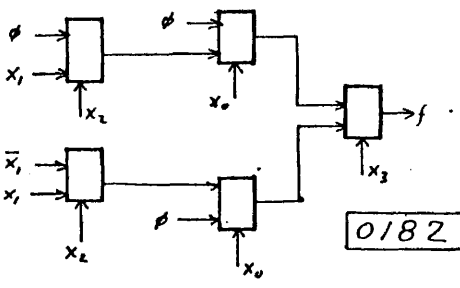
017F

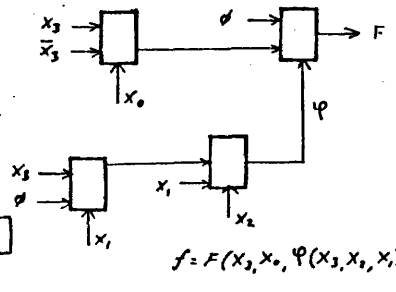




$f = F(x_3, x_1, \varphi(x_3, x_2, x_0))$

0180

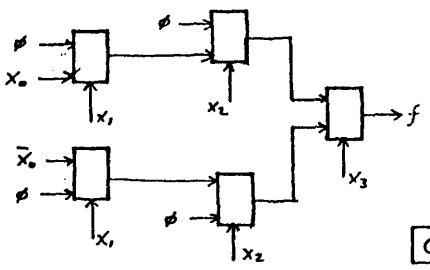




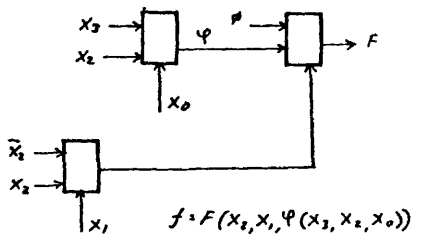
$f = F(x_3, x_0, \varphi(x_3, x_2, x_1))$

0182

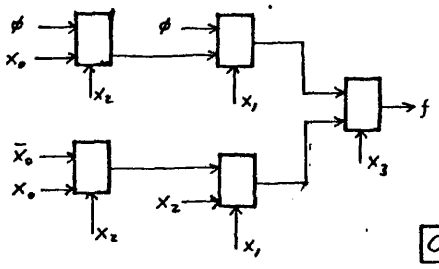
DSND



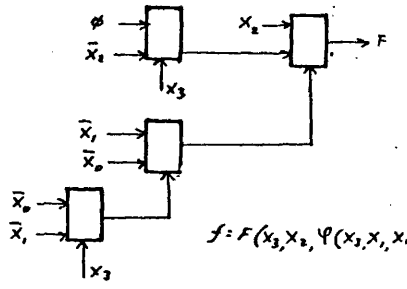
0183



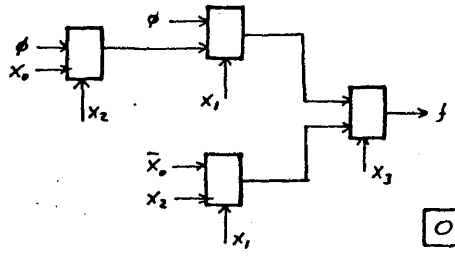
$f = F(x_2, x_1, \varphi(x_3, x_2, x_0))$



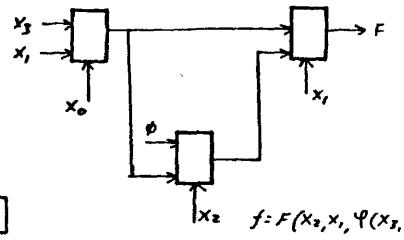
0187



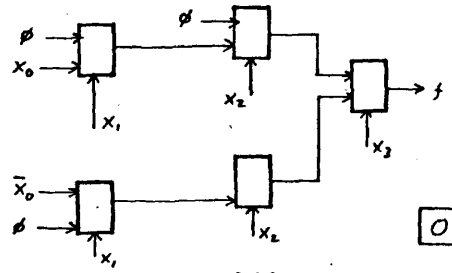
$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$



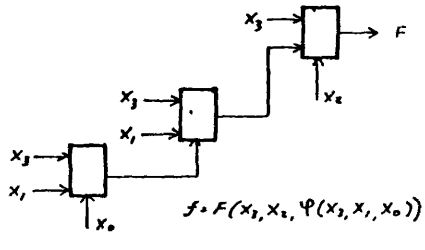
0188



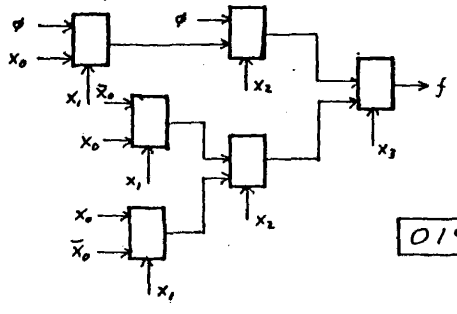
$f = F(x_2, x_1, \varphi(x_3, x_1, x_0))$



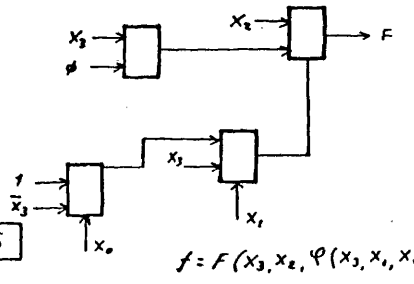
0189



$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

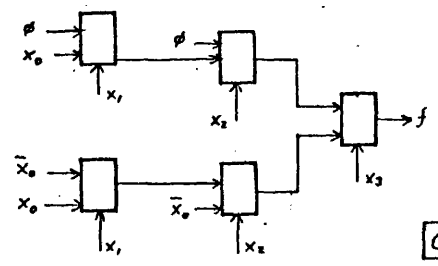


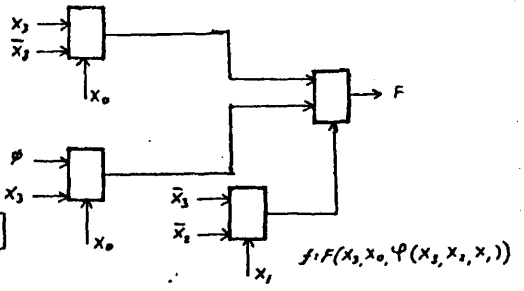
0196



$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

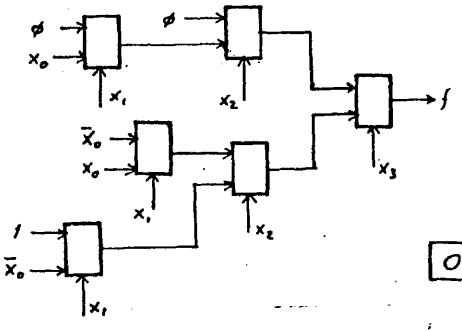
DSND

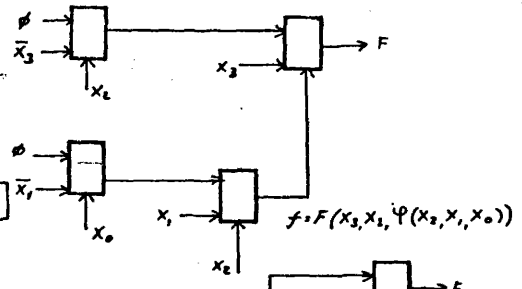




$f = F(x_3, x_0, \varphi(x_3, x_2, x_1))$

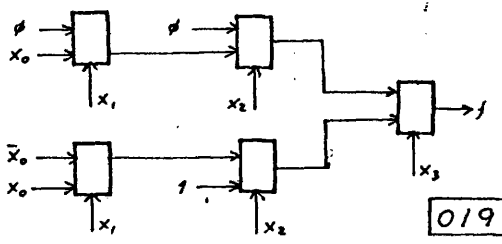
019A

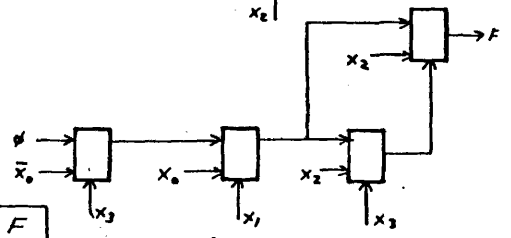




$f = F(x_3, x_1, \varphi(x_2, x_1, x_0))$

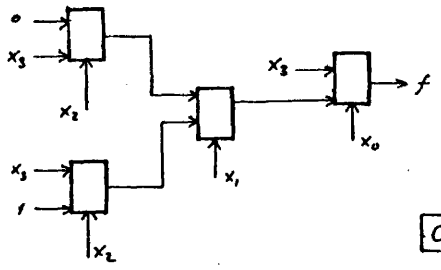
019E

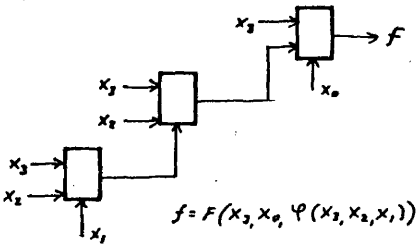




$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

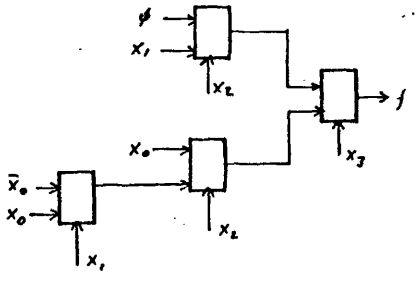
019F

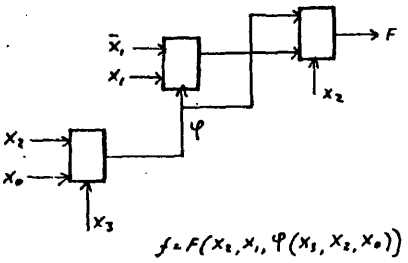




$f = F(x_3, x_0, \varphi(x_3, x_2, x_1))$

01BF

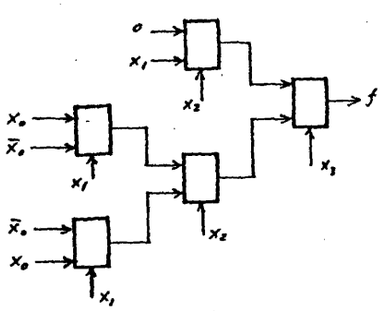




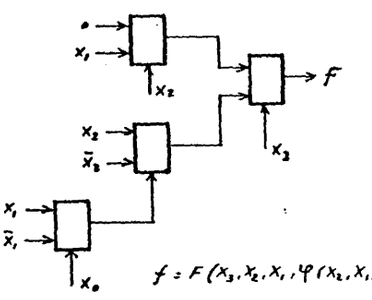
$f = F(x_2, x_1, \varphi(x_3, x_2, x_0))$

0359

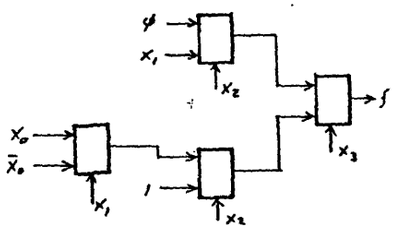
D5ND



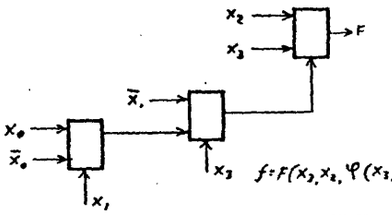
0369



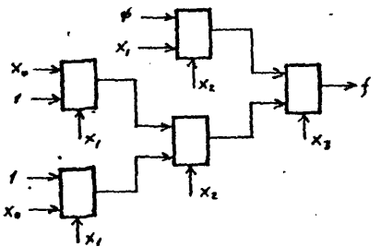
$f = F(x_3, x_2, x_1, \varphi(x_2, x_1, x_0))$



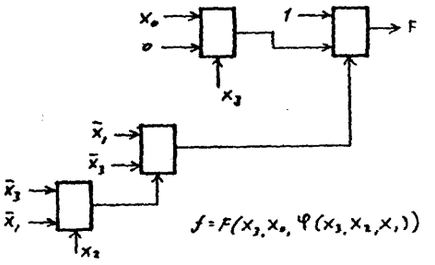
036F



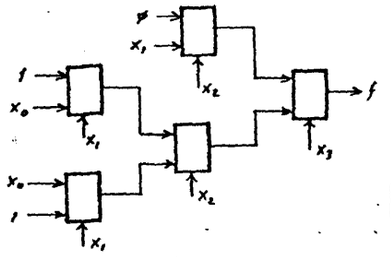
$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$



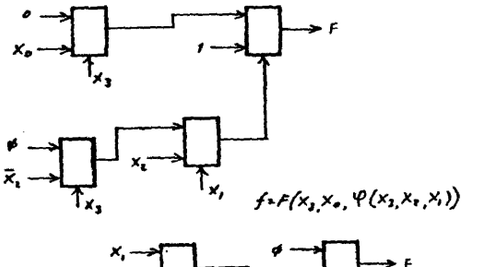
037D



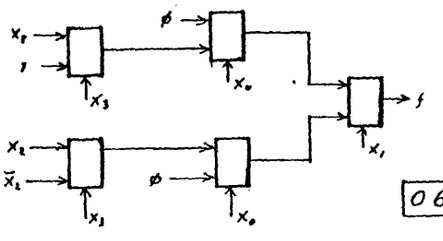
$f = F(x_3, x_0, \varphi(x_3, x_2, x_1))$



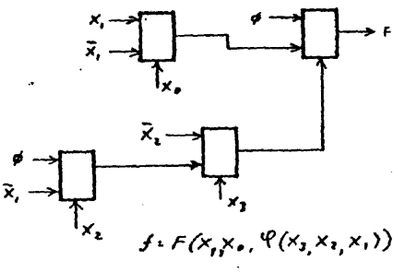
03D7



$f = F(x_3, x_0, \varphi(x_3, x_2, x_1))$

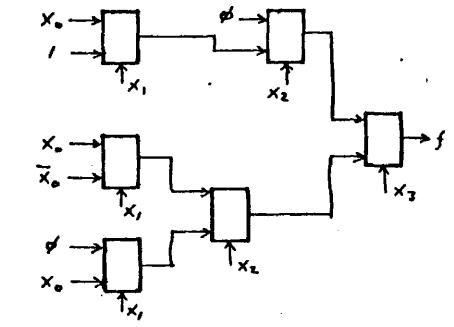


0662

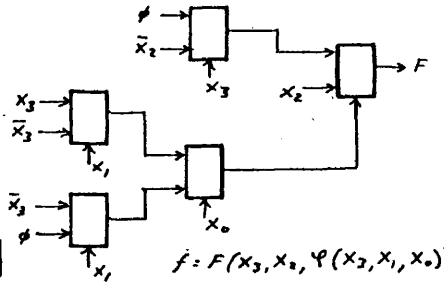


$f = F(x_1, x_0, \varphi(x_3, x_2, x_1))$

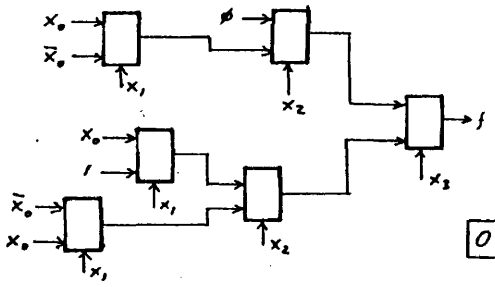
DSND



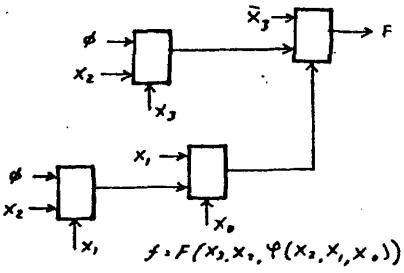
0678



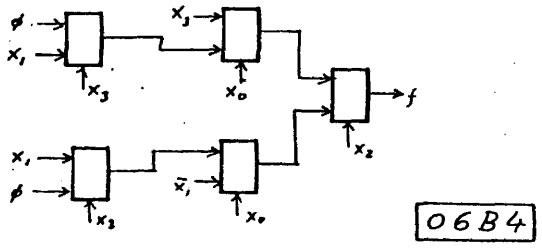
$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$



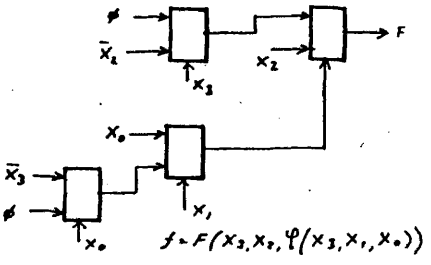
0679



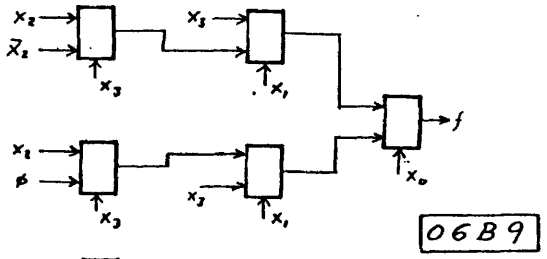
$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$



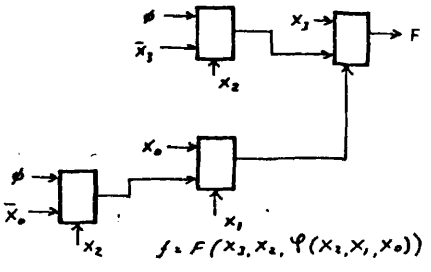
06B4



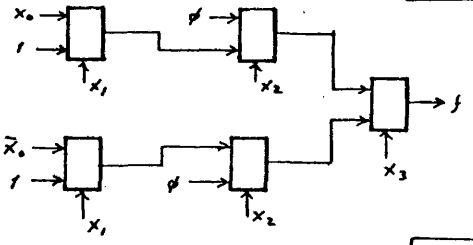
$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$



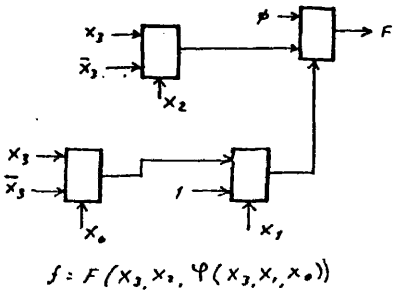
06B9



$f = F(x_3, x_2, \varphi(x_2, x_1, x_0))$



07B0



$f = F(x_3, x_2, \varphi(x_3, x_1, x_0))$

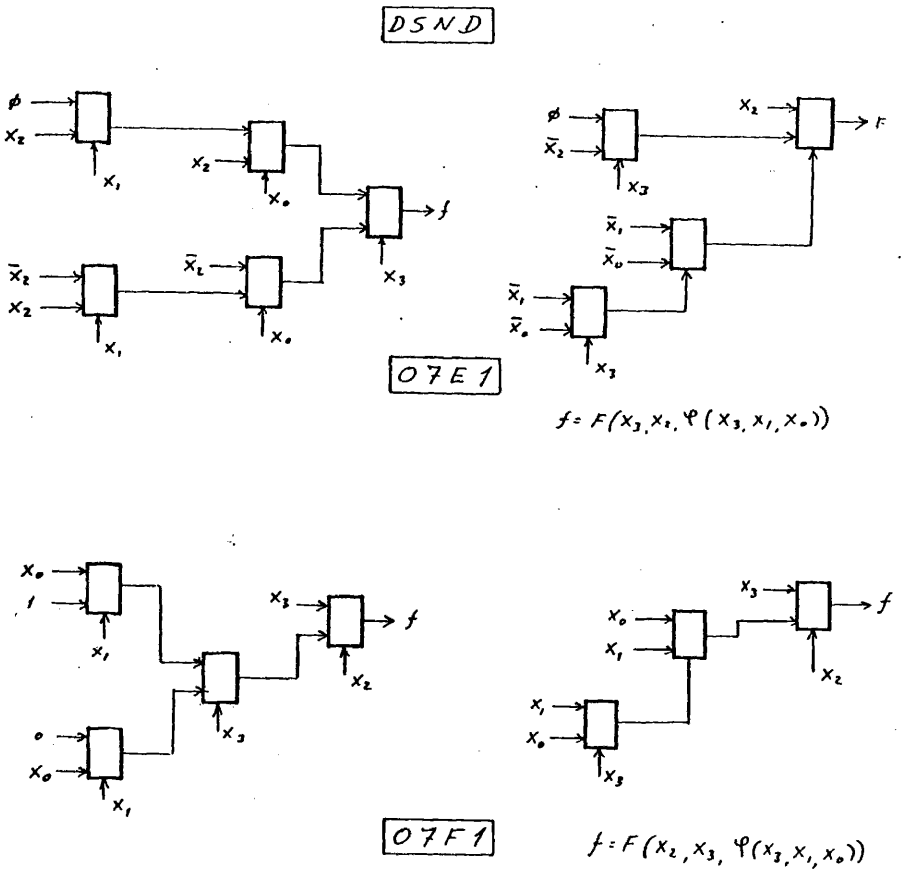


Figura 2.38.- Catálogo de las funciones representativas de las clases de equivalencia n-p-n con d.s.n.d. que mejoran la síntesis de la estructura arborescente óptima.

a) $\bigcup_{i=1}^{m} X_i = X$

b) $X_i \cap X_j = \emptyset \quad \forall\ i, j \in [1,m]\ ;\quad i \neq j$

Diremos que $f(X)$ tiene una descomposición múltiple disjunta (ver Figura 2.39 a) cuando se verifique:

$$f(X) = F[\varphi_1(X_1), \varphi_2(X_2), \ldots, \varphi_m(X_m)]$$

ó

$$f(X) = F[\varphi_1(X_1), \varphi_2(X_2), \ldots, \varphi_{m-1}(X_{m-1}), X_m] \qquad (2.32)$$

Asimismo $f(X)$, presenta una descomposición iterativa disjunta (ver Figura 2.39b) cuando

$$f(X) = F[\varphi_m[\varphi_{m-1}[\ldots \varphi_2[\varphi_1(X_1), X_2]\ldots]X_{m-1}], X_m] \qquad (2.33)$$

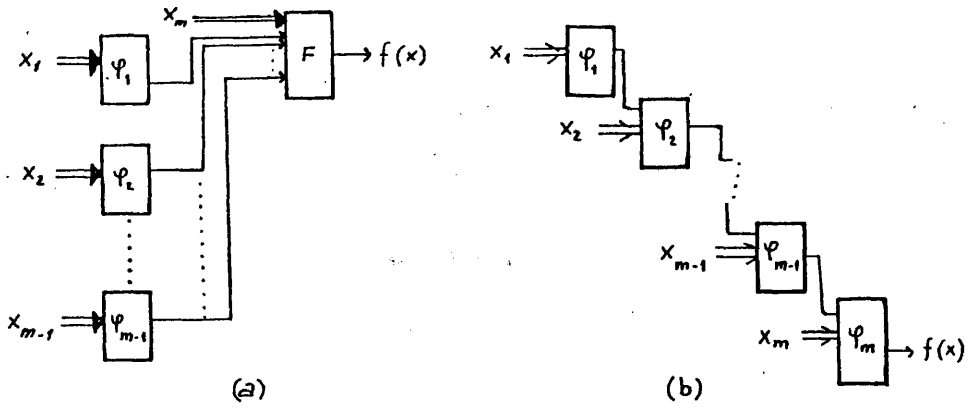


Figura 2.39.- Estructura de la descomposición múltiple disjunta (Fig. a) y de la descomposición iterativa disjunta (Fig. b).

Finalmente, se denomina descomposición compleja disjunta a cualquier forma híbrida de los dos tipos anteriores de descomposiciones.

Se supone siempre que los subconjuntos de variables X_i tienen más de un elemento (card $X_i > 1$), pues de lo contrario, las funciones simples que aparecen en la definición son triviales.

Definición 2.12.- Consideremos el retículo de los conjuntos ligados de una función de conmutación $f(X)$ dada. Los conjuntos ligados máximos para $f(X)$ son los elementos del retículo que están cubiertos por X.

El conjunto de los conjuntos ligados máximos goza de algunas propiedades importantes que se enuncian a continuación en forma de teorema, aunque sin demostrarlos.(Ver Davio-Deschamp y Thayse([11])).

Se representa por $\overline{X}_i$ el complemento de X_i con respecto a X ($\overline{X}_i = X - X_i$).

Teorema 2.9.- Sea $f(X)$ una función de conmutación que depende de todas sus variables y $\{X_0,\ldots,X_{M-1}\}$ el conjunto de sus conjuntos ligados máximos. Una de las dos propiedades siguientes se verifica:

a) $X_i \cap X_j = \emptyset \quad \forall\ i,j \in [0,1,\ldots,M-1] \quad i \neq j$

b) $X_i \cap X_j \neq \emptyset,\ X_i \cup X_j = X \Rightarrow \overline{X}_i \cap \overline{X}_j = \emptyset \quad \forall\ i,j \in [0,1,\ldots,M-1] \quad i \neq j$

El Teorema 2.9 permite una clasificación de las funciones de conmutación en funciones tipo 1 y tipo 2, según que verifiquen la propiedad a) o b). Estas funciones vienen caracterizadas por los Teoremas siguientes:

Teorema 2.10.- Sea $f(X)$ una función de conmutación de tipo 1, que depende de todas sus variables y $\{X_0,X_1,\ldots,X_{M-1}\}$ el conjunto de sus conjuntos ligados máximos. Entonces f admite una descomposición múltiple disjunta del tipo:

$$f(X) = F[\varphi_{M-1}(X_{M-1}),\ldots,\varphi_0(X_0)]$$

Teorema 2.11.- Sea $f(X)$ una función de conmutación del tipo 2, que depende de todas sus variables y $\{X_0, X_1, \ldots, X_{M-1}\}$ el conjunto de sus conjuntos ligados máximos. Entonces f admite una descomposición disjunta del tipo:

$$f(X) = \varphi_{M-1}(\overline{X}_{M-1}) \; T \ldots T \; \varphi_0(\overline{X}_0)$$

donde T representa uno de los tres operadores θ, + y ..

Ejemplo 2.16.- Se desea sintetizar con MLUM(1) la siguiente función de 6 variables:

$$f(x_5, x_4, x_3, x_2, x_1, x_0) = x_2x_1x_0 + x_3x_1x_0 + x_4x_1x_0 + x_5x_1x_0 + x_5x_2 + x_5x_3 + x_5x_4 \quad (2.34)$$

Esta función está tomada del libro "Discrete and Switching Functions", de Davio-Deschamp y Thayse, McGraw-Hill, 1978.

El diagrama de Hasse del retículo correspondiente a los conjuntos ligados de la función (2.34) se da en la figura 2.40.

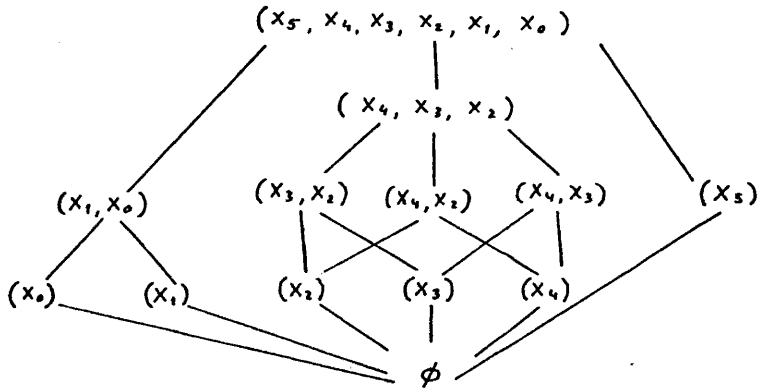


Figura 2.40.- Retículo de los conjuntos ligados de la función del Ejemplo 2.16.

Del diagrama de Hasse de la Figura 2.40, se deducen las dos d.s.d. siguientes:

$$f(X) = F_1\ (x_5, x_1, x_0, \varphi_1(x_4, x_3, x_2))$$
$$f(X) = F_2\ (x_5, x_4, x_3, x_2, \varphi_2\ (x_1, x_0)) \qquad (2.35)$$

Los conjuntos ligados máximos de $f(X)$ son: $X_0 = (x_1, x_0)$, $X_1 = (x_4, x_3, x_2)$ y $X_2 = (x_5)$

X_0, X_1 y X_2 satisfacen la condición a) del Teorema 2.9 y, por lo tanto, $f(X)$ es una función tipo 1. Aplicando el Teorema 2.10, $f(X)$ admite la descomposición múltiple disjunta (Ver Figura 2.41)

$$f(X) = F(x_5, \varphi_1(x_4, x_3, x_2), \varphi_2(x_1, x_0))$$

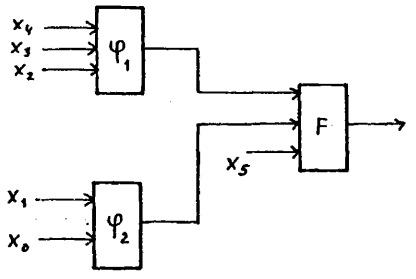
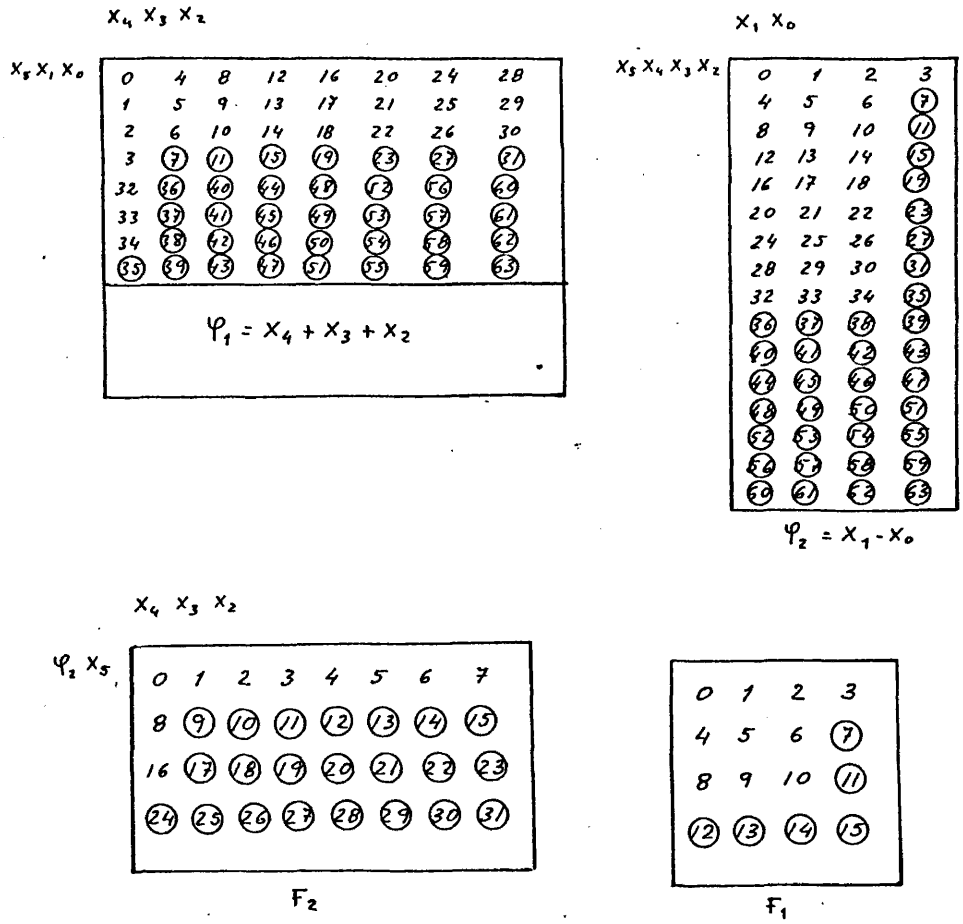


Figura 2.41.- Descomposición múltiple disjunta de la función del ejemplo 2.16.

La determinación de las funciones φ_1, φ_2 y F se presentan en las cartas de descomposición de la Tabla 2.13.

Las ecuaciones de la descomposición son:

$$\varphi_1(x_4, x_3, x_2) = x_4 + x_3 + x_2\ ; \quad \varphi_2\ (x_1, x_0) = x_1\ x_0$$
$$F(\varphi_2, \varphi_1, x_5) = \bar{\varphi}_2\ \varphi_1\ x_5 + \varphi_2\ \varphi_1\ \bar{x}_5 + \varphi_2\ x_5 \qquad (2.36)$$



$$F = \overline{\varphi}_2\, \varphi_1\, x_5 + \varphi_2\, \varphi_1\, \overline{x}_5 + \varphi_2\, x_5$$

TABLA 2.13.- Cartas de descomposición para la determinación de las funciones φ_1, φ_2 y F del Ejemplo 2.16.

Las funciones φ_1 y F de 3 variables pertenecen a las clases de equivalencia n-p-n números 2 y 8 respectivamente. Su síntesis con MLUM(1) se dan en la Figura 2.42 a,b, y c). La estructura generalizada resultante al combinar las funciones φ_1, φ_2 y F es la de la Figura 2.42d).

A toda descomposición se le puede asociar una cota superior en el número de módulos necesitados para su implementación que será la suma de las cotas superiores de las subfunciones que intervienen en la descomposición. Sin embargo, hay una descomposición (si existe) que hace mínima dicha cota superior.

Veamos esto en el caso de la descomposición iterativa disjunta.

Teniendo en cuenta la Figura 2.39b. Sea:

$$Card(X_1) = n_1$$
$$Card(X_i) = n_i - 1 \qquad i = (2,\ldots,m)$$

Si la síntesis se va a efectuar con MLUM(p), y en el caso general de que p no divida exactamente a n_i-p, será necesario utilizar para cada subfunción $\varphi_i (i=1,\ldots,m)$ como máximo el número de módulos siguientes (ver expresión (1-11) del capítulo anterior).

$$J_i = 1 + 2^{p_{1i}} \left(\frac{k^{\ell_i - 1} - 1}{k-1} \right) \qquad (2.37)$$

donde $k = 2^p$

$\ell_i = \lceil n_i - 1/p \rceil$, $\lceil x \rceil$ es el menor entero mayor o igual a x

$p_{1i} = (n_i - 1) - p\,(\ell_i - 1)$; $\quad 1 \leq p_{1i} \leq p$

p_{1i} es el número de entradas de control utilizadas en el módulo del primer nivel de la función φ_i.

La cota superior en el número de módulos utilizados será:

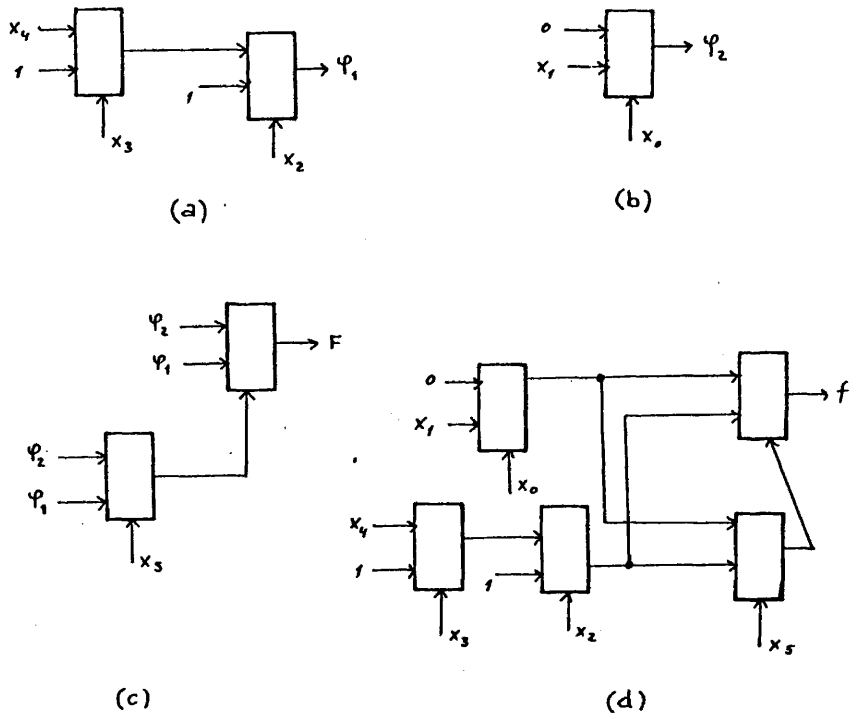


Figura 2.42.- Síntesis de la función del Ejemplo 2.16, mediante una descomposición múltiple disjunta.

$$J = \sum_{i=1}^{m} J_i = \sum_{i=1}^{m} \left[1 + 2^{p_{1i}} \left(\frac{k^{\ell_i - 1} - 1}{k - 1} \right) \right] \qquad (2.38)$$

Sujeto a la siguiente ligadura:

$$\Psi (n_1, \ldots, n_m) = \sum_{i=1}^{m} n_i - m - n + 1 = 0 \qquad (2.39)$$

Aplicando el método de los multiplicadores de Lagrange:

$$\frac{\partial J}{\partial n_i} + \lambda \frac{\partial \Psi}{\partial n_i} = 0 \qquad i = (1, \ldots, m)$$

Como $\partial \Psi / \partial n_i = 1 \quad \forall\, i$ derivando J con respecto a n_i y n_j e igualando da:

$$2^{p_{1i} - p_{1j}} = k^{n_j - n_i / p} \qquad \forall\ i, j \qquad (2.40)$$

Si $p_{1i} = p_{1j}$, de (2.40) se deduce que $n_j = n_i \quad \forall\ i,j$. Sea $N = n_1 = n_2 = , \ldots, = n_m$, sustituyendo en la ligadura (2.39) se deduce:

$$N = \frac{n-1}{m} + 1 \qquad (2.41)$$

que nos fijan los tamaños óptimos de las particiones disjuntas del conjunto X.

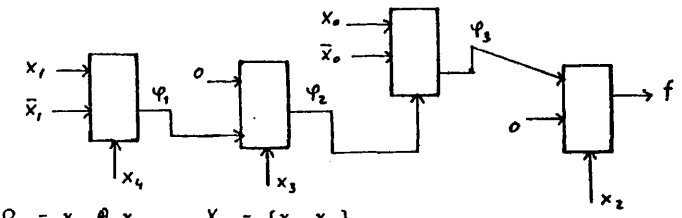
Ejemplo 2.17.- Se desea sintetizar la función de 5 variables siguiente (Ver Torrent ([12]))

$$f(x_4, x_3, x_2, x_1, x_0) = \sum (1,3,9,10,17,19,24,27) = \bar{x}_3 \bar{x}_2 x_0 + \bar{x}_4 \bar{x}_2 \bar{x}_1 x_0 + x_4 \bar{x}_2 x_1 \bar{x}_0 + x_4 x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0 + \bar{x}_4 x_3 \bar{x}_2 x_1 \bar{x}_0 \qquad (2.42)$$

(2.42) admite una descomposición iterativa disjunta del tipo:

$$f(x_4,x_3,x_2,x_1,x_0) = \varphi_4\ (x_2, \varphi_3(x_0, \varphi_2(x_3, \varphi_1(x_4,x_1))))$$

En este caso $n=5$ y $m=4$, de (2.41) se sigue que $N=2$ y la descomposición tiene una cota superior óptima ($J=4$) en el número de módulos necesarios para implementar la función (Ver Figura 2.43).



$\varphi_1 = x_4 \oplus x_1$ $\quad X_1 = \{x_4, x_1\}$

$\varphi_2 = x_3 \cdot \varphi_1$ $\quad X_2 = \{x_3\}$

$\varphi_3 = x_0 \oplus \varphi_2$ $\quad X_3 = \{x_0\}$

$\varphi_4 = \bar{x}_2 \cdot \varphi_3$ $\quad X_4 = \{x_2\}$

Figura 2.43.- Síntesis de la función $f(x_4,x_3,x_2,x_1,x_0) = \sum(1,3,9,10,17,19,24,27)$

En el caso de que en la Definición (2.11) no se cumpla la condición b), es decir, que se verifique $X_i \cap X_j \neq \emptyset$ para algún par (i,j), entonces las descomposiciones múltiples e iterativas dadas por las expresiones (2.32) y (2.33) pasan a ser de tipo no disjunto.

Un estudio sistemático de este tipo de descomposiciones no ha sido aún efectuado, sin embargo, vamos a poner de manifiesto en un ejemplo cómo una función de 4 variables que no presenta d.s.d. ni d.s.n.d., sí tiene una descomposición múltiple no disjunta que requiere para su síntesis 3 módulos MLUM(1), mientras que la estructura arborescente óptima requiere 4 MLUM(1).

Ejemplo 2.18.- Se desea sintetizar la función de 4 variables

$$\delta(x_3,x_2,x_1,x_0) = \sum(7,8,10,12,13,15)$$

$$\delta(x_3,x_2,x_1,x_0) = x_2x_1x_0 + x_3\bar{x}_2\bar{x}_0 + x_3x_2\bar{x}_1 = (\bar{x}_2\bar{x}_0 + x_2x_0)(x_2x_1 + \bar{x}_2x_0 + x_1x_0) +$$

$$+ x_3(\bar{x}_2\bar{x}_0 + x_2\bar{x}_1)$$

Llamando:

$$\varphi_1(x_0,x_2) = \bar{x}_2\bar{x}_0 + x_2x_0$$

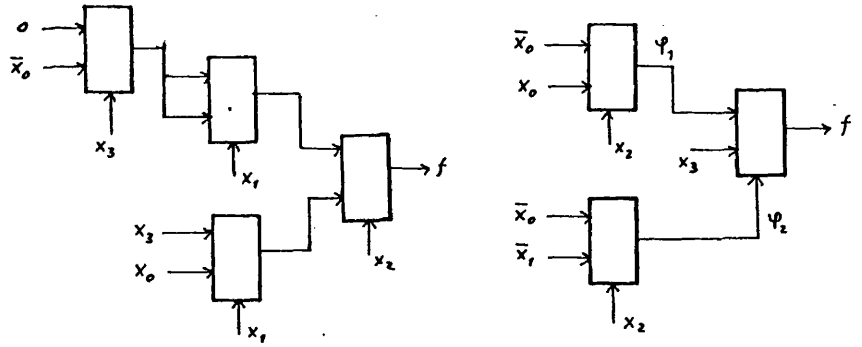
$$\bar{\varphi}_2(x_0,x_1,x_2) = x_2x_1 + \bar{x}_2x_0 + x_1x_0$$

Se puede expresar δ de la forma siguiente:

$$\delta(x_3,x_2,x_1,x_0) = \varphi_1(x_0,x_2)\,\bar{\varphi}_2(x_0,x_1,x_2) + x_3\varphi_2(x_0,x_1,x_2) = \quad (2.43)$$

$$= F(\varphi_1(x_0,x_2), \varphi_2(x_0,x_1,x_2), x_3)$$

A esta descomposición múltiple no disjunta le corresponde la síntesis de la Figura 2.44b) que, en este caso, podemos afirmar que es óptima.



a) Estructura arborescente óptima

b) Descomposición múltiple no disjunta.

Figura 2.44.- Síntesis de la función $\delta(x_3,x_2,x_1,x_0) = \sum(7,8,10,12,13,15)$

Este ejemplo pone de manifiesto cómo el tipo de descomposición compleja no disjunta juega un papel importante en la síntesis de estructuras generalizadas óptimas

2.12.- SINTESIS DE FUNCIONES MULTIPLES CON MLUM(1): EMPLEO DE LA DESCOMPOSICION FUNCIONAL

La extensión de los métodos de descomposición funcional a la síntesis de funciones múltiples es un problema que no ha sido tratado al menos en nuestro conocimiento. En esta sección se indica una posible línea de enfoque para el problema y se define el concepto de descomposición simple disjunta múltiple (d.s.d.m.) que permite en determinados casos efectuar la síntesis de funciones múltiples.

Sin embargo, esta línea de acción deberá, en un futuro próximo, ser objeto de una mayor investigación.

Definición 2.12.- Dada una función de conmutación múltiple, $\underline{f}(X)$: $B^n \rightarrow B^m$, completamente especificada, $X = (x_{n-1},\ldots,x_0)$, $B^i = \underbrace{BxBx,\ldots,xB}_{i \text{ veces}}$; $B = (0,1)$.

Sea $\{X_1|X_2\}$ una partición disjunta del conjunto X. Diremos que $\underline{f}(X)$ admite una descomposición simple disjunta múltiple (Ver Figura 2.45) si se puede expresar como:

$$\underline{f}(X) = \underline{F}(X_1, \varphi(X_2))$$

$$\varphi: B^t \rightarrow B(card(X_2) = t),\ F: B^{s+1} \rightarrow B^m\ (card(X_1) = s),\ s + t = n$$

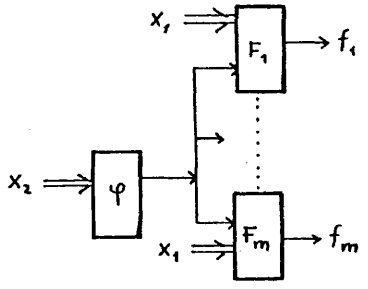


Figura 2.45.- Descomposición simple disjunta múltiple (d.s.d.m.).

Algoritmo para la determinación de descomposiciones disjuntas múltiples

El algoritmo para determinar d.s.d.m. es una extensión del algoritmo de Deschamp ([10]) para d.s.d. Como se sabe, el conjunto de los conjuntos ligados (es decir, de las particiones disjuntas $X_1|X_2$ del conjunto X que dan lugar a d.s.d.) tiene estructura de retículo. Esta propiedad se explota por Deschamp para desarrollar un método analítico de tipo no exhaustivo que permite encontrar todos los conjuntos ligados.

En el procedimiento de determinación de d.s.d.m. se van determinando sucesivamente las listas A_i de conjuntos ligados, con i variables en el conjunto X_2, de las funciones simples (δ_j) que componen la función múltiple desde $i = 1,..,n$ $j=1,..,m$. En cada paso se van almacenando informaciones de aquellos conjuntos ligados, que son idénticos, y a los que corresponde la misma función φ. El proceso termina cuando, o bien no existe una lista de conjuntos ligados o no hay ninguna función φ común, al menos, en las listas correspondientes a dos funciones.

Ejemplo 2.19.- Se desea sintetizar la siguiente función múltiple de 4 variables

$$X = (x_3, x_2, x_1, x_0)$$
$$\delta_1(X) = \sum(1,2,4,7,8,11,13,14)$$
$$\delta_2(X) = \sum(1,2,4,7,8,9,10,11,12,13,14,15)$$
$$\delta_3(X) = \sum(9,10,12,15)$$

El diagrama de Hasse del retículo correspondiente a los conjuntos ligados de cada función se da en la Figura 2.46.

Se observa que el conjunto ligado $X_2 = (x_2, x_1, x_0)$ da lugar a la misma función $\varphi(x_2)$ en la d.s.d. de cada una de las funciones. Así pues, en este caso,

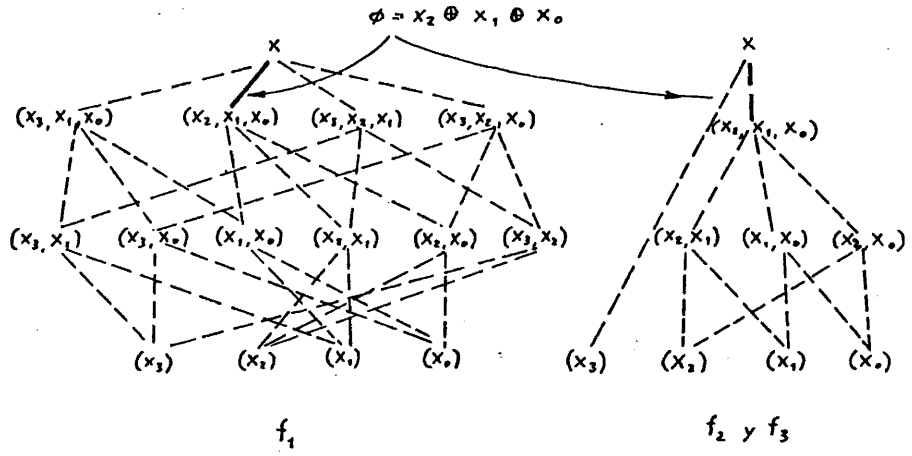


Figura 2.46.- Diagrama de Hasse de la función múltiple del Ejemplo.

la función múltiple $\underline{f}(x)$ admite d.s.d.m. En la Figura 2.47 está representada su implementación con módulos lógicos universales tipo multiplexor de una entrada de control(MLUM(1)).

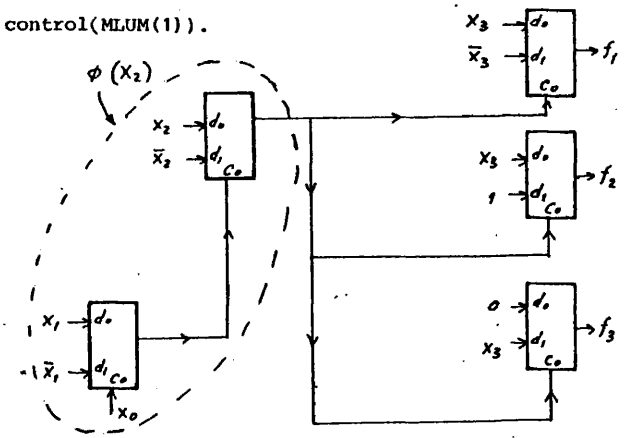


Figura 2.47.- Desarrollo con MLUM(1) de la función múltiple del Ejemplo

REFERENCIAS

(1) VOITH, R.P., "ULM implicants for minimization of universal logic module Circuits". IEEE Trans. on Computers, vol. C-26, May 1977, pp. 417-424.

(2) GLOVER, F., "A multiphase-dual algorithm for zero-one integer programming problem", Operations Research, Vol. C-21, June 1972, pp. 573-582.

(3) DORMIDO, S., and CANTO, M.A., "Systematic Synthesis of combinational circuits using multiplexers", Electronics Letters, Voo. 14, n° 18, 31 August 1978, pp. 588-590.

(4) YAU, S.A., and TANG, C.K., "Universal Logic circuitos and their modular realizations", Proceedings AFIPS, 1968, pp. 297-305.

(5) VAN HOLTEN, C., "Double multiplexer logic capability". Electronic Design, August 1974, pp. 86-89.

(6) BLAKESLEE, T.R., "Digital design with standard MSI and LSI", John Wiley 1975, pp. 65-73.

(7) WHITEHEAD, D.G., "Algorithm for logic-circuit synthesis by using multiplexers", Electronics Letters, Vol. 13, n° 12, june 1977, pp. 355-356.

(8) LANGDON, G.G., "A decomposition chart technique to aid in realizations with multiplexers", IEEE Trans. on Computers, Vol. C-27, Feb. 1978, pp. 157-159.

(9) CURTIS, M.A. "A new approach to the design of switching circuits", Van Nostrand, 1962.

(10) DESCHAMPS, J.P., "Binary simple decompositions of discrete functions". Digital Processes, Vol. 1, 1975, pp. 123-140.

(11) DAVID, M., DESCHAMPS, J.P. and THAYSE, A., "Discrete and switching functions", McGraw-Hill, 1978.

(12) TORRENT, J., "Descomposición simple disjunta de funciones booleanas completas", Tesina de Licenciatura en C. Fís. Univ. Autónoma de Barcelona, 1978.

SEGUNDA PARTE

CAPITULO III

SINTESIS DE CONTADORES PARALELOS GENERALIZADOS

3.1.- INTRODUCCION

Con la aparición de los circuitos integrados de una elevada escala de integración VLSI (very large scale integration), se ha hecho muy notorio que los métodos de síntesis de circuitos aritméticos necesitaban un tratamiento diferente al empleado hasta ahora.

En el presente capítulo se aborda el estudio de contadores paralelos generalizados como bloque fundamental en la síntesis de circuitos aritméticos.

Este tipo de unidad tiene, entre otras, como aplicaciones inmediatas, la síntesis de sumadores de entrada múltiple ([1])([2])([3]), de multiplicadores tipo paralelo ([4]) (cuestión que abordaremos en el próximo capítulo), de procesadores asociativos ([5]), etc.

El camino que se va a seguir es el siguiente: en primer lugar, se define formalmente qué entendemos por un circuito aritmético, para lo cual seguimos, en cierta medida, la línea de pensamiento desarrollada por Meo ([6]). Dentro de esta categoría de circuitos nos centramos sobre una clase especial que Dadda ([7]) denomina contadores paralelos y que posteriormente han sido desarrollados por Foster y Stockton ([5]), Swartzlander ([8]), Kobayashi y Ohara ([9]) y Current y Mow ([10]). A partir de estos trabajos se define un nuevo tipo de contador paralelo que denominamos generalizado que incluye y amplía todos los resultados obtenidos hasta el momento presente.

Se desarrolla un algoritmo de reducción que permite efectuar la síntesis de cualquier contador paralelo generalizado que se desee, utilizando únicamente un solo tipo de módulo en la misma. Este hecho es de indudable interés en la síntesis de circuitos aritméticos VLSI.

En determinados casos se demuestra que el procedimiento de síntesis es óptimo en cuanto a que minimiza el número de módulos básicos (primitivos) utilizados. En el caso general se da una cota inferior del número de módulos que se necesitan. Pruebas extensivas del algoritmo en un ordenador digital muestran que el procedimiento presenta resultados muy satisfactorios en todos los casos. Asimismo, se han determinado dos cotas (superior e inferior) para el número de niveles necesarios para efectuar la síntesis, y que posibilitan el evaluar rápidamente el retardo que se produce.

Conviene indicar que este tipo de unidad está bien adecuada para procesos tipo "pipeline", que aumentarían grandemente su velocidad operativa.

3.2.- CIRCUITOS ARITMETICOS: CONTADORES PARALELOS

Definición 3.1.- Un circuito de conmutación combinacional se denomina aritmético si es posible encontrar una partición del conjunto {E} de sus variables de entrada en los subconjuntos

$$E_i = \{e_{i1},\ldots,e_{im_i}\} \qquad i = 0,1,\ldots,p$$

y una partición del conjunto {S} de variables de salida en tantos subconjuntos como los de entrada

$$S_i = \{s_{i1},\ldots,s_{in_i}\} \qquad i = 0,1,\ldots,p$$

tal que:

$$\sum_{i=0}^{p} \sum_{j=1}^{m_i} e_{ij}\, 2^i = \sum_{i=0}^{p} \sum_{j=1}^{n_i} s_{ij}\, 2^i \qquad (3.1)$$

para cualquier conjunto de valores de las variables independientes e_{ij} (ver Figura 3.1).

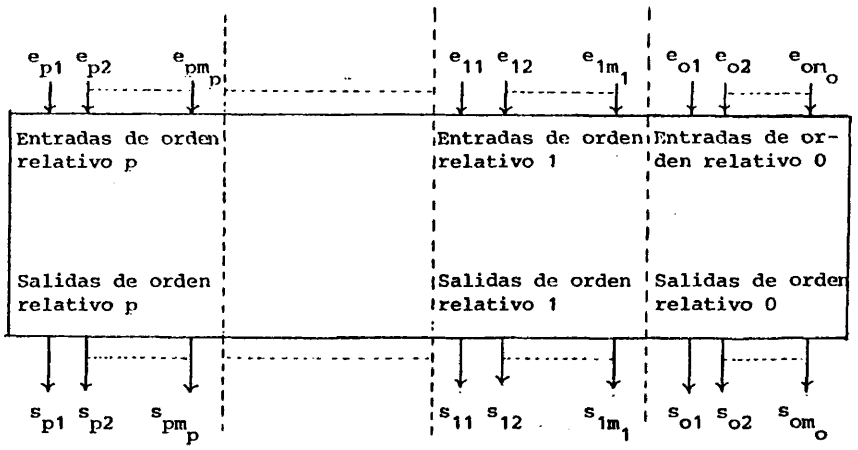


Figura 3.1.- Representación general de un circuito aritmético

En otras palabras, un circuito aritmético es un circuito combinacional que procesa un conjunto de dígitos binarios de entrada (en general de pesos diferentes) para dar un conjunto de dígitos binarios de salida cuya suma es igual a la suma de los dígitos de entrada.

Ejemplos clásicos de circuitos aritméticos son los sumadores binarios convencionales y los circuitos utilizados en los multiplicadores tipo paralelo para procesar la matriz de productos elementales.

El primero de los dos subíndices de una variable e_{ij} o s_{ij} se denomina el orden relativo de e_{ij} o s_{ij} en el circuito dado; el valor 2^i es el peso relativo correspondiente y el producto $e_{ij}2^i$ o $s_{ij}2^i$ corresponde al valor relativo de la variable. El lado izquierdo de (3.1) será pues el valor relativo total de la entrada y el lado derecho el valor relativo total de la salida.

En general, se considerarán sistemas compuestos de circuitos aritméticos conectados de tal forma que los pesos de las señales de orden menor (e_{oj} o s_{oj}) de un circuito particular serán diferentes de las de otro circuito en el mis-

mo sistema.

En un circuito como en el de la Figura 3.1, las señales aplicadas a las entradas de orden relativo 0, tendrán un peso de 2^{c_k}; la constante c_k asociada con dicho circuito dentro del sistema es un entero que se denomina orden del circuito. De esta forma $c_k + i$ será el orden absoluto de la variable e_{ij} o s_{ij}, 2^{c_k+i} el peso absoluto y $e_{ij}\,2^{c_k+i}$ o $s_{ij}\,2^{c_k+i}$ el valor absoluto de la variable en cuestión.

De (3.1) se sigue que:

$$\sum_{i=0}^{p}\sum_{j=1}^{m_i} e_{ij}\,2^{c_k+i} = \sum_{i=0}^{p}\sum_{j=1}^{n_i} s_{ij}\,2^{c_k+i} \qquad (3.2)$$

El lado izquierdo de (3.2) será el valor absoluto total de la entrada y el lado derecho el valor absoluto total de la salida.

Definición 3.2.- Un circuito aritmético se denomina triangular si cada uno de los conjuntos de salida s_i contienen únicamente un elemento s_{i1}.

Esta clase de circuitos aritméticos son importantes desde el punto de vista de las aplicaciones. La salida de tales circuitos es un número binario cuyo valor es igual al valor total de las entradas.

Definición 3.3.-La interconexión de circuitos aritméticos forma un sistema regular si se verifican las dos condiciones siguientes:

a) ninguna salida se puede conectar a más de una entrada y viceversa.

b) las entradas y salidas que se conecten entre sí deben tener el mismo peso absoluto.

Un problema importante que surge inmediatamente asociado con la definición de sistema regular de circuitos aritméticos es el de su estabilidad. Dichos

sistemas pueden ser inestables en el sentido usual de que no existe un estado de equilibrio (para toda posible combinación de entradas) que se alcance en tiempo finito desde el instante en que se aplica la entrada.

En la Figura 3.2 está representado el caso de un sumador completo (full-adder) en el que una de sus salidas se conecta a una de sus entradas.

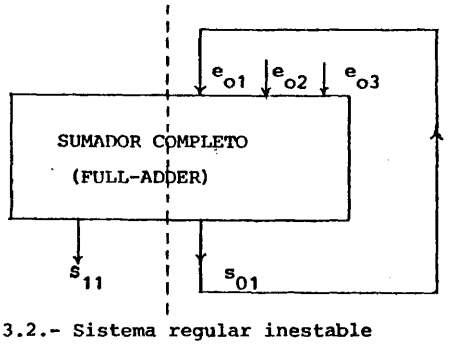


Figura 3.2.- Sistema regular inestable

Sea Δt el retardo del circuito. Se puede observar una condición de inestabilidad si comenzando desde un estado inicial estable con $e_{01} = e_{02} = e_{03} = s_{01} = s_{11} = 0$, se aplican en t_0 las señales $e_{02} = 0$, $e_{03} = 1$

	t_0	$t_0+\Delta t$	$t_0+2\Delta t$	$t_0+3\Delta t$
s_{01}	0	1	0	1
s_{11}	0	0	1	0

Este caso es un ejemplo de una condición general que puede ocurrir cuando se introduce realimentación.

Conocido un sistema, el conjunto de conexiones para un peso dado puede describirse por un grafo orientado, en el cual los nodos corresponden a los circuitos componentes y las aristas representan la existencia de una conexión entre los dos circuitos asociados por dicha arista. Un grafo de este tipo se denomina grafo

particular de orden p. Se denomina grafo total del sistema, al grafo que se obtiene superponiendo los grafos particulares de todos los órdenes del sistema.

Un grafo se dice cerrado si existe, al menos, una secuencia dirigida de nodos de la forma $A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_{n-1} \rightarrow A_n \equiv A_0$ si no, se dirá abierto.

Si el grafo total es abierto se llamará completamente abierto y cuasi-abierto en el caso de que todos los grafos particulares sean abiertos pero no el grafo total.

Dos condiciones de estabilidad para sistemas abiertos y cuasiabiertos son las siguientes:

Proposición 3.1.- Un sistema abierto es siempre estable.

Proposición 3.2.- En un sistema cuasiabierto, tal que cualquier circuito componente verifica la condición de que las salidas de cualquier peso son independientes de las variables de los pesos de orden superior es siempre estable. Esto permite establecer la siguiente proposición.

Proposición 3.3.- Una condición suficiente para que un sistema triangular sea estable es que sea cuasiabierto.

En lo que sigue, los sistemas que se consideran cumplen alguna de las proposiciones anteriores y, por tanto, se puede garantizar a priori su estabilidad.

Los contadores paralelos introducidos por Dadda ([7]) como una unidad básica en la realización de multiplicadores paralelos corresponden a un tipo particular de circuitos aritméticos triangulares.

Definición 3.4.- Un contador paralelo $(p;d)$ es un circuito combinacional con d salidas y $p \leq 2^d - 1$ entradas, donde el número binario representado por las d salidas es el número de "unos" presentes en las entradas (ver Figura 3.3).

De la Definición 3.5 se sigue que el número de salidas d debe verificar que:

$$2^d - 1 \geq \sum_{i=0}^{n-1} p_i \, 2^i > 2^{d-1} - 1 \qquad (3.3)$$

Un contador paralelo presenta utilización completa si se cumple la condición:

$$2^d - 1 = \sum_{i=0}^{n-1} p_i \, 2^i \qquad (3.4)$$

En este caso, el contador paralelo generalizado se dirá que está saturado.

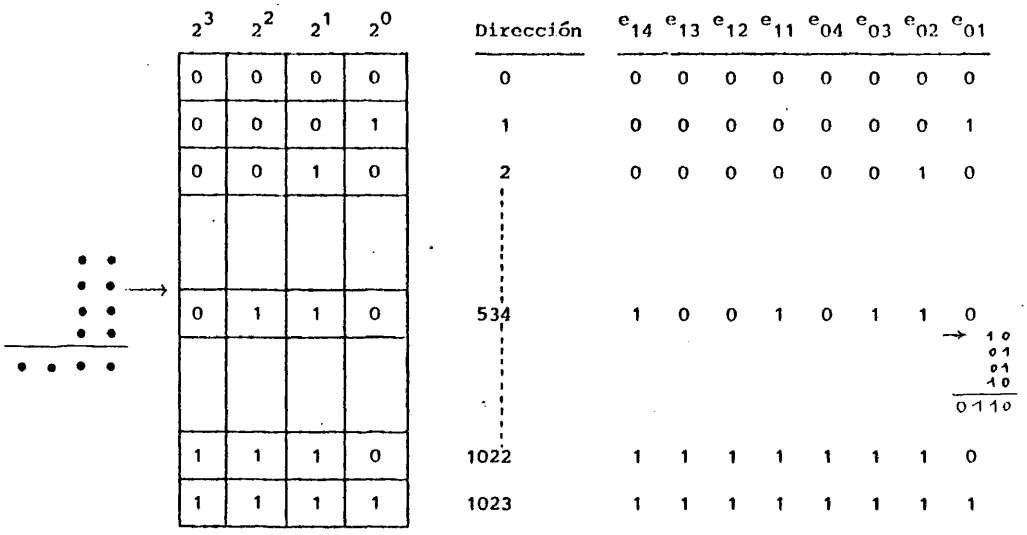
En un contador saturado, todas las combinaciones de salida son significativas, mientras que en un contador no saturado, algunas combinaciones de salida nunca estarán presentes.

Un contador paralelo generalizado puede ser realizado con una ROM de

$$2^{\sum_{i=0}^{n-1} p_i} \times d$$

bits, que se utiliza como una tabla de consulta a la que se accede dando como dirección la configuración correspondiente de sus entradas. En la Figura 3.5 está representada la implementación de un contador paralelo (5,5;4) con una ROM de 1024 x 4 bits.

En la Tabla 3.1(pag.208) están representados aquellos contadores que con la tecnología actual pueden ser implementados en un solo chip. Unicamente se consideran aquellos contadores en los que el número de bits por columnas es idéntico, es decir, $p_0 = p_1 = \ldots = p_{n-1}$. El problema que surge es que cuando $d > 8$, el tamaño de la memoria se hace excesivamente grande y no se puede realizar en un único circuito integrado, lo que se debe en parte al elevado grado de redundancia que exis-



2^3	2^2	2^1	2^0	Dirección	e_{14}	e_{13}	e_{12}	e_{11}	e_{04}	e_{03}	e_{02}	e_{01}
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	1	0	2	0	0	0	0	0	0	1	0
0	1	1	0	534	1	0	0	1	0	1	1	0
1	1	1	0	1022	1	1	1	1	1	1	1	0
1	1	1	1	1023	1	1	1	1	1	1	1	1

Figura 3.5.- Realización de un contador paralelo (5,5;4) con una ROM de 1024 x 4 bits.

ten en las configuraciones de entrada. Esto nos lleva al problema de cómo conectar contadores paralelos generalizados de una determinada capacidad para formar uno mayor. Este problema se conoce como el problema de reducción que pasamos a tratar seguidamente.

3.3.- PROBLEMA DE REDUCCION

Un problema general para sistemas regulares de circuitos aritméticos se puede formular como sigue:

Dados:

1) Un conjunto de señales binarias de entrada, cada una con un peso asignado.

La matriz de productos elementales $x_i y_j$, que se obtiene al efectuar la multiplicación de dos números $X = (x_{n-1},\ldots,x_0)$ e $Y = (y_{m-1},\ldots,y_0)$ por puertas Y y la suma de m números de n bits cada uno son ejemplos de tales conjuntos.

2) Uno o más tipos de circuitos aritméticos elementales que se van a utilizar como componentes primitivos del sistema regular.

3) Un conjunto de salidas, a cada una de las cuales se le asigna un peso. Por ejemplo, en el caso más simple, éste es el conjunto de los bits de un número binario.

Construir un sistema general de suerte tal que se minimice una cierta función de coste asignada a priori. Posibles funciones de coste que se pueden asignar son, por ejemplo, el número de componentes primitivas utilizadas para construir el sistema o el retardo que se produce para obtener el resultado final.

El problema así planteado tiene una gran generalidad, es, pues, conveniente hacer algunas restricciones al mismo.

- Para un sistema dado, únicamente se utilizará un único tipo de circuito primitivo. Estos sistemas se llaman uniformes.
- Las componentes primitivas serán triangulares.
- Sobre el conjunto de entradas no se impone ligaduras, sin embargo, el conjunto de salidas puede pertenecer a uno de los dos tipos siguientes:
  a) Existe como máximo una salida para cualquier peso que se considere. Este problema se denomina de reducción a uno.
  b) Existe como máximo dos salidas para cualquier peso que se considere. Este problema se denomina de reducción a dos.

En este capítulo, únicamente se va a considerar el problema de reducción a uno. En el problema de reducción a dos se suele emplear como última etapa del

proceso de reducción un sumador con previsión de arrastre para acelerar la obtención del resultado.

- Se considera una clase particular de contadores paralelos generalizados de tipo saturado que verifican las siguientes ligaduras:

$$\begin{aligned} & p_0 = p_1 = \ldots = p_{n-1} = N \\ & d = \delta \cdot n \end{aligned} \qquad (3.5)$$

$\delta \geq 2$ y n son enteros.

La condición de saturación implica que:

$$N = \frac{2^d - 1}{2^n - 1} = \frac{2^{\delta \cdot n} - 1}{2^n - 1} \qquad (3.6)$$

Contadores de este tipo se representarán por $(n \times N; \delta \cdot n)$ y corresponden al caso de que todas las columnas (n) tienen igual altura (N) y, por lo tanto, consumen una matriz rectangular de N filas por n columnas.

El problema de reducción se puede formular entonces en los siguientes términos: dado $(n_1 \times N_1; \delta_1 \cdot n_1)$, construir $(n_2 \times N_2; \delta_2 \cdot n_2)$ donde $n_2 = \alpha\, n_1$, siendo α un número entero mayor que 1.

Por lo tanto, el problema es reducir $(n_2 \times N_2; \delta_2 \cdot n_2)$ a un número binario igual al valor total de sus entradas, utilizando únicamente un tipo de componente primitiva $(n_1 \times N_1; \delta_1 \cdot n_1)$.

En la Figura 3.6 se representa gráficamente el esquema de reducción para generar el contador paralelo (6 x 65; 12) con módulos del tipo (2 x 5; 4).

3.4.- ALGORITMO DE REDUCCION

El algoritmo de reducción determina para cada nivel el número de bits en la columna de peso 2^j $j = 0,1,2,\ldots$ y el número de contadores que deben utilizarse.

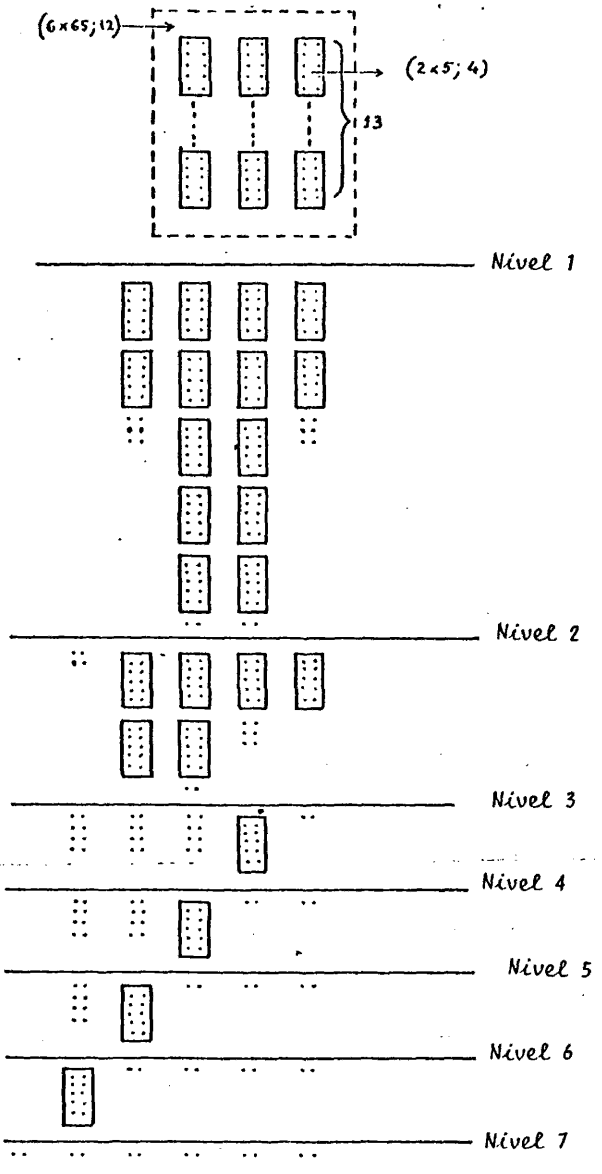


Figura 3.6.- Generación de (6 x 65; 12) con (2 x 5; 4)

Caso a.-

$$\delta_1 = \delta_2 = 2$$

En este caso particular, como se verá, el número de contadores elemen-tales que se utiliza es mínimo y se obtiene una expresión exacta que calcula dicho número.

Sea m_i^j el número de bits de peso 2^j en el nivel i. De la simetría del circuito primitivo que se utiliza se sigue que:

$$m_i^{n_1 \cdot j} = m_i^{n_1 j+1} = \ldots = m_i^{n_1(j+1)-1} \qquad \forall\ i,\ j$$

Esto hace posible el que únicamente se consideren las columnas de peso $2^{n_1 \cdot j}$ $(j = 0,1,2,\ldots)$. Considerando estos números como las componentes de un vector, el problema será reducir un vector inicial, $\underline{\ell_0}$ definido por

$$\underline{\ell_0} = (N_2, N_2, \ldots, N_2) \text{ con } dim\ \underline{\ell_0} = a$$

a un vector final

$$\underline{\ell_q} = (1,1,\ldots,1) \text{ con } dim\ \underline{\ell_q} = 2a$$

Sea ℓ_i^j la componente j del vector $\underline{\ell_i}$, verificándose $\ell_i^j = m_i^{n_1 \cdot j}$ y c_i^j el número de contadores que se utilizan en el nivel i-ésimo en la columna de peso $2^{n_1 \cdot j}$. Podemos considerar estos números como las componentes de un vector $\underline{c_i}$.

El emplazamiento de los contadores elementales en un nivel de reducción dado i se determina recorriendo el vector $\underline{\ell_i}$ de derecha a izquierda, e insertando tantos contadores como sea preciso para reducir al máximo posible las alturas de las columnas.

Las reglas para calcular c_i^j y ℓ_{i+1}^j son:

$$c_i^j = \left\lfloor \frac{\ell_i^j}{N_1} \right\rfloor \qquad (3.7)$$

$$\ell^{j}_{i+1} = \left\lfloor \frac{\ell^{j-1}_{i}}{N_1} \right\rfloor + \left\lfloor \frac{\ell^{j}_{i}}{N_1} \right\rfloor + (\ell^{j}_{i}) \bmod N_1 \qquad (3.8)$$

donde $\lfloor x \rfloor$ es el mayor entero menor que o igual a x.

Un ejemplo de una reducción para este caso se observa en la Figura 3.7, para generar un contador (2 x 5; 4) con contadores elementales (1 x 3; 2) (full-adders).

Sea c^j el número total de contadores en la columna de peso $2^{n_1 \cdot j}$

El problema de minimizar el número de contadores elementales $(n_1 \times N_1; 2n_1)$ en la síntesis de $(n_2 \times N_2; 2n_2)$ se puede formular en los términos siguientes.

$$\text{Minimizar } J = \sum_{i=0}^{k} c^i \qquad (3.9)$$

donde $k = 2(a - 1)$

Sujeto a las ligaduras siguientes:

$$(N_1 - 1)\, c^0 \geq N_2 - 1$$

$$-c^0 + (N_1 - 1)\, c^1 \geq N_2 - 1$$

$$\vdots$$

$$-\sum_{i=0}^{a-2} c^i + (N_1 - 1)\, c^{a-1} \geq N_2 - 1$$

$$-\sum_{i=1}^{a-1} c^i + (N_1 - 1)\, c^a \geq -1$$

$$-\sum_{i=2}^{a-1} c^i + (N_1 - 1)\, c^{a+1} \geq -1$$

$$\vdots$$

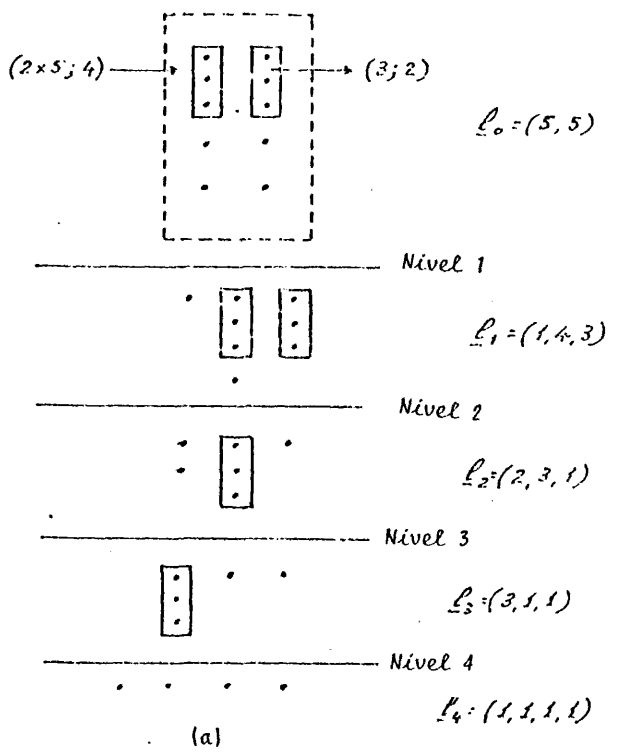


Figura 3.7.- Generación de (2 x 5; 4) con (1 x 3; 2) (Full-adders).

$$\sum_{i=k+1-a}^{k-1} c^i + (N_1 - 1)\, c^j \geq -1 \qquad (3.10)$$
$$- c^k \geq -1$$

La resolución de (3-9) y (3-10) corresponde a un problema de programación lineal entera. En el caso particular que estamos considerando se obtienen tomando el signo de igualdad en las ligaduras y resolviendo para las c^i. Después de una serie de cálculos se obtiene:

$$\begin{aligned}
c^{2a-2} &= 1 \\
c^{2a-3} &= 1 + 2^{n_1} \\
c^{2a-4} &= 1 + 2^{n_1} + 2^{2n_1} \\
&\vdots \\
c^{a-1} &= 1 + 2^{n_1} + \ldots + 2^{(a-1)n_1} \\
c^{a-2} &= 2^{n_1} + 2^{2n_1} + \ldots + 2^{(a-1)n_1} \\
c^{a-3} &= 2^{2n_1} + \ldots + 2^{(a-1)n_1} \\
&\vdots \\
c^1 &= 2^{(a-2)n_1} + 2^{(a-1)n_1} \\
c^0 &= 2^{(a-1)n_1}
\end{aligned} \qquad (3.11)$$

Ejemplo 3.1.- Consideremos la síntesis de un contador paralelo generalizado (6 x 65; 12) a partir de módulos (2 x 5; 4) que está representada en la Figura 3.6. En este caso, las expresiones (3.9) y (3.10) se reducen a:

$$\text{minimizar } J = \sum_{i=0}^{4} c^i \qquad (3.12)$$

sujeto a las ligaduras

$$\begin{aligned}
4\, c^0 &\geq 64 \\
-c^0 + 4\, c^1 &\geq 64
\end{aligned} \qquad (3.13)$$

$$\begin{aligned} -c^1 + 4\,c^2 &\geq 64 \\ -c^2 + 4\,c^3 &\geq -1 \\ -c^3 + 4\,c^4 &\geq -1 \\ -c^4 &\geq -1 \end{aligned} \qquad (3.13)$$

que dan como resultado

$$\begin{aligned} c^0 &= 2^4 = 16 \\ c^1 &= 2^2 + 2^4 = 20 \\ c^2 &= 2^0 + 2^2 + 2^4 = 21 \\ c^3 &= 2^0 + 2^2 = 5 \\ c^4 &= 2^0 = 1 \end{aligned}$$

Estos valores coinciden con los que se deducen de la Figura 3.6.

Una característica notable de este caso particular es que en el procedimiento de síntesis, los contadores elementales se ajustan perfectamente para conformar un contador paralelo de mayor dimensión.

Caso b.-

$$\delta_1 \neq 2 \text{ y/ó } \delta_2 \neq 2$$

La única restricción obvia que se pone en este caso es de que

$$d_2 = n_2\,\delta_2 > n_1\,\delta_1 = d_1$$

es decir, que la capacidad del contador que se va a generar (d_2), sea mayor que la de sus componentes elementales (d_1).

La complicación adicional que surge en el algoritmo de reducción es de que ya no es posible, en principio, un ajuste exacto de los contadores elementales para formar el contador que se desea.

Es decir, en el proceso de reducción se puede llegar a un punto en que la introducción de un contador elemental no utilice todas sus entradas. Esto re-

presenta un mal uso de dicho contador, pues no se está utilizando con su capacidad plena. La manera de eliminar este inconveniente viene dictada por la siguiente consideración: Toda entrada a un contador elemental colocado en la columna de peso $2^{n_1 j}$ es equivalente a utilizar esa entrada 2^{kn_1} veces en un contador que esté disponible en la columna de peso $2^{n_1(j-k)}$. Debe observarse que el proceso inverso no es posible.

Así por ejemplo, un contador (2 x 5; 4) se puede realizar con un contador (1 x 15; 4). Basta para ello que las entradas de peso 2^1 del contador (2 x 5; 4) se dupliquen como entradas del contador (1 x 15; 4) (ver Figura 3.8).

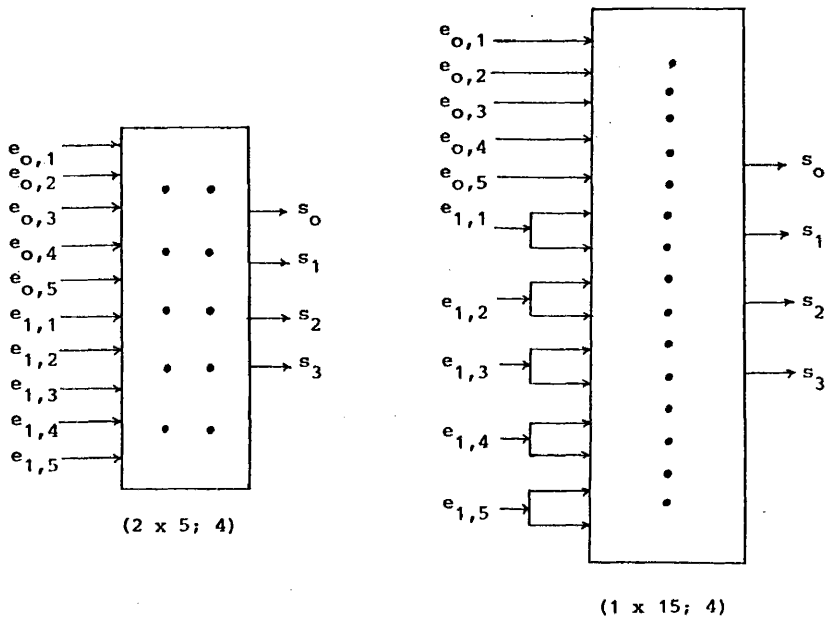


Figura 3.8.- Realización de un contador (2 x 5; 4) con un contador (1 x 15; 4)

Sin embargo, el proceso en sentido contrario no es factible.

Esto nos lleva a definir el grado de reducción de un contador paralelo en comparación con otro de su misma capacidad (mismo valor de d).

Definición 3.6.- Dados dos contadores paralelos de igual capacidad $(n_a \times N_a; d)$ y $(n_b \times N_b; d)$ diremos que el primer contador presenta un grado de reducción mayor que el segundo si $n_a > n_b$.

Así pues, el grado de reducción va asociado al número de columnas del contador paralelo generalizado, de manera que, dado un contador paralelo, podremos realizar con él todos aquellos contadores que, teniendo igual capacidad (o menor evidentemente) que él, presentan un mayor grado de reducción.

Esto lleva consigo una mayor complejidad a la hora de implementar contadores con un grado de reducción pequeño.

El menor grado de reducción posible corresponde a $n = 1$. Así, en una realización con ROMS, el contador (1 x 15; 4) requiere una memoria de 2^{15} x 4 bits, mientras que el contador (2 x 5; 4) únicamente necesita 2^{10} x 4 bits.

Este hecho nos será de utilidad posteriormente cuando se vaya a determinar cotas sobre el número de niveles para efectuar la síntesis.

En este caso, para desarrollar el algoritmo de reducción, definimos:

$$\{j_1 = \min \{j \mid \ell_i^j > 1\} \quad \text{inicialmente } j_1 = 0$$

$$\{j_2 = \max \{j \mid \ell_i^j > 0\} \qquad " \qquad j_2 = a - 1$$

j_1 y j_2 son, pues, dos punteros que en el proceso de reducción nos van a indicar en cualquier nivel dónde se deben comenzar a insertar contadores elementales (j_1) y hasta dónde debemos de continuar (j_2).

En cada nivel i se tiene que cumplir la siguiente ligadura

$$c_i^{j_1} > 0 \qquad (3.14)$$

Las reglas para calcular c_i^j y ℓ_{i+1}^j son:

$$c_i^j = \left\lfloor \frac{\ell_i^j}{N_1} \right\rfloor \qquad j = j_1, j_1+1, \ldots, j_2 \qquad (3.15)$$

$$\ell_{i+1}^j = \sum_{m=j-(\delta_1-1)}^{j} c_i^m + (\ell_i^j) \bmod N_1 \qquad j = j_1, \ldots, j_2+\delta_1-1 \qquad (3.16)$$

con el convenio de que $c_i^m = 0$ si $m \notin [j_1, \ldots, j_2]$ y $\ell_i^j = 0$ si $j > j_2$.

(3-16) se puede calcular de forma recursiva como sigue:

$$\ell_{i+1}^{j_1} = c_i^{j_1} + (\ell_i^{j_1}) \bmod N_1$$

$$\ell_{i+1}^{j} = \ell_{i+1}^{j-1} + c_i^{j} - c_i^{j-\delta_1} + (\ell_i^{j}) \bmod N_1 - (\ell_i^{j-1}) \bmod N_1$$

$$j = j_1+1, \ldots, j_2+\delta_1-1$$

Si se verifica que $c_i^{j_1} = 0$ (es decir $1 < \ell_i^{j_1} < N_1$) se está violando la ligadura (3.14) y esto implica que el contador elemental dispuesto más a la derecha, tiene algunas entradas vacías que pueden llenarse con entradas de peso superior, de acuerdo con la consideración ya apuntada.

Es, entonces, necesario modificar las componentes del vector $\underline{\ell}_i$ en el algoritmo. Así, si la componente j del vector $\underline{\ell}_i$ se cambia de ℓ_i^j a $\ell_i^j - 1$, el proceso de reducción no se modifica si la componente ℓ_i^{j-k} se cambia a $\ell_i^{j-k} + 2^{kn_1}$.

Teniendo esto en cuenta podemos ya dar el siguiente algoritmo general de reducción:

Algoritmo

Entrada: n_1, a, δ_1, δ_2

1) Fase de inicialización

$d_1 = \delta_1 n_1$; $n_2 = a\, n_1$; $d_2 = \delta_2 n_2$

$$N_1 = \frac{2^{d_1}-1}{2^{n_1}-1} \quad ; \qquad N_2 = \frac{2^{d_2}-1}{2^{n_2}-1}$$

$i = 0$; $j_1 = 0$; $j_2 = a-1$

$\ell_0^j = N_2$ para $j = j_1, j_1+1, \ldots, j_2$

2) $c_i^j = \left\lfloor \frac{\ell_i^j}{N_1} \right\rfloor \qquad j = j_1, \ldots, j_2$

Si $c_i^{j_1} = 0$ entonces ir al paso 4), si no ir al paso 2a)

a) $\ell_{i+1}^{j_1} = c_i^{j_1} + (\ell_i^{j_1}) \bmod N_1$

$$\ell_{i+1}^{j} = \ell_{i+1}^{j-1} + c_i^{j} - c_i^{j-\delta_1} + (\ell_i^{j}) \bmod N_1 - (\ell_i^{j-1}) \bmod N_1$$

$j = j_1+1, \ldots, j_2+\delta_1-1$

b) $j_2 = \max \{j,\ j \in [j_2, \ldots, j_2+\delta_1-1] \mid \ell_{i+1}^{j} \neq 0\}$

3) Si $\ell_{i+1}^{j_1} = 1$ entonces $j_1 = j_1 + 1$

Si $j_1 = \delta_2\, a-1$ entonces Parar

si no ir al paso 3)

si no $i = i + 1$ ir al paso 2)

4) $m = 1$, $r = \frac{N_1 - \ell_i^{j_1}}{2^{n_1}}$, $\qquad \ell_i^{j_1} = N_1$

5) $\ell_i^{j_1+m} = \ell_i^{j_1+m} - r$

si $\ell_i^{j_1+m} \geq 0$ entonces ir al paso 2)

si no $\qquad r = \frac{-\ell_i^{j_1+m}}{2^{n_1}}$

$\ell^{j_1+m} = 0$

$m = m + 1$

ir al paso 5).

Ejemplo 3.2.- Se da a continuación un ejemplo paso a paso del algoritmo de reducción presentado, para la síntesis de un contador (8 x 257; 16) con contadores elementales (2 x 85; 8).

Entrada: $n_1 = 2, \quad a = 4, \quad s_1 = 4, \quad s_2 = 2$

1) $d_1 = 8; \quad n_2 = 8; \quad d_2 = 16; \quad N_1 = 85; \quad N_2 = 257$

$i = 0; \quad j_1 = 0; \quad j_2 = 3$

$\underline{\ell}_0 = (257, 257, 257, 257)$

2) En las sucesivas iteraciones, los vectores $\underline{c}_i$ y $\underline{\ell}_{i+1}$ toman los siguiente valores:

$\underline{c}_0 = (0,0,0,3,3,3,3) \rightarrow \underline{\ell}_1 = (3,6,9,14,11,8,5)$ modificado a $\underline{\ell}_1 = (3,6,9,14,8,0,85)$

$\underline{c}_1 = (0,0,0,0,0,0,1) \rightarrow \underline{\ell}_2 = (3,6,9,15,9,1,1,)$ " $\underline{\ell}_2 = (3,6,8,0,85,1,1)$

$\underline{c}_2 = (0,0,0,0,1) \rightarrow \underline{\ell}_3 = (3,7,9,1,1,1,1,)$ " $\underline{\ell}_3 = (85,1,1,1,1)$

$\underline{c}_3 = (0,0,1,) \rightarrow \underline{\ell}_4 = (1,1,1,1,1,1,1,1)$

Las modificaciones en el vector $\underline{\ell}_i$ tienen lugar en los pasos 4) y 5) (cuando $c_i^{j_1} = 0$) para poder insertar un contador en la posición más a la derecha que viene determinada por el apuntador j_1. De esta forma, para utilizar todas las entradas del contador se utilizan entradas de un peso superior, tal como se ha indicado.

En la Figura 3.9 se representa gráficamente la síntesis de un contador (3 x 9; 6) con módulos elementales tipo (1 x 7; 3).

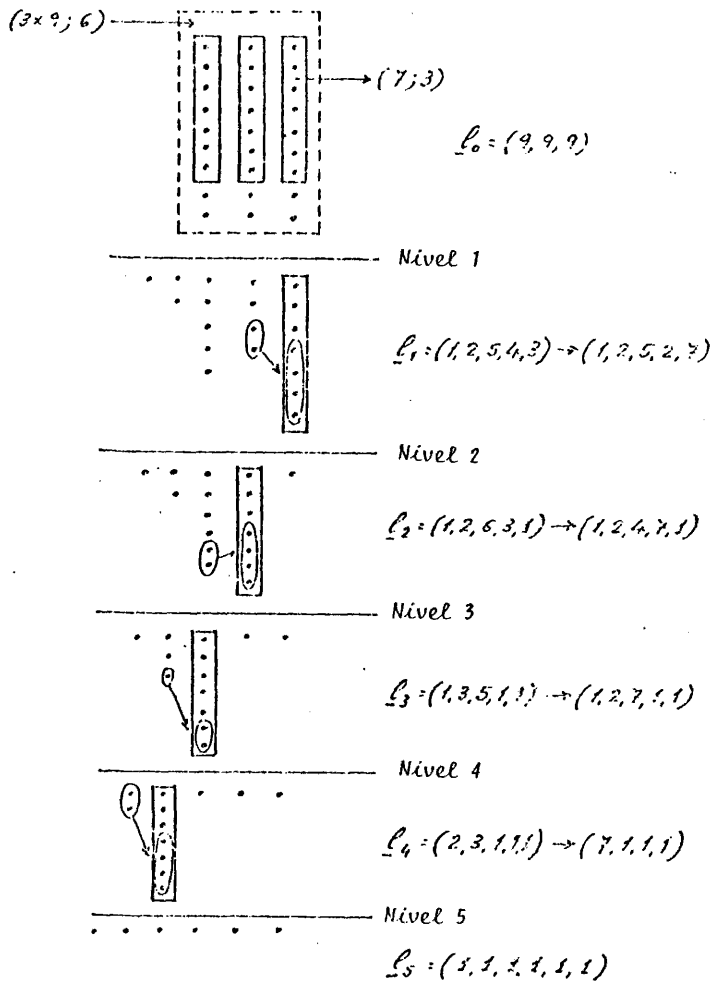


Figura 3.9.- Síntesis de un contador (3 x 9; 6) con (1 x 7; 3)

3.5.- NUMERO DE CONTADORES ELEMENTALES UTILIZADOS EN EL ALGORITMO DE REDUCCION

El número total C de módulos elementales $(n_1 \times N_1;\ d_1)$ utilizados en la síntesis de $(n_2 \times N_2;\ d_2)$ puede aproximarse mediante el siguiente razonamiento:

a) El contador paralelo generalizado $(n_2 \times N_2;\ d_2)$ es un circuito combinacional que, desde el punto de vista de número de entradas número de salidas, reduce $n_2 \cdot N_2$ líneas de entrada a $d_2 = \delta_2 \cdot n_2$ líneas de salida.

b) Asimismo, los contadores paralelos generalizados elementales $(n_1 \times N_1;\ d_1)$ reducen un conjunto de $n_1 \cdot N_1$ líneas de entrada a $d_1 = \delta_1 \cdot n_1$ líneas de salida.

Estas consideraciones nos llevan a la siguiente cota inferior para el número C de módulos elementales

$$C \geq \frac{n_2N_2 - n_2\delta_2}{n_1N_1 - n_1\delta_1} = \frac{a(N_2 - \delta_2)}{N_1 - \delta_1} \qquad (3.17)$$

En el caso particular de $\delta_1 = \delta_2 = 2$, la cota inferior corresponde al número C de elementos utilizados.

Este resultado se puede obtener tomando en cuenta la expresión (3.11).

$$C = \sum_{i=0}^{2a-2} c^i = a(1 + 2^{n_1} + \ldots + 2^{(a-1)n_1}) = \frac{a(N_2 - 2)}{(N_1 - 2)} \qquad (3.18)$$

Esta concordancia cabía esperarla si tenemos en cuenta que, para este caso particular, tal como se hizo notar, el proceso de reducción garantiza la minimización del número de módulos elementales utilizados.

En el algoritmo general esto se corresponde con el hecho de que no es necesario modificar las componentes del vector $\underline{\ell}_i$ en ningún nivel (esto es, no se hace ninguna entrada en los pasos 4 y 5 del algoritmo). En esta circunstancia, las entradas de los contadores elementales corresponden a los pesos de las columnas en donde se inserta dicho contador.

En la Tabla 3.2a) se representa el número de contadores elementales que se utilizan cuando $\delta_1 = \delta_2 = 2$, en la síntesis de determinados contadores. Asimismo, en la Tabla 3.2b) están representados la cota inferior y el número real de contadores que se necesitan en el caso particular $\delta_1 = 3$ y $\delta_2 = 2$.

Tipo de contador elemental

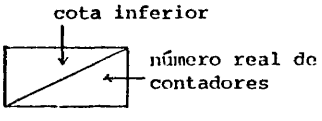
Contador que se desea sintetizar	(1 x 3; 2)	(2 x 5; 4)	3 x 9; 6)
(2 x 5; 4)	6	x	x
(3 x 9; 6)	21	x	x
(4 x 17; 8)	60	10	x
(5 x 33; 10)	155	x	x
(6 x 65; 12)	378	63	18
(7 x 129; 14)	889	x	x

TABLA 3.2a) Caso $\delta_1 = \delta_2 = 2$. Número de contadores elementales utilizados en la síntesis de determinados contadores. Las x corresponden a que el valor de "a" no es un número entero mayor que 1 y, por tanto, la síntesis no está permitida.

Contador que se desea sintetizar

tipo de contador elemental	(2x5;4)	(3x9;6)	(4x17;8)	(5x33;10)	(6x65;12)	(7x129;14)
(1x7;3)	1 / 2	5 / 7	15 / 16	38 / 42	94 / 97	222 / 226
(2x21;6)	x / x	x / x	1 / 2	x / x	10 / 13	x / x
(3x73;9)	x / x	x / x	x / x	x / x	1 / 2	x / x

TABLA 3.2b) Caso $\delta_1 = 3$ y $\delta_2 = 2$



3.6.- NUMERO DE NIVELES UTILIZADOS EN EL ALGORITMO DE REDUCCION

Sea q el número de niveles necesarios para reducir a un número binario el contador paralelo generalizado $(n_2 \times N_2;\ d_2)$ usando módulos elementales $(n_1 \times N_1;\ d_1)$. Teniendo en cuenta la definición 3.6 relativa al grado de reducción de un contador paralelo, vamos a ser capaces de determinar una cota inferior (q_i) y una cota superior (q_s) de q.

La Figura 3.10 ilustra gráficamente la línea de razonamiento que se sigue.

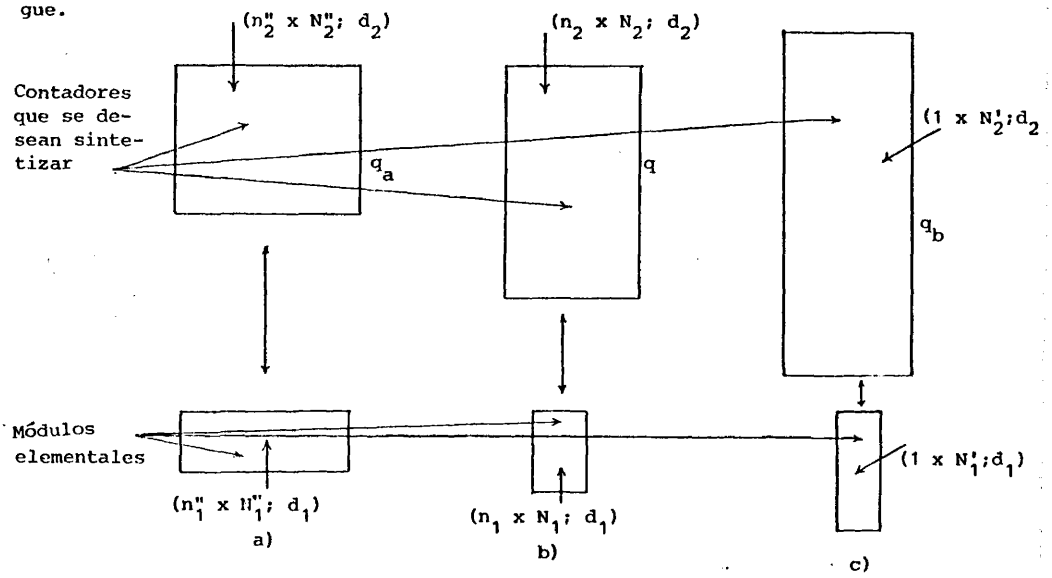


Figura 3.10.- Cálculo de las cotas superior e inferior para el número de niveles en el proceso de reducción

Sean q_a, q y q_b el número de niveles necesarios para efectuar la síntesis de los contadores generalizados de la Figura 3.10 usando los contadores elementales que se muestran. Se observa que el número de salidas $(d_2$ y $d_1)$ de ambos módulos es idéntico y lo único que se modifica es el grado de reducción de los contadores. Si se toma en consideración el grado de reducción, se veri-

fica que:

$$q_b \geq q \geq q_a \qquad (3.19)$$

Lo que vamos a determinar son una cota superior (q_s) a q_b y una cota inferior (q_i) a q_a que, teniendo en cuenta la relación (3.19), serán a su vez, respectivamente, cota superior e inferior de q.

Como es fácil deducir estas cotas serán tanto mejores cuanto menos haya que modificar la estructura original de la Figura 3.10b) para pasar a la de las Figuras 3.10a) y 3.10c).

3.6.1.- Determinación de la cota superior q_s

El cálculo de q_s, a partir de la síntesis de la Figura 3.10c), se basa en una generalización del procedimiento desarrollado por Swartzlander ([8]) en el caso de que los módulos elementales fueran sumadores completos (contadores tipo (3; 2)).

La determinación de q_s se fundamenta en los dos Teoremas siguientes:

Teorema 3.1.-

La síntesis de $(N_2^1;\ d_2)$ con módulos elementales $(N_2^2;\ d_2-1)$ requiere únicamente 3 niveles.

En efecto, suponiendo los contadores saturados se tiene la siguiente relación entre N_2^1 y N_2^2:

$$\left.\begin{aligned} N_2^2 &= 2^{d_2-1} - 1 \\ N_2^1 &= 2^{d_2} - 1 \end{aligned}\right\} \Longrightarrow N_2^1 = 2\,N_2^2 + 1 \qquad (3.20)$$

La situación de forma general es la que se muestra en la Figura 3.11. En el último nivel, tal como se observa, se hace necesario utilizar un sumador completo (contador tipo (3; 2)).

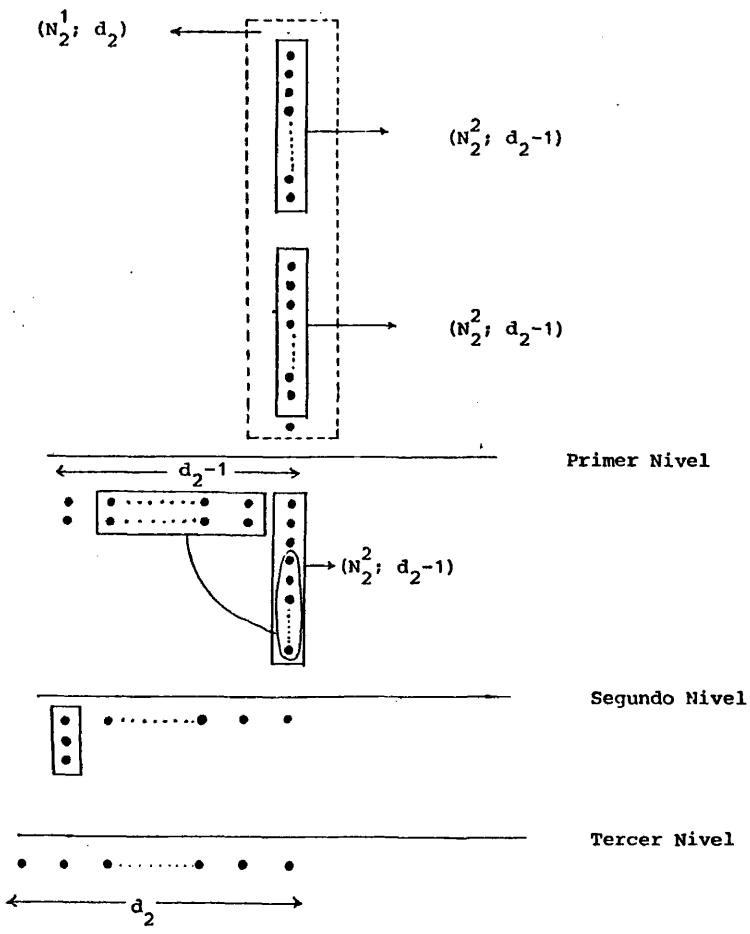


Figura 3.11.- Síntesis de $(N_2^1;\ d_2)$ con módulos elementales $(N_2^2;\ d_2-1)$.

En la Figura 3.12 se representa la síntesis de un contador paralelo (15; 4) con módulos (7; 3).

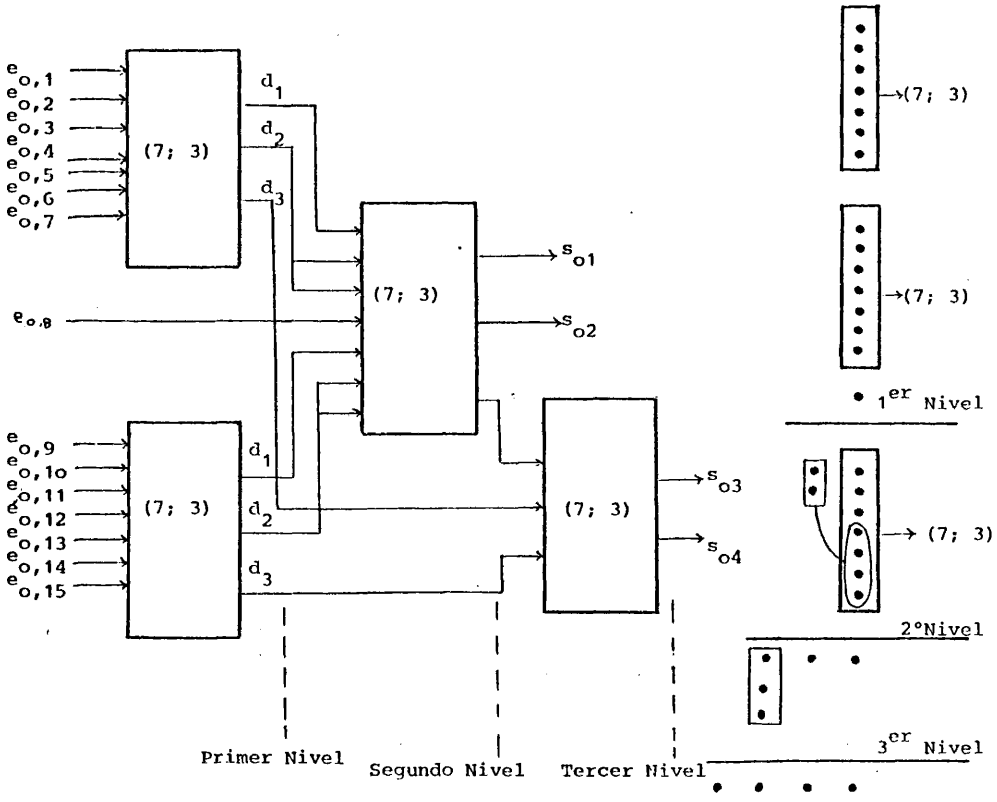


Figura 3.12.- Síntesis del contador (15; 4) con módulos elementales (7; 3).

Teorema 3.2.-

Si la síntesis de $(N_2'; d_2-1)$ con módulos elementales $(N_1; d_1)$ se hace en M niveles, entonces la síntesis de $(N_2; d_2)$ requiere a lo más $M+2$ niveles.

De forma esquemática, el proceso de reducción que garantiza esta cota superior se da en la Figura 3.13. El objetivo es utilizar, tan pronto como se generen, las salidas que se vayan produciendo en la síntesis de $(N_2'; d_2-1)$ con módulos elementales $(N_1; d_1)$. De esta forma cada d_2-1 niveles podemos completar un nuevo módulo elemental y reducir a uno d_1-1 bits de la salida.

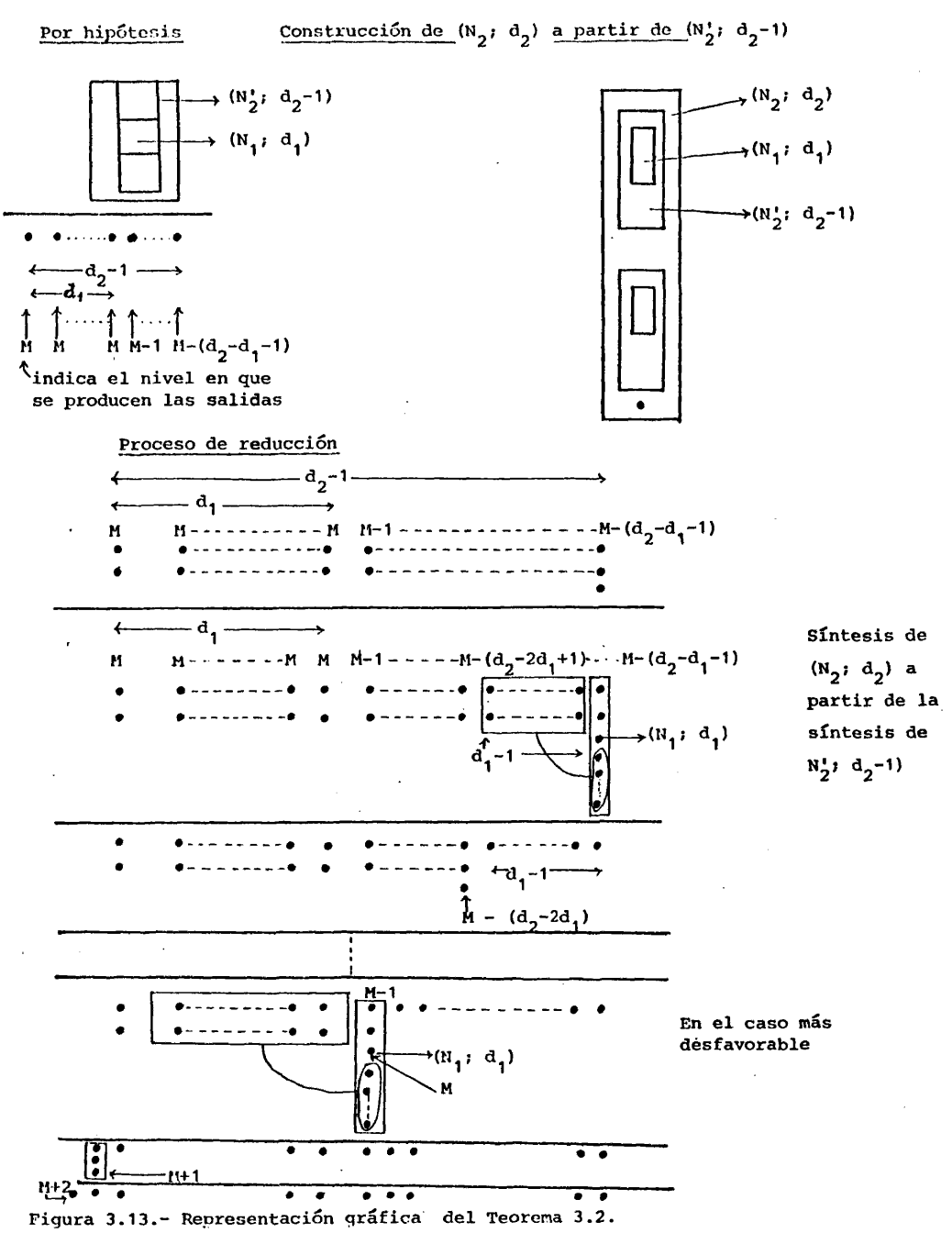


Figura 3.13.- Representación gráfica del Teorema 3.2.

Teniendo en cuenta los Teoremas 3.1 y 3.2 se puede determinar fácilmente la cota superior q_s.

$$(N_1;\ d_1) \longrightarrow (N_2^1;\ d_1+1) \longrightarrow (N_2^2;\ d_1+2) \longrightarrow \ldots \longrightarrow (N_2^{d_2-d_1};\ d_2) \equiv (N_2;\ d_2)$$

3 niveles *3+2 niveles* *3+2(d₂-d₁-1) niveles*

Es decir

$$q_s = 2(d_2 - d_1) + 1 \qquad (3.21)$$

En la Figura 3.14 se da un caso concreto de síntesis de un contador paralelo (31; 5) con módulos elementales (7,3). Se hace uso para ello de la síntesis de contadores paralelos (15; 4) con módulos (7; 3). (Ver Figura 3.12).

Conviene indicar que este proceso de construcción únicamente ha sido ideado con el fin de determinar una cota superior en el número de niveles y que, por lo tanto, no es óptimo. En la Figura 3.15 se representa la síntesis óptima que únicamente necesita 4 niveles y 7 módulos elementales en lugar de los 4 niveles y 10 módulos requeridos en la Figura 3.14, en los cuales están dos de ellos infrautilizados (se usan como contadores (3; 2)).

3.6.2.- Determinación de la cota inferior q_i

El cálculo de q_i se fundamenta en sustituir el módulo elemental $(n_1 \times N_1;\ d_1)$ por uno de igual capacidad (mismo valor de (d_1)) pero con el número de columnas (n_1) igual al del contador que se desea sintetizar. Es decir, de acuerdo con la Definición 3.6, se pasa a un contador equivalente que presenta un mayor grado de reducción y que por consideraciones de simetría permite tratar el problema como si de contadores paralelos no generalizados se tratase.

Se transforma, pues, el contador paralelo $(n_1 \times N_1;\ d_1)$ en un contador $(n_2 \times N_1';\ d_1)$ donde:

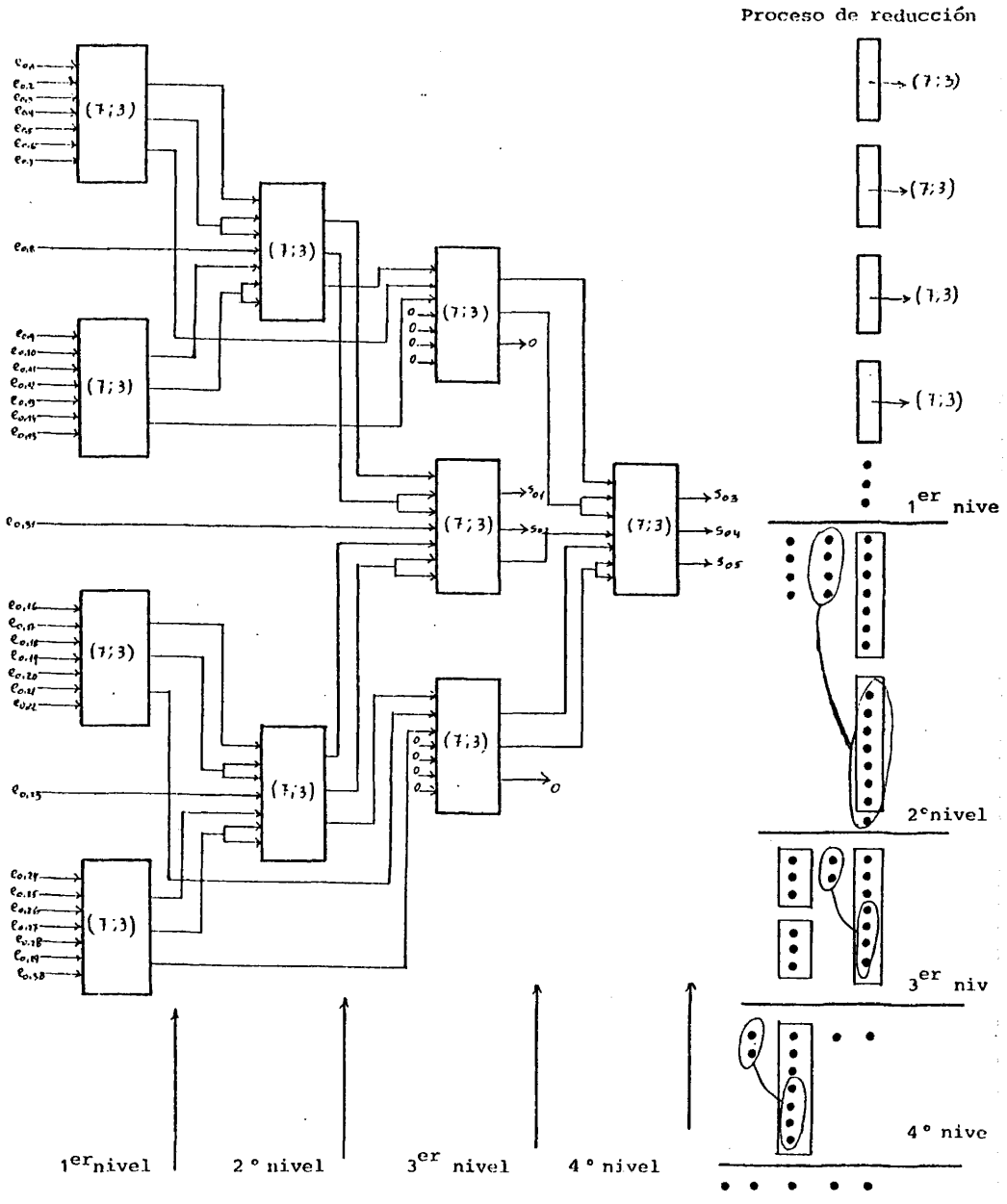


Figura 3.14.- Síntesis del contador (31; 5) con módulos (7; 3) (Procedimiento de determinación de q_δ)

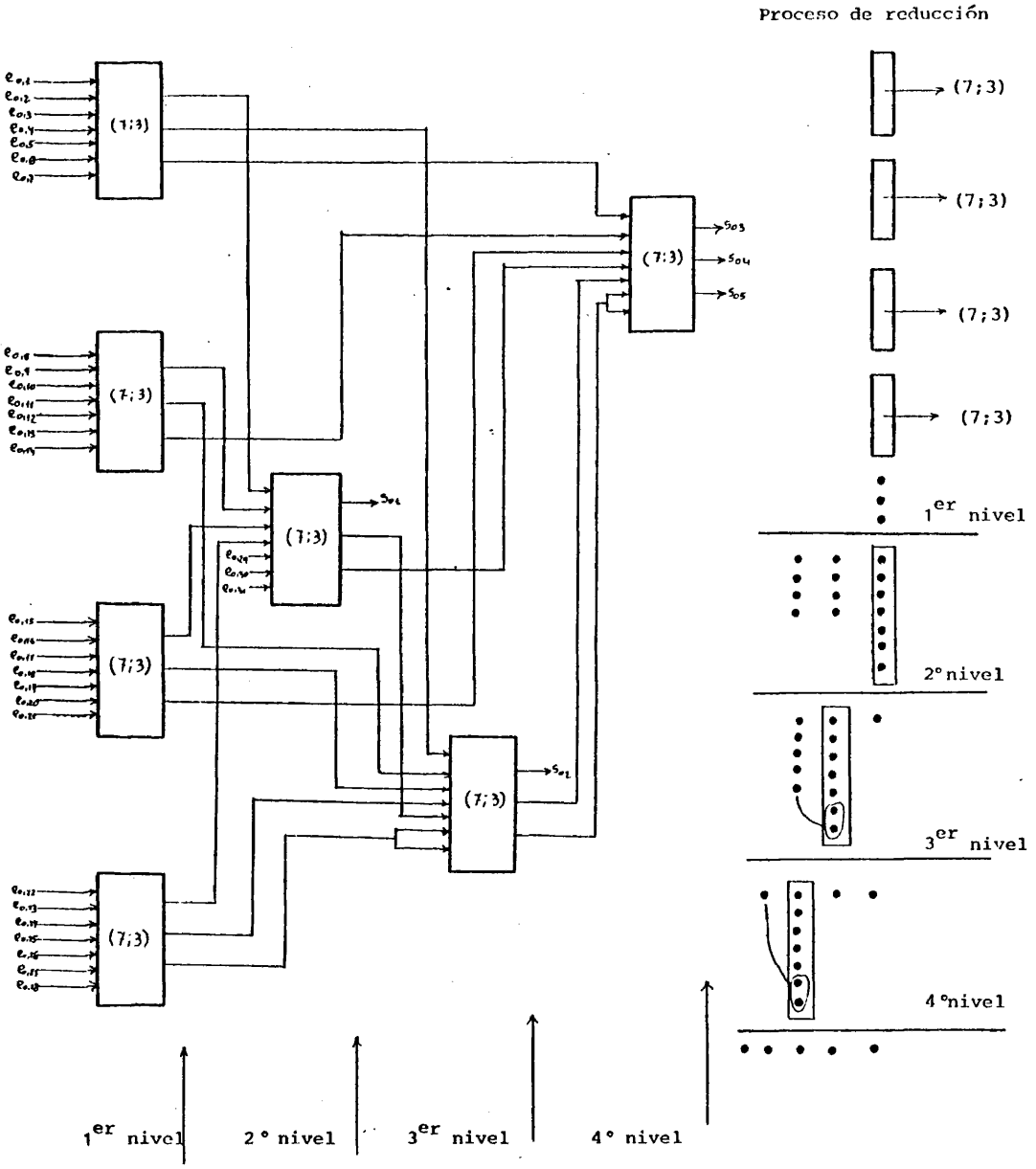


Figura 3.15.- Síntesis óptima del contador (31; 5) con módulos (7; 3)

$$N_1' = \frac{2^{d_1} - 1}{2^{n_2} - 1} \qquad (3.22)$$

Para que (3.22) tenga sentido se supone que $d_1 > n_2$.

Se trata así de determinar una cota inferior para la síntesis de $(n_2 \times N_2;\ d_2)$ con módulos elementales $(n_2 \times N_1';\ d_1)$ que es equivalente al de la síntesis de $(N_2;\ d_2)$ con módulos $(N_1';\ d_1)$.

De los resultados de Kobayashi y Ohara ([9]) se sigue que:

$$q_i = \lceil \ell og_{N_1'} N_2 \rceil + (d_2 - d_1) \qquad (3.23)$$

Donde: $q_{1i} = \lceil \ell og_{N_1'} N_2 \rceil$ representa el número de niveles necesarios para reducir la columna de peso 2° de N_2 bits a un bit. A partir del nivel q_{1i}, la transición de $\underline{\ell}_{q_{1i}+i}$ a $\underline{\ell}_{q_{1i}+i+1}$ deja inalteradas las i componentes de menor peso (cuyo valor es 1) y reduce la componente $i + 1$ a 1; salvo en el último nivel, donde el contador que se inserta determina los d_1 bits finales de d_2.

En la Tabla 3.3 se representa para los casos dados en las Tablas 3.2, el número de niveles realmente necesitado en la síntesis (calculado por programa) y las cotas superior e inferior que le corresponden de acuerdo con las relaciones (3.21) y (3.23).

Tipo de contador	Tamaño de la memoria Palabras x Bits
(3;2)	8x2
(5,5;4)	1024x4
(7;3)	128x3
(15;4)	2^{15}x4

TABLA3.1.-Contadores elementales realizados en un solo circuito integrado.

Tipo de contador elemental

Contador que se desea sintetizar	(1x3;2)			(2x5;4)			(3x9;6)		
(2x5;4)	2	4	5	x			x		
(3x9;6)	3	6	9	x			x		
(4x17;8)	5	9	13	3	4	9	x		
(5x33;10)	7	12	17	x			x		
(6x65;12)	8	15	21	6	7	17	4	4	13
(7x129;14)	10	18	25	x			x		

a) Caso $\delta_1 = \delta_2 = 2$

Tipo de contador elemental

Contador que se desea sintetizar	(1x7;3)			(2x21;6)			(3x73;9)		
(2x5;4	0	2	3	x			x		
(3x9;6)	2	5	7	x			x		
(4x17;8)	3	7	11	0	2	5	x		
(5x33;10)	4	10	15	x			x		
(6x65;12)	6	10	19	3	5	13	0	2	7
(7x129;14)	7	13	23	x			x		

b) Caso $\delta_1 = 3$, $\delta_2 = 2$

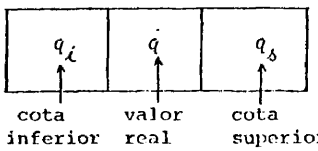


TABLA 3.3.- Número de niveles utilizados en la síntesis de determinados contadores. Las x corresponden a que el valor de "a" no es un número entero mayor que 1 y, por tanto, la síntesis no está permitida.

REFERENCIAS

(1) MELLADO, M., HERNANDEZ CACHERO, A. y ALIQUE, J.R., "Sumadores de entrada múltiple". Anales de la Real Soc. Esp. de Fís. y Quím. Vol. 60, pág. 49, nº 1 a 3. Enero-Febrero-Marzo, 1972.

(2) MELLADO, M., HERNANDEZ CACHERO, A. y ALIQUE, J.R., "Multiplicador binario basado en sumadores de entrada múltiple". Revista de Informática y Automática, nº 19. 1974.

(3) HERNANDEZ CACHERO, A., "Síntesis de circuitos aritméticos de alta velocidad". Tesis Doctoral. Universidad Complutense, 1971.

(4) STENZEL, J., KUBITZ, J. and GARCIA, H., "A compact high-speed parallel multiplication scheme". IEEE Trans. Computers. Vol. C-26, nº10, October, 1977.

(5) CAXTON, C., FOSTER and STOCKTON, F.D., "Counting responders in an associative memory", IEEE Trans. on Computers, December, 1971.

(6) MEO, R., "Arithmetic networks and their minimization using a new line of elementary units", IEEE Trans. on Computers, Vol. C-24, nº 3, March, 1975.

(7) DADDA, L., "Some schemes for parallel multipliers", Alta Frequenza, Vol. XXXIV, nº 5, May 1975, pp. 349-356.

(8) SWARTZLANDER, E., "Parallel counters", IEEE Trans. on Computers, Vol. C-22, nº 11, November, 1973.

(9) KOBAYASHI, H., and OHARA, H., "A synthesizing method for large parallel counters with a network of smaller ones", IEEE Trans. on Computers, Vol. C-27, nº 8, August, 1978.

(10) CURRENT, W. and MOW, A., "Implementing parallel counters with four-valued threshold logic", IEEE Trans. on Computers, Vol. C-28, nº 3, March 1979.

CAPITULO IV

SINTESIS DE MULTIPLICADORES BINARIOS

4.1.- INTRODUCCION

En cálculos de tipo científico, el tiempo que se invierte en la multiplicación es un parámetro fundamental a la hora de evaluar la potencia de cálculo del sistema. Este tiempo es generalmente mucho mayor que el necesitado para otras operaciones, como por ejemplo la suma, y su influencia puede llegar a ser dominante en programas que requieran un elevado número de ellas. Se estima que en este tipo de cálculos, el número de multiplicaciones puede alcanzar el 30% del número total de operaciones aritméticas.

Estas consideraciones demuestran la importancia del estudio de multiplicadores de alta velocidad. Como en todo problema de síntesis existirá un compromiso de velocidad-coste y la elección óptima estará obviamente en función de la aplicación en concreto a la que se destine el multiplicador.

Con el fin de simplificar la estructura del capítulo consideraremos únicamente la multiplicación de números binarios sin signo. Esto no supone restricción a los métodos que se propongan que pueden extenderse con facilidad al caso de números binarios con signo y con cualquier tipo de representación que se adopte.

La manera más obvia de multiplicar dos números binarios positivos es la de inspeccionar secuencialmente los bits del multiplicador desde el menos significativo al más significativo. Si un bit dado es un "1", el multiplicando se suma a la mitad más significativa de un acumulador de longitud doble; si es un "0" no se efectúa dicha suma. El contenido del acumulador se desplaza un bit a la derecha cuando se inspecciona cada bit del multiplicador. Cuando todos los

bits del multiplicador han sido considerados, el acumulador contiene el producto. Esta es una forma básica del conocido esquema de multiplicación de suma y desplazamiento. La Figura 4.1 ilustra una implementación con componentes tipo MSI de un multiplicador secuencial construido con un sumador y un registro de desplazamiento. Para claridad, únicamente se representan las líneas de datos. Este método secuencial es adecuado para multiplicaciones de baja velocidad. Cuando la velocidad se hace un parámetro crítico se está forzado ineludiblemente a utilizar un punto de vista combinacional a la hora de efectuar la síntesis del multiplicador.

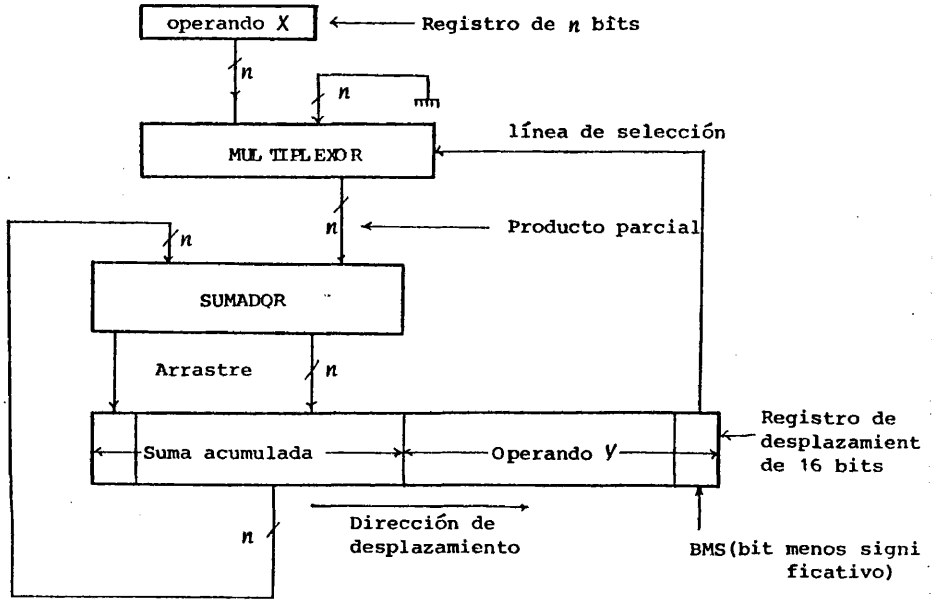


Figura 4.1.- Multiplicador secuencial construido con componentes MSI. Algoritmo de suma y desplazamiento.

En este capítulo vamos a considerar las siguientes cuestiones:

En primer lugar, se efectúa una clasificación de los multiplicadores digitales atendiendo a la forma en que se van presentando los bits de los operandos al multiplicador (serie o paralelo). A partir de esta clasificación, analizamos con especial atención los multiplicadores tipo paralelo y sus formas de realización. Básicamente existen dos categorías de multiplicadores paralelos que son: a) de tipo iterativo y b) basados en esquemas de reducción de la matriz de productos parciales. Dentro del primer tipo se propone un nuevo tipo de multiplicador iterativo basado en una modificación del esquema de Guild ([1]). Para los del tipo b) se generaliza el algoritmo de Stenzel y col. ([2]) y se utiliza de forma extensiva la síntesis de contadores generalizados vista en el capítulo anterior. Cuando la longitud de los operandos es elevada ($n > 32$), una combinación de ambos tipos de multiplicadores (utilizando los de tipo iterativo como módulos elementales de multiplicación) parece aconsejable. Asimismo se generaliza el multiplicador tipo cuasi-serie de Swartzlander([3]) desarrollándolo de forma serie-paralelo, lo que acelera el proceso de cálculo en función del grado de paralelismo que se desee.

Finalmente, se propone un multiplicador tipo paralelo para números fraccionarios de muy bajo coste, que ha sido implementado como elemento potenciométrico en las unidades de integración digital que se verán en el próximo capítulo.

4.2.- CLASIFICACION DE LOS MULTIPLICADORES DIGITALES

Un multiplicador digital es un circuito cuyas entradas son los bits de los dos factores y cuyas salidas son los bits del producto. En este apartado, vamos a efectuar una clasificación de los multiplicadores basados en las distintas formas en que los bits constitutivos de los factores se pueden presentar a las

entradas del multiplicador y en cómo se pueden generar los bits del producto a su salida.

En cuanto a los factores, cada uno de ellos pueden ser o serie o paralelo (no consideramos una presentación tipo serie-paralelo). En el primer caso, los bits de los factores se transmiten secuencialmente a las entradas, empezando por los menos significativos. En el caso paralelo, los bits de los factores se presentan simultáneamente en las n entradas. Por tanto, se pueden considerar 3 casos para la presentación de los dos factores: paralelo-paralelo (ambos factores paralelos), serie-paralelo y serie-serie. Las relaciones temporales entre los factores son las siguientes:

a) En el caso paralelo-paralelo, los factores se presentan simultáneamente y, por tanto, el multiplicador es un circuito con $2n$ entradas.

b) En el caso serie-paralelo, el factor paralelo se presenta simultáneamente junto con el bit menos significativo (BMS) del factor serie; entonces, el multiplicador tendrá $n+1$ entradas.

c) En el caso serie-serie, los bits correspondientes de los dos factores se presentan simultáneamente en las dos entradas, comenzando con el par menos significativo.

En el caso a), se podría considerar también la posibilidad de que los dos factores no sean simultáneos. Este caso no tiene interés desde un punto de vista lógico, ya que, evidentemente, ningún bit del producto se puede generar si solamente se conoce uno de los factores. No obstante, una presentación no simultánea puede ser útil en el diseño, si el factor que se presenta primero puede "preparar" al circuito para que reciba al segundo, de forma que el retardo resultante sea menor que en el caso de que exista simultaneidad.

En los casos b) y c) se observa que el primer bit del producto se pue-

de generar únicamente después de que estén disponibles los BMS de los factores; el segundo bit requiere el conocimiento de los segundos bits de los factores, etc. Después de la presentación del último par de bits de los factores, ninguna información nueva entrará al multiplicador, aunque la mitad más significativa (y posiblemente algunos bits de la mitad menos significativa) del producto tienen todavía que generarse. Por tanto, estos tipos de multiplicadores contienen necesariamente una cierta capacidad de memoria.

Un parámetro más a considerar en esta clasificación es las diferentes formas de generar los bits del producto. En el caso a), incluso aunque la generación del producto puede, en principio, ser o serie o paralelo, únicamente se considerará el último caso. De esta forma, se puede pensar en el multiplicador como un circuito estrictamente combinacional. Más aún, si los productos tienen que generarse en forma secuencial, deberá dotarse al multiplicador con un procedimiento de serialización y, de esta forma, el multiplicador estaría en los casos b) y c).

En el caso b) cada bit de la mitad menos significativa del producto, como ya se ha argumentado, no estará nunca disponible antes que el bit correspondiente del factor serie. La mitad más significativa no se puede generar hasta que se suministre el último bit del factor serie al multiplicador; su presentación puede ser tipo serie como la primera mitad o paralelo. Una observación similar se puede hacer para el caso c).

La Tabla 4.1 resume la clasificación descrita.

4.3.- MULTIPLICADORES TIPO PARALELO

Un multiplicador paralelo es un circuito combinacional que tiene como entrada los bits de los dos factores y como salida los bits del producto. Teóricamente al menos, se podría realizar como un circuito combinacional de salida

Tipo de multiplicador	Factores		Producto	
	Multiplicando	Multiplicador	Mitad menos significativa	Mitad más significativa
Paralelo	Paralelo	Paralelo	Paralelo	Paralelo
Serie - Paralelo	Paralelo - Serie	Serie - Paralelo	Serie	Paralelo - Serie
Serie	Serie	Serie	Serie	Paralelo - Serie

TABLA 4.1.- Clasificación de multiplicadores digitales.

múltiple en dos niveles, pero su coste sería prohibitivo.

Una segunda posibilidad es utilizar una ROM de tamaño adecuado que se programa como una tabla de consulta en la que los bits de los factores nos determinan la dirección de la memoria que tiene como contenido el producto correspondiente de forma análoga a como se hizo con los contadores paralelo ([4]) ([5]).

Si m y n representan, respectivamente, las longitudes en bits del multiplicando y multiplicador, la ROM que se necesita debe tener una capacidad de 2^{m+n} palabras $x(m+n)$ bits (Ver Figura 4.2).

La utilización de una ROM no es factible incluso para valores de m y n moderados (m y $n > 5$).

Básicamente hay dos esquemas para multiplicadores paralelos:

- de generación de una matriz de productos parciales y reducción posterior.
- de tipo iterativo.

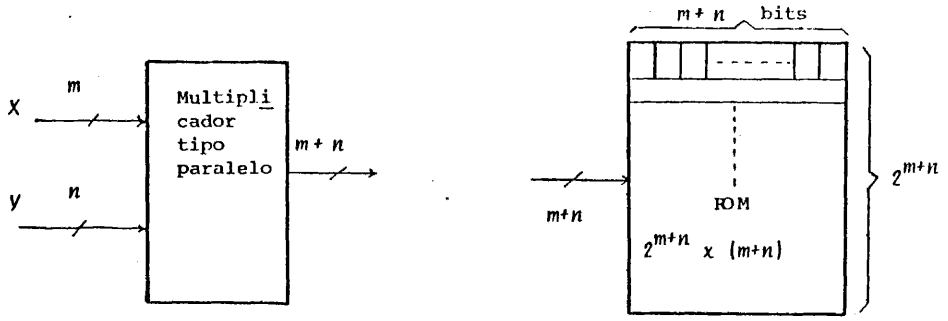


Figura 4.2.- Síntesis de un multiplicador paralelo mediante una ROM

4.3.1.- Tipo de generación y reducción

En esta clase de multiplicadores paralelos, el proceso de multiplicación se efectúa en tres pasos básicos, aun cuando no existe ninguna separación entre las correspondientes tres partes del circuito (Ver Figura 4.3)

a_1) Generación de la matriz M_0 de los productos de cada bit (o grupo de bits) del multiplicando por cada bit (o grupo de bits) del multiplicador.

a_2) Reducción del número de filas en esta matriz a una de dos filas M_ℓ cuya suma es igual al resultado.

a_3) Suma de estas dos filas para obtener el producto final.

En el proceso de reducción se hace uso extensivo de los contadores paralelos discutidos ya previamente. Normalmente se suele efectuar una reducción a dos (de acuerdo con la Definición 3.4), aunque se podría efectuar una reducción a uno, en cuyo caso sobraría el paso a_3). En este último paso, la suma se efectúa por un sumador rápido dotado con previsión de arrastre ([6]).

En la Figura 4.4 se representa esquemáticamente un ejemplo de este tipo de multiplicador cuando los factores son de 4 bits cada uno.

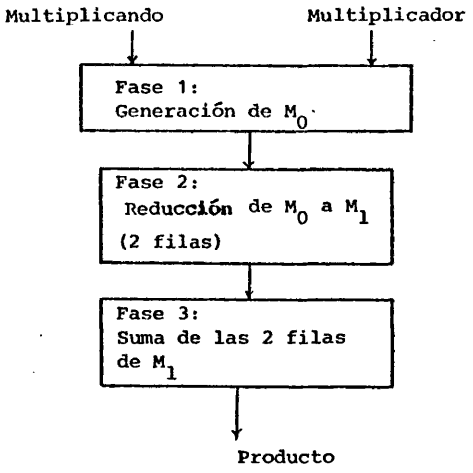


Figura 4.3.- Fases de un multiplicador paralelo del tipo de generación y reducción.

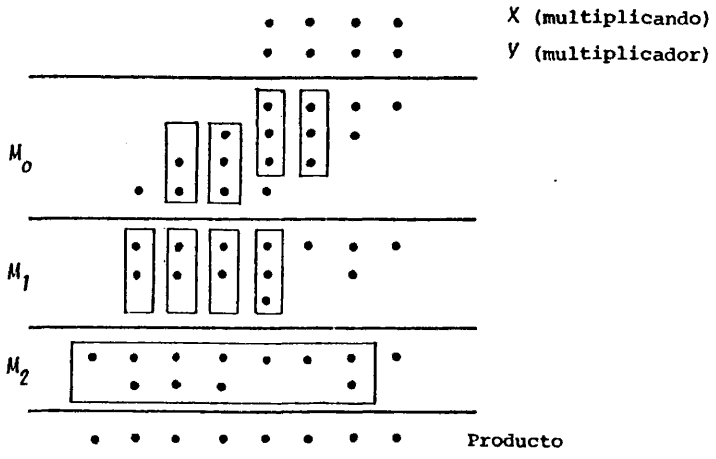


Figura 4.4.- Esquema de un multiplicador paralelo de 4 x 4 bits, utilizando contadores paralelos tipo (3; 2).

El ejemplo de la Figura 4.4 muestra que el problema planteado en los multiplicadores paralelos es uno de reducción, tal como se definió en la sección 3.3 del capítulo anterior.

Para acelerar al máximo la obtención del resultado caben dos acciones:

1) Reducir el número de filas en la matriz inicial utilizando como componentes primitivas módulos elementales de multiplicación $m \times p$ en lugar de las puertas Y (multiplicadores (1 x 1)).

En efecto, el uso de multiplicadores 1 x 1 da para la matriz M_0 las siguientes alturas:

$$\begin{aligned} h_i &= i + 1 \qquad \text{para} \quad i = 0,1,\ldots,n-1 \\ h_i &= 2n-1-i \qquad " \quad i = n,\ldots.2n-2 \end{aligned} \tag{4.1}$$

donde h_i es la altura de la columna de peso 2^i y n es la longitud de palabra del multiplicando y multiplicador.

Sin embargo, si la generación de la matriz inicial se efectúa con multiplicadores elementales de capacidad $m \times p$, la altura de las distintas columnas de la matriz viene dada por:

$$\begin{aligned} h_i &= \left\lfloor \frac{i}{m} \right\rfloor + \left\lfloor \frac{i}{p} \right\rfloor + 1 \qquad i = 0,1,2,\ldots,n-1 \\ h_i &= \left\lfloor \frac{2n-1-i}{m} \right\rfloor + \left\lfloor \frac{2n-1-i}{p} \right\rfloor + 1 \qquad i = n,\ n+1,\ldots,2n-1 \end{aligned} \tag{4.2}$$

h_i tiene el mismo significado anterior, y $\lfloor x \rfloor$ representa el mayor número entero menor o igual a x. Si el multiplicador elemental $m \times p$ se efectúa con una ROM, su capacidad deberá ser de $2^{m+p} \times (m+p)$ bits. (Así una ROM de 256 x 8 puede programarse para ser utilizada como un multiplicador elemental 4 x 4).

2) Usar contadores paralelos generalizados de la mayor capacidad posi-

ble para reducir al máximo el número de niveles en el proceso de reducción.

Conviene advertir en este punto que si lo que se pretende es reducir el número de niveles, los algoritmos desarrollados en el capítulo anterior no son los más adecuados, puesto que aquéllos estaban orientados fundamentalmente a minimizar el número de componentes primitivas utilizadas, dando en este sentido soluciones óptimas o cuasióptimas. Así, por ejemplo, en el caso de la Figura 4.4, la síntesis quedaría tal como indica la Figura 4.5.

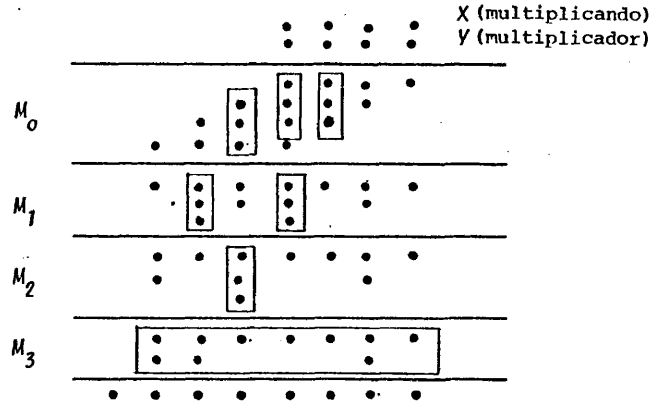


Figura 4.5.- Esquema alternativo de un multiplicador paralelo de 4 x 4 bits, utilizando contadores paralelos tipo (3; 2).

Comparando ambas figuras se observa que el número de módulos elementales (contador (3; 2)) ha sido reducido de 8 a 6 pero a expensas de aumentar en 1 nivel el proceso de reducción.

Una buena heurística cuando el criterio es minimizar el número de niveles fue inicialmente propuesta por Dadda ([7]) utilizando como componentes primitivas contadores tipo (3; 2) y posteriormente Stenzel y col. ([2]) generalizaron el procedimiento.

Su idea es la siguiente:

Se parte, como datos iniciales, con los siguientes parámetros:

1) Longitud de palabra del multiplicando y multiplicador.

2) Módulo elemental de multiplicación para la generación de M_0.

3) Contador paralelo generalizado utilizado en el proceso de reducción.

4) Problema de reducción a 2.

Del conocimiento de 1) y 2), teniendo en cuenta (4.1) ó (4.2), se tiene determinada la matriz inicial M_0.

Los contadores paralelos generalizados pertenecen a la categoría de rectangulares y saturados, es decir, son de la forma $(k \times n; d)$

donde:

$$n = \frac{2^d - 1}{2^k - 1} \qquad (4.3)$$

k juega el papel de n en el capítulo anterior, así como n el de N (el cambio obedece a evitar confusión con la longitud de palabra del multiplicador que tradicionalmente se denota por n).

Además, se impone la ligadura de que el número de columnas de entrada $(d = \delta \cdot k)$ para que la expresión (4.3) corresponda a un número entero.

El problema queda reducido a cómo ir insertando contadores para pasar de la matriz inicial M_0 a una equivalente con solamente 2 filas.

Dadda y Stenzel invierten el razonamiento y lo formulan en los siguientes términos: Dado un tipo de contador, existirá una altura máxima para cada nivel de reducción, de forma que si la matriz sobrepasa dicha altura se hace necesario un nivel adicional para la reducción. Se trata pues de determinar esta secuencia de alturas máximas para cada nivel, lo que nos permitirá, conocida la altura máxima de la matriz M_0, el calcular el número de niveles necesarios en la reducción. Esto permitirá posteriormente dar un algoritmo de inserción de contadores

elementales de forma que se vaya recorriendo la secuencia determinada en orden inverso.

El efecto de una cadena de contadores adyacentes del tipo $(k \times r; d)$ es la de producir d bits de salida en intervalo de k bits. Las salidas de los contadores se alinean de forma tal que no más de s salidas contribuyen a una columna dada. En la Figura 4.6 se muestra el caso de contadores (3 x 9; 6).

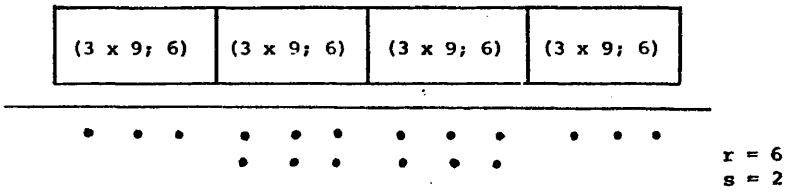


Figura 4.6.- Efecto de una serie de contadores adyacentes tipo (3 x 9; 6)

La Figura 4.6 nos muestra que una matriz de r bits de altura se puede reducir a una de s bits mediante el uso de un nivel de contadores.

Esta consideración permite calcular el número de niveles de contadores necesitados para reducir una matriz inicial de más de r bits a una de s bits. Sea ℓ_j la altura máxima de la matriz que puede reducirse a s bits usando j niveles. Con esta notación $\ell_0 = s$ y $\ell_1 = r$, conocida ℓ_j se puede determinar ℓ_{j+1} si se tiene en cuenta que los ℓ_j bits representan la salida de una fila de $\lfloor \ell_j / s \rfloor$ contadores en el nivel $j+1$ más un residuo de $(\ell_j) mod\ s$ bits que no fueron reducidos por los contadores.

Cada uno de los $\lfloor \ell_j / s \rfloor$ contadores utilizados consumen r bits de salida, por tanto

$$\ell_{j+1} = r \left\lfloor \frac{\ell_j}{s} \right\rfloor + (\ell_j) \bmod s \qquad (4.4)$$

Esta expresión, con las condiciones iniciales, dadas, permite determinar las secuencias que dan las alturas de reducción máxima para un tipo de contador dado.

En la Tabla 4.2 se dan estas secuencias para algunos contadores típicos.

Tipo de contador	Secuencia de reducción
(3; 2)	2 3 6 9 13 ....
(7; 3)	3 7 15 35 79 ....
(2 x 5; 4)	2 5 11 26 65 ....

TABLA 4.2.- Secuencias de alturas de reducción máxima de determinados contadores.

Mediante el uso de esta tabla y de las expresiones (4.1) ó (4.2), según sea el caso, se tiene determinado el número de niveles necesarios para la reducción.

Así, por ejemplo, en la síntesis de un multiplicador de 32 x 32 bits con módulo de multiplicación elemental tipo 4 x 4 y contadores tipo (2 x 5; 4) se necesitan 3 niveles, ya que la altura máxima de la matriz inicial sería de 15 y la secuencia de reducción correspondería a: 15 $\longrightarrow$ 11 $\longrightarrow$ 5 $\longrightarrow$ 2

Conviene advertir que determinados tipos de contadores no dan una reducción a dos, como por ejemplo el (7; 3), en este caso se necesitaría un nivel adicional de contadores (3; 2) para efectuar la reducción final.

Esto nos lleva al siguiente algoritmo de Stenzel y col., modificado al permitir módulos de multiplicación elemental del tipo m x p.

Algoritmo de Stenzel y col. modificado

Entrada: n, m, p, k, r, d

n → longitud del multiplicando y del multiplicador

$m \times p$ → multiplicador elemental

$(k \times r;\ d)$ → contador utilizado

1) Fase de inicialización:

a) Si $m = p = 1$

$$h_i = i + 1 \qquad i = 0,1,\ldots,n-1$$

$$h_i = 2n-1-i \qquad i = n,\ldots,2n-2$$

b) Si $m \neq 1$ ó $p \neq 1$

$$h_i = \left\lfloor \frac{i}{m} \right\rfloor + \left\lfloor \frac{i}{p} \right\rfloor + 1 \qquad i = 0,1,\ldots,n-1$$

$$h_i = \left\lfloor \frac{2n-1-i}{m} \right\rfloor + \left\lfloor \frac{2n-1-i}{p} \right\rfloor + 1 \qquad i = n,n+1,\ldots,2n-1$$

$$H = \max\ (h_i) \qquad i = 0,\ldots,2n-1$$

H va a almacenar siempre la altura máxima de la matriz en un nivel de reducción dado. Inicialmente corresponde a la altura de la matriz inicial M_0.

2) Generación de la secuencia ℓ_j, hasta un j tal que

$$\ell_j < H \text{ y } \ell_{j+1} \geq H, \quad T = \ell_j$$

T va a almacenar siempre la altura máxima de la matriz que resulta después de efectuada la reducción en curso. Inicialmente corresponde al mayor término de la secuencia menor que H.

3) Si $H=\delta$ entonces. Parar si no hacer 3.1) para $i = 0,1,\ldots,2n-1$

3.1) Si $h_i \leq T$ no hacer nada, si no insertar un contador en este punto, ajustar las alturas de las columnas y repetir 3.1).

Las reglas para ajustar las alturas de las columnas son:

$h_{i+j} = max\ \{T, (h_{i+j}-n+1)\}$ para $j=0,1,\ldots,k-1$

$h_{i+j} = h_{i+j} + 1 \qquad j=k,\ldots,d-1$

4) $H = T$

T = próximo término en la serie de reducción

ir a 3)

En la Tabla 4.3 se da un listado de la síntesis de un multiplicador de 32 x 32 bits utilizando como módulos elementales, multiplicadores de 2 x 2 bits y contadores paralelos (2 x 5; 4). Los resultados se presentan de la forma siguiente: en cada etapa o nivel del algoritmo de reducción, el primer grupo de números indica la inserción de contadores (su número y el lugar donde se ubican) y el segundo grupo refleja cómo queda la matriz después de la reducción.

4.3.2.- Tipo iterativo

Hennie ([8]) define un circuito iterativo como aquel que está compuesto de un número de sub-circuitos o celdas idénticas conectadas de forma regular. La idea de utilizar tales tipos de estructuras iterativas es particularmente atractiva en la actualidad, ya que conducen sus diseños de forma natural a circuitos con elevada escala de integración (LSI). Se han propuesto ya diversos tipos de estructuras iterativas para la multiplicación de números binarios. La multiplicación se efectúa o usando un algoritmo convencional de suma y desplazamiento ([9])([10]), o una realización directa de los productos parciales y de su suma ([1]). El multiplicador iterativo que hemos diseñado ([11]) pertenece a esta última categoría.

Sean X e Y dos números de $2n$ bits cada uno

MATRIZ INICIAL

1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15	17	17	19	19
21	21	23	23	25	25	27	27	29	29	31	31	31	31	29	29	27	27	25	25
23	23	21	21	19	19	17	17	15	15	13	13	11	11	9	9	7	7	5	5
3	3	1	1																

ALGORITMO DE REDUCCION

ETAPA NUMERO 1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	1	0	2	0	2	0	2	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0																

1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15	17	17	19	19
21	21	23	23	25	25	26	26	26	26	26	25	26	25	25	25	26	26	26	26
23	23	21	21	19	19	17	17	15	15	13	13	11	11	9	9	7	7	5	5
3	3	1	1																

ETAPA NUMERO 2

0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	2	0	3	0
4	0	4	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0
5	0	4	0	3	0	3	0	2	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0																

1	1	3	3	5	5	7	7	9	9	11	11	11	11	11	11	11	11	11	11
11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
11	11	11	11	11	11	11	11	11	11	11	11	11	11	10	10	7	7	5	5
3	3	1	1																

ETAPA NUMERO 3

0	0	0	0	0	0	1	0	2	0	2	0	2	0	2	0	2	0	2	0
2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0
2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	1	0	1	0
0	0	0	0																

1	1	3	3	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
4	4	1	1																

ETAPA NUMERO 4

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

1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2																

Tabla 4-3.- Síntesis de un multiplicador de 32x32 bits utilizando multiplicadores elementales de 2x2 bits y contadores paralelos (2x5;4).

$$X = (x_{2n-1}, x_{2n-2}, \ldots, x_0)$$
$$Y = (y_{2n-1}, y_{2n-2}, \ldots, y_0)$$

Su producto viene dado por:

$$X.Y = \sum_{i=0}^{2n-1} x_i 2^i \sum_{j=0}^{2n-1} y_j 2^j = \sum_{j=0}^{2n-1} \left(\sum_{i=0}^{2n-1} x_i 2^i y_j \right) 2^j \qquad (4.5)$$

La ecuación (4.5) es la base para el algoritmo de suma y desplazamiento.

Alternativamente se puede ver el producto en términos de la suma de las columnas de la matriz de productos de bits inicial. Es decir,

$$X.Y = \sum_{i=0}^{2(2n-1)} \left(\sum_{k=0}^{i} x_k y_{i-k} \right) 2^i \qquad (4.6)$$

con x_i, $y_i = 0$ para $i < 0$ ó $i > 2n-1$

Definamos:

$$\hat{x}_{2k} = x_{2k+1} \cdot 2 + x_{2k}$$
$$\hat{y}_{2k} = y_{2k+1} \cdot 2 + y_{2k} \quad \text{para } k=0, 1, \ldots, n-1 \qquad (4.7)$$

Entonces,

$$X.Y = \sum_{k=0}^{2(n-1)} s_{2i} \, 2^{2i} \qquad (4.8)$$

donde

$$s_{2i} = \sum_{k=0}^{i} \hat{x}_{2k} \, \hat{y}_{2(i-k)} \qquad (4.9)$$

con $\hat{x}_{2k}$, $\hat{y}_{2k} = 0$ para $k < 0$ ó $k > n-1$

En la Fig. 4.7 se muestra una celda utilizada en el multiplicador iterativo que se describe. Esta celda puede efectuar la función aritmética $p = ab+c+d$, donde a, b, c y d tienen la misma longitud de palabra de dos bits.

La suma de dos términos tales como c y d no aumenta la longitud de palabra del producto, ya que: $(2^2-1)(2^2-1)+2(2^2-1) = 2^4-1$.

Esta celda básica se puede implementar fácilmente sobre una ROM de 256x4 bits utilizada como tabla de consulta.

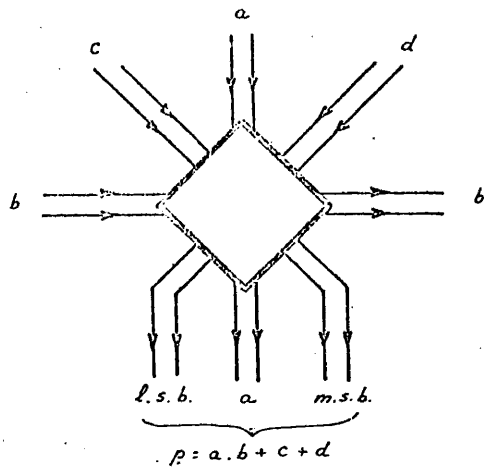


Fig. 4.7.- Celda básica del multiplicador iterativo

Para 2n = 8, en la Fig. 4.8, se da una realización práctica del multiplicador iterativo. Cuando se utilizan todas las entradas primarias del multiplicador iterativo (es decir, las entradas libres) se puede realizar una función más compleja $XY + U + V$, donde U y V tienen la misma longitud de palabra que X e Y . En la figura se muestra, asimismo, un ejemplo específico; las entradas en forma binaria de X, Y, y V son:

$$X = (1\ 0\ 1\ 1\ 0\ 1\ 1\ 1)$$

$$Y = (1\ 0\ 1\ 0\ 0\ 1\ 0\ 1)$$

$$U = (0\ 1\ 1\ 0\ 1\ 1\ 0\ 0)$$

$$V = (1\ 0\ 0\ 0\ 0\ 1\ 1\ 1)$$

Para un multiplicador de 2nx2n bits, el tiempo de cálculo del propuesto por Guild es $(4n-1)\tau$ y en los de Chung-Bedrosian ([12]), ([13]) son $6n\tau$ para el tipo 1 y $4n\tau$ para el tipo 2, mientras que para el que hemos presentado es de $(2n-1)\tau$ donde τ es el retardo producido por una única celda. En cuanto a la memoria necesitada, el multiplicador desarrollado necesita n^2 celdas, mientras que para los tipos 1 y 2 de Chung y Bedrosian, únicamente son necesarias $4n$ y $2n$ celdas respectivamente. Sin embargo, la regularidad de la estructura del multiplicador propuesto lo hace muy atractivo para su fabricación LSI.

Este multiplicador puede generalizarse fácilmente a uno de $pnxpn$ bits con

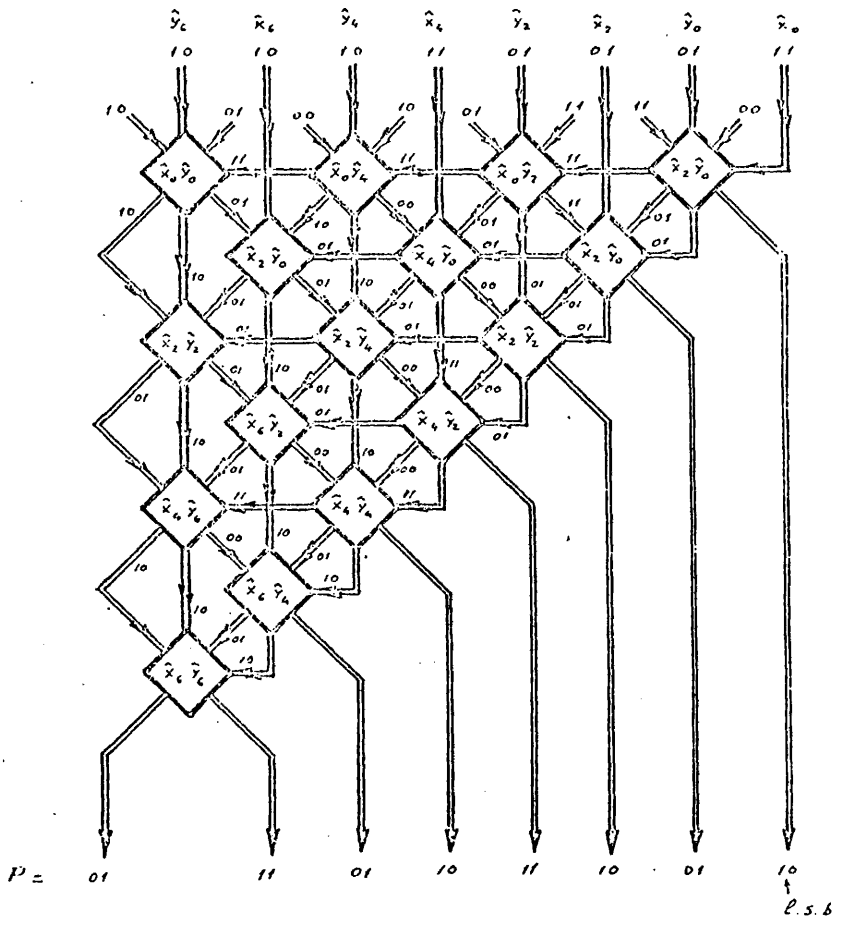


Fig. 4.8.- Esquema del multiplicador iterativo de 8x8 bits

$p > 2$; en este caso, la celda básica deberá tener una capacidad de $2^{4p} \cdot 2p$ bits (la suma de dos términos de longitud de palabra p bits no aumenta la longitud de palabra del producto, ya que $(2^p-1)^2+2(2^p-1) = 2^{2p}-1$).

Sin embargo, como se ve, el tamaño de la ROM se hace excesivo incluso para p no muy elevados. Una posible solución a este problema sería adoptar un esquema tipo generación y reducción para la celda básica, tal como se describió en el apartado anterior.

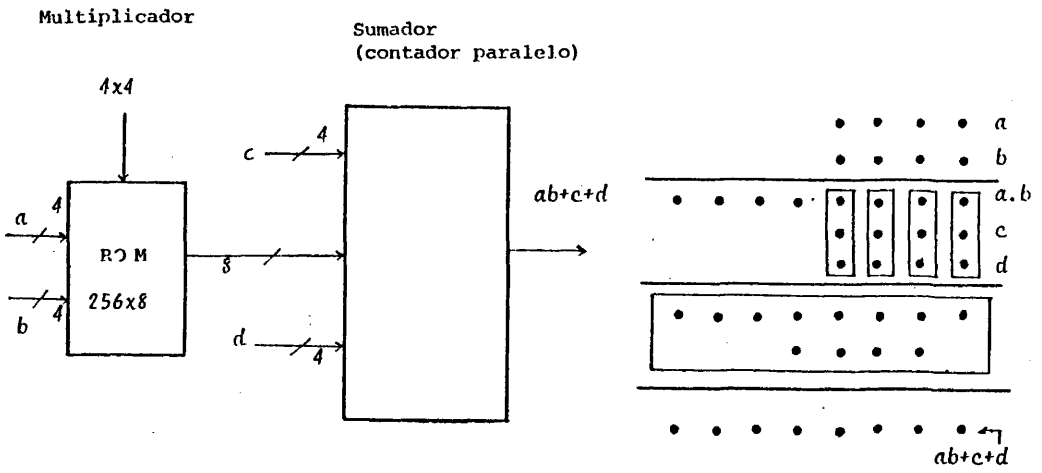


Fig. 4.9.- Celda básica para el caso $p=4$.

En la Fig. 4.9 se presenta un esquema para el caso $p=4$. Se utiliza una ROM de 256x8 para realizar un multiplicador 4x4 y a la salida se suman los dos números de 4 bits utilizando un esquema de reducción. Esta interrelación entre multiplicadores tipo generación y reducción e iterativos que aquí se ha llevado del primero al segundo, para construir la celda básica del multiplicador iterativo, se puede también invertir utilizando a éstos como módulos de multiplicación elemental en la generación de la matriz inicial M_0

4.3.3.- Consideraciones tecnológicas

Entre los parámetros más críticos y que caracterizan a un multiplicador cabe considerar los siguientes:

- Velocidad de multiplicación
- Disipación de potencia
- Longitud de palabra de los operandos
- Capacidad de expansión
- Forma de representación de los datos.

Por lo que respecta al rango de velocidades en multiplicadores integrados en un sol

chip oscilan entre 40 nanosegundos del multiplicador de 2x4 bits AMD25S05 ([14]) hasta los 800 nanosegundos del multiplicador de 16x16 bits MMI67516 ([15]). El primero es de tipo iterativo y realiza la función $xy+k$, el segundo pertenece a la categoría de generación y reducción. De forma general podemos decir que para los multiplicadores iterativos el retardo aumenta linealmente con la longitud de los operandos (n), mientras que,en los de generación y reducción, dicho retardo únicamente varía con el logaritmo de n.

La disipación de potencia de estos multiplicadores oscila entre los 300 miliwatios del AMD25505 y los 5 watios del TRW-MPY16 ([16]) que requiere un elemento de refrigeración para no sobrepasar los límites de temperatura que puede tolerar.

La capacidad de expansión se refiere a la posibilidad de construir multiplicadores de mayor longitud de palabra, utilizando el menor número posible de circuitos integrados que no sean de este tipo.

La forma de representación de los datos es o sin signo o con signo (normalmente en complemento a 2). El segundo tipo de representación es el más utilizado en los multiplicadores monolíticos (integrados en un solo chip), aunque requieren un hardware adicional para acomodar el bit de extensión correspondiente al signo y el tener que efectuar la complementación durante el proceso de expansión. Algunos, como el MMI ([15]) (multiplicador de 8x8 bits) pueden acomodar ambos tipos de representación.

La tabla 4.4 tomada de S. Waser ([17]) da las características fundamentales de los multiplicadores monolíticos más comunes.

La tendencia actual para el futuro parece que va en la línea de extender la capacidad de los multiplicadores monolíticos haciéndolos verdaderos procesadores aritméticos programables. Esto se ejemplifica por dos unidades recientes: la primera es el MMI67516, que además de la multiplicación efectúa la división y la suma de diversos productos; la segunda es la unidad aritmética para procesamiento de señal

Tipo	Configuración	N° de pins	Retardo máximo (ns)	Potencia máxima (vatios)	Producto normalizado vel x potencia	Representación de datos	Tipo de paralelismo	Algoritmo
TRW MPY-8	8x8	40	170	1,5	4,0	Compl-2	Total	Generación
TRW MPY-12	12x12	64	200	3,75	5,2	Compl-2	Total	y
TRW MPY-16	16x16	64	230	5	4,5	Compl-2	Total	reducción
MMI 67558	8x8	40	125	1,4	2,7	Compl-2 o sin signo	Total	Booth
MMI 67516	16x16	24	800	1,3	4,1	Compl-2	Multando y 2 bits del multiplicador	Booth
MMI 67508	8x8	20	400	0,75	4,7	Compl-2	Total	
AMD 25S05	2x4	24	40	0,9	4,5	Compl-2	Total	Iterativo
AMD 25L14	8x1	16	50	0,6	3,7	Compl-2	Multando y 1 bit del multiplicador	Booth
TI 74S274 (ROM)	4x4	20	70	0,6	2,6	Sin signo	Total	ROM
MOT 10183	2x4	24	25	1,0	2,5	Compl-2	Total	Iterativo
AMD 25LS2516	8x8	40	400	1,4	8,7	Compl-2	Igual que MMI67516	Booth

Tabla 4.4.- Características de algunos multiplicadores monolíticos disponibles en el mercado

de TRW que realiza suma, resta y multiplicación en forma paralela, cambios de escala y multiplexing en registros paralelos ([18]).

4.4 DEL MULTIPLICADOR CUASI-SERIE AL MULTIPLICADOR PARALELO

El multiplicador cuasi-serie ([3]) obtiene los bits del producto de forma serie, desde el menos significativo al más significativo. El procedimiento es contar (con un contador paralelo) el número de unos en la columna correspondiente de la matriz de productos parciales con la adición de los arrastres previos. Nuestro objetivo en esta sección es demostrar que, utilizando el concepto de contadores paralelos generalizados, es posible ir del multiplicador cuasi-serie al multiplicador paralelo. La consecuencia es, entonces, la posibilidad de una elección más flexible en cuanto a la estructura del multiplicador se refiere, teniendo en cuenta el compromiso velocidad-coste.

Sean X e Y el multiplicando y el multiplicador respectivamente de longitud de palabra n. Se utiliza un módulo de multiplicación elemental de $\ell x \ell$ bits, de forma tal que $n = kx\ell$.

Entonces, el producto P se puede expresar como:

$$P = XY = \sum_{i=0}^{2(\ell k-1)} \left(\sum_{k=0}^{i} x_k \, y_{i-k} \right) 2^i = \sum_{i=0}^{2n-1} p_i \, 2^i \qquad (4.10)$$

con $x_i, y_i = 0$ para $i < 0$ ó $i \geq \ell k-1$.

Definimos:

$$\hat{x}_{m\ell} = \sum_{j=0}^{\ell-1} x_{m\ell+j} \, 2^j \qquad (4.11)$$

$$\hat{y}_{m\ell} = \sum_{j=0}^{\ell-1} y_{m\ell+j} \, 2^j \text{ para } m = 0,1,\ldots,k-1$$

Entonces:

$$P = XY = \sum_{i=0}^{2(k-1)} s_{i\ell} \cdot 2^{i\ell}$$

donde

$$s_{i\ell} = \sum_{m=0}^{i} \hat{x}_{m\ell}\, \hat{y}_{(i-m)\ell} \qquad (4.12)$$

con $\hat{x}_{m\ell}$, $\hat{y}_{m\ell} = 0$ para $m < 0$ ó $m > k-1$.

Para formar P se utiliza el siguiente mecanismo recursivo:

$$\begin{aligned} \hat{p}_{o\ell} &= (s_{o\ell}) \bmod 2^{\ell}, \quad c_o = 0 \\ \hat{p}_{i\ell} &= (s_{i\ell} + c_{i-1}) \bmod 2^{\ell} \\ c_i &= (s_{i\ell} + c_{i-1} - \hat{p}_{i\ell}) \,/\, 2^{\ell} \end{aligned} \qquad i = 1,2,\ldots,2k-1 \qquad (4.13)$$

De (4.13) es fácil ver que:

$$\hat{p}_{i\ell} = \sum_{j=0}^{\ell-1} p_{i\ell+j}\, 2^{j}$$

$$P = \sum_{i=0}^{2k-1} p_{i\ell}\, 2^{i\ell}$$

El algoritmo definido por (4.11), (4.12) y (4.13) puede ser realizado con los siguientes elementos:

- Un registro de desplazamiento de (2n-1) bits.
- Un registro de memoria de n bits.
- Un contador paralelo generalizado tipo $(2\ell x k; r]$ donde

  $$r = \lceil \ell \log_2 \{k(2^{\ell} - 1)^2 + 1\} \rceil \qquad (4.14)$$

- k módulos de multiplicación elemental de $\ell x \ell$ bits cada uno.
- Un registro de r bits con posibilidad de sumar y desplazar la información que denominamos registro de rebose.

La Fig. 4.10 es un diagrama de bloques del multiplicador.

Para comenzar el proceso, el multiplicando se almacena en la parte izquierda del registro de desplazamiento de (2n-1) bits y el multiplicador se introduce en el registro de memoria de n bits.

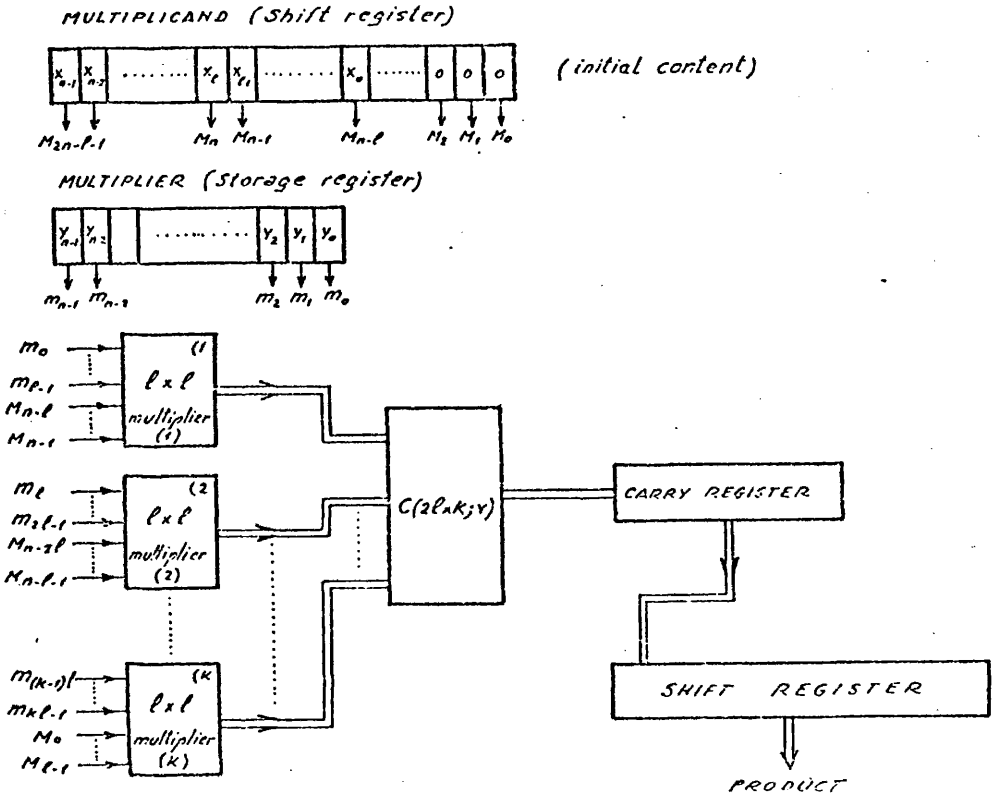


Fig. 4.10.- Diagrama de bloques del multiplicador cuasi-serie-paralelo

La ejecución del ciclo i-ésimo de la multiplicación da $\delta_{i\ell}$ de (4.12), como la salida del contador paralelo generalizado y c_i de (4.13) como el contenido del registro de rebose cuyos ℓ bits menos significativos son sacados como ℓ bits del producto.El multiplicando se desplaza a la derecha ℓ bits y comienza un nuevo ciclo. Cuando un total de $(2k-1)$ de estos ciclos se han completado, el resultado ha sido obtenido con grupos de ℓ bits en cada ciclo, salvo en el último, en el que son 2ℓ bits.

En la Fig. 4.11, se considera un ejemplo que demuestra el procedimiento

	MULTIPLICAND MULTIPLIER	MULTIPLIER MODULE	GENERALIZED COUNTER	CARRY REGISTER AFTER ADD	CARRY REGISTER AFTER SHIFT	PRODUCT SI
	110111110000000	(1) 001100				
CYCLE 1	101111010	(2) 000000	001100	001100	001	100
		(3) 000000				
	000110111110000	(1) 001110				
CYCLE 2	101111010	(2) 101010	111000	111001	111	001
		(3) 000000				
	000000110111110	(1) 001100				
CYCLE 3	101111010	(2) 011110	1011011	1100010	1100	010
		(3) 110001				
	000000000110111	(1) 000000				
CYCLE 4	101111010	(2) 101010	1001101	1011001	1011	001
		(3) 100011				
	000000000000110	(1) 000000				
CYCLE 5	101111010	(2) 000000	011110	101001		101001
		(3) 011110				

PRODUCT P= 101001001010 001 100

446 x 378 = 168588

Fig. 4.11.- Pasos en el cálculo de 446x378 con el multiplicador cuasi-serie-paralelo descrito.El problema que se resuelve con este algoritmo es 446x378.En este caso,

$$\left.\begin{matrix} n = 9 \\ \ell = 3 \end{matrix}\right\} \longrightarrow k = 3 \longrightarrow \text{de (4.14)} \quad n = 8$$

El grado de paralelismo viene dado por el parámetro k ; así, cuando $k=n$ se tiene un multiplicador cuasi-serie y en el otro extremo, cuando $k=1$, el multiplicador resulta uno de tipo paralelo, en el que únicamente es necesario un módulo de multiplicación.

Debe observarse que, tanto en la síntesis de los multiplicadores elemen

tales como en la del contador paralelo generalizado, podemos utilizar las técnicas desarrolladas previamente. Eso hace que esta estructura tenga una gran flexibilidad en cuanto al compromiso de velocidad-coste que venga impuesto como ligadura de diseño.

4.5 DISEÑO DE UN MULTIPLICADOR PARALELO DE MUY BAJO COSTE PARA NUMEROS FRACCIONARIOS

En esta sección se presenta el diseño de un multiplicador paralelo de muy bajo coste, aunque de precisión limitada, que ha sido incorporado como escalador en las unidades de integración digital que se discuten en el próximo capítulo.

Dado un número Y, nuestro objetivo es multiplicarlo por un número P que está normalizado $(0 \leq P \ll 1)$ (ver Fig. 4.12a). Con el fin de mantener el coste en unos límites razonables, el valor de P se redondea únicamente a 4 bits, utilizando para ello un registro adicional P' (ver Fig. 4.12). Antes de que el cálculo comience, P' se presetea a 2^{-4}, de forma que los 4 bits más significativos de P se redondean al cuanto más cercano a 2^{-3}.

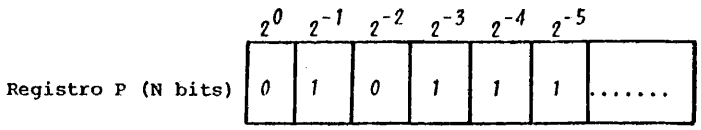
Así, para formar, por ejemplo, una fracción de 23/32 de un número Y con una precisión de 4 bits (m) (es decir, $m=k/8$ donde $0 \leq k \leq 8$), requiere que m tome la siguiente secuencia de valores:

$$m = \underbrace{\frac{3}{4}, \frac{3}{4}, \frac{5}{8}, \frac{3}{4}}, \underbrace{\frac{3}{4}, \frac{3}{4}, \frac{5}{8}}, \ldots$$

tal como se deduce de la Tabla 4.5.

m+P'	1/16	25/32	24/32	23/32	26/32	etc.	....
m=P redondeado a 4 bits	0	6/8	6/8	5/8	6/8	etc.	....
P' (N-4)bits	1/16	1/32	0	3/32	1/16	etc.	....
t	0	1	2	3	4	5	....

Tabla 4.5.- Redondeo del registro P a 4 bits



a) Formato del registro P

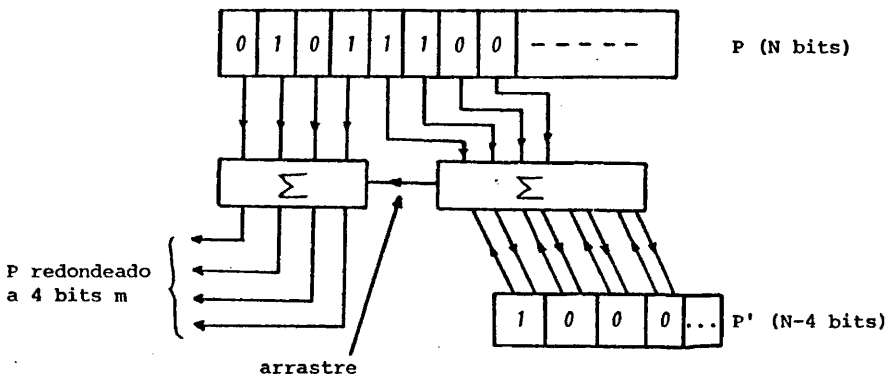


Fig. 4.12.- Estructura del registro P

Los nueve valores que puede tomar $m[m = 0, 1/8, \ldots, 8/8]$ pueden obtenerse como suma de dos sumandos de la forma $\pm 2^{-i}$, tal como indica la Tabla 4.6.

m	0	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8	
1er sumando	0	0	1/2	1/2	1/2	1/2	1	1	1	
2º sumando	0	1/8	-1/4	-1/8	0	1/8	-1/4	-1/8	0	

Tabla 4.6 Obtención de m como suma de dos sumandos de la forma $\pm 2^{-i}$

Si $m = (m_a, m_b, m_c, m_d)$ ($2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$), las funciones lógicas $\delta_\alpha(\alpha = 0, 2^{-1}, 2^0, 2^{-3}, -2^{-2}, -2^{-3})$ que se obtienen son:

$$\delta_0 = \bar{m}_a\ \bar{m}_b\ \bar{m}_c \quad (1^{er}\ \text{sumando})$$

$$\delta_{2-1} = m_b \oplus m_c$$

$$\delta_1 = m_a + m_b\, m_c$$

$$\delta_0 = \bar{m}_c\, \bar{m}_d \quad (2^\circ \text{ sumando})$$

$$\delta_{-2-2} = m_c\, \bar{m}_d$$

$$\delta_{2-3} = \bar{m}_c\, m_d$$

$$\delta_{-2-3} = m_c\, m_d$$

$$\delta_- = \delta_{-2-2} + \delta_{-2-3} = m_c$$

La selección del valor que corresponde a cada uno de los sumandos se realiza mediante un conjunto de multiplexores con una entrada de control (SN74157 para el primer sumando) y dos entradas de control (SN74153 para el 2° sumando) (V. Fig. 4.13a y b)

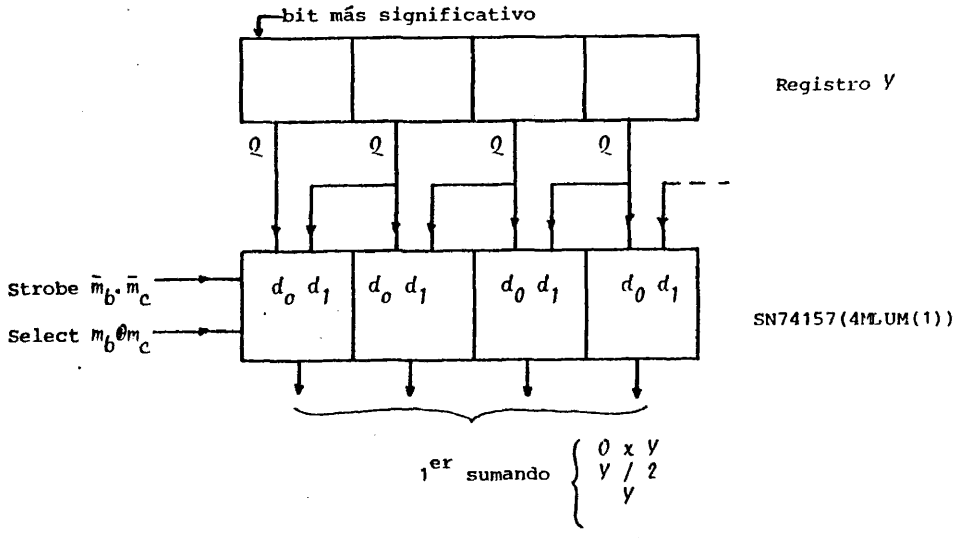


Fig. 4.13-a.- Síntesis con MLUM(1) [SN74157] del 1er sumando

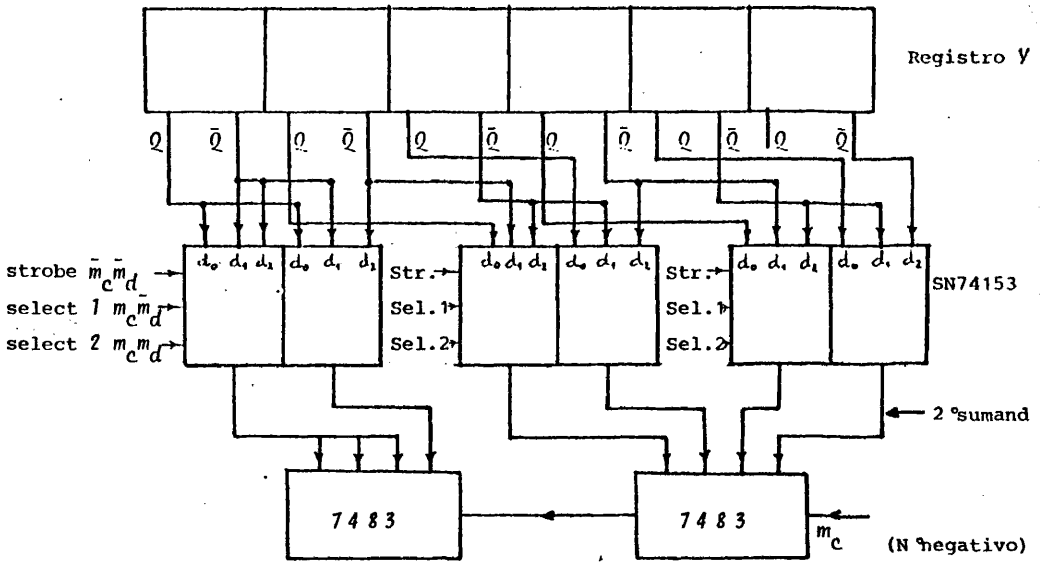


Fig. 4.13-b Síntesis con MLUM(2)[SN74153] del 2 ° sumando

Teniendo esto en cuenta, el coste en componentes MSI del multiplicador desarrollado es el siguiente: (41 es la longitud del registro Y)

- Multiplexores de dos entradas de datos (SN74157) → 1
- Multiplexores de cuatro entradas de datos (SN74153) → 21
- Sumadores de cuatro bits (7483) → 21
- Registro auxiliar P' (7495) → 1-1
- Generación de las señales de control para los multiplexores → 1

Así pues, si representamos por C_1 el coste de un circuito integrado de 16 patillas, el coste C será:

$$C = 6\ell$$

Si no se utilizase la capacidad de retención de residuo del registro P y este simplemente se trunease a los 4 bits mas significativos, el coste total se reduciria a:

$$C = 4\ell + 1$$

REFERENCIAS

(1) GUILD, H., "Fully iterative fast array for binary multiplication and addition" Electronics Letters, 12th., vol. 5, No. 12, pp. 263, June 1969

(2) STENZEL, J., KUBITZ, J. and GARCIA, H. "A compact high-speed parallel multiplication scheme" IEEE Trans Computers, vol. C-26, No. 10, pp. 948-957, Oct. 1977.

(3) SWARTZLANDER, E. Jr. "The quasi-serial multiplier" IEEE Trans. Computers, vol. C-22 No.4, pp. 317-321, April 1973.

(4) SWARTZLANDER, E. Jr. "Parallel counters" IEEE Trans. Computers, col. C-22, pp. 1021-24, Nov. 1973.

(5) KOBAYASHI, H. and OHARA, J. "A synthesizing method for large parallel counters with a netwaork of smaller ones" IEEE Trans. Computers, vol. C-27, pp. 753-57, ug. 1978.

(6) Mc SORLEY, L. "High speed arithmetic in binary computers" Proc. of the IRE, pp. 67-91, Jan. 1961.

(7) DADDA, L. "Some schemes for parallel multipliers"Alta Frequenza, vol. XXXIV, No. 5 pp. 349-56, Maggio 1965.

(8) HENNIE, F.C. "Iterative arrays of logical circuits" MIT 1961.

(9) HOFFMANN, C., LACAZE, B., CSILLAG, P. "Multiplieur parallele a circuits logiques itératifs" Electronics Letters, vol. 4, No. 9, May 1963.

(10) DEAN, K.J. "Design for a full multiplier" Proc. IEE, vol. 115, pp. 1592-94, 1978.

(11) DORMIDO, S. and CANTO Ma. A., "High-speed iterative array for binary multiplication" Electronics Letters, vol. 14, no. 23, pp. 742-3, Nov. 1978.

(12) CHUNG, J. and BEDROSIAN, D. "Iterative digital multiplier based on cellular arrays of R.O.M.S." Elect. Letters, vol 11, No. 18, pp. 426-28, ept. 1975.

(13) CHUNG, J. and BEDROSIAN, "Alternative iterative digital multiplier based on cellular arrays of R.O.M.." Elect. Letters, vol. 12, No. 17, pp. 447-8, Aug.1976.

(14) "Advanced Micro Devices, Inc., Low Power Schottky Data Book, Sunnyvale, Calif. 1977.

(15) "Monolithic Memories, Inc. 67516/67508 Data Sheet, Sunnyvale, Calif. Oct. 1977.

(16) "TRW, MPY-Series Multipliers", Redondo Beach, Calif., Oct. 1977

(17) WASER, S. "High-speed monolithic multipliers for real-time digital signal processin Computer, pp. 19-29, Oct. 1978.

(18) BUIE, L. and ZIMMERMAN, A. "Very large scale integrated circuits for digital signal processing" Circuits and Systems, pp. 2-8, Feb. 1977.

CAPITULO V

INTEGRADORES DIGITALES DE RESOLUCION EXTENDIDA (IDRE) CON TRANSMISION DE LAS DIFERENCIAS DE SEGUNDO ORDEN

5.1 BREVE REVISION HISTORICA DE LAS MAQUINAS DISEÑADAS PARA RESOLVER ECUACIONES DIFERENCIALES (ANALIZADORES DIFERENCIALES).

El desarrollo de la regla de cálculo por OUGHTRED, en 1630, marca el comienzo del cálculo analógico ([1]). Un Analizador Diferencial (AD) es el nombre genérico asignado a una clase de dispositivos de cálculo, con la tarea específica de resolver ecuaciones diferenciales (ED). El concepto de una máquina con la cual resolver ED no es una innovación reciente. Esta idea se atribuye normalmente al gran matemático LEIBNITZ (1675) ([2]), quien concibió un dispositivo que utilizaba complejas interrelaciones mecánicas como sustituto a los procesos de cálculo. Para la solución de ED, estos procesos se pueden dividir en dos clases:

a) Procesos aritméticos

b) Procesos infinitesimales de cálculo.

Debido a problemas tecnológicos propios de la época, los conceptos de Leibnitz no fueron plasmados en realizaciones prácticas, excepto para máquinas aritméticas simples, tales como la máquina de sumar de PASCAL.

J. SANG, en 1850, describía un dispositivo conocido como "planímetro", que, esencialmente, era un integrador mecánico del tipo de rueda y disco ([3]) y que tuvo gran popularidad en la época, pues hacia 1880 se habían ya construido unas 12.000 unidades ([4]).Después de ver el planímetro de SANG, en una demostración en 1850, J. MAXWELL propuso una mejora al incorporar el uso de dos esferas ([5]).

Durante el periodo 1875-78, E THOMSON (posteriormente Lord KELVIN) en una serie de trabajos ([6]),([7])([8]) describía los siguientes resultados:

- Un integrador mecánico del tipo de disco (inventado por su hermano J. Thomson ([9])).
- El concepto importante de resolver ED por integraciones mecánicas sucesivas (lazo cerrado).
- Diversas aplicaciones a ED con coeficientes variables, ecuaciones integrales que llevaban consigo el producto de dos funciones y un analizador armónico.

En realidad, la primera máquina que se construyó y se utilizó para integrar una función se hizo en 1925, en el M.I.T. por V. BUSH, F.D. GAGE y H.R. STEWART ([10]). Es interesante destacar que estos investigadores norteamericanos, en el momento de construir su máquina, lo hicieron desconociendo el trabajo de los hermanos Thomson, ya reseñado. El integrador de rueda y disco y la máquina que lo incorpora los describe Bush en ([11]).

A partir del analizador diferencial mecánico de Bush, se fueron realizando muchas mejoras en el periodo 1930-50. El desarrollo de esta actividad se puede encontrar en la referencia ([12]).

Con la aparición de la válvula electrónica (De Forest, 1906), C.A. LOVELL, de Bell Telephone Laboratories ([13]) desarrolló un amplificador operacional, el cual con una realimentación adecuada realizaba la integral,con respecto al tiempo, de cualquier función continua representada como una tensión variable aplicada a su entrada. Esto condujo a la aparición de los Analizadores Diferenciales Electrónicos, más comúnmente conocidos como Calculadoras Analógicas.

El alto coste y complejidad inicial de este tipo de calculadoras motivaron a F.G. STEELE, P.E. ECKDAHL y I.S. REED, en 1950 ([14]) a desarrollar una versión digital del analizador diferencial mecánico de Bush (A.D.D.).Esta unidad denominada MADDIDA (Magnetic Drum Digital Differential Analyzer) fue construida por NORTHROP AIRCRAFT Co., resultando un sistema económico y eficaz para la solución de E.D. que, durante un cierto periodo de tiempo,no tuvo rival para este tipo de tarea.

Durante la década de los 60, la tecnología se desarrolla espectacularmente y los campos del cálculo analógico y digital realizan un aumento muy sustancial en su capacidad, debido principalmente más a las mejoras de los dispositivos que a cambios profundos en la estructura de los mismos.

El único concepto estructural nuevo que aparece, desde el punto de vista de los A.D., fue la configuración híbrida que combina en un sistema integral tanto elementos analógicos como digitales.

El A.D.D. pertenece a la clase de calculadoras digitales de propósito especial, del tipo incremental. Es decir, frente a una calculadora digital de propósito universal, presenta, desde el punto de vista de utilización como analizador diferencial, las dos características distintivas siguientes:

1) Una máquina incremental tiene un algoritmo básico de cálculo que, en el caso del A.D.D., es el de integración.
2) Una máquina incremental efectúa la transferencia de información entres sus unidades operacionales sobre la base de transmitir no variables globales sino los cambios que se efectúan en dichas variables en el proceso de cálculo.

Esto implica que los algoritmos de cálculo que desarrollan sus unidades se tengan que expresar, necesariamente, en forma incremental. Como punto de partida analicemos muy sucintamente el diseño de MADDIDA, poniendo de manifiesto, tanto sus innovaciones como los puntos débiles que presentaba.

La unidad básica del ADD, y en particular del MADDIDA, es el integrador. Este aceptaba como entrada dos trenes de impulsos, de velocidades dx/dt y dy/dt y daba a su salida un tren de impulsos dz/dt, dado por la expresión (5.1).

$$\frac{dz}{dt} = Cy \frac{dx}{dt} \qquad (5.1)$$

donde C es una constante.

La integral definida de (5.1) da:

$$z(t) = z(t_o) + C \int_{t_o}^{t} y(\tau) \frac{dx}{d\tau} \, d\tau \qquad (5.2)$$

Como la evaluación de esta integral se iba a efectuar sobre un dispositivo digital, se aproxima (5.2) por un proceso de sumación, expresado en (5.3):

$$z(t) \simeq z(t_o) + C \sum_{i=0}^{n} y(t_i) \, \Delta x(t_i) \qquad (5.3)$$

donde $\Delta x(t_i) \quad x(t_{i+1}) - x(t_i)$.

La primera innovación de MADDIDA fue formular el algoritmo de integración en forma incremental. Utilizando la integración simple de Euler (aproximación rectangular), la función incremento resulta:

$$\Delta z_n = (z_{n+1} - z_n) = y_n \, \Delta x_n \qquad (5.4)$$

La generación de cada incremento lleva consigo dos acciones:

a) La acumulación de los incrementos Δy para formar y_n .

b) La multiplicación de y_n por el incremento de Δx_n en la variable independiente.

Si y_n y Δx_n fueran variables globales almacenadas cada una de ellas en una palabra de memoria, como sucedería en la implantación de este algoritmo en una calculadora digital de propósito general, el producto necesitaría una palabra de longitud doble a la de sus factores. Como el incremento que se transfiere se da en simple precisión, el producto se redondea para producir el resultado con dicha precisión, lo que introduce inevitablemente un error de redondeo. Debido a que los incrementos Δz_n que se generan tendrán que acumularse en alguna parte, se observó que se podía conseguir un aumento en la precisión, si en lugar de redondear el producto, la segunda parte del mismo se retenía y se iba acumulando.

La segunda innovación que introduce MADDIDA fue eliminar la necesidad de efectuar la operación aritmética de multiplicación en el cálculo de $y_n \Delta x_n$, al imponer que el valor de la entrada Δx podía ser +1 ó -1. El uso de este esquema de

codificación resultaba en una codificación de tipo binario entre integradores y la reducción de la operación de multiplicar a una simple suma o resta.

El desarrollo del integrador se explica mejor haciendo referencia a la Figura 5.1. Este diagrama de bloques ilustra las características salientes de cómo fue estructurado el integrador de MADDIDA. Los valores incrementales del integrando eran acumulados en el registro Y a través de un circuito sumador/sustractor.

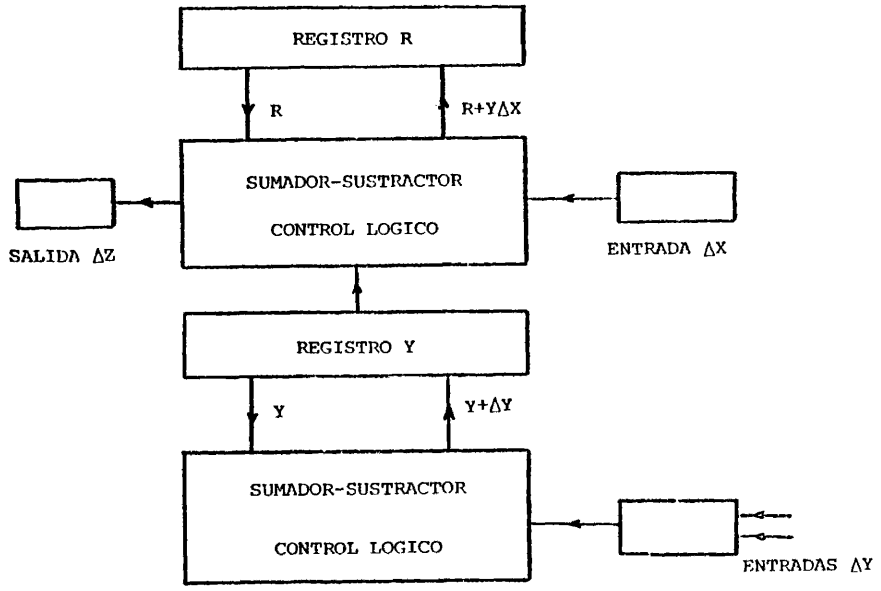


Figura 5.1: a) Diagrama de bloques del integrador MADDIDA

b) Representación esquemática.

Esta operación era seguida por la suma o resta del registro Y, con el registro R dependiendo del valor de ΔX. El registro R contiene el residuo o parte menos significativa del producto acumulado $Y.\Delta X$. Así pues, el integrador elemental se presenta bajo la forma siguiente: un sistema formado por dos registros Y y R, que reciben dos tipos de entradas incrementales:

ΔX llamada entrada primaria

ΔY llamada entrada secundaria,

dando, como resultado, una salida incremental ΔZ.

Sobre la base de este integrador,MADDIDA estaba organizado de forma serie, con una única unidad operacional de cálculo. Los registros Y y R se registraban sobre dos pistas de una memoria tipo tambor magnético. Una tercera pista se utilizaba para registrar los valores de los incrementos ΔZ. Finalmente, sobre una cuarta pista se registraba la información relativa a la interconexión de los integradores para resolver el problema.

Aunque MADDIDA representa "per se" un sistema con una concepción nueva, sufría de grandes desventajas. La primera limitación era la pérdida de significancia del contenido del registro Y debido, en parte, a la falta de un estado que detectase el cero en la entrada ΔY. A causa de la comunicación binaria entre integradores, una entrada nula para ΔY venía representada por una serie consecutiva de incrementos positivos y negativos. La segunda limitación provenía de que, por usar algoritmos de integración extremadamente simples, la solución numérica obtenida se iba deteriorando rápidamente y acumulando un error muy significativo frente a la solución verdadera.

Las especificaciones de MADDIDA eran las siguientes:

- Algoritmo de integración - Euler (rectangular)
- Aritmética - Tipo serie

- Comunicación entre integradores - Binaria
- Procesamiento - Secuencial
- Memoria - Tambor magnético
- Velocidad de iteración - 60 iteraciones/seg./integrador
- Número de integradores - 44
- Longitud de palabra - 28 bits + bit de signo.

Básicamente, pues, vemos que el ADD consiste, en esencia, en una colección de integradores digitales que se pueden interconectar entre sí para resolver problemas dinámicos. Intimamente ligado con el ADD está el problema de la programación de los mismos que consta de tres pasos:

1°) Formulación de un diagrama de interconexionado que describa la interrelación entre integradores para resolver un problema específico.

2°) Cambio de escala, es decir, efectuar una transformación lineal de las variables del problema en las variables máquina, de forma tal que éstas permanezcan dentro de los límites preestablecidos y no existiendo, por lo tanto, desbordamientos (overflow) en los contenidos de los registros Y. Este paso es necesario para cualquier máquina que opere con aritmética en coma fija.

3°) Codificar el problema de manera que sea introducido en la máquina para su ejecución. Este paso, obviamente, está ligado a las características particulares de cada ADD.

Un estudio detallado de estos puntos puede encontrarse en ([15])([16]).
Una vez analizado, desde un punto de vista básico, en qué consiste un ADD, se pueden sistematizar las posibles estructuras de los mismos (V. Dormido ([17])), atendiendo a las características que lo definen y que han dado lugar a una evolución hacia sistemas más elaborados y complejos. Estas características son:

a) Velocidad de la unidad aritmética
b) Forma en que son procesados los integradores

c) Forma de generación de los incrementos de salida.

d) Grado de resolución de los incrementos de salida.

e) Mecanismos de interconexión de los integradores.

En nuestro trabajo, estamos fundamentalmente interesados en la característica d).

Ya hemos visto que una de las limitaciones del MADDIDA era el sistema de comunicación binario entre sus integradores $\Delta Z\ell(-1, +1)$, así pues, la transición a un modo ternario de comunicación $\Delta Z\ell(-1, 0, +1)$ fue una extensión lógica y natural para aumentar la precisión del ADD. Como se necesitaban dos dígitos binarios para un sistema de comunicación ternario, la cantidad de memoria ΔZ requerida era doble que en el caso previo.

Sin embargo, tanto un esquema como el otro presentaban básicamente el inconveniente de que la anchura de banda de los integradores era necesariamente muy baja. Si, por ejemplo, se desea generar una onda sinusoidal con una resolución de una parte en 10^6, entonces, puesto que la velocidad máxima con que un integrador puede acumular es de 1 bit por ciclo de integración, el tiempo requerido para generar un ciclo completo de la sinusoide es:

$$T = \frac{2\pi \times 10^6}{m}$$

donde m representa la velocidad de iteración. Así pues, incluso si la velocidad de iteración es igual a 10^6 por segundo, la sinusoide de frecuencia más elevada que con la máxima resolución se puede generar posee un periodo de 2π . Esto es, por supuesto, muy inferior al ancho de banda de una calculadora analógica y fue, en gran medida, una de las causas que motivaron que los ADD no ganasen más adeptos como AD de uso general.

Con el fin de mejorar la respuesta en frecuencia de los integradores

McGhee y Nilsen ([18]) han propuesto un ADD de resolución extendida en el que la longitud del incremento de salida, ΔZ, es un parámetro libre.

Recientemente, Elshoff y Hulina ([19]) extienden el concepto de los integradores digitales de manera que trabajan en coma flotante evitando así los problemas de cambio de escala que sus predecesores llevaban consigo.

Nuestra línea de trabajo ha sido precisamente modificar el mecanismo de transmisión de incrementos, al transmitir no las ΔZ sino los cambios incrementales que se producen en dichos incrementos ($\Delta^2 Z$). El punto de partida fué la observación de que en la resolución de un problema, si la velocidad de muestreo para los integradores digitales es rápida,comparada con las velocidades de cambio de los contenidos de los registros Y, entonces, el cambio de ΔZ entre iteraciones es únicamente de cero o una unidad. Expresado en forma de diferencias: $\Delta^2 Z_i = \Delta Z_i - \Delta Z_{i-1}$, estaba limitado en valor a ($\pm$ 1, 0) unidades.

Este argumento se puede aplicar tanto si el sistema de transferencia entre integradores es del tipo ternario, como si es de resolución extendida,aunque es evidente que los pequeños cambios que puedan haber en el registro Y, se reflejaran tanto mas pronto cuanto mayor sea la resolución de ΔZ.

Una excepción a este argumento se puede producir durante las primeras iteraciones si se aplica por ejemplo, como entrada al sistema, un salto de posición, lo cual puede resultar en grandes variaciones de ΔZ, que a su vez se reflejarán en variaciones, ya no tan pequeñas, de $\Delta^2 Z$.

Con el fin de minimizar en lo posible este efecto adverso, se ha adoptado como medida de compromiso un sistema quinario de transmisión ($\Delta^2 Z \in \{\pm 2, \pm 1, 0\}$). En cualquier caso,no obstante, se puede dotar al integrador de un sistema de interrupción de forma que cuando se superen estos márgenes, se utilicen tantos ciclos de integración como sean necesarios, sin embargo, esto complicaria excesivamente la estructura del integrador.

5.2 PRINCIPIOS BASICOS DE INTEGRADORES DIGITALES CON TRANSMISION DE LAS DIFERENCIAS DE SEGUNDO ORDEN (Δ^2).

Con el fin de mantener el compromiso de precisión-coste en límites razonables, se ha escogido el método de integración de Adams-Bashfort de 2° orden, como el esquema básico de integración, sin embargo, la metodología que vamos a seguir tiene carácter general y se puede aplicar igualmente a cualquier otro algoritmo de integración.

Al mismo tiempo, el integrador básico tiene una estructura muy simplificada y que no difiere en gran medida de las soluciones previamente adoptadas. La originalidad de la solución que se propone estriba en que la comunicación entre integradores no se realiza en base a transmitir los incrementos (Δ) de las variables, sino las diferencias de 2° orden Δ^2 de las mismas. En principio, esto implicaría que cualquier variable (por ejemplo, Y) se tendría que generar para cualquier elemento máquina a dos niveles de acumulación. Ver Figura 5.2.

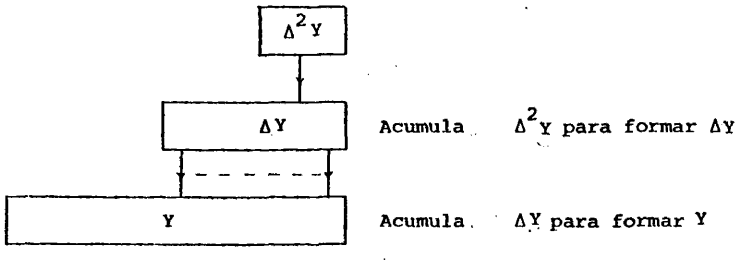


Figura.- 5.2 Integrador básico para diferencias de 2° orden

En forma de transferencia de registro esto es:

$$\Delta y_i = \Delta y_{i-1} + \Delta^2 y_i$$

$$y_i = y_{i-1} + \Delta y_i = y_{i-1} + \Delta y_{i-1} + \Delta^2 y_i$$

El proceso inverso, esto es, la generación de diferencias de 2° orden de la información procesada en un elemento máquina, es también simple. Por ejen-

plo, un integrador que efectúe el proceso de integración con respecto al tiempo, en su forma más simple consistiría de la estructura de registros que se muestra en la Figura 5.3

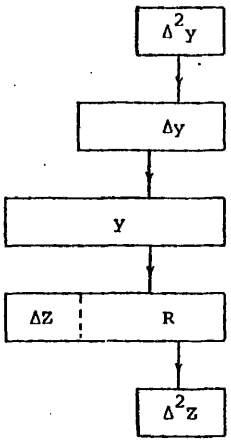


Figura.- 5.3 Integrador temporal

Cuyas ecuaciones de transferencias de registros son:

$$\Delta y_i = \Delta y_{i-1} + \Delta^2 y_i$$
$$y_i = y_{i-1} + \Delta y_i$$
$$\Delta Z_i + R_i = R_{i-1} + y_i$$
$$\Delta^2 Z_i = \Delta Z_i - \Delta Z_{i-1}$$

Obviamente, en el proceso de sumar y_i a R_{i-1} un cambio en ΔZ puede ocurrir únicamente si el bit más significativo de R_i transmite un arrastre a ΔZ_i en orden a incrementar ΔZ en 1. No obstante y_i puede también diferir de y_{i-1} en la misma posición que el bit menos significativo de ΔZ a través de la ecuación:

$$y_i = y_{i-1} + \Delta y_i$$

Si $|\Delta Y_{max}| < R_{max}$ esta diferencia queda restringida a ± 1.

Así pues, si tomamos en cuenta ambos cambios, tanto en Y como los arrastres de R, resulta que los cambios en ΔZ están limitados como máximo a ± 2, es decir, el sistema de transferencia de información entre integradores se reduce a $\Delta^2 \in \pm 1, \pm 2, 0$.

La lógica para generar $\Delta^2 Z$ tiene ciertas implicaciones que no se hacen obvias de la discusión anterior.

a) Como realmente la información que se va a comunicar entre distintos elementos es Δ^2 y no Δ, se ve que no es necesario que ΔZ esté contenido en ningún registro. Realmente, aunque ΔZ no exista de forma explícita, $\Delta^2 Z$ se puede obtener de cambios en otros registros y/o sumadores en la iteración previa.

b) Teniendo en cuenta la observación anterior, la parte del registro Y que tiene el mismo peso que ΔZ, se hace por lo tanto innecesaria. Por lo tanto, el registro Y se puede truncar a una longitud igual a la del registro R.

c) El procesamiento de $\Delta^2 Z$ a partir de los arrastres (overflow) de los registros se complica, ya que los contenidos de dichos registros representan números con signo. En particular si se utiliza la representación en complemento a dos, el rebose de un sumador debido a un operando negativo no dará lugar a un cambio en Δ^2, mientras que la ausencia de arrastre un decremento de 1 en el resultado. Como se verá, desde el punto de vista lógico, la complicación extra que se añade es mínima.

d) Los métodos basados en las diferencias de 2° orden presentan un considerable ahorro y simplificación a la hora de efectuar interconexiones entre diferentes unidades.

En efecto, en una máquina típica, como ya hemos visto $\Delta^2 \in \{\pm 2, \pm 1, 0\}$, lo cual se puede realizar mediante un sistema quinario de transferencia de información que vendría soportado sobre tres hilos nada más. Esto se compara muy favorablemente con respecto a los integradores de resolución extendida de McGhee y

Nilsen [18], en los que las transmisiones efectuadas eran las ΔZ globalmente.

e) Los integradores digitales de resolución extendida con transmisión de las diferencias de 2° orden, evitan en principio las multiplicaciones que se producen cuando lo que se transmite son las diferencias de primer orden (supuesto que el ID es de resolución extendida). Esto se debe a que los productos de una iteración dada se pueden construir del valor de dicho producto en la iteración previa más una serie de incrementos deducidos a partir de las diferencias de 2° orden.

Por ejemplo, si se desea calcular $K \times Y_i$ sería necesario un multiplicador (supuesto que las longitudes de los registros K e Y fueran mayores que 1, $\ell(K) > 1$ y $\ell(Y) > 1$). El punto de vista alternativo, basado en las diferencias de 2° orden nos daría:

$$K\Delta Y_i = K(\Delta Y_{i-1} + \Delta^2 Y_i) = K\Delta Y_{i-1} + K\Delta^2 Y_i$$
$$K\,Y_i = K(Y_{i-1} + \Delta Y_i) = K\,Y_{i-1} + K\Delta Y_i$$

$K\Delta^2 Y_i$ es un proceso de multiplicación trivial, ya que uno de los factores es una diferencia de 2° orden, por lo tanto, $K\Delta Y_i$ se puede generar mediante una simple suma y, por tanto, KY_i se obtiene, asimismo, únicamente mediante suma.

Utilizando una técnica análoga se puede formar $Y_i \Delta X_i$ al reducir ambos factores Y y ΔX a sus diferencias de 2° orden y construir el producto mediante la acumulación de los dos conjuntos de variable

$$Y_i = Y_{i-1} + \Delta Y_{i-1} + \Delta^2 Y_i$$
$$\Delta X_i = \Delta X_{i-1} + \Delta^2 X_i$$

de donde:

$$Y_i \Delta X_i = (Y_{i-1} + \Delta Y_{i-1} + \Delta^2 Y_i)(\Delta X_{i-1} + \Delta^2 X_i) =$$
$$= \underbrace{Y_{i-1}\Delta X_{i-1} + \Delta Y_{i-1}\Delta X_{i-1}}_{\text{valores previos}} +$$

$$+ Y_{i-1}\Delta^2 X_i + \Delta Y_{i-1}\Delta^2 X_i + \Delta^2 Y_i \Delta^2 X_i + \Delta X_{i-1}\Delta^2 Y_i \quad (*)$$

f) El sistema de diferencias de 2° orden permite utilizar grandes valores para las diferencias de primer orden (gran resolución) sin tener que transmitir éstas entre los distintos elementos operacionales. Para poder efectuarlo, se coloca una restricción sobre las diferencias de 2° orden. Así pues, aunque las pendientes de las variables pueden tomar valores grandes, no resulta lo mismo sobre la variación de dichas pendientes.

g) Una cuestión que puede surgir es por qué no utilizar en las transferencias entre integradores, diferencias de orden superior a dos. Es claro que esto implicaría de principio más acumulaciones para retornar a las variables originales, pero esto es relativamente poco importante, especialmente si se piensa en una fabricación de estos elementos de cálculo mediante técnicas LSI.

La razón para parar en las diferencias de 2° orden se pueden comprender por el siguiente razonamiento.

Consideremos que una variable está creciendo lentamente en magnitud, en estas condiciones si se representan las distintas diferencias se obtiene el siguiente resultado (ver Figura 5.4).

Se puede ver que un salto de posición unitario en la amplitud de Y origina que las diferencias sucesivas de Y cambien cada vez de manera más acusada. Hasta las diferencias de 2° orden, las amplitudes de dichas diferencias se comportan de una manera regular, pero a partir de ellas las desviaciones se van haciendo más grandes, de manera que los registros necesitados para almacenar tales variables se harían cada vez mayores en lugar de menores, tal como era nuestro objetivo inicial.

(*) productos en que al menos uno de los factores representa una diferencia de 2° orden.

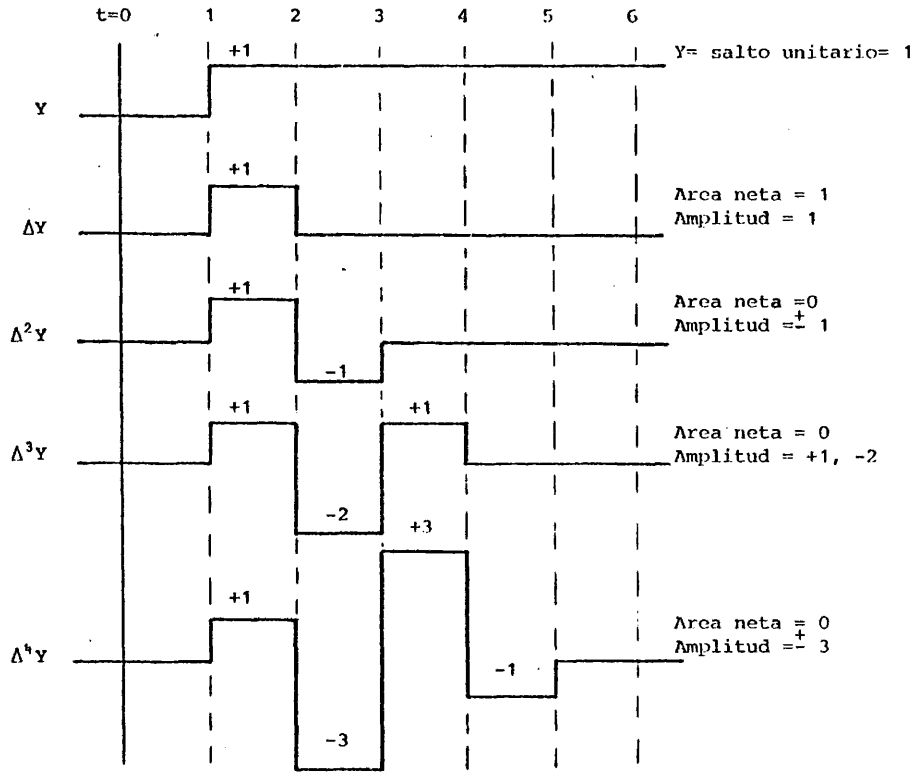


Figura 5.4.- Comportamiento de las diferencias de orden superior a un salto de posición de la variable Y.

Partiendo de estas ideas, en este capítulo se van a desarrollar los siguientes puntos:

En primer lugar, las nociones de base de un integrador digital de resolución extendida (IDRE) con el desarrollo detallado de su diseño lógico. Las características de este integrador son totalmente originales, en el conocimiento del autor. A continuación y con el fin de dotarle de una mayor flexibilidad operativa se estudia la inclusión en el mismo de un registro que se asimile a un coeficiente potenciométrico. La estructura del multiplicador que se desarrolla es de una gran sencillez y su precisión, aunque moderada, utiliza las ventajas de retención de residuo, normalmente utilizada en los procesos de integración pero

nunca en los de multiplicación. Esta innovación no complica excesivamente, como veremos, la estructura del integrador.

Los integradores desarrollados hasta este punto son válidos si la integral se efectúa con respecto al tiempo; así pues, con el fin de eliminar esta ligadura, se desarrolla un integrador generalizado con respecto a cualquier variable, que incluye a los anteriores como casos particulares.

Finalmente y con el fin de poner de manifiesto los resultados obtenidos, se muestra verificación experimental de los mismos en la resolución de determinadas ecuaciones diferenciales.

5.3 ESTRUCTURA BASICA DEL INTEGRADOR DIGITAL DE RESOLUCION EXTENDIDA (IDRE) QUE SE PROPONE.

Una estructura típica de un integrador digital (ID), del tipo más común encontrado en un ADD, es el de la Figura 5.5.

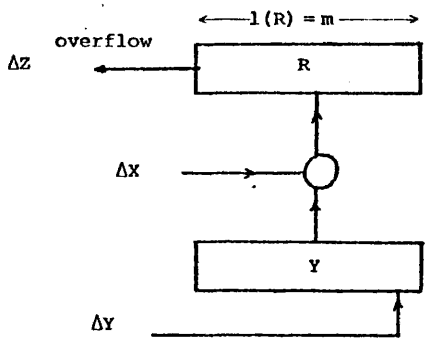


Fig. 5.5: Integrador digital convencional

donde $\Delta Z \in \{-1,0,1\}$, supuesta una transferencia ternaria entre integradores.

El ID efectúa la operación $z = \int ydx$ incrementalmente, de acuerdo con un algoritmo de integración previamente escogido. Así, por ejemplo, como casos

más usuales, se tienen:

$$\Delta Z_{n+1} \cong Y_n . \Delta X_n \quad (Euler)$$

$$\Delta Z_{n+1} \cong \{Y_n + \Delta Y_n\} . \Delta X_n \quad (Euler)$$

$$\Delta Z_{n+1} \cong \{Y_n + 1/2\ \Delta Y_n\} . \Delta X_n \quad (Trapezoidal)$$

$$\Delta Z_{n+1} \cong \{Y_{n+1} + 1/2\ \Delta Y_n\} . \Delta X_n \quad (Adams\text{-}Bashfort)$$

donde $n(0,1,2,...)$ es el instante de tiempo discretizado.

Así, si se escoge el método de integración de Adams-Bashfort, las ecuaciones de transferencias de registros serían:

Paso 1: $Y_{n+1} = Y_n + \Delta Y_n$; $R^*_{n+1} + \Delta Z^*_{n+1} = R_n + 1/2\ \Delta Y_n . \Delta X_n$

Paso 2: $\Delta Z_{n+1} + R_{n+1} = R^*_{n+1} + \Delta Z^*_{n+1} + Y_{n+1} . \Delta X_n$

Es decir, el método de Adams-Bashfort (de 2° orden) suministra una corrección trapezoidal retrospectiva a una predicción de integrando constante (V. Fig. 5.6). Sea $\Delta X = \Delta T$.

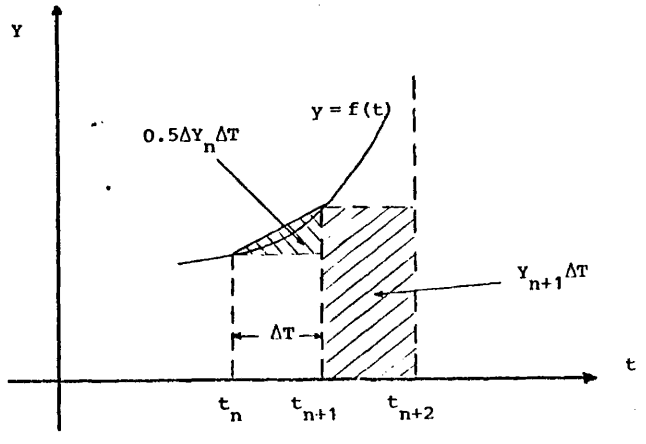


Figura 5.6: Integración de $y = f(t)$ con corrección trapezoidal retrospectiva.

BIBLIOTECA

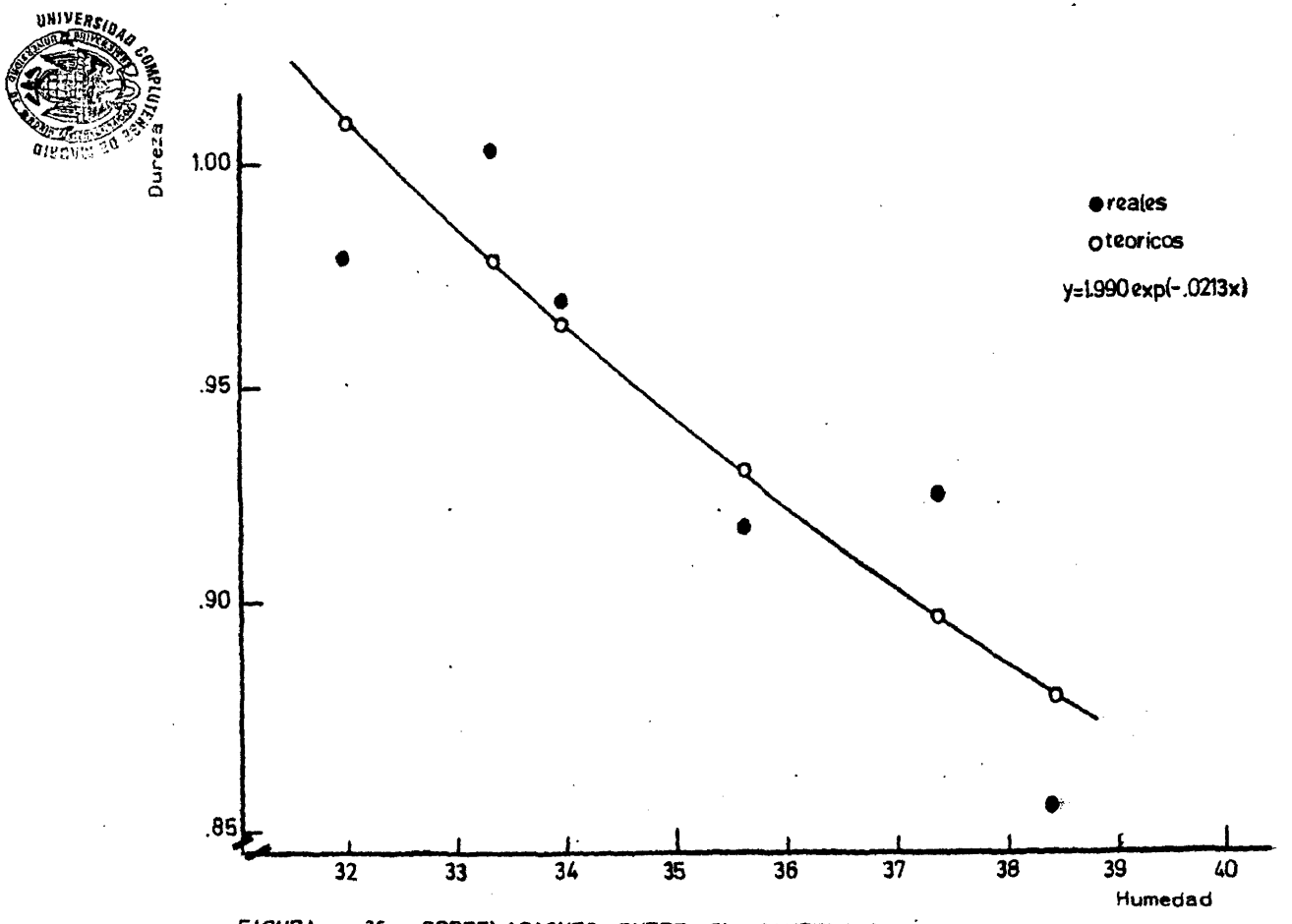


FIGURA.- 36. CORRELACIONES ENTRE EL CONTENIDO EN HUMEDAD Y LA DUREZA A LO LARGO DE LA MADURACION GLOBAL DE TODOS LOS LOTES.

gistros R_B e Y_B.

La Fig. 5.8 muestra como ejemplo el diagrama de conexión de dos IDRE para generar las funciones sinusoidales $sen x$ y $cos x$.

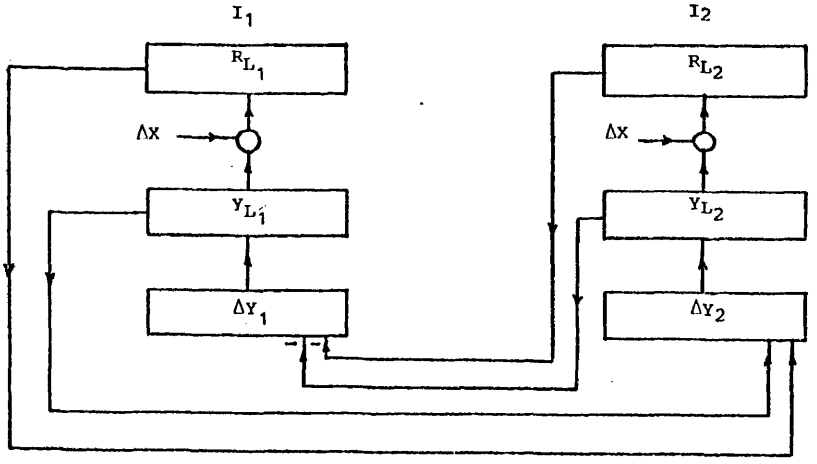


Figura 5.8: Generación $sen\,x$ y $cos\,x$.

El ID de la Fig. 5.5 corresponde a uno cuya entrada ΔX toma únicamente los valores +1, 0 ó -1. Sin embargo, en el caso general ΔX será la salida ΔZ de otro integrador que, como sabemos, es un número de m bits. En este caso, se necesita una estructura más compleja, tal como se esquematiza en la Fig. 5.9. Se compone de cuatro registros de $2m$ bits cada uno, R, $\Delta Y.\Delta X$, $Y.\Delta X$ e Y y cuatro registros de m bits ΔY, $\Delta^2 Y$, ΔX y $\Delta^2 X$.

En cada periodo de iteración, estos registros contienen los números R_n, $\Delta Y_{n-1}.\Delta X_{n-1}$, $Y_n.\Delta X_{n-1}$, Y_n, ΔY_{n-1}, $\Delta Y_n - \Delta Y_{n-1}$, ΔX_{n-1} y $\Delta X_n - \Delta X_{n-1}$, respectivamente.

Utilizando los contenidos de los registros que se representan, una iteración del algoritmo de integración de Adams-Bashfort se puede realizar efectuando, en el orden que se indica, las ecuaciones siguientes:

$$\Delta Y_n \cdot \Delta X_n = \underline{\Delta Y_{n-1} \cdot \Delta X_{n-1}} + \underline{\Delta X_{n-1}\{\Delta Y_n - \Delta Y_{n-1}\}} + \\ + \underline{\Delta Y_{n-1} \cdot \{\Delta X_n - \Delta X_{n-1}\}} + \{\underline{\Delta Y_n - \Delta Y_{n-1}}\} \cdot \{\underline{\Delta X_n - \Delta X_{n-1}}\} \qquad (5.5)$$

$$\{Y_n + \Delta Y_n\} \cdot \Delta X_n + 2^{2m} \cdot 0|Y \cdot \Delta X| = \underline{Y_n \cdot \Delta X_{n-1}} + \underline{Y_n} \cdot \{\underline{\Delta X_n - \Delta X_{n-1}}\} + \underline{\Delta Y_n \cdot \Delta X_n} \qquad (5.6)$$

$$R_{n+1} + 2^{2m} \cdot 0|R| = \underline{R_n} + \{\underline{Y_n + \Delta Y_n}\} \cdot \Delta X_n + 1/2 \ \underline{\Delta Y_n \cdot \Delta X_n} \qquad (5.7)$$

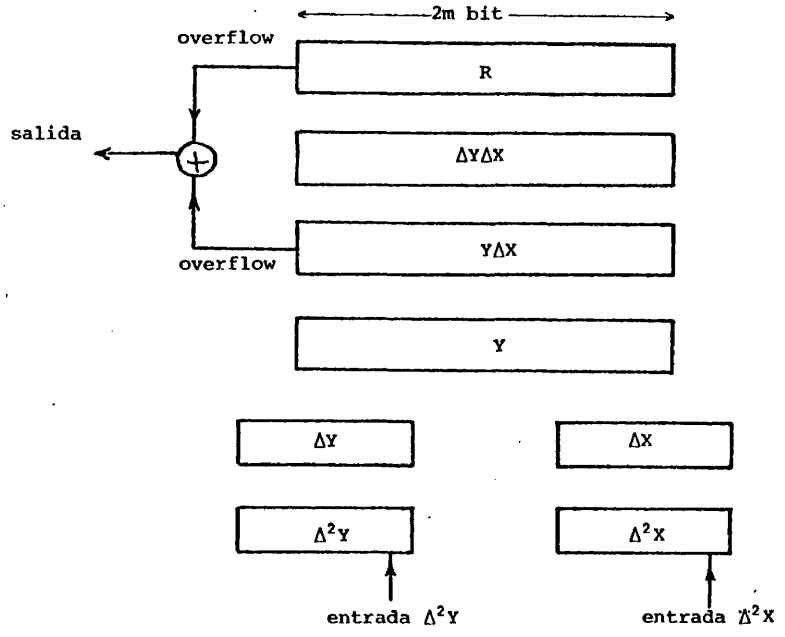


Figura 5.9: Estructura completa del IDRE generalizado

Los términos subrayados en las ecuaciones (5.5) a (5.7) están todos contenidos en los registros de la Fig. 5.9. Entonces, realizar una iteración (es decir, modificar los contenidos de los registros) se corresponde al miembro izquierdo de las ecuaciones anteriores y, por lo tanto, únicamente son necesarias suma y multiplicación por dos (realizada por un desplazamiento de un bit a la

izquierda). En estas expresiones $O[Y.\Delta X]$ y $O[R]$ son respectivamente overflows de los registros $Y.\Delta X$ y R y la suma de estos overflows es el incremento de la salida ΔZ del integrador (o lo que es lo mismo $\Delta^2 Z$) el cual se transmite a otros integradores.

5.3.1 Estructura concreta del IDRE con respecto al tiempo

En primer lugar vamos a considerar el caso de la integración digital con respecto a la variable tiempo, pues en éste, la estructura del IDRE se hace tan simple como la de los ID convencionales.

Consideramos la integral de una determinada función del tiempo $y(t)$.

$$z(t) = \int_{t_o}^{t} y(t)\, dt \; ; \qquad |y(t)| < 1 \tag{5.8}$$

Como ya se ha dicho, se va a utilizar el algoritmo de integración de Adams-Bashfort de segundo orden. En este caso:

$$\Delta z(t_n) \equiv z(t_{n+1}) - z(t_n) = \{y(t_n) + 1/2\ \Delta y(t_{n-1})\}\Delta t \tag{5-9}$$

donde $\Delta t = t_n - t_{n-1} \quad \forall\, n$

La Fig. 5.10 da la estructura de un ID convencional que efectúa el algoritmo de Adams-Bashfort de forma digital.

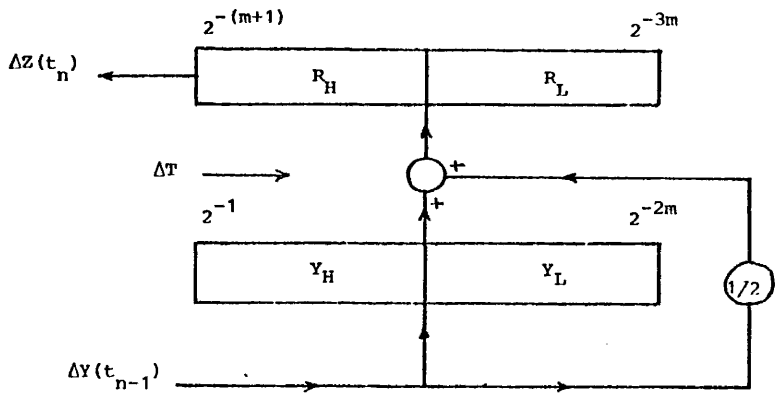


Figura 5.10: Integrador digital (Algoritmo de Adams-Bashfort de 2° orden)

La operación de integración se efectúa como sigue: Cuando el tiempo t aumenta en una cantidad constante Δt, que, en nuestro caso, es 2^{-m}, el lado derecho de la ecuación (5.9) se suma al registro R. Así pues, el lugar más significativo del registro R tiene un peso de $2^{-(m+1)}$. Debido a la suma, el registro R puede presentar o no rebose (overflow). Es claro que el valor de este rebose será -2^{-m}, 0, ó $+2^{-m}$ y que representa una aproximación a $\Delta Z(t_n)$ que es el lado izquierdo de la ecuación (5.9).

Definamos para el razonamiento que sigue un pulso unitario que tiene de peso 2^{-m}. Resumiendo lo anterior, el ID tiene una salida $\Delta Z(t_n)$ que es el pulso de rebose del lugar más significativo del registro R, una entrada ΔT que representa el incremento de tiempo y una entrada $\Delta Y(t_{n-1})$ que, en general, será la salida de otro integrador. Así pues, $\Delta Z(t_n)$, ΔT y $\Delta Y(t_{n-1})$ son cantidades trivaluadas, esto es su valor es de -1, 0, ó $+1$ pulso. Por tanto, una iteración de la operación de integración se realiza de la forma siguiente: cuando llega un pulso eléctrico al terminal de entrada, ΔT, la entrada $\Delta Y(t_{n-1})$ se suma al registro Y y produce $Y(t_n)$ e $Y(t_n) + 1/2\Delta Y(t_{n-1})$ se suma al registro R para producir la salida $\Delta Z(t_n)$.

Así, si la entrada ΔT por ejemplo es de mil pulsos por segundo, se ejecutarán mil iteraciones de integración en un segundo, de manera que el tiempo t en la integración procede a una velocidad de 1000×2^{-m} por segundo.

No obstante debe observarse que los registros de longitud $2m$, tal como se muestran en la Fig. 5.10 no son necesarios, ya que son suficientes registros R_H e Y_H de m bits. Pero la mitad de los registros R_L e Y_L se han añadido con vistas a poner de manifiesto las diferencias estructurales y funcionales entre los ID convencionales y el IDRE que se propone.

En el IDRE, la salida del integrador no es el pulso de rebose del registro R_H, sino que es el propio registro R_H, el cual es la suma de Y_H y el rebose

de la cifra más significativa de la mitad del registro R_L. Así la salida es un número de m bits que es una representación mucho más precisa de $\Delta Z(t_n)$ en la ecuación (5.9). Como han demostrado Mc Ghee y Nilsen ([18]), la introducción de una resolución extendida reduce considerablemente el error de redondeo en la integración digital.

Para resolver una ecuación diferencial el número de m bits, que es la salida de un integrador digital, debe de trasmitirse a la entrada ΔY de otro integrador. Pero esta transmisión es costosa en tiempo, si m es grande, lo cual resultará, en definitiva, tanto en una estructura más compleja del integrador digital, como en una disminución de su velocidad de cálculo. Esta desventaja se elimina en el IDRE de la manera siguiente:

La cantidad de cambio en cada iteración de la integración del número de m bits ΔZ que es la salida, es la suma del rebose del lugar más significativo del registro Y_L y el cambio del rebose del lugar más significativo del registro R_L. Representamos esta cantidad por $\Delta^2 Z$. Es claro que el valor de $\Delta^2 Z$ es, a lo más, un número de dos bits que tiene de peso 2^{-2m}, esto es $\Delta^2 Z$ es $-2x2^{-2m}$, $-1x2^{-2m}$, 0, $+1x2^{-2m}$ ó $+2x2^{-2m}$. Por tanto, la transmisión del número de m bits ΔZ se puede realizar efectivamente mediante la transmisión de $\Delta^2 Z$ a condición de que el registro Y_H se coloque como tercer registro del integrador digital (registro de entrada ΔY) al cual se transmitirá la salida $\Delta^2 Z$ del integrador que tenga conectado.

Usando este hecho, resulta para el IDRE, con respecto al tiempo, la estructura que se observa en la Fig. 5.11 y en la cual $\mathcal{D}$ representa un elemento que retarda una iteración o $\Delta t = 2^{-m}$ (FF-D).

Vamos a justificar matemáticamente la estructura de la Fig. 5.11.

Las ecuaciones algorítmicas que rigen el proceso de integración de Adams-Bashfort son:

$$Y_{i+1} = Y_i + \Delta Y_i \qquad (5.10)$$

$$R_{i+1} = R_i + Y_{i+1} + 1/2\ \Delta Y_i \qquad (5.11)$$

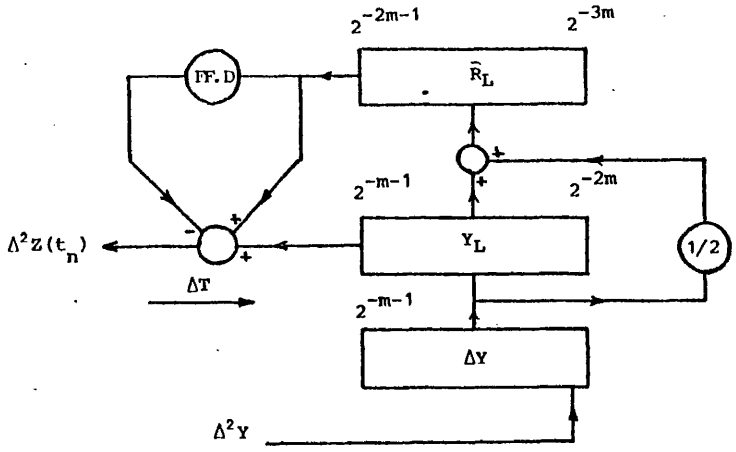


Figura 5.11: IDRE con respecto al tiempo: Estructura concreta

supuesto que no consideramos los reboses que se producen debido a la longitud finita de los registros. Si se tienen éstos en cuentan se tendrían que modificar de la forma siguiente:

$$O(Y_{i+1}) + Y^*_{i+1} = Y^*_i + \Delta Y^*_i \qquad (5.12)$$

$$O(R_{i+1}) + R^*_{i+1} = O(R_i) + R^*_i + Y^*_{i+1} + 1/2\ \Delta Y^*_i \qquad (5.13)$$

De (5.12) se deduce:

$$Y^*_{i+1} = -O(Y_{i+1}) + Y^*_i + \Delta Y^*_i \qquad (5.14)$$

Sustituyendo (5.14) en (5.13), se obtiene:

$$O(R_{i+1}) - O(R_i) + O(Y_{i+1}) + R^*_{i+1} = R^*_i + Y^*_i + 3/2\ \Delta Y^*_i \qquad (5.15)$$

Así pues, $\Delta^2 Z$ vendrá expresado por:

$$\Delta^2 Z = O(R_{i+1}) - O(R_i) + O(Y_{i+1}) \qquad (5.16)$$

tal como queda puesto de manifiesto en la Fig. 5.11.

5.3.1.1 Diseño lógico del IDRE con respecto al tiempo

El diseño lógico del IDRE con respecto al tiempo se ha efectuado utilizando circuitos integrados TTL de la serie 7.400 de media escala de integración. Nuestro ob-

jetivo ha sido, simplemente, el construir un prototipo que pusiese de manifiesto la simplicidad del diseño.

La representación utilizada para los números negativos ha sido la de complemento a dos. El IDRE opera bajo control de un reloj (y de determinadas señales que, posteriormente, se explicitarán), de forma que,a cada impulso, los registros de que consta almacenan los resultados de la iteración previa. Fundamentalmente se han utilizado dos tipos de circuitos integrados: el sumador de 4 bits (7483) y un registro de 4 bits (7495).

En el diseño lógico del IDRE conviene distinguir tres partes:

- Algoritmo de integración propiamente dicho.
- Circuito generador de $\Delta^2 Z$.
- Introducción de condiciones iniciales.

El algoritmo de integración se realiza de acuerdo con el esquema de la Figura 5.12 al que corresponden las ecuaciones siguientes:

$$\begin{aligned} \Delta Y' &= \Delta Y + \sum \Delta^2 Y \\ Y' &= Y + \Delta Y' \\ R' &= R + 1/2\ \Delta Y' \\ R'' &= R + Y \end{aligned} \qquad (5.17)$$

Los valores de ΔY, Y y R corresponden a los contenidos de los registros, como resultado de la iteración previa. Por otra parte, $\Delta Y'$, Y', R' y R'' son las salidas de los sumadores que componen el integrador. Al finalizar la iteración, se almacenan $\Delta Y'$, Y' y R'' en ΔY, Y y R respectivamente.

$$\begin{aligned} \Delta Y &= \Delta Y' \\ Y &= Y' \\ R &= R'' \end{aligned} \qquad (5.18)$$

Las expresiones (5.18) están marcadas en trazos discontinuos en la Fig. 5.12. Para una longitud de palabra del integrador de 4ℓ su coste será:

Número de registros (de 4 bits cada uno) = 3ℓ

Número de sumadores (de 4 bits cada uno) = 4ℓ

Así pues, el número de circuitos integrados será de 7ℓ, de forma que para una longitud de palabra de 8 bits ($\ell = 2$) se necesitan 14 unidades. En lo que sigue, se representa por $S(.)$ el bit de signo de $(.)$.

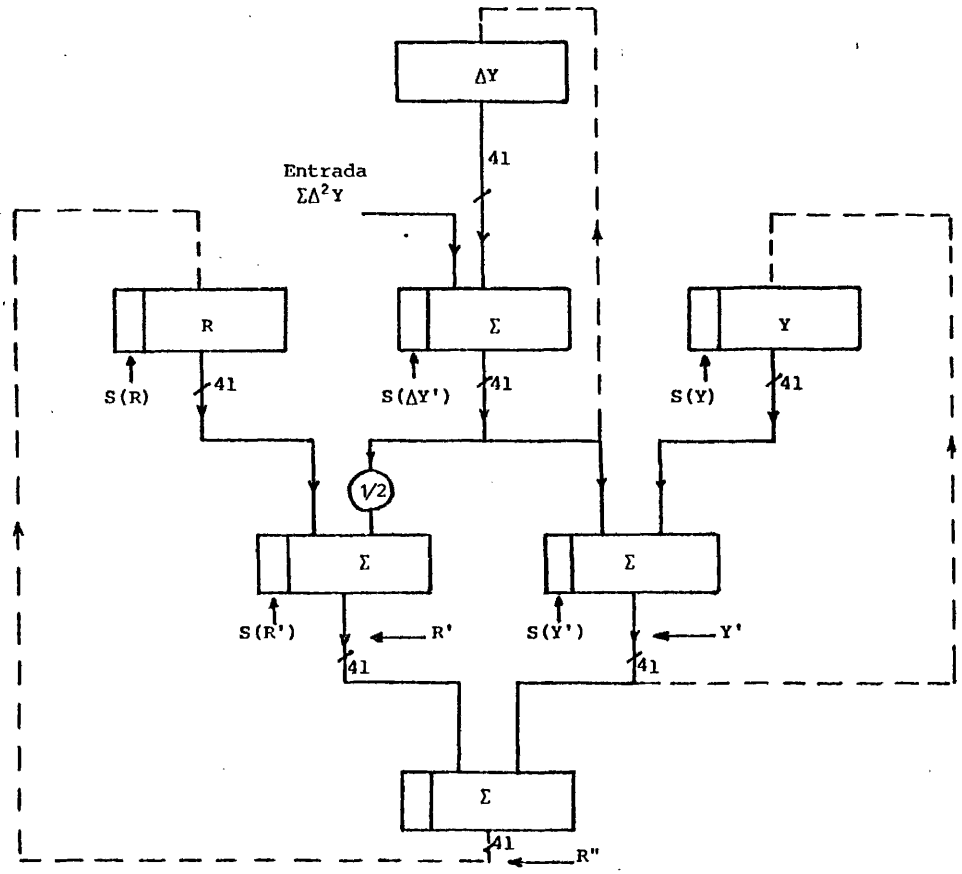


Figura 5.12: Diseño lógico del IDRE

El diagrama funcional del circuito generador de $\Delta^2 Z$, teniendo en cuenta (5.16) será el de la Fig. 5.13.

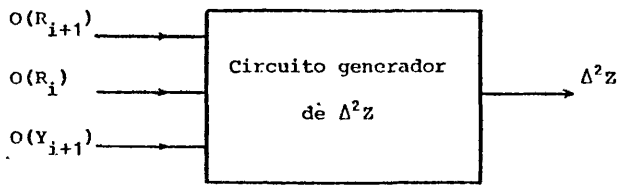


Figura 5.13: $\Delta^2 Z = O(R_{i+1}) - O(R_i) + O(Y_{i+1})$

La expresión (5.16) pone de manifiesto que se hace necesario almacenar el $O(R_i)$ de la iteración previa. Como $O(.) \in \{-1,0,1\}$ se precisan 2FF-D (D^+ y D^-) para guardar dicha información con el convenio siguiente:

$$O(R_i) = -1 \rightarrow D^+ = 0 \quad D^- = 1$$
$$O(R_i) = 0 \rightarrow D^+ = 0 \quad D^- = 0$$
$$O(R_i) = 1 \rightarrow D^+ = 1 \quad D^- = 0$$

De la Fig. 5.12, se sigue que en el IDRE los overflows se pueden producir a la salida de los sumadores Y', R' y R''. (Un overflow a la salida del sumador $\Delta Y'$ nunca puede ocurrir si el problema tiene bien ajustados los factores de escala). Designemos por $O^+(.)$ y $O^-(.)$ respectivamente la aparición de un overflow positivo y negativo (nunca pueden ocurrir los dos simultáneamente) a la salida del registro (.). Las expresiones lógicas para O^+ y O^- de los registros Y', R' y R'' son:

$$O^+(Y') = \overline{S(Y)}.S(\overline{\Delta Y'}).S(Y') \; ; \quad O^-(Y') = S(Y).S(\Delta Y').\overline{S(Y')}$$
$$O^+(R') = \overline{S(R)}.S(\overline{\Delta Y'}).S(R') \; ; \quad O^-(R') = S(R).S(\Delta Y').\overline{S(R')} \qquad (5.19)$$
$$O^+(R'') = \overline{S(R')}.\overline{S(Y')}.S(R''); \quad O^-(R'') = S(R').S(Y').\overline{S(R'')}$$

De esta forma, el valor de $\Delta^2 Z$ se puede expresar como una suma algebraica:

$$\Delta^2 Z = (O^+(Y') + O^+(R') + O^+(R'') + D^-) - (O^-(Y') + O^-(R') + O^-(R'') + D^+) \qquad (5.20)$$

(5.20) se puede calcular añadiendo al primer sumando del miembro de la derecha el complemento a dos del segundo sumando. En la Fig. 5.14 está representado el esquema lógico correspondiente y, como se ve, se hace uso extensivo del concepto de contador paralelo desarrollado en el capítulo 3.

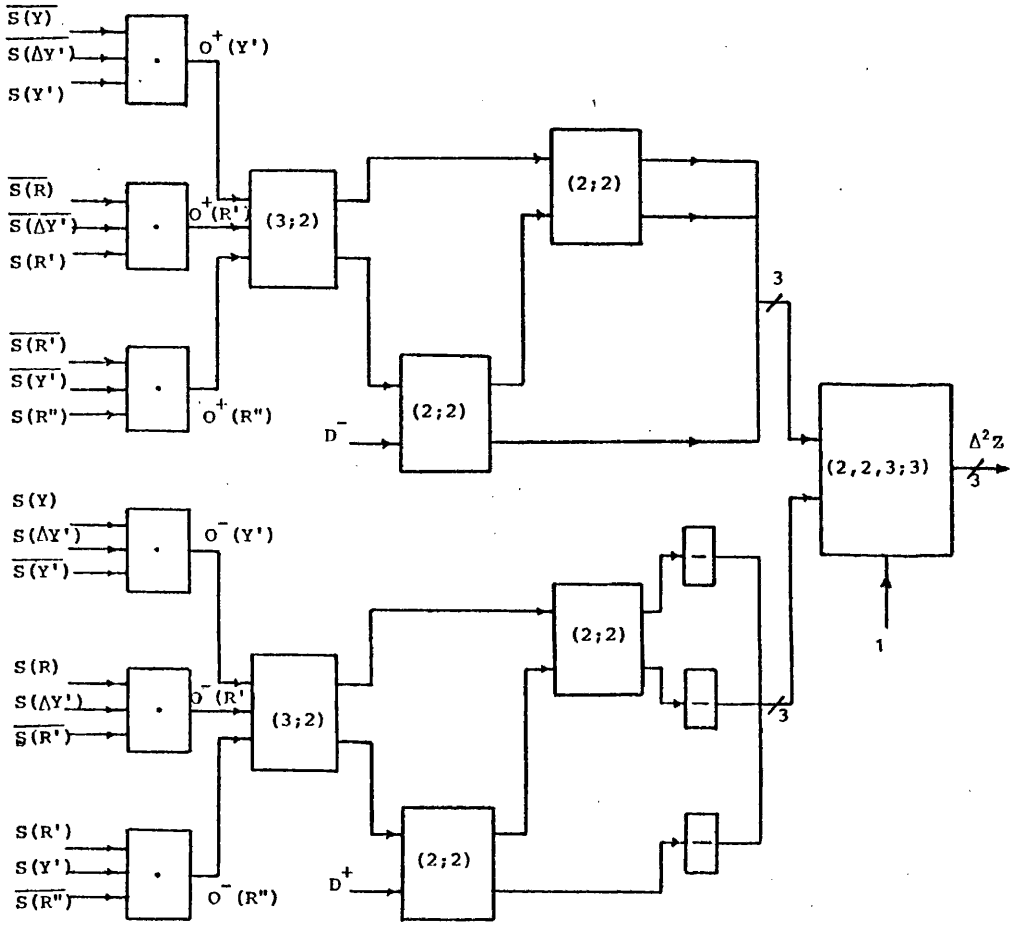


Figura 5.14: Diseño lógico de $\Delta^2 Z$

Quedan por sintetizar las entradas a los dos FF-D (D_E^+ y D_E^-) que corresponderán al valor de $O(R_i)$ en la próxima iteración.

$$D_E^+ = f_1(O^+(R'), O^-(R'), O^+(R''), O^-(R''))$$
$$D_E^- = f_2(O^+(R'), O^-(R'), O^+(R''), O^-(R'')) \qquad (5.21)$$

Es fácil deducir las siguientes expresiones lógicas:

$$D_E^+ = O^-(R').O^+(R'') + O^+(R').O^-(R'')$$
$$D_E^- = O^-(R').O^+(R'') + O^+(R').O^-(R'') \qquad (5.22)$$

a las que corresponde el diagrama lógico de la Fig. 5.15.

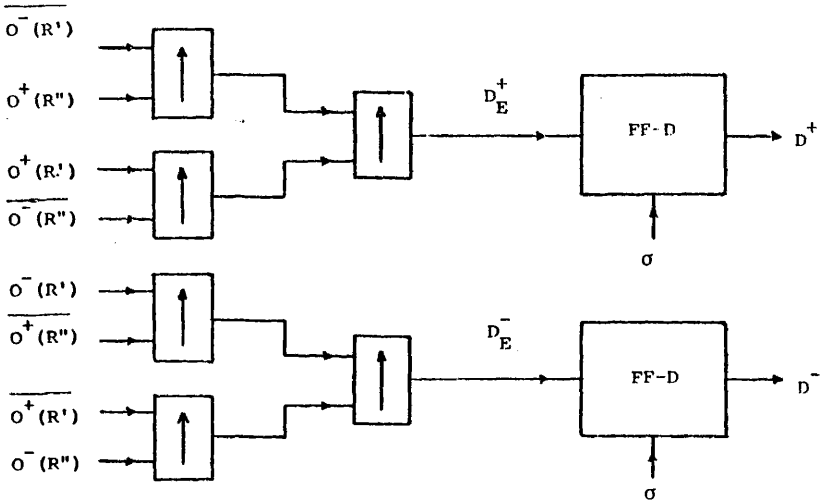
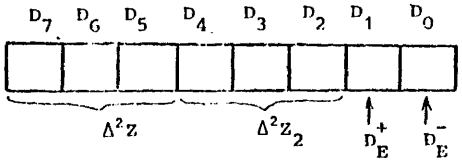


Figura 5.15: Síntesis de las entradas a los FF-D

El diseño lógico de $\Delta^2 Z$ se puede realizar de forma compacta utilizando una memoria ROM de 256x8 bits. Las ocho entradas de la memoria corresponden a D^+, D^-, $S(Y')$, $S(Y)$, $S(R)$, $S(Y')$, $S(R')$ y $S(R'')$, que nos determinan una dirección cuyo contenido almacena la siguiente información:



donde $\Delta^2 Z_2$ representa el complemento a dos de $\Delta^2 Z$, pues conviene tener como salida ambos valores, ya que,en determinados casos, en la interconexión entre integradores, se hace necesario un cambio de signo.Así, por ejemplo, en el caso particular de que $\mathcal{D}^+ = 0$, $\mathcal{D}^- = 0$, $S(\Delta Y') = 1$, $S(Y) = 0$, $S(R) = 1$, $S(Y') = 1$, $S(R') = 0$ y $S(R'') = 0$ en la dirección

$$\begin{array}{cccccccc} 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 = 44_{10} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \mathcal{D}^+ & \mathcal{D}^- & S(\Delta Y') & S(Y) & S(R) & S(Y') & S(R') & S(R'') \end{array}$$

se almacenaría

1	1	1	0	0	1	0	1

que corresponde a $\Delta^2 Z = -1, \Delta^2 Z_2 = 1$, $\mathcal{D}_E^+ = 0$ y $\mathcal{D}_E^- = 1$. Esta solución parece más atractiva que la primera porque, aparte de ser más compacta (utiliza un único chip) y por tanto más fiable, es más económica y posibilita una mayor velocidad operativa del IDRE, al requerir menor número de niveles lógicos. Para finalizar el diseño lógico del IDRE queda por resolver el problema de cómo introducir las condiciones iniciales en el mismo. La condición inicial para el integrador es el valor ΔY_0 que se debe almacenar antes de comenzar a operar en el registro ΔY. Los registros Y y R no necesitan ninguna condición inicial de arranque, si bien, si se desea minimizar los errores de redondeo, existen unos valores iniciales óptimos (ver Zugasti ([16])) que se pueden introducir por el mismo procedimiento que el que se indica para el registro ΔY; esto lleva consigo un encarecimiento del diseño.

En la Fig. 5.16, se indica un procedimiento de carga en paralelo de la condición inicial, que utiliza un multiplexor de dos entradas de datos (el 74157, que incorpora 4 MLUM (1)), la entrada de control al multiplexor fija los modos de operación de toma de condiciones iniciales (reset) y de ejecución (operate). Existe

una segunda señal de control no indicada sobre el esquema de la Fig. 5.12 y que actúa sobre los registros R, ΔY e Y impidiendo su carga y que fija el modo de retención (hold). Es decir, estando el IDRE en modo de ejecución, si se activa el modo de retención, los contenidos de los registros permanecen fijos en el valor alcanzado en ese momento.

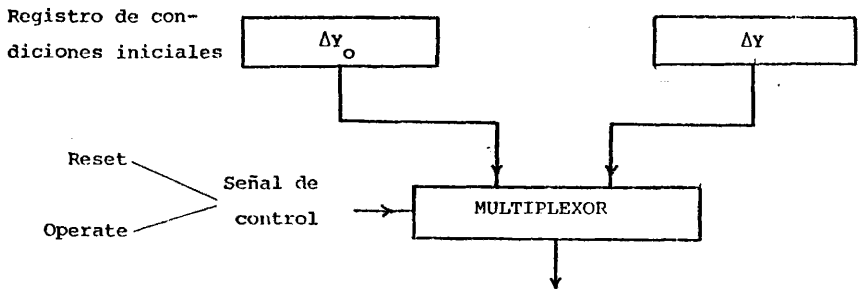


Figura 5.16: Introducción de condiciones iniciales en el registro ΔY.

En resumen, si representamos por C_1 el coste de un circuito integrado de 16 patillas y por C_2 el de uno de 24 patillas (se adopta, pues, el segundo procedimiento para generar el $\Delta^2 Z$), el coste total C del IDRE con respecto al tiempo y con una longitud de palabra de 4ℓ, será:

$$C = C_1(7\ell + \ell + 1) + C_2 = C_1(8\ell + 1) + C_2 \qquad (5.23)$$

donde $C_1 7\ell$=coste del diseño lógico del IDRE

$C_1\ell$ = coste de introducir condiciones iniciales en ΔY

C_1 = coste de los 2 FF-D necesarios para almacenar $O(R_i)$

El número de circuitos integrados utilizado es $N = 8\ell + 2$.

Para una longitud de palabra de 8 bits ($\ell = 2$) se necesitan 18 circuitos integrados.

5.3.2 Introducción de un registro P como coeficiente potenciométrico del IDRE.

En una calculadora analógica típica existen, normalmente, el doble de potenciómetros que de amplificadores operacionales, ya que su uso se hace muy frecuente en la resolución de problemas. En los ADD iniciales, el coste de cada ID permanecía muy bajo, ya que cada integrador se podía representar por dos o tres posiciones de un tambor magnético. La consecuencia inmediata de ello fue el uso de integradores como potenciómetros. Así, utilizándolos como indica la Fig. 5.17, la lógica de control se simplificaba porque no se tenía que diferenciar entre integradores genuinos y potenciómetros.

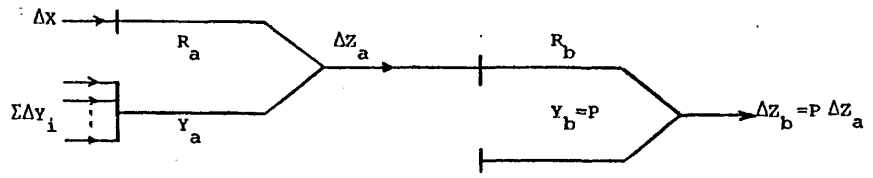


Figura 5.17: Uso de un integrador como un potenciómetro

Cuando la estructura del ID se va haciendo cada vez más compleja, esta solución inicial presenta dos claras desventajas:

a) Se están infrautilizando los recursos de un integrador, y, si tenemos en cuenta que el empleo de factores de escala diferente es muy normal, en último término, lo que estamos es degradando considerablemente la capacidad operativa del ADD.

b) Como consecuencia de lo anterior, se provoca una retardación de la solución, al aumentar considerablemente el número de retardos lógicos.

Parece, pues, natural intentar modificar la estructura del ID para que incorpore dentro del mismo un registro adicional P que pueda actuar como coeficiente potenciométrico.

Estas consideraciones nos han movido a modificar el IDRE desarrollado en la sección anterior, para dotarle de esta posibilidad de uso. En este caso, las ecuaciones algorítmicas que describen el proceso de integración de Adams-Bashfort

de segundo orden son:

$$\begin{aligned} \Delta Y_i &= \Delta Y_{i-1} + \Sigma\, \Delta^2 Y_i \\ Y_i &= Y_{i-1} + P\Delta Y_i \\ R_i &= R_{i-1} + P(Y_i + 0{,}5\Delta Y_i) \\ \Delta^2 Z_i &= O(Y_i) + O(R_i) - O(R_{i-1}) \end{aligned} \qquad (5.24)$$

A primera vista, un examen de las Ecs. (5.24) nos revelan que, a la hora de realizar físicamente el IDRE, aparecen dos multiplicaciones por el factor P, lo cual encarece considerablemente el coste del IDRE.

Con el fin de minimizar en lo posible este aumento de coste, desarrollamos en el capítulo anterior (sec. 4.5) un multiplicador paralelo de muy bajo coste para números fraccionarios que incorporamos ahora al IDRE. El algoritmo de integración (5.24) se realiza ahora de acuerdo con el esquema de la Fig. 5.18. Por lo que respecta al circuito generador de $\Delta^2 Z$ y a la introducción de condiciones iniciales, no sufren ninguna modificación y no serán tratados de nuevo.

Comparando las Figs. (5.18) y (5.12) se observa que la incorporación de un potenciómetro en el IDRE lleva consigo la introducción de dos multiplicadores. Teniendo en cuenta el coste de estas unidades (desarrollado en 4.5), el coste total del IDRE con respecto al tiempo, con potenciómetro incorporado y con una longitud de palabra de 4ℓ será:

$$C = C_1(18\ell + 1) + C_2 \qquad (5.25)$$

Comparando (5.23) y (5.25), se deduce que el coste se ha duplicado. Esto se debe al hecho de que el registro P está dotado de capacidad para retener el residuo que se va generando en la formación del factor de escala. Si se elimina esta posibilidad y se restringe el registro P directamente a una longitud de 4 bits, el coste total pasa a ser:

$$C' = C_1(16\ell + 3) + C_2 \qquad (5.26)$$

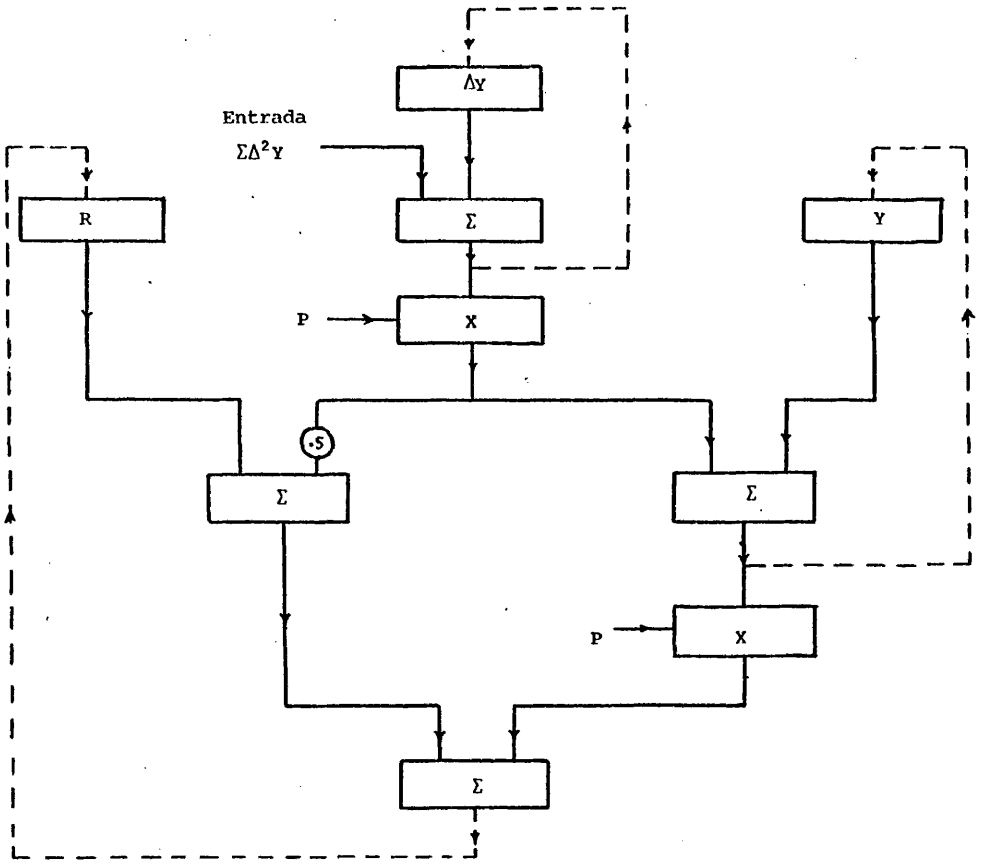


Figura 5.18: Diseño lógico del IDRE con coeficiente potenciométrico P

5.4 INTEGRADOR DIGITAL DE RESOLUCION EXTENDIDA GENERALIZADA (IDREG)

El IDRE desarrollado en las dos últimas secciones calcula la integral de cualquier función respecto a una variable independiente (normalmente el tiempo). En el caso de una integración de Stieltjes (como sucede si el problema no es lineal)

el IDRE no puede ser utilizado. Nuestro objetivo en esta sección es desarrollar un IDRE generalizado (IDREG) deduciendo una fórmula de integración numérica de Stieltjes.

Se desea calcular la integral

$$z = \int y(t)dx(t) \qquad (5.27)$$

donde t es la variable independiente que, en esta fórmula, será el tiempo.

Admitiendo que $x(t)$ es derivable respecto a t, se obtiene:

$$z = \int y(t)\dot{x}(t)dt \qquad (5.28)$$

Si empleamos la fórmula de integración de Adams-Bashfort de segundo orden que hemos venido utilizando, se deduce:

$$\Delta z_n \equiv z_{n+1} - z_n = \{\dot{z}_n + 1/2(\dot{z}_n - \dot{z}_{n-1})\}\Delta t = \{y_n\dot{x}_n + 1/2(y_n\dot{x}_n - y_{n-1}\dot{x}_{n-1})\}\Delta t \qquad (5.29)$$

El primer problema que se plantea es obtener un valor aproximado de $\dot{x}_n$. Para ello, aproximamos la función $x(t)$ por el polinomio de interpolación en diferencias regresivas de Newton de grado r, $L_r(t)$ que toma valores x_i en t_i ($i = n-r, \ldots, m-1, n$) ([20]).

$$L_r(t) = x_n\varphi_0(t) + \frac{\nabla x_n}{1!\Delta t}\varphi_1(t) + \ldots + \frac{\nabla^r x_n}{r!\Delta t^r}\varphi_r(t) \qquad (5.30)$$

donde :

$$\varphi_0(t) = 1$$

$$\varphi_k(t) = \prod_{j=n-k+1}^{n}(t-t_j) \qquad 1 < k < r$$

$$\nabla^0 x_n = x_n$$

$$\nabla^{j+1}x_n = \nabla^j x_n - \nabla^j x_{n-1} \qquad j = 0, 1, 2, \ldots.$$

El error $e(t)$ en la aproximación para $t \in [t_{n-r}, t_n]$ se puede evaluar como:

$$e(t) \equiv x(t) - L_r(t) = \frac{\prod_{j=n-r}^{n}(t-t_j)}{(r+1)!}x^{r+1}(\zeta) \qquad (5.31)$$

donde $x^{n+1}(\zeta)$ representa el valor de la derivada de orden $(n+1)$ de $x(t)$en un cierto instante de tiempo $\zeta\epsilon[t_{n-n}, t_n]$. Entonces, $\dot{x}(t)$ se puede aproximar derivando la Ec. (5.30). La fórmula de aproximación de $\dot{x}(t)$ en $t = t_n$ para $n = 1,2$ y 3 son respectivamente:

$$\dot{x}_n = \begin{cases} 1/\Delta t\ (x_n - x_{n-1}) & (n=1) \\ 1/2\Delta t\ (3x_n - 4x_{n-1} + x_{n-2}) & (n=2) \\ 1/6\Delta t\ (11x_n - 18x_{n-1} + 9x_{n-2} - 2x_{n-3}) & (n=3) \end{cases} \qquad (5\text{-}32)$$

Los errores para las aproximaciones (5.32) se pueden obtener de (5.31) derivando $e(t)$ y estos errores son:

$$\dot{e}(t_n) = \begin{cases} 1/2\ x^{(2)}(\zeta)\Delta t & (n=1) \\ 1/3\ x^{(3)}(\zeta)\Delta t^2 & (n=2) \\ 1/4\ x^{(4)}(\zeta)\Delta t^3 & (n=3) \end{cases} \qquad (5.33)$$

(5.33) demuestra que el error de aproximación de la fórmula de diferenciación numérica es proporcional a Δt^n. La siguiente consideración nos lleva a utilizar la fórmula de aproximación correspondiente a $n = 2$. Si $x(t) = t$ entonces (5.29) se convierte en la fórmula de ADAMS-BASHFORT de segundo orden que se ha estudiado previamente y el error de integración es del orden Δt^2. Al resolver ecuaciones diferenciales, el valor de y_n en (5.29) será, en general, la suma de las salidas de otros integradores. Por tanto, y_n tendrá también un error de Δt^2 . Por otra parte, los ID tienen una estructura tal que producen en sus operaciones errores de redondeo del orden de Δt^2. De estos hechos es pues razonable aproximar $\dot{x}_n$ en (5.29) con un error en la aproximación del mismo orden, de Δt^2.

Sustituyendo la segunda fórmula de diferenciación numérica de (5.32) para $\dot{x}_n$ en (5.29), obtenemos la siguiente expresión:

$$\Delta z_n = 3/4\ y_n(3\Delta x_{n-1} - \Delta x_{n-2}) - 1/4\ y_{n-1}(3\Delta x_{n-2} - \Delta x_{n-3}) \qquad (5.34)$$

La ecuación (5.34) se puede escribir como:

$$\Delta z_n = y_n \Delta \delta_{n-1} + 1/2 [y_n \Delta \delta_{n-1} - y_{n-1} \Delta \delta_{n-2}] = y_n \Delta \delta_{n-1} + 1/2 \Delta (y_n \Delta \delta_{n-1}) =$$
$$= y_n \Delta \delta_{n-1} + 1/2 (y_n \Delta^2 \delta_{n-1} + \Delta y_n \Delta \delta_{n-1}) \qquad (5.35)$$

De (5.35) se puede deducir la siguiente estructura del IDREG, que representa el diagrama de conexión entre registros (V. Fig. 5.19). Los pesos de los lugares más y menos significativos de cada registro se dan en la misma figura, donde $\Delta t = 2^{-m}$.

Las entradas al IDREG son $\Delta^2 Y_{n-1}$, $\Delta^2 X_{n-1}$ y $\Delta^2 X_{n-2}$ (retardada de $\Delta^2 X_{n-1}$) que son, a su vez, salidas de otros ID en la iteración precedente de operación. En el caso especial de $\Delta^2 X_{n-1} = \Delta^2 X_{n-2} = 0$ y $\Delta F_{n-1} = 1$ se efectúa la integración con respecto al tiempo y el IDREG se transforma en un IDRE.

En cada una de las iteraciones, estas entradas producen en el registro $Y_n \Delta F_{n-1}$ la parte menos significativa en representación binaria de la integral que es menor que 2^{-2m}. La parte más significativa de este valor ha sido almacenada de antemano como condición inicial en los registros ΔY_{n-1} de los ID a los cuales se transmiten los $\Delta^2 Z$ del integrador actual.

Asimismo en cada iteración , la salida $\Delta^2 Z_n$ del IDREG es la suma del overflow del lugar más significativo del registro $Y_n \Delta F_{n-1}$ y el overflow del registro R_n de forma análoga a como sucedía en el IDRE.

A pesar de la estructura un poco compleja que presenta el diagrama de bloques de la Fig. 5.19, todas las operaciones se realizan mediante sumadores y unos pocos multiplicadores (denotados por ⊗) muy simples, porque, en el peor de los casos, uno de los factores es un número de dos o tres bits.

En la próxima sección se presentan algunos resultados experimentales del IDREG en la resolución de determinadas ecuaciones diferenciales.

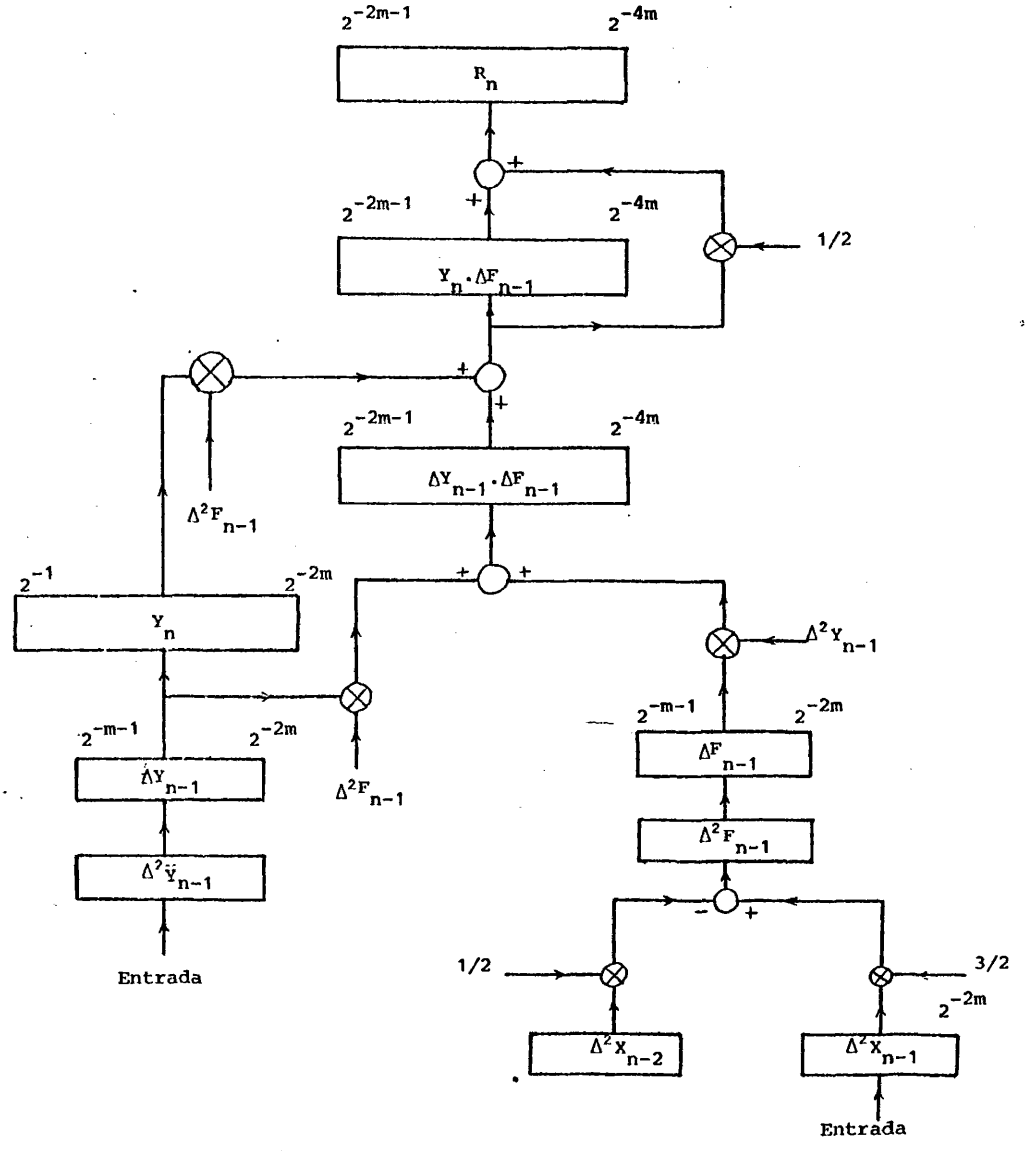


Figura 5.19: Diagrama de bloques del IDREG

5.5 ALGUNOS RESULTADOS EXPERIMENTALES.

El trabajo experimental llevado a cabo se puede dividir en dos partes perfectamente diferenciadas: en la primera se efectuó el desarrollo de dos prototipos de ID, siendo el primero un IDRE con respecto al tiempo, tal como se expuso en la sección 5.3.1.1. Posteriormente se modificó su estructura para permitir la introducción de un registro P como coeficiente potenciométrico del IDRE. Se tomó una longitud de palabra de 8 bits ($\ell = 7$) lo que hacía que el número de circuitos integrados envueltos en el diseño fuese pequeño.

Los resultados experimentales en la resolución de una Ecuación Diferencial lineal de coeficientes constantes de primer orden se dan en las Figs. 5.20, en el caso del IDRE con respecto al tiempo, y en la 5.21, cuando se introduce el registro P como coeficiente potenciométrico.

Como nuestro objetivo final no era la construcción de un ADD sino simplemente demostrar la posibilidad de realización experimental de unos prototipos, no seguimos nuestro estudio por esta línea. En su lugar (y esta es la segunda parte) se pasaron a efectuar comprobaciones por simulación en una calculadora digital de propósito general (el sistema utilizado ha sido un HP 1000). En el caso del IDREG, los únicos resultados obtenidos han sido por simulación.

Las pruebas fueron extensivas y aquí solamente se presentan algunas de las mismas. Por lo que se refiere al IDRE con respecto al tiempo, en lo que sigue lo representaremos simbólicamente por el diagrama de la Fig. 5.22

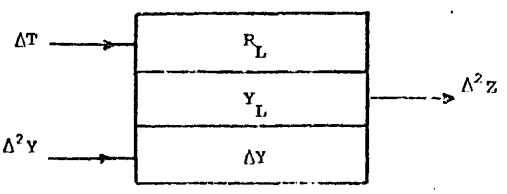


Fig. 5.22: Representación diagramática del IDRE con respecto al tiempo

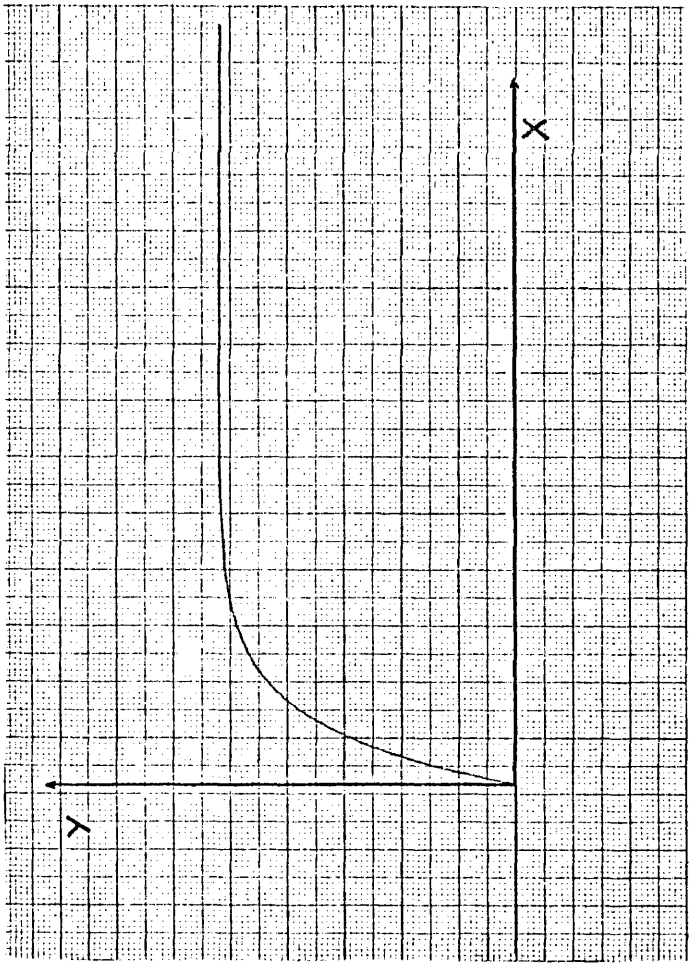


Fig. 5.20: $y' = -y \quad y(0) = y_0$

Solución con un IDRE con respecto al tiempo

X = .25 seg/cm

Y = 1 volt/cm

σ = 1000 hertz

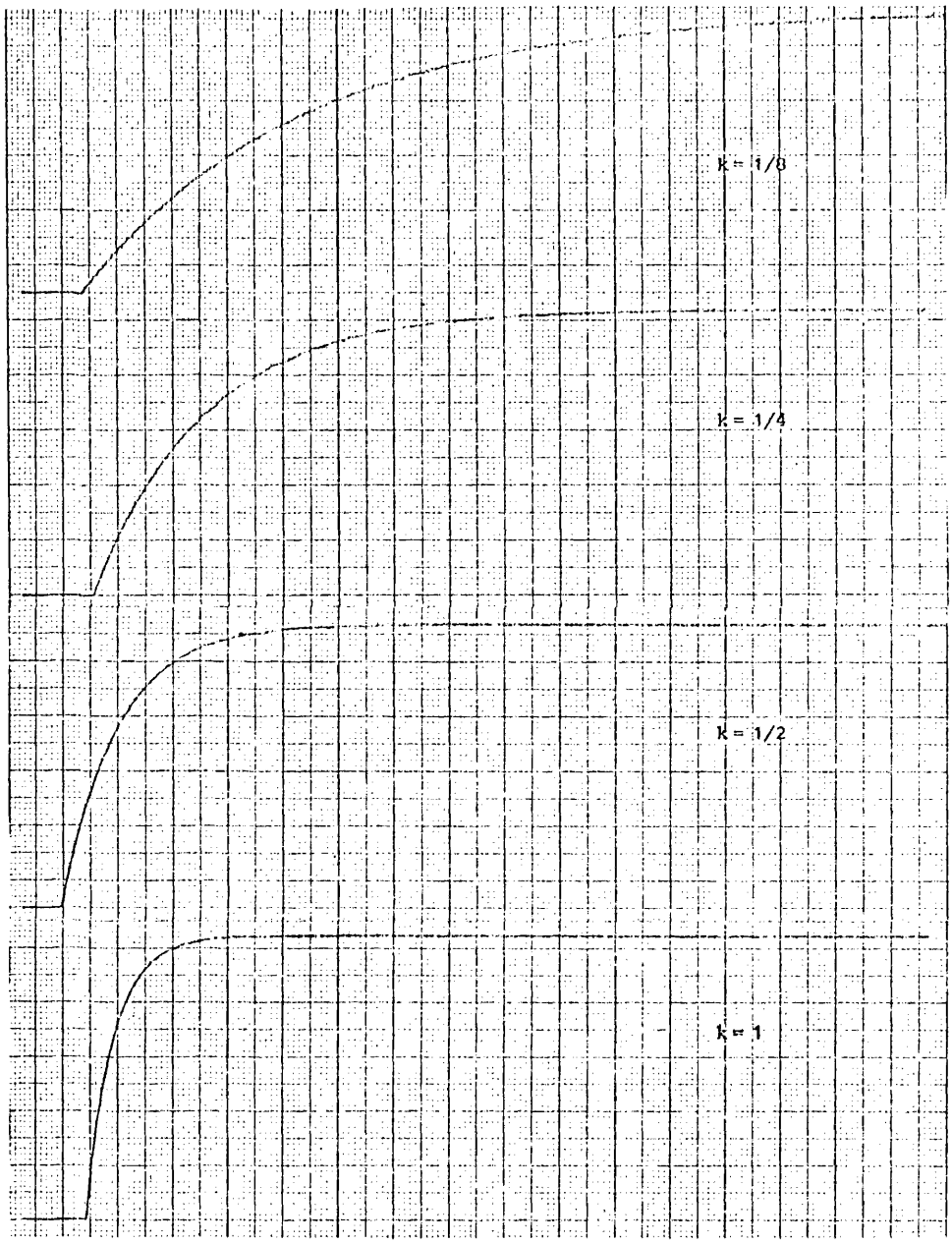


Figura 5.21: $y' = -ky$, $y(0) = y_0$. Solución con un IDRE con respecto al tiempo con coeficiente potenciométrico P

X = .25 seg/cm ; Y = 1 volt/cm ; σ = 2000 hertz

Representamos los valores iniciales de los registros R_L, Y_L y ΔY por los números enteros contenidos en dichos registros.

Ejemplo 5.1: Función exponencial decreciente.

De la ecuación:

$$y(t) = A e^{-t} \quad \text{con} \quad A = cte. \tag{5.36}$$

se deduce:

$$\Delta y(t_n) = -\{y(t_n) + 1/2\ y(t_{n-1})\}\Delta t \tag{5.37}$$

De (5.37) se obtiene el diagrama de interconexión que se muestra en la Fig. 5.23. Debe observarse que el valor actual de $y(t_n)$ viene dado por $-N(\Delta Y)2^{-m}$ o de forma más precisa por $-\{N(\Delta Y)2^{-m} + N(Y_L)2^{-2m}\}$, donde $-N(\Delta Y)$ y $-N(Y_L)$ son respectivamente los números enteros contenidos en los registros ΔY e Y_L.

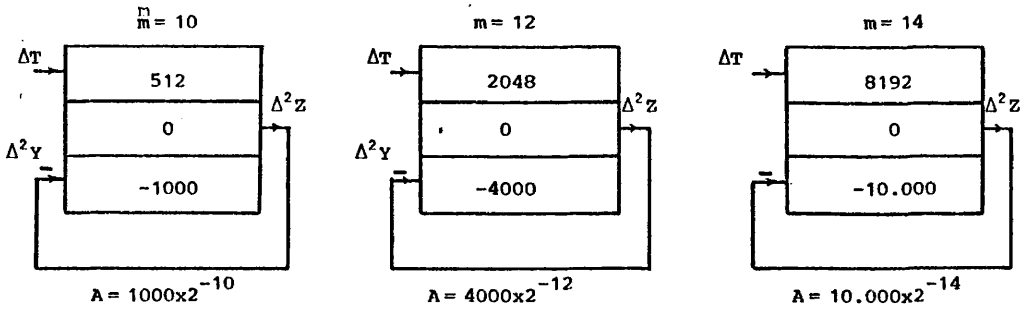


Figura 5.23: Diagrama de generación de Ae^{-t}

El error de cálculo

$$E = Ae^{-t} + N(\Delta Y)2^{-m} \tag{5.38}$$

se da para cada uno de los tres casos en la Tabla 5.1. Esta tabla muestra que el valor de E para cualquier t es menor que un pulso, esto es, menor que 2^{-m}. Hay que hacer notar que estos resultados no dan una estimación exacta de la precisión de cálculo, ya que se ha despreciado el término menos significativo $N(Y_L)\ 2^{-m}$ en (5.38). El error de cálculo es exactamente:

$$E_e = Ae^{-t}\{+ N(\Delta Y)2^{-m} + N(Y_L)\ 2^{-2m}\} \tag{5.39}$$

t	$E(x2^m)$		
	$A = 10^3 x2^{-10}$	$A = 4x10^3 x2^{-12}$	$A = 10^4 x2^{-14}$
0.00	0.0	0.0	0.0
0.25	0.8	0.2	0.0
0.50	0.5	0.1	0.3
0.75	0.4	0.5	0.7
1.00	0.9	0.5	0.8
1.25	0.5	0.0	0.0
1.50	0.1	0.5	0.3
1.75	0.8	0.1	0.7
2.00	0.3	0.3	0.3
2.25	0.4	0.6	0.0
2.50	0.1	0.3	0.9
2.75	0.9	0.7	0.3
3.00	0.8	0.1	0.9
3.25	0.8	0.1	0.7
3.50	0.2	0.8	0.0
3.75	0.5	0.1	0.2
4.00	0.3	0.3	0.2
4.25	0.3	0.1	0.6
4.50	0.1	0.4	0.1
4.75	0.7	0.6	0.5
5.00	0.7	0.0	0.4
5.25	0.2	0.0	0.5
5.50	0.1	0.3	0.9
5.75	0.2	0.7	0.8
6.00	0.5	0.9	0.9
6.25	0.9	0.7	0.3
6.50	0.5	0.0	0.0
6.75	0.2	0.7	0.7
7.00	0.9	0.6	0.1
7.25	0.7	0.8	0.1
7.50	0.6	0.2	0.5
7.75	0.4	0.7	0.3

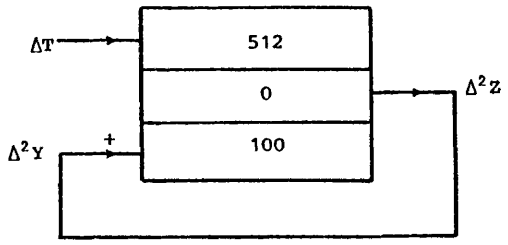
Tabla 5.1: Error de cálculo para Ae^{-t}

Ejemplo 5.2: Función exponencial creciente.

En la Fig. 5.24, se dan los diagramas de interconexión para generar $(100/2^{10})e^{t}$ y $(1000/2^{14})e^{t}$. Tanto en la Fig. 5.23 como en la 5,24, el registro R_L contiene inicialmente un número próximo a 2^{m-1} con el fin de reducir al máximo los errores de redondeo en la operación de integración.

m= 10

A= $100x2^{-10}$



a)

m= 14

A= $1000x2^{-14}$

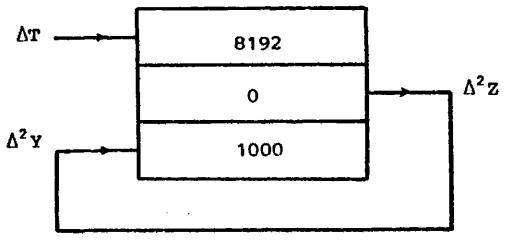


Figura 5.24: Diagrama de generación de Ae^{t}

El error de cálculo

$$E = Ae^{t} - N(\Delta Y).2^{-m} \qquad (5.40)$$

se da en la Tabla 5.2. De la tabla se deduce que este error se mantiene menor que un pulso (2^{-m}) mientras $Ae^{t} < 1$; cuando $Ae^{t} \geq 1$ se produce overflow en el registro ΔY y no es posible por lo tanto continuar el cálculo .

t	$Ex(2^{-10})$ $A = 100x2^{-10}$	$Ex(2^{-14})$ $A = 1000x2^{-14}$
0.000	0.0	0.0
0.125	0.3	0.1
0.250	0.4	0.0
0.375	0.5	0.0
0.500	0.9	0.7
0.625	0.8	0.2
0.750	0.7	0.0
0.875	0.9	0.9
1.000	0.8	0.3
1.125	0.0	0.2
1.250	0.0	0.3
1.375	0.5	0.1
1.500	0.2	0.7
1.625	0.8	0.4
1.750	0.5	0.6
1.875	0.1	0.8
2.000	0.9	0.1
2.125	0.3	0.9
2.250	0.7	0.7
2.375	x	0.0
2.500	x	0.5
2.625	x	0.6
2.750	x	0.6

Tabla 5.2: Error de cálculo para Ae^t

Ejemplo 5.3: Función sinusoidal.

De las ecuaciones:

$$y_1(t) = A\ sen\ t$$
$$y_2(t) = A\ cos\ t \qquad (5.41)$$

se deduce el siguiente par de ecuaciones en diferencias:

$$\Delta y_1(t_n) = \{y_2(t_n) + 1/2\ \Delta y_2(t_{n-1})\}\Delta t$$
$$\Delta y_2(t_n) = -\{y_1(t_n) + 1/2\ \Delta y_1(t_{n-1})\}\Delta t \qquad (5.42)$$

De (5.42) se obtiene el diagrama de interconexión que se muestra en la Fig.(5.25).

El valor calculado de $A\,sen\,t$ viene representado por:

$$N(\Delta Y_1)2^{-m} + N(Y_{L_2})2^{-2m} \qquad (5.43)$$

donde ΔY_1 es el registro ΔY del integrador I_1 e Y_{L_2} es el registro Y_L del integrador I_2.

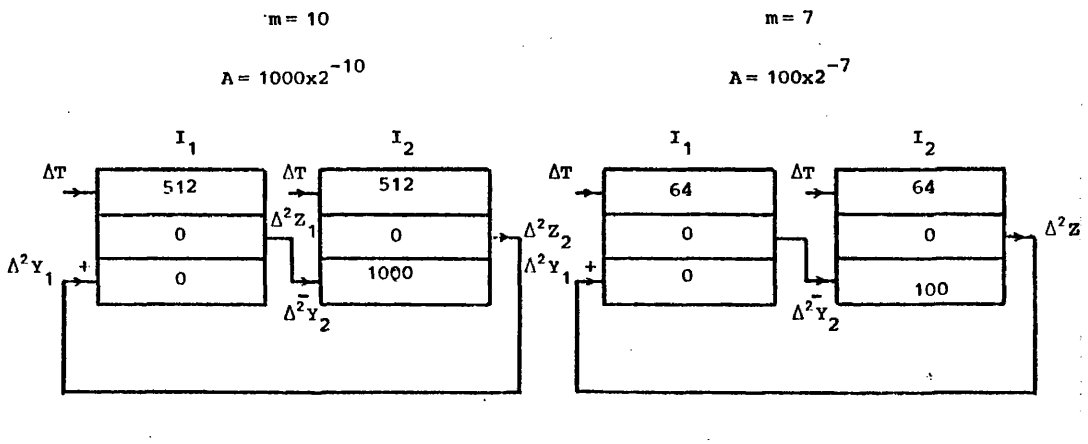


Figura 5.25: Diagrama de generación de $A\,sen\,t$.

El error de cálculo para$(1000/2^{10})$sen t se da en la tabla 5.3

t	$E(x2^{-10})$	t	$E(x2^{-10})$	t	$E(x2^{-10})$	t	$E(x2^{-10})$	t	$E(x2^{-10})$
0.0	0.0	37.5	0.2	88.0	0.4	138.0	0.0	182.0	0.2
0.5	0.4	38.0	0.4	88.5	0.2	138.5	0.7	182.5	0.7
1.0	0.5	38.5	0.0	89.0	0.1	139.0	0.1	183.0	0.7
1.5	0.5	39.0	0.8	89.5	0.4	139.5	0.1	185.5	0.2
2.0	0.3	39.5	0.6	90.0	0.0	140.0	0.2	184.0	0.6
2.5	0.5	40.0	0.1	90.5	0.7	140.5	0.4	184.5	0.9
3.0	0.1	40.5	0.1	91.0	0.0	141.0	0.1	185.0	0.6
3.5	0.2	41.0	0.4	91.5	0.3	141.5	0.0	185.5	0.5
4.0	0.5	41.5	0.4	92.5	0.5	142.0	0.2	186.0	0.0
4.5	0.5	42.0	0.5	92.5	0.6	142.5	0.3	186.5	0.9
5.0	0.1	42.5	0.9	93.0	0.7	143.0	0.7	187.0	0.8
5.5	0.5	43.0	0.2	93.5	0.0	143.5	0.4	187.5	0.9
6.0	0.6	43.5	0.2	94.0	0.8	144.0	0.0	188.0	0.5
6.5	0.1	44.0	0.7	94.5	0.9	144.5	0.7	188.5	0.5

Tabla 5.3: Error de cálculo para $(1000/2^{10})$ *sen* t

Ejemplo 5.4: Aplicación del IDREG

Se desea calcular la siguiente integral de Stieltjes:

$$z(t) = \int_0^t sen\ t\ d(cos\ t) \qquad (5.44)$$

cuyo valor exacto es:

$$z(t) = (sen\ 2t/4) - t/2 \qquad (5.45)$$

El diagrama de interconexión de ID para calcular la integral (5.44) se da en la Fig. 5.26.

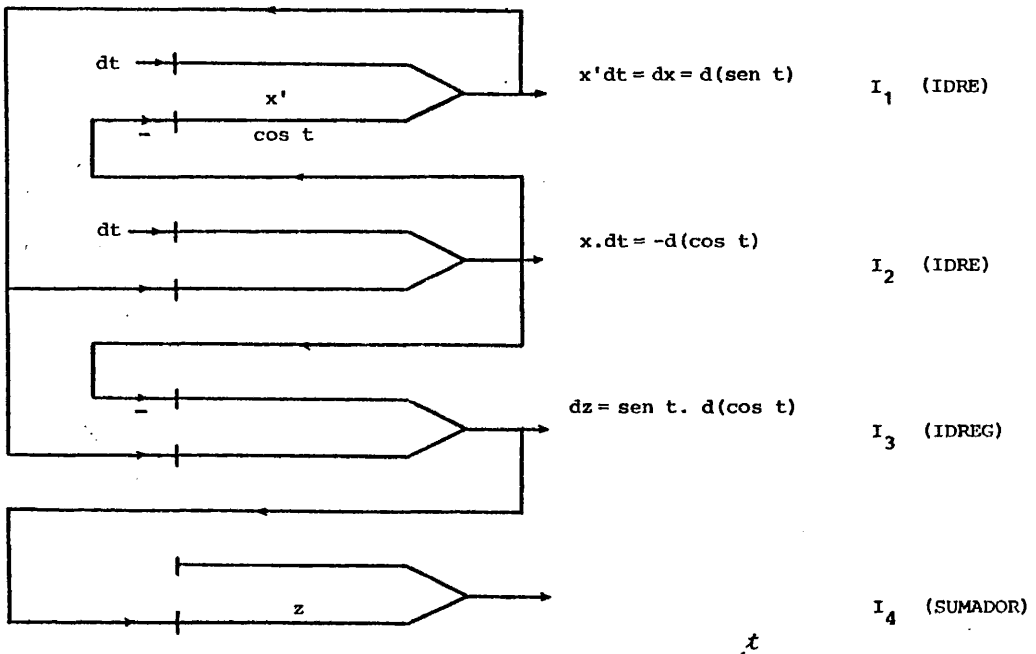


Figura 5.26: Diagrama de conexión de ID para calcular $z(t) = \int_0^t sen\ t\ d(cos\ t)$

En la fig. 5.26 los ID I_1 e I_2 corresponden a IDRE con respecto al tiempo, que resuelven la ecuación armónica (generan $sen t$ y $cos t$), el I_3 corresponde a un ID y el I_4 se utiliza simplemente como sumador para calcular la integral.

En la tabla 5.4 se dan los errores cometidos en el cálculo.

t	z(t)	Error ($\times 10^{-3}$)	
		$\Delta t = .01$	$\Delta t = .005$
50	-25'12659	-0.8585	-0.2147
100	-50.21832	-1.7100	-0.4275
150	-75.24994	-2.5493	-0.6374
200	-100.21273	-3.3749	-0.8449
250	-125.11694	-4.1888	-1.0474
300	-149.98895	-4.9965	-1.2493
350	-174.86401	-5.8049	-1.4514
400	-199.77651	-6.6209	-1.6554
450	-224.75055	-7.4493	-1.8624
500	-249.79328	-8.2915	-2.0729
550	-274.98293	-9.1449	-2.2863
600	-300.02207	-10.004	-2.5011
650	-325.14513	-10.862	-2.7156
700	-350.22822	-11.712	-2.9280
750	-375.24848	-12.549	-3.1373
800	-400.20031	-13.372	-3.3432
850	-412.65404	-14.184	-3.5463
900	-449.96695	-14.992	-3.7482
950	-474.84602	-15.801	-3.9505
1000	-499.76749	-16.619	-4.1549

Tabla 5.4: Error de cálculo de $z(t) = \int_0^t sen\ t\ d(\cos t)$

Se observa que si el incremento Δt se reduce por el factor 1/2, el error de cálculo se reduce, aproximadamente, por el factor 1/4. Esto significa que el error de cálculo debido a la fórmula de integración numérica (5.35) es del orden de Δt^2

Ejemplo 5.5

Se desea resolver la siguiente ecuación diferencial:

$$y''-ty'+y=0; \qquad y(0)=C_0,\ y'(0)=C_1 \qquad (5.46)$$

La solución analítica de (5.46) viene dada por:

$$y=C_1t+C_0\left(1-\frac{1}{2}t^2-\frac{1}{4!}t^4-\frac{3}{6!}t^6-\frac{15}{8!}t^8-\ldots\right) \qquad (5.47)$$

De (5.46) se deduce fácilmente $dy' = tdy - ydt$ (5.48)

A (5.48) corresponde el diagrama de interconexión de la Fig. 5.27

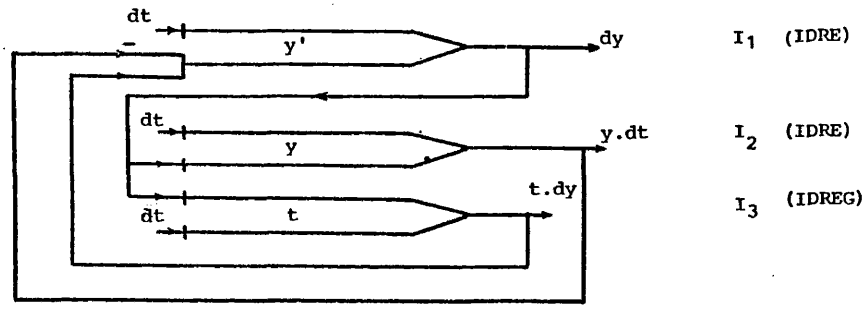


Fig. 5.27: Diagrama de conexión para la resolución de $y''-ty'+y=0;\ y(0)=C_0, y'(0)=C_1$

En la tabla 5.5 se da la solución de (5.46) con condiciones iniciales

$t(x2^{-8})$	y(t)	Error$(x10^{-5})$
20	1.07507	0.82289
40	1.14402	0.77299
60	1.20678	0.86600
80	1.26327	0.83651
100	1.31335	0.95259
120	1.35683	0.92017
140	1.39350	1.04392
160	1.42307	1.09190
180	1.44522	1.13309
200	1.45995	1.17014
220	1.46559	1.30691
240	1.46279	1.43100

Tabla 5.5: Solución de $y''-ty'+y=0;\ y(0)=1, \quad y'(0)=1$

REFERENCIAS

(1) SIZER, T.R.M., ed. "The Digital Differential Analyzer", Chapman and Hall, 1968, London.

(2) BUSH, V. and CALDWELL, S.H., "A new type of differential analyzer" J. Franklin Inst., vol. 240, 1945, pp. 255-326.

(3) SANG, J., "Description of a platometer, an instrument for measuring the areas of figures drawn on paper". Trans. Royal Scottish Soc. Arts., vol. 4, 1850-56, pp. 119-129.

(4) SLOTNIK, D.L., "The fastest computer" Scientific American, vol. 224, n°2, Feb. 1971, pp. 76-87.

(5) MAXWELL, J.C., "Description of a new form of platometer, an instrument for measuring the areas of plane figures drawn on paper". Trans. Royal Scottish Soc. Arts, vol. 4, 1850-56, pp. 420-429.

(6) THOMSON, W. "On an instrument for calculating $(\int\phi(x)\psi(x)dx)$, the integral of the product of two given functions". Proc. Roy. Soc. (London), vol. 24,

(7) THOMSON, W., "Mechanical integration of the linear differential equations of the second order with variable coefficients" Proc. Roy. Soc. (London), vol. 24, 1875-76, pp. 269-271.

(8) THOMSON, W. "Mechanical integration of the general linear differential equation of any order with variable coefficients" Proc. Roy. Soc. (London), vol. 24, 1875-76, pp. 271-275.

(9) THOMSON, J. "On an integrating machine having a new kinematic principle" Proc. Roy. Soc. (London), vol. 24, 1875-76, pp. 267-269.

(10) BUSH, V. and HAZEN, H.L. "Integraph solution of differential equations" J. Franklin Inst., vol. 204, 1927, pp. 575-615.

(11) BUSH, V. "The differential analyzer. A new machine for solving differential equations", J. Franklin Inst., vol. 212, 1931, pp. 447-488.

(12) SOROKA, W., "Analog methods in computation and simulation". Mc. Graw Hill, New York, 1954.

(13) FLEMING et al., "Thermionic Values 1904-1954" Institution of Electrical Eng. 1955.

(14) BARTEE, T.C., LEBOW, I.L., and REED, I.S. "Theory and design of digital machines" Mc Graw Hill, 1962.

(15) MELLADO, M. y col., "Desarrollo de sistemas de cálculo y control mediante técnicas discretas". Memoria -Resumen para la Comisión Asesora de Investigación Científica y Técnica. Depto. de Informática y Automática, Univ. de Bilbao, 1973

(16) ZUGASTI, V., "Investigación y desarrollo de un Analizador Diferencial Digital (ADD) para control en tiempo real". Tesis doctoral, Bilbao, 1979.

(17) DORMIDO, S. , "Estudio sistemático de los analizadores diferenciales digitales" En Cibernética: aspectos y tendencias actuales, ps.89-107. Real Academia de Ciencias Exactas, Físicas y Naturales. Madrid, 1980.

(18) Mc GHEE, R.B. and NILSEN, R.N., "The extended resolution digital differential analyzer: A new computing structure for solving differential equations" IEEE Trans. on Computer, vol. C-19, n° 1, January 1970, pp. 1-9.

(19) ELSHOFF, J.L. and HULINA, P.T. "The binary floating point digital differential analyzer" in 1970 Fall Joint Comput. Conf. AFIPS, vol. 37, Washington, D.C., Spartan, 1970, pp. 369-376.

(20) BECKETT, R. and HURT, J. "Numerical calculations and algorithms" Mc Graw Hill, 1967.

PRINCIPALES APORTACIONES Y CONCLUSIONES

Se pueden resumir en los siguientes puntos:

1) En la síntesis de una función de conmutación con MLUM(p), se define una nueva clase de estructuras que se denominan generalizadas. Se estudian sus propiedades fundamentales, y se comparan con las estructuras arborescentes demostrandose, que en general, son preferibles a estas. Para ambos tipos, únicamente son necesarias estudiar la función representativa de las clases de equivalencia n-p-n, para tener definidas las estructuras óptimas.

2) Basado en una modificación del método de Quine-Mc Cluskey, se obtienen condiciones necesarias y suficientes para la síntesis de una función de conmutación de n variables, mediante un único MLUM(p), $(n > p+1)$.

3) Se efectua una generalización del método de Voith, de manera que puedan tomarse en cuenta la existencia de términos inoperantes en la función a sintetizar. Asimismo, se plantea,de manera formal, la síntesis óptima de estructuras arborescentes con MLUM(p), como un problema de programación lineal entera 0-1. Para su resolución se emplea una variante del método de enumeración implícita de Glover, permitiendose la existencia de funciones residuos idénticas.

4) Partiendo del concepto de descomposición funcional de una función booleana, se desarrolla un procedimiento muy directo y de naturaleza cuasioptima, para la síntesis con MLUM(p). Sus características mas sobresalientes son las siguientes:

   4a) Permite la obtención tanto de estructuras arborescentes como generalizadas. En el caso de las segundas, puede dar lugar a una disminución en el número deMLUM(p) necesitados, cuando se la compara con la estructura arborescente óptima.

   4b) Cuando se implementan los algoritmos a que da lugar, la aplicación del método es muy rápida en tiempo de cálculo. De hecho, el método lo que hace es una optimización por niveles, eligiendo como variables del primer nivel aquellas que dan lugar a una reducción máxima en el número de MLUM(p) nece

sitados en ese nivel, e iterando el procedimiento a los niveles sucesivos.

5) Utilizando el procedimiento descrito en el punto anterior, se han conseguido los siguientes resultados:

5a) Dar las estructuras óptimas (arborescentes o generalizadas) para todas las clases de equivalencia n-p-n, de una función de conmutación de tres variables sintetizada con MLUM(1).

5b) La síntesis de cualquier función de cuatro variables con MLUM(1), no requiere nunca utilizar la cota máxima de 7 módulos que seria imprescindible, si únicamente se tomase en cuenta las estructuras arborescentes.

5c) Determinar todas las estructuras generalizadas con MLUM(1) que admiten d.s.d. y d.s.n.d. para funciones de cuatro variables. En todos los casos, el coste es inferior o igual al de la estructura arborescente óptima correspondiente.

6) Se aborda el estudio de contadores paralelos generalizados como bloque fundamental en la síntesis de circuitos aritméticos, desarrollándose un algoritmo de reducción que permite generar cualquier contador utilizando únicamente un solo tipo de módulo. En determinados casos, se demuestra que el procedimiento es óptimo en cuanto a que minimiza el número de módulos básicos que se utilizan.

7) Dentro de los multiplicadores paralelos de tipo iterativo, se ha desarrollado uno basado en una modificación del esquema de Guild. En el caso de los multiplicadores paralelos basados en esquemas de reducción, se generaliza el algoritmo de Stenzel.

8) Se ha desarrollado un integrador digital de resolución extendida con transmisión de las diferencias de segundo orden (Δ^2) (IDRE) con respecto al tiempo, que incorpora un coeficiente potenciométrico. El algoritmo de integración utilizado ha sido el de Adams-Bashfort de segundo orden. Se construye un prototipo experimental con circuitos integrados MSI.

9) Se propone un integrador digital de resolución extendida con transmisión de las dife-

rencias de segundo orden (Δ^2) (IDREG) con respecto a cualquier variable que incluye al anterior como caso particular.

Las posibilidades que de la presente memoria pueden emanar para abordar nuevos problemas, se pueden resumir en los puntos siguientes:

a) Procedimiento sistemático de síntesis óptima de estructuras generalizadas con MLUM(p).

b) Síntesis de contadores paralelos generalizados utilizando mas de un tipo de módulo en su construcción, y con minimización de determinadas funciones de coste.

c) Aplicación de la teoría de descomposición funcional a la síntesis de multiplicadores tipo paralelo.

d) Generalización de los integradores digitales de resolución extendida con transmisión de las diferencias de segundo orden (Δ^2) cuando se utilizan métodos de integración de varios pasos lineales.

UNIVERSIDAD COMPLUTENSE DE MADRID
BIBLIOTECA