ALGORITMOS ALTERNATIVOS PARA LA SIMPLIFICACIÓN DE FUNCIONES BOOLEANAS

ALTERNATIVE ALGORITHMS TO SIMPLIFY THE BOOLEANS FUNCTIONS

SERGIO ADRIÁN MARTÍN
Docente e Investigador de la
Universidad Francisco Gavidia

REALIDAD Y REFLEXIÓN
Reality and Reflection

Año 5, Nº 13, San Salvador, El Salvador, Centro América Revista Cuatrimestral enero-abril 2005

ALGORITMOS ALTERNATIVOS PARA LA SIMPLIFICACIÓN DE FUNCIONES BOOLEANAS

ALTERNATIVE ALGORITHMS TO SIMPLIFY THE BOOLEANS FUNCTIONS

Sergio Adrián Martín
Docente e Investigador de la
Universidad Francisco Gavidia

The author states a method to simplify the Booleans functions that could be practical if it is done in a computer. The well known methods are proposed and taken into account by other authors and the degree of difficulty of each method is estimated. It is also taken into account the total of all possible tests that the computer has to accomplish to a considerable input of variables. Besides, the way that the prototype has been developed is shown, for that reason the surroundings and language's pros and cons are analyzed. The proposed procedures to face the problems that deal with the Booleans functions do not need the use of graphics and can be applied with accessible technology in El Salvador, preferable if you can count with a 800 MHz and 32 MB computer. MARTÍN, SERGIO BRAN; ALGORITHMS; BOOLEAN ALGEBRA; LOGIC MATHS.

INTRODUCCIÓN

Durante mucho tiempo se ha especulado sobre el proceso continuo de la convergencia digital de la gran mayoría de las disciplinas derivadas de la electrónica. Sin embargo, se han mantenido muchas de las metodologías de diseño, al punto que aparte de los procesos de miniaturización y de la adopción de nuevos materiales, no se han hecho contribuciones sustanciales a los métodos de resolución de funciones booleanas. En este artículo doy a conocer un enfoque nuevo en esta área y lo comparo con los métodos planteados por diversos autores. Particularmente hago hincapié en los métodos aplicables para su uso en computadoras, no en una orientación para su uso por humanos.

El objetivo de la investigación en sí es el de implementar una serie de algoritmos con el fin de que la computadora pueda guiar a un diseñador, o a un estudiante de electrónica a resolver mucha de la problemática de los sistemas digitales sin recurrir a métodos tradicionales.

PROPUESTAS DE SOLUCIÓN

Base elemental

Si se desea simplificar una función lógica, han existido durante años métodos tradicionales con ese fin. Sin embargo algunos de ellos son métodos gráficos y otros representan problemas no resueltos aun en una computadora.

La base de una simplificación efectiva es reducir la función a un numero mínimo de operaciones y operandos. Tradicionalmente han existido dos formatos clásicos:

a) Sumatoria lógica: se puede expresar una función como la suma lógica de términos más simples. Generalmente cada término es el resultado de una función AND de las variables independientes de la función. A los términos de este tipo se les llama minterms. Por ejemplo:
Y=f+g ó y=ab'+a'b (1)

b) Multiplicatorio lógico: se puede expresar una función como un multiplicatorio lógico, es decir como una serie de operaciones AND entre términos más simples. Generalmente cada uno de esos términos es el resultado de una sumatoria lógica de algunas de las variables independientes de la función. A los términos de este tipo se les llama maxterms. Por ejemplo:
Y=g.h ó y=(a+b)(a'+b') (2)

Hacia 1927, un matemático ruso llamado Zegalkin planteo el hecho de que las funciones lógicas se pueden escribir usando principalmente operadores EXOR y operadores AND. Sin embargo, estas teorias permanecieron desconocidas en occidente durante los siguientes 30 años, cuando dos investigadores llamados Reed y Muller establecieron los llamados polinomios de Reed Muller de la siguiente forma:

$y=f_1 \oplus f_2 \oplus f_3$ (3)

donde es una expresión lógica, generalmente de tipo minterm.

B. Métodos basados en Reed Muller.

Una función booleana expresada en la forma de polinomios de Reed Muller, puede expresarse de la siguiente forma:

$y=k0 \oplus kla \oplus k2b \oplus k3c \oplus k4ab \oplus k5ac \oplus k6bc \oplus k7abc$ (4)

Resulta del arreglo de funciones exor y and de las combinaciones posibles de las variables de entrada, donde k0, k1, k2... etc., puede asumir valores de 1 ó 0. El planteamiento de funciones lógicas de esta forma implica que los circuitos a construir podrían tener un costo ma-

yor que los que se construyan de acuerdo a minterms, dado que el costo de los integrados con compuertas exor es mayor. Sin embargo, de darse el caso que la cantidad de integrados a usarse en el formato de Reed Muller es menor para una función dada, el costo total podría ser menor al final.

Si se hace una revisión completa de la expresión lógica anterior, que es la típica para una función que dependa de 3 variables de entrada (en este caso la cantidad de términos del polinomio es de 8), entonces la cantidad total de términos siempre será de 2^n.

Obviamente, a primera vista, dado que la cantidad de términos posibles dentro de una expresión es inferior a la cantidad de términos de una expresión canónica de minterms, se podría suponer que resulte más práctico un algoritmo basado en Reed Muller que uno basado en minterms, mas no necesariamente. Por ejemplo la simple función lógica OR como polinomio de Reed Muller sería:

$$y=a\oplus b\oplus ab \quad (5)$$

Mientras que la función exor en forma de minterms es:

$$y=ab'+a'b \quad (6)$$

Sin embargo, en vista de ese tipo de dificultades, se han planteado tres variantes de la expresión de los polinomios de Reed Muller:

a) La llamada "polaridad positiva", donde todas las entradas siempre se expresarán en su valor verdadero, nunca el negado.
b) La de "polaridad fija", donde se admite que algunas entradas se expresen en forma negada, pero de ser así, deberán mantener esta forma en todos los términos en que aparezcan en el polinomio final.
c) La de "polaridad mixta", donde no habría restricciones en la presencia de variables que en ciertos términos aparezcan en su forma positiva o en su forma negada.

La consecuencia inmediata de asumir variantes en la forma de expresar los polinomios de Reed Muller es que algunas de estas variantes al implementarse resulten en circuitos tan simples o más que los de las expresiones canónicas de minterms.

Los trabajos de Dragan Jankovic[1] de la Universidad de Nis en Yugoslavia, plantean que se puede plantear una solución usando matrices para encontrar los coeficientes de Reed-Muller que definen a una función booleana. En principio habría que partir de que si se conoce la tabla de verdad de una función de n variables, se puede construir para un polinomio del tipo de polaridad positiva con coeficientes desconocidos, cada uno de los cuales determina si un minterm en especial estará presente en el polinomio de Reed Muller representativo de la función.

Así,

$$y=k0\oplus kla\oplus k2b\oplus k3c\oplus k4ab\oplus k5ac\oplus k6bc\oplus k7abc \quad (7)$$

Ya que se tienen 2^n incógnitas, habría que contar con 2^n ecuaciones. Cada término de la ecuación se construye evaluando los minterms del polinomio para los correspondientes valores de las entradas, y el término independiente de cada ecuación sería el valor de la función que se evalúa, de acuerdo a su tabla de verdad.

Si se escribiera en forma matricial resultaría:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} k0 \\ k1 \\ k2 \\ k3 \\ k4 \\ k5 \\ k6 \\ K7 \end{bmatrix} = \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \end{bmatrix} \quad (8)$$

Un hecho interesante de que la matriz que define al sistema de ecuaciones no depende del vector de términos independientes, conlleva a que si se puede encontrar la inversa de ésta, se puede resolver el sistema independientemente de la función que se esté analizando.

Esta matriz en especial tiene algunas características particulares:

1-Es simétrica respecto a la diagonal secundaria, de tal forma que para todo $a_{ij} = a_{kl}$ tal que $i = l = j = k = n = 1$donde n es el orden de la matriz.

2-Si se evalúa su determinante usando las operaciones propias de los grupos de Galois, el valor del determinante siempre será 1, independientemente del orden de la matriz.

3-Su inversa, obtenida respetando las operaciones propias de los grupos de Galois es igual a la matriz original.

4-La matriz de n orden puede obtenerse mediante una expansión Kronecker, a partir de la matriz para un sistema de 1 variable:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad (9)$$

La propiedad 4 facilita construir la matriz del sistema mediante procesos iterativos, aun para sistemas de orden elevado, y el hecho de que esta matriz sea igual a su inversa evita el tener que recurrir a procesos de inversión de matrices para resolver el sistema.

De no ser por estas características, para construir la matriz habría que hacer que cada columna de la matriz coincidiera con la tabla de verdad de cada uno de los minterms que forman parte del polinomio planteado como solución general.

Aunque la mecanización de este método resulta simple, para una función como $\Sigma(1,2,3)$. La respuesta sería $y=a\oplus b\oplus ab$, lo cual resulta más complejo que la solución de la forma canónica obtenida por el método de Martín y=a+b.

Se podría pensar entonces, que la forma canónica es una manera más simple de llegar a una solución en todos los casos; la realidad es que no. De hecho, aparte de la polaridad positiva existen dos tipos más de polinomios de Reed Muller, los de polaridad fija y los de polaridad mixta.

En el caso de un polinomio de polaridad fija, cabe la posibilidad de que se presenten una o más variables negadas dentro de los minterms que forman el polinomio, pero deberán mantener esta polaridad todas las veces que aparezcan en cualquiera de los minterms de la expresión.
Así por ejemplo

$$y = k0\oplus kla\oplus k2b'\oplus k3ab' \quad (10)$$

Sería un polinomio de polaridad fija para un sistema de dos variables, en que b aparece negado en todo momento. Como es de esperar, de buscar una solución matricial para un sistema de este tipo, no se podría recurrir a la misma matriz que

para los casos de polaridad positiva. De hecho, tomando en cuenta las posibles variaciones de las combinaciones de las variables de entrada llegaran a existir 2^n posibles matrices, una de las cuales coincidiría con la matriz usada para polaridad positiva, y las otras serían diferentes. Por lo tanto, para una misma función se esperarían 2^n soluciones diferentes.

Aunque la forma más obvia de obtener cada una de estas matrices es a través de la evaluación de las tablas de verdad de cada minterm, estas matrices tendrán ciertas características a considerar:

1- No son simétricas respecto a ninguna de las dos diagonales.

2- Siempre se puede encontrar su inversa por métodos como Gauss, respetando las condiciones propias de los grupos de Galois.

3-La inversa de cada matriz es igual a la transpuesta de la matriz respecto a la diagonal secundaria (no es una transposición clásica).

Si:

$$A=\begin{bmatrix}1&0&1&0\\1&1&1&1\\1&0&0&0\\1&1&0&0\end{bmatrix} \text{ entonces } A^{-1}=\begin{bmatrix}0&0&1&0\\0&0&1&1\\1&0&1&0\\1&1&1&1\end{bmatrix} \quad (11)$$

4-Una forma de desarrollar la matriz es usando el producto Kronecker[2], tomando como base la matriz normal.

Cuando el término tiene polaridad positiva y la matriz completaría que correspondería a

$$\begin{bmatrix}1&0\\1&1\end{bmatrix} \quad (12)$$

Así para el sistema de 2 variables basado en b' y a:

$$\begin{bmatrix}1&1\\1&0\end{bmatrix}\oplus\begin{bmatrix}1&0\\1&1\end{bmatrix}=\begin{bmatrix}1&0&1&0\\1&1&1&1\\1&0&0&0\\1&1&0&0\end{bmatrix} \quad (13)$$

5-Se puede obtener la inversa de la matriz del sistema si se aprovecha una propiedad del producto Kronecker: la inversa de un producto Kronecker es igual al producto Kronecker de las matrices inversas:

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\oplus\begin{bmatrix}1&0\\1&1\end{bmatrix}=\begin{bmatrix}0&0&1&0\\0&0&1&1\\1&0&1&0\\1&1&1&1\end{bmatrix} \quad (14)$$

Así que aprovechando estas propiedades es factible desarrollar un programa, que aunque resulte más complejo que usado para polaridad positiva, permita llegar a una respuesta valida.

Nuevamente cabe la posibilidad de que la respuesta obtenida al final resulte en algunos casos tan compleja o más que la expresión de la función en su forma canónica.

Una tercera alternativa implicaría buscar la respuesta para un polinomio de polaridad mixta. En un polinomio de polaridad mixta, las variables pueden tener polaridad positiva o negativa, sin que se respete ningún orden en particular. Así por ejemplo:

$$y=k0\oplus kla\oplus k2b\oplus k3a'b \quad (15)$$

La dificultad radicaría, en que no se podría recurrir a métodos basados en el producto Kronecker para desarrollar la matriz característica del sistema. Además existen dos problemáticas adicionales: la cantidad de matrices posibles a usar, y la aparición de matrices singulares.

Número de matrices de polaridad mixta. Ya que un polinomio de polaridad mixta puede formarse a partir de cualquiera de los minterms posibles formado con n variables. Usando el equivalente a la lógica de 3 estados para formar un minterm cualquiera, entonces caben tres posibilidades:

a) *Que una variable no esté presente en la formación de un minterm.*
b) *Que esté presente y su valor esté negado.*
c) *Que esté presente y su valor no esté negado.*

Tomando en cuenta estas posibilidades, entonces, al tomar en cuenta que hay 3^n minterms, y que una solución matricial requería usar 2^n de esos minterms, las posibles matrices en total estarían dadas por el combinatorio

$$\binom{3^n}{2^n} \qquad (16)$$

El cual puede llegar a tener valores extremadamente grandes conforme n crece. Así:

Tabla 1

N	Matrices
2	126
3	2220075
4	33594090947249085
5	9.81229441228878084272647123397 48e + 39

La magnitud en que crecen la cantidad de matrices a evaluar es superior al crecimiento exponencial del total de minterms para la forma canónica, la cual crece exponencialmente de acuerdo a la formula 3^n.

Una clara referencia de la diferencia en el crecimiento de estas magnitudes se muestra en la siguiente gráfica.

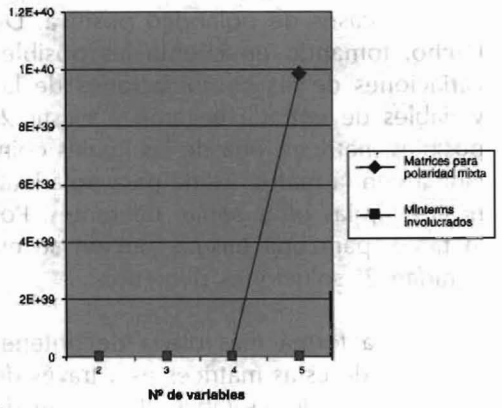


Fig 1. Crecimiento comparativo de matrices usadas para polaridad mixta y el crecimiento del número de minterms.

Este crecimiento de soluciones plantea que al dejar que un usuario escoja por sí mismo cuál de las posibles matrices no lo llevará a la solución más simple, lo llevaría a una cantidad prácticamente inconmensurable de soluciones.

El problema de las matrices singulares

Dentro de un sistema de n variables, si se escogen dos minterms que sean uno el negado del otro, al construir la matriz de polaridad mixta, se podría llegar a una matriz singular, de tal forma que ésta no tendrá inversa, y por lo tanto a través de ella no se podrá llegar a una solución válida. Así, para la combinación 1,a,a', ab se tendría:

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \qquad (17)$$

Y esta es una matriz que no tiene inversa. De hecho, entre mayor sea el número de variables, mayor será el número de matrices singulares que pueden llegar a aparecer dentro del universo de matrices generables para un sistema de polaridad mixta.

Soluciones no matriciales

Cabe la posibilidad entonces de buscar alternativas no matriciales que resuelvan funciones en forma de polinomios de Reed Muller. Los trabajos de Per Lingren[3] de la Universidad Tecnológica de Lulea, en Suecia, y los de otros plantean algunas alternativas no matriciales.

Una alternativa relativamente simple consiste en asumir que cada 1 en la tabla de verdad corresponde a un minterm cuya tabla de verdad sólo tiene un valor verdadero. Así que, al hacer la operación exor con cualquiera de los otros minterms, no habrá anulación de unos en el resultado final. Usando esta lógica, la función que tenga mayor cantidad de 1 en su tabla de verdad resultará más compleja, y cada minterm siempre estará compuesto por combinaciones de las n variables que intervienen en la función. Así, la expresión $y=\Sigma(1,3,5,7)$ correspondería a $y=c'b'a \oplus c'ba \oplus cb'a \oplus cba$. Aunque esta respuesta es válida, dista de ser la más sencilla de implementar en un circuito, y su única ventaja radica en que el procedimiento es muy simple para la computadora.

Otra alternativa no matricial, consiste en un método de aproximaciones sucesivas, según el cual, se puede descomponer una función basada en un polinomio de Reed Muller, si se asume que cada minterm tiene una cantidad de 1 menor que la función final, y aprovechando el siguiente teorema:

Si $y = x \oplus z$ entonces también es cierto que $x \oplus y = z$

Esto llevaría a que sí y es una función compleja, con una buena cantidad de 1, al hacer la operación exor con alguno de los minterms que la forman el resultado será en cualquier caso:

a) Un minterm relativamente simple y fácil de identificar o
b) Una función más simple que y tal que tendrá una cantidad menor de 1 en su tabla de verdad.

La esencia del procedimiento consistirá en poder generar las tablas de verdad de todos los minterms posibles, y en llevar conteo de cómo se reduce el número de 1 en la función resultante con forme se somete a prueba cada minterm.

Un atajo dentro de este procedimiento consiste en asumir que si la cantidad total de 1 de la función original es mayor a la mitad de todos los valores posibles de la tabla de verdad, entonces resultaría más fácil trabajar con el negado de la función original en lugar de con ella misma, y aplicar la siguiente propiedad:

$$f' = 1 \oplus f. \qquad (18)$$

En muchas ocasiones el uso de este atajo lleva a una respuesta de tipo polaridad mixta que es mucho más simple que la que se obtiene por otros métodos.

Vale la pena señalar que los algoritmos aplicables a Reed Muller ya fueron implementados como parte de proyecto de graduación de Alejandro Fabián[4], en la Universidad Don Bosco (2003).

C. Solución propuesta para expresiones canónicas.

Buscando una alternativa, que no incluya los métodos tradicionales, ni que esté relacionada con los polinomios de Reed Muller, cabe la posibilidad de llegar a soluciones más simples, y que al no depender de funciones XOR, resulten más baratas de implementar.

Al buscar una forma de sistematizar la simplificación de las funciones en las formas canónicas, propongo un algoritmo basado en las siguientes premisas.

1- Si se puede hacer que para un número n de variables la computadora pueda evaluar cada uno de los minterms que podrían obtenerse al combinar la n variables, el proceso de saber cuál minterm es parte de una función dada se convierte en un proceso de eliminación.

2- Si un minterm "m" es parte de una función f entonces

$$f = m + n + h$$
$$f' = m'n'h' \qquad (19)$$
$$\Rightarrow f'm=0$$

Luego, si se conoce la tabla de verdad de la función que se desea simplificar, la tabla de verdad de un minterm generado por la computadora, si m es parte de f, al aplicar la última ecuación, en ningún caso el resultado deberá dar diferente de 0. Si en algún caso el resultado es verdadero, habría que descartar ese minterm como parte de la ecuación final.

Por ejemplo, para la función $y=\Sigma(1,2,3)$, la tabla de verdad resulta ser la siguiente:

Tabla 2

b	A	Y
0	0	0
0	1	1
1	0	1
1	1	1

Los minterms que pueden formar esta función serán: 0,1, a a', b, b', a'b', ab', a'b y ab. Un total de 10 términos. Ya que los primeros 2 no vale la pena probarlos, al probar el caso de a tendríamos:

Tabla 3

y'	A	Y'a
1	0	0
0	1	0
0	0	0
0	1	0

Como a es un minterm incluido en Y, entonces todas las combinaciones de Y'a darán cero, siempre.

Para el caso de b' resultará distinto

Tabla 4

y'	b'	y' b'
1	1	1
0	1	0
0	0	0
0	0	0

Dado que al menos en un caso el resultado de combinación es 1, es fácil deducir que b' no es un minterm de Y.

3- El seguir el procedimiento anterior para todos los minterms posibles no elimina un problema con que generalmente se encuentran los estudiantes: la presencia de términos redundantes en la expresión final, es decir, que algunos minterms que se han incluido dentro de la expresión pueden de hecho ser absorbidos por otros.

4- Para resolver el problema anterior se puede partir de los principios planteados en el numeral 2 y determinar qué minterms ya están incluidos en otros, de tal forma que habría que descartarlos.

5- Una vez sistematizados todos estos pasos el resultado será una expresión de la forma canónica reducida a su forma más simple.

D. Relación con la teoría de conjuntos

Si se relacionan los operadores booleanos con operaciones de conjuntos, habría que considera que la función AND corresponde a la de intercepción, que la OR es la de unión, y que la NOT es simplemente señalar todo aquello que no pertenece al conjunto a que se aplica.

Así, una gráfica que señalara a un subconjunto m como incluido en un conjunto F, sería tanto como afirmar que m es un minterm de F.

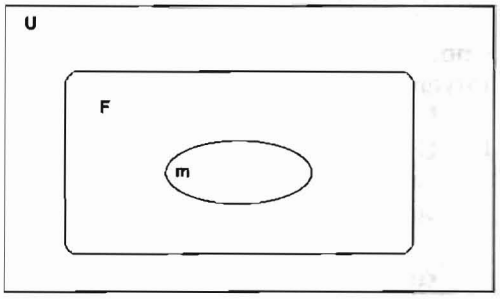


Fig. 2. m como un minterm de F

Toda la región, dentro del universo que no pertenece a f sería f':

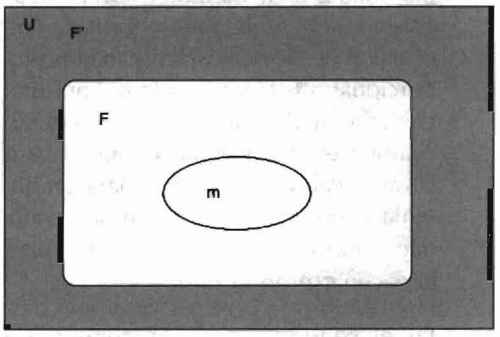


Fig 3 El conjunto de F'

Si buscamos el intercepto de F' y m, sólo llegaríamos a un espacio vacío, ya que no tienen ningún espacio en común, por lo tanto, Se cumple el principio propuesto.

Si se hubiera trabajado con un conjunto p no incluido dentro de F (una expresión que no es un minterm de F) entonces, la representación gráfica sería:

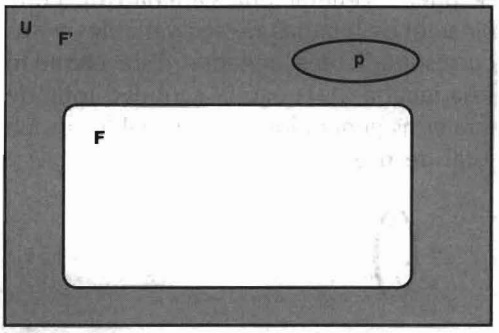


Fig. 4. p no es minterm de F pero está incluido en F'

Dado que p queda incluido dentro de F', entonces el resultado de la intercepción sería diferente a un conjunto vacío.

E. Implicaciones

Un método como el descrito implica probar una cantidad considerable de posibilidades, lo obvio es que si se sistematiza el método, cuantos más minterms se prueben en el sistema, más tiempo se llevará la simplificación.

Cuando se trabaja con una sola variable, los minterms posibles son: 0, 1, a, a'. Un total de 4. Para el caso de 2 variables ya se había comprobado que los minterms posibles son 10. Si se analiza un caso de 3 variables el número total de términos posibles es de 28. Al analizar esta secuencia de números se deduce que la cantidad de minterms al final para n variables de entrada es de 3^n+1.

Una comprobación de esta fórmula para un caso general partiría de este principio: En un caso simple de formarse un minterm con k variables de n variables de entrada, se formaría un numero de combinaciones de variables dado por:

$$\binom{n}{k} \qquad (20)$$

Pero de cada una de estas combinaciones se puede generar una cantidad de combinaciones binarias de los variables y sus correspondientes negados. Esta cantidad será igual a 2^k. Luego la cantidad total de minterms generados con k variables de un total de n es:

$$2^k \binom{n}{k} \qquad (21)$$

La sumatoria de todos los términos posibles desde k=1 hasta n, da como resultado una expresión similar a la del polinomio de Newton para la potencia de un binomio, donde 2 es uno de los monomios y 1 el otro, es decir:

$$\sum_{k=0}^{n} 2^k \binom{n}{k} = (2+1)^n. = (3)^n \qquad (22)$$

Sin embargo, como no se está considerando el caso de k = 0, entonces la cantidad de minterms formados con las variables resulta $3^n.-1$. Considerando los minterms que no dependen de las variables de entrada, es decir 0 y 1, entonces la cantidad de minterms posibles será $3^n.+1$.

F. La problemática en torno al algoritmo

Uno de los puntos de interés en torno a generar una aplicación que utilice el procedimiento propuesto, radica en la necesidad de generar las tablas de verdad de minterms y a partir de allí seguir el proceso de eliminación. Hay dos alternativas que pueden ser viables:

1- Dado que los minterms a usar serán siempre los mismos para un número dado de variables de entrada, y sus tablas de verdad no dependen de la tabla de verdad de la función a analizar, entonces sería factible tener almacenadas todas estas tablas en una base de datos, de la cual se consultarían las tablas de verdad sólo para llevar a cabo el procedimiento de eliminación descrito, no con el fin de alterar en nada los resultados almacenados.

2- Recurrir a un procedimiento que dinámicamente genere las tablas de verdad en cada ocasión en que haya que analizar una función en especial.

Ambas alternativas ofrecen ventajas y desventajas:

a) En el primer caso, habría que alimentar los datos manualmente a la tabla de verdad, y para que la aplicación tuviera la capacidad de resolver funciones con 5 variables de entrada, la base tendría que tener 243 tablas de 243 minterms, donde cada tabla tendría 32 valores asignados. De esta forma, habría entonces 7,776 elementos binarios almacenados en la base de datos. En otras palabras la base de datos necesitaría una cantidad de elementos almacenados proporcional a 6 elevado a la n. Para una cantidad n de variables a considerar. Aunque el orden de las operaciones a llevar a cabo es pequeño para un sistema informático, resulta en una cantidad enorme de datos a digitarse para almacenarse en la base.

b) En el segundo caso, la generación dinámica de las tablas de verdad implica considerar para cada variable al menos 3 condiciones posibles:

   i) El que la variable esté ausente en el minterm a analizar.

ii) El que la variable esté presente, pero deba operarse con su valor actual.
iii) Que la variable esté presente, pero deberá operararse siempre negada.

Al final, habrá que llevar a cabo un total de $(6)^n$ evaluaciones, de todos los valores que pueden asumir los minterms a analizar. El factor determinante en este caso es el tiempo máquina necesario para llevar a cabo todas estas evaluaciones cuando se quiera analizar una función que dependa de un número de variables elevado.

C. Métodos alternativos

En este momento vale la pena preguntarse si no sería viable buscar la forma de simplificar funciones booleanas por otros métodos, como el uso de una extensa base de datos en la que se buscara una forma simplificada de una función comparándola con su tabla de verdad almacenada. Aunque ésta parece una alternativa bastante simple, hay que evaluar elementos importantes como el tamaño de la base de datos o la magnitud del tiempo de búsqueda en la misma; y sopesar esto contra las características de los métodos de Martin.

De acuerdo a la metodología de Martin, si se estima que se requieren n variables para representar una función, entonces existirán 3^n minterms que pueden ser parte de la función simplificada, y el proceso de simplificación propuesto eliminará aquellos minterms que no pertenecen a la función analizada, y también aquellos que son absorbidos por otros dentro de la misma función. El proceso implica que el programa en Visual Basic desarrolle las tablas de verdad de cada uno de los minterms, y ya que en su formato más simple, cada tabla de verdad estaría formada por 2^n bits, así que el archivo que utilizaría el programa en clisp tendría un tamaño de 6^n bits. Así para simplificar funciones de sistemas de 6 variables se necesitaría de un archivo de al menos 46 Kb. En la práctica los archivos usados son más grandes porque no se expresan las tablas de verdad con 1 y 0 sino que con los términos **t** y **nil** que son propios del lenguaje Lisp.

A estas alturas vale la pena preguntarse: ¿Cuán grande sería una base de datos como sustituto al método de Martin? En primer lugar, sería necesario que esta base contara al menos con 2 campos, uno con la expresión algebraica de la función en su forma más simple, y otro con la tabla de verdad correspondiente. Obviamente el tamaño de cada campo depende en buena medida del número máximo de variables que sea capaz de procesar el sistema, y ello influirá directamente en el tamaño de la base de datos. En segundo lugar, el tamaño de la base de datos también depende del número de registros a contener, y deberá contener todas las funciones posibles que se puedan generar con n variables. La cuestión ahora es ¿cuántas funciones pueden generarse con n variables?

Para responder a la pregunta anterior habría que recurrir a buscar cuántas formas mínimas se pueden obtener combinando minterms, o partir de otro modelo que permita llegar a la misma respuesta de una forma más directa. Si se utilizan las afirmaciones de Reed y Muller de que toda función booleana puede expresarse en forma de uno de sus polinomios, si se llevan a expresiones de tipo de polaridad fija, la cual es su forma más simple, se presentan en formas como la siguiente:

$$y=k_0 \oplus k_1 a \oplus k_2 b \oplus k_3 ab \qquad (23)$$

La cual sería un polinomio de Reed-Muller para 2 variables. En general habría que

decir que un polinomio de este tipo podría estar formado por 2^n términos. Existirán coeficientes k, cada uno de los cuales asumiría valores de 1 ó 0, así que el número de posibles combinaciones que se generarían es de 2^{2n}. Si mediante un polinomio de Reed-Muller puede representarse cualquier función, entonces el número total de funciones distintas que se pueden generar con n variables corresponde al mismo valor de combinaciones generadas por un polinomio de Reed-Muller.

Así, para un sistema de 6 variables se generarían 2^{64} funciones distintas, es decir $18,446,744,073,709,551,616=1.8x10^{18}$. Si a este número corresponde el número de registros de la base de datos, aunque cada registro apenas ocupase un bit (que de hecho no será así), el tamaño mínimo de la base sería de 2147483648 Gbytes.

De lo anterior se deduce que la alternativa de usar un sistema de bases de datos es impráctica. El sistema con que cuenta actualmente no representa ni siquiera 100kbytes de consumo del disco duro ni exige más allá de los 2MB de ram que requiere cualquier programa en Clisp[5] para funcionar.

El uso de una base de datos como alternativa al método usado hasta ahora, involucraría disponer de un disco duro con una capacidad muy por encima de cualquier fabricado a la fecha. El tiempo promedio de búsqueda de una respuesta al sistema sería proporcional al tamaño de la base de datos, así que sin importar cuánto se prolongue el proceso de Martin, siempre será un valor aceptable comparado al de hacer una búsqueda en una base de trillones de Gbytes.

$$F=m.n.p \qquad (24)$$

Donde m, n y p son a su vez maxterms, al aplicar el teorema de D' Morgan se tendría:

$$F'=m'+n'+p' \qquad (25)$$

Si se lleva a cabo la siguiente operación, entonces tendríamos:

$$F'+m=m+m'+n'+p'=1 \qquad (26)$$

O simplemente

$$F'+m=1 \qquad (27)$$

De aquí se podría deducir que para cualquier maxterm m que sea parte de una función F deberá cumplirse el principio anterior. Por ejemplo:

Si: $F=\prod(2,3)$

Entonces la tabla de verdad será:

Tabla 4

b	a	F
0	0	0
0	1	0
1	0	1
1	1	1

Entre los posibles maxterms estarán: a,b,a',b', (a'+b'),(a'+b),(a+b') y (a+b), Un total de 8 maxterm, sin tomar en cuenta a 0, con lo cual podrían ser un total de 9 posibles (3^n).

Al llevar a cabo la expresión 1 tomando m=a, tendríamos:

Tabla 5

F'	m	F'+m
1	0	1
1	1	1
0	0	0
0	1	1

Se observa que en un caso en especial el resultado no es 1, por lo tanto a no es un maxterm de la función F.

Si por otra parte se prueba lo mismo con b, se puede observar un resultado diferente:

Tabla 6

F'	m	F'+m
1	0	1
1	0	1
0	1	1
0	1	1

De ahí se deducirá que para b si es un maxterm de F.
Así pues, si m es un maxterm de F, se cumplirá que:

$$F'+m=1 \qquad (28)$$

O aplicando el teorema de DeMorgan:

$$Fm'=0 \qquad (29)$$

I. El método complementario desde el punto de vista de la teoría de conjuntos

Si relacionamos los operadores booleanos con las operaciones de conjunto, tal como se señaló en el apartado B. Una función expresada en forma de maxterms sería el equivalente a la intercepción de dos o más conjuntos. Así $F=m.n$, se presentaría gráficamente como:

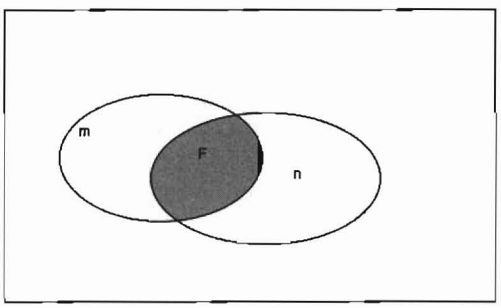


Fig. 5. m como un maxterm de F.

Mientras que m', correspondería a todo lo que fuera del conjunto m:

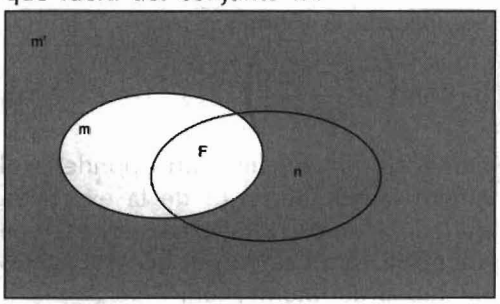


Fig. 6. El conjunto de m'.

Cuando se busca la intercepción de F y m' lo que se consigue es un conjunto vacío, lo cual concuerda con lo que había planteado anteriormente.

J. Problemáticas adicionales

Hasta ahora, se ha partido de que se puede aplicar los métodos propuestos de la premisa de que es posible llevar a cabo las operaciones señaladas entre la tabla de verdad de la función a simplificar y todas las otras posibles tablas de verdad de los minterms o maxterms que deban evaluarse para el sistema. Sin embargo, hay varias interrogantes a aclarar: ¿Cuántas variables son las necesarias para simplificar una determinada función? ¿Cómo se generarán los 3^n minterms? ¿Cómo se generarán las tablas de verdad de los minterms?.

La primera dificultad es la más simple de resolver, y es similar al problema del número de bits necesarios para representar una serie de símbolos. La diferencia radical está en que en este caso, la cantidad de bits no será función del número total de símbolos sino del valor del número mayor dentro del conjunto de números en la representación de la función como una sumatoria.

Así para $y=\sum(1,3,5,7)$, nmax=7

Se aplicaría la fórmula:

$$b=1+int\left(\frac{\log(n\ max)}{log2}\right) \quad (30)$$

Donde la función int correspondería al entero menor del valor de la expresión entre paréntesis. Esta expresión es bien conocida como el Id2 por los estudiantes de Teoría de información.

La siguiente dificultad, como obtener los 3^n minterms no debería ser mayor dificultad de no ser por:

No se está recurriendo a minterms ya previamente almacenados, sino a un método para generarlos dinámicamente en función al número b de variables involucradas en cada caso

Es más fácil para la computadora trabajar con un código numérico representativo de cada mister a tener que trabajar con expresiones algebraicas.

Como una forma de afrontar este tipo de problema se debe recurrir a lo que se conoce como un código ternario, es decir un sistema numérico basado en tres símbolos representativos. En un sistema ternario se necesitarían n cifras para representar a cada uno de los 3^n minterms. Se seguiría la siguiente convención:

a) Cada posición dentro de la expresión ternaria está relacionada con una de las variables a intervenir dentro del minterm.
b) Un valor de 0 en una posición indica que la variable correspondiente estará negada.
c) Un valor de 1 en una posición indica que la variable correspondiente se tomará con su valor sin alterar.
d) Un valor de 2 en una posición indica que esa variable no es parte de ese minterm en particular, así que no importa qué valor tome.

Siguiendo esta lógica, en un sistema de 3 variables, el minterm b'a se representaría por 201, y el minterm c por 122. Para el símbolo 222, el cual indicaría el minterm es verdadero independientemente de las variables, sólo cabe la posibilidad de que sea 1.

De lo planteado, la rutina para la generación de los minterms se limitará a iniciar un conteo de los números de 0 a 3^n -1. Cada valor del contador es convertido a código ternario y luego cada código es almacenado junto con su equivalente algebraico.

El tipo de representación planteado puede extenderse para expresar maxterms, Ya que después de todo, para n variables existirán 3^n maxterms.

La última dificultad, es decir la de evaluar la tabla de verdad de cada minterm, involucra partir de los valores ternarios almacenados para cada minterm. Cada tabla de verdad a evaluar tiene 2^n posiciones dentro de ella misma a considerar, correspondientes a todas las combinaciones binarias de las n variables. De ahí, que se recurra al uso de un contador que asume valores enteros entre 0 y 2^n -1, cada valor es convertido a un código binario, y cada bit es interpretado de acuerdo a la configuración de números ternarios que representar al minterm a evaluar. Si el bit de una variable es 0 para un valor binario y hay un 1 ternario en la posición de esa variable la posición en la tabla de verdad pasará a ser 0. Si el bit de una variable es 1 para un valor binario y hay un 0 ternario en la posición de esa variable, la posición n la tabla de verdad pasará a ser

1. Cuando el valor ternario de un variable es 2, el valor de la variable no afectará a la posición en la tabla de verdad. Por conveniencia se asume antes de evaluar cada posición que esta tendrá un valor inicial de 1.

K. Formas de desarrollar la solución propuesta

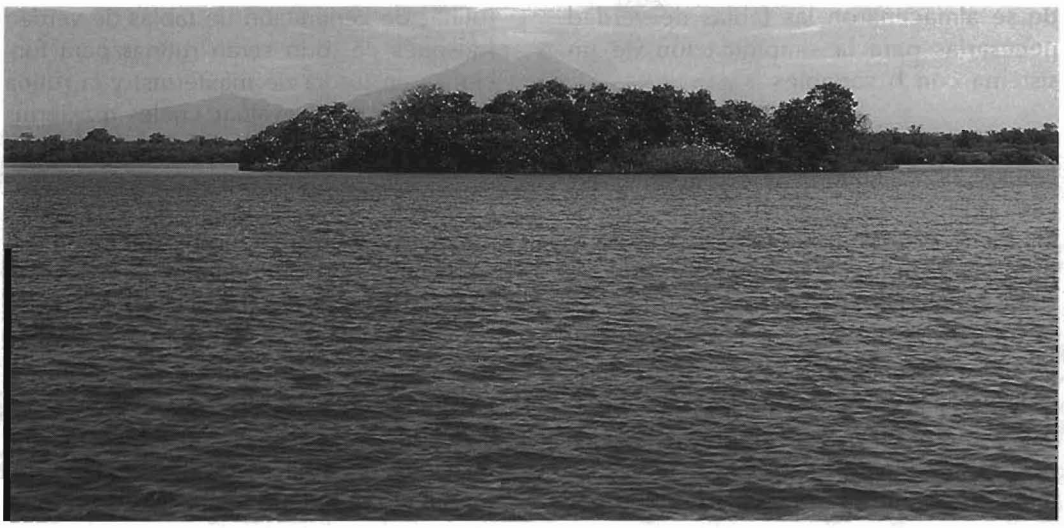
Dentro de todo el conjunto de pasos a efectuar para obtener un sistema funcional, que pueda resolver todas estas problemáticas se han planteado varias posibilidades:

Un solo sistema en Visual Basic que pueda ejecutar todos los algoritmos planteados, y otros relacionados con la conversión entre formas de representación de las funciones.

Un sistema hecho en Visual C que pueda migrarse más fácilmente a otras plataformas como una mejor alternativa.

Un sistema hecho totalmente en una plataforma CPL, que pueda llevarse a un entorno Windows o a uno Linux sin mayor dificultad.

En realidad cada propuesta tiene ventajas y desventajas. Por una parte tanto Visual Basic como Visual C presentan entornos gráficos amigables para el usuario, pero un programa en Visual Basic que lo resuelva todo consumiría extensivamente recursos de memoria, y tendría una cantidad considerable de líneas de código dadas las dificultades para manejar un gran número de vectores representativo de cada tabla de verdad. Un versión Visual C sería más fácil de migrar a otros entornos, pero la cantidad de líneas de código sería igual o mayor que para el caso de Visual Basic. Un lenguaje idóneo para este tipo de procesamiento es CLISP, ya que se puede efectuar el tratamiento de las tablas de verdad como listas, y con un mínimo de líneas de código. Sin embargo la dificultad estriba en que su entorno es de tipo texto, trabaja mejor con valores lógicos o numéricos que con expresiones algebraicas, y no hay una interfaz por defecto para conectarse con un entorno gráfico en particular.



La solución desarrollada al momento consiste en "usar lo mejor de ambos mundos". Es decir, recurrir a una plataforma de entrada y salida de datos en forma gráfica, y en transferir las operaciones booleanas más complejas a CLISP.

Así pues, la deducción del número de variables, de los minterms y de sus respectivas tablas de verdad se puede efectuar a nivel de Visual Basic. Posteriormente los resultados de estas tablas se almacenan en un archivo de texto, respetando la convención de símbolos usados por CLISP.

Un programa en Visual Basic se encarga de escribir una rutina en CLISP que invoca la evaluación de la tabla de verdad de la función a evaluar, recurriendo a una librería en CLISP que contiene las funciones de evaluación basadas en los algoritmos de Martin.

Posteriormente el programa en Visual Basic manda a ejecutar un archivo de procesamiento por lotes que provoca que CLISP evalúe la rutina creada desde Visual Basic. Mientras la rutina CLISP se ejecuta, recurre a leer del archivo donde se almacenaron las tablas de verdad necesarias para la simplificación de un sistema con b variables.

Una vez obtenida la simplificación de la función, el resultado es una lista en la cual cada posición corresponde a cada uno de los minterms que podrían ser parte de la función evaluada. Donde aparezca un valor verdadero en esta lista se estará indicando que el minterm correspondiente si forma parte de la expresión final, donde aparezca un valor falso, indica que ese minterm no esta contenido en la respuesta final, ya sea porque no es un minterm válido o porque es absorbido por otro de los minterms señalados dentro de la función final.

Ya que mientras la rutina en CLISP procesa la información, Visual Basic deberá esperar, lo cual se logra mediante una rutina de pérdida de tiempo controlada por un evento: Visual Basic registra la fecha y hora en que transfirió los datos a CLISP y también registra la fecha y hora de la última modificación a un archivo en que CLISP escribirá los resultados de sus cálculos. Mientras a la hora de la última actualización de ese archivo sea menor a la hora en que se transfirieron los datos, el programa en Visual Basic continuará con su rutina de pérdida de tiempo. Sólo cuando la fecha de la última actualización cambia, se puede tener la seguridad de que Clisp ha terminado sus cálculos y ha colocado el resultado final en el archivo de salida. Este procedimiento es necesario ya que no hay una forma directa de pasar parámetros entre los entornos VB y CLISP.

Este tipo de procedimiento es similar al empleado para simplificar funciones en forma de maxterms, aunque cambian las rutinas de generación de tablas de verdad (después de todo serán rutinas para funciones en forma de maxterms) y la rutina den CLISP para evaluar cuáles maxterms sí son válidos en la expresión final.

Para las funciones en forma de Reed Muller, aunque se puede recurrir a procedimientos similares, ya que muchos de ellos son de tipo matricial, resulta de CLISP no ofrece ventajas particulares para el procesamiento de matrices. El resultado en un programa que resulta tan largo como el desarrollado por A. Fabián en su trabajo de graduación, con la diferencia de que él no usó en ningún momento algún tipo de progra-

mación en CLISP, simplemente se limitó a diseñar sus rutinas en Visual Basic.

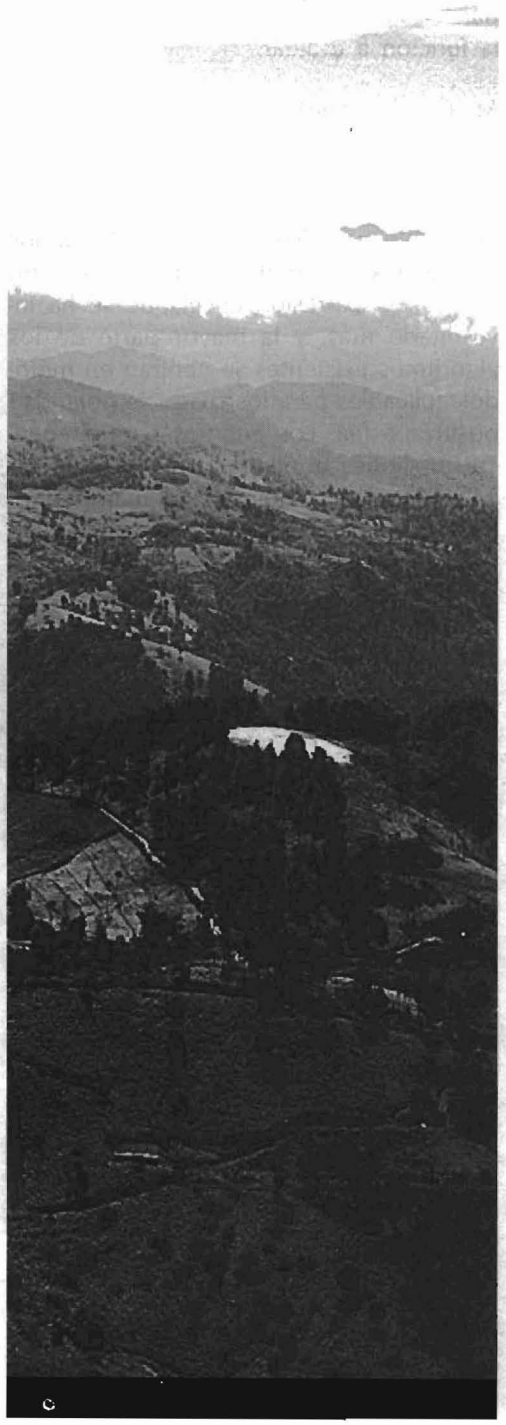
Otras metodologías

Hay algunos autores que han desarrollado sus propios métodos para afrontar los problemas relacionados con funciones booleanas. Así por ejemplo, Denis V. Popel[6] de Baker University, propone el uso de uniones de Sierpinski para la representación de funciones. Esta es más bien una metodología gráfica, y su sistematización sería más bien un reto a asumir en un futuro distante.

Como una forma de ayudar a los diseñadores a llegar a funciones más simples de implementar circuitalmente, basándose en compuertas exor, Alan Mishchenko[7] de Portland State University, propone un algoritmo llamado de Bi-descomposición. Este tipo de enfoque sigue la línea de que no importa tanto el tipo de expresión que se obtenga al final, sino que el resultado se pueda plantear en base a bloques simples de entrada y un uso mínimo de funciones. El resultado final está más cerca de un polinomio de Reed Muller de polaridad mixta que de cualquier otro tipo de expresión. El resultado de esa investigación llevó a un programa que tiene aplicación en el diseño de FPGA[8].

Debatosh Debnath[9], de Kyushu Institute of Technology de Japón, ha desarrollado un sistema para la minimización de funciones booleanas llevándolo a un polinomio de Reed Muller de polaridad fija, no usan un método de desarrollo de matrices, más bien usan una técnica de diagramas de decisión binario multi-terminal.

Otro investigador en este campo es B. Steinbach[10], de Freiberg University of Mining and Technology, quien ha desarro-



llado un método basado en derivadas parciales de las expresiones algebraicas de la función a evaluar.

M. Conclusiones

Existen muchas metodologías para enfrentar los problemas relacionados con las funciones booleanas. Desde el desarrollo de los polinomios de Reed Muller, éste ha sido el tipo de expresión en el que se ha investigado más, y la mayor parte de los algoritmos existentes se centran en métodos aplicables para los casos de polaridad positiva o fija. Los intentos para simplificar funciones booleanas en forma de expresiones de polaridad mixta, aunque prometedores, no representan necesariamente la mejor alternativa.

Cualquier intento por desarrollar un método basado en búsquedas simples usando bases de datos es impráctico para sistemas que requieran más de 4 variables.

Los métodos propuestos por el autor, difieren enormemente de la tendencia de otros investigadores alrededor del mundo. No requieren del uso de gráficos, y se pueden implementar con tecnología accesible a nuestro país. Las pruebas preliminares del sistema prototipo demuestran que es preferible contar con una computadora de más de 800 Mhz de velocidad, y más de 32 Mb de ram. Estos requerimientos de hardware caen dentro de límites aceptables para las disponibilidades actuales de la región, así que es factible implementar versiones futuras como un instrumento de enseñanza de Sistemas Digitales. Con ello se solucionaría en parte las limitaciones y deficiencias en el aprendizaje Sistemas digitales que se observa entre muchos de los estudiantes que han cursado esta asignatura.

RECONOCIMIENTOS

Expreso mi agradecimiento al doctor Otto Lange, del Technische Universität Hamburg-Harburg, por el curso dado sobre Teoría de Información. También agradezco al Dr. John Paxton, de Montana State University, por el curso que impartió sobre lenguaje CLISP, ya que es una herramienta valiosa sin la cual no se contaría en la actualidad con un prototipo del sistema de simplificación de funciones booleanas. También extiendo mi agradecimiento a Alejandro Fabián, cuyo trabajo de investigación en torno a Reed Muller representa una alternativa a los algoritmos que he desarrollado.

Referencias

[1] *Información general sobre Clisp, http://Clisp.cons.org/ .*

[2] *Dragan Jankovic, "Efficient Calculation of Fixed-Polarity Polynomial Expressions for Multiple-Valued Logic Functions", http://csdl.computer.org/comp/proceedings/ismvl/2002/1462/00/14620076abs.html.*

[3] *Biografía de Kronecker disponble : http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Kronecker.html.*

[4] *P. Lindgren "IMPROVED MINIMIZATION METHODS OF PSEUDO KRONECKER EXPRESSIONS FOR MULTIPLE OUTPUT FUNCTIONS", http://www.sm.luth.se/~pln/publications/publications.html.*

[5] *A. Fabián, "Método alternativo para la reducción de funciones por polinomios Reed-Muller". Universidad don Bosco, Tesis 621.381 F118 2003.*

[6] *D .V. Popel. "Sierpinski Caskets for Functions Representation", http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=21788 .*

[7] *Alan Mishenko, "An Algorithm for Bi-Decomposition of Logic Functions", http://www.ee.pdx.edu/~alanmi/publications/index.htm*

[8] *FPGA. http://www.mrc.uidaho.edu/fpga/fpga.html*

[9] *Younes, Ahmed "Representation of Boolean quantum circuits as reed-Muller expansions". Fuente: International Journal of Electronics; Jul2004, Vol. 91 Issue 7, p431, 14p (Obtenido a través de EBSCO Host).*

[10] B. Steinbach, "Minimization of AND-ExOR Expressions", http://www.informatik.tu-freiberg.de/prof2/publikationen/index.html

[11]Butler, Jon T.,Shmerko, Vlad P. "Comments on 'Sympathy: Fast Exact Minimization of Fixed Polarity Reed-Muller Expansion for Symmetric Functions.'.". Fuente: IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems; Nov2000, Vol. 19 Issue 11, p1386, 0p, 2 charts (Obtenido a través de EBSCO Host).

[12]Yang, Ke1; :Zhao, Qianchuan2 "The balance problem of min-max systems is co-NP hard.". Fuente: Systems & Control Letters; Nov2004, Vol. 53 Issue 3/4, p303, 8p (Obtenido a través de EBSCO Host).

Notas

1 Dragan Jankovic, "Efficient Calculation of Fixed-Polarity Polynomial Expressions for Multiple-Valued Logic Functions".

2 Es interesante el hecho de que Kronecker ideó su producto originalmente para operaciones con números racionales, pero que ello no tiene mayor aplicación práctica en ninguna área, y aunque en su época ya existían los trabajos de Boole, ha sido hasta recientemente que se está aplicando a esa área.

3 P. Lindgren "IMPROVED MINIMIZATION METHODS OF PSEUDO KRONECKER EXPRESSIONS FOR MULTIPLE OUTPUT FUNCTIONS"

4 Alejandro Fabián, "Método alternativo para la reducción de funciones por polinomios Reed-Muller", Universidad Don Bosco 2003.

5 Common Lisp: versión de lenguaje Lisp (List proccesing). Lenguaje con licencia GNU usado en aplicaciones de inteligencia artificial y para el procesamiento de listas complejas.

6 D.V. Popel. "Sierpinski Caskets for Functions Representation"

7 Alan Mishenko, "An Algorithm for Bi-Decomposition of Logic Functions"

8 FPGA: Arreglos de campos de compuertas programables, algunos de estos arreglos se pueden usar para implementar funciones lógicas y otros se usan en el diseño de sistemas de redes neuronales.

9 D. Debnath "Exact Minimization of Fixed Polarity Reed-Muller Expressions for Incompletely Specified Functions"

10 B. Steinbach, "Minimization of AND-ExOR Expressions".