UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA
ESCUELA DE ELECTRÓNICA

TRABAJO DE GRADUACIÓN:

METODO ALTERNATIVO PARA LA REDUCCIÓN DE FUNCIONES BOOLEANAS POR POLINOMIOS REED-MULLER

PARA OPTAR AL GRADO DE:

INGENIERO EN ELECTRONICA

PRESENTADO POR:
FABIAN ESPINO, ALEJANDRO ALBERTO

OCTUBRE DEL 2003

SOYAPANGO - ELSALVADOR - CENTROAMERICA

DEDICATORIA

A **JEHOVÁ mi Dios**, quien con su bondad inmerecida me ha permitido llegar hasta donde estoy.

AGRADECIMIENTO

A mis padres, que a pesar de muchas cosas siempre han estado allí, pendientes por mi, los amo.
A ti mamá por la paciencia que tuviste, el apoyo que me has dado y el amor que sigues teniendo por mi, mil gracias.

Cachorra, gracias por todo, si, vos, gracias, vos entendés.
A mi esposa y a mis dos hijos, mis tres amores, los amo.
Un agradecimiento a mi familia por el apoyo incondicional, a mis amigos que me apoyaron a todos.
A mis hermanos, saludos congregación Monte Blanco.

ALEJANDRO ALBERTO FABIAN ESPINO.

INDICE

CONTENIDO **PAG.**

LIMITACIONES

- Se desarrollará un software como aplicación de Reed-Muller, para minimizar funciones lógicas y no para los otros métodos conocidos.

- El software a utilizar se desarrollará en la plataforma visual Basic , en entorno gráfico.

- El software como aplicación solo se limita a la aplicación de Reed-Muller para minimizar funciones lógicas, solamente.

- El programa de aplicación solamente proporciona la minimización de una función booleana con compuertas Exor, y no un software para diseñar dispositivos FPGA.

ALCANCES

- Con la investigación se logra a dar a conocer las diferentes utilidades del código Reed-Muller en el campo de electrónica.

- Lograr poseer otra herramienta para la minimización de funciones lógicas

OBJETIVOS

OBJETIVO GENERAL:

- Dar a conocer otro método de reducción de funciones lógicas utilizando polinomios Reed-Muller

OBJETIVOS ESPECIFICOS:

- Proporcionar información de este tipo de herramientas con fines educativos y destacar su utilidad en las áreas de Electrónica.

- Dar a conocer esta herramienta eficaz, y las diferentes representaciones de la misma, mediante aplicaciones a través de la implementando de un lenguaje de programación para la descomposición de este método.

- Dar a conocer el método de corrección de errores mediante Reed-muller de primer orden

- Proporcionar una fuente bibliográfica como referencia para estudios futuros.

- Dar a conocer la implementación de Reed-Muller en los dispositivos FPGA.

INTRODUCCIÓN

Actualmente existe información sobre el diseño de circuitos digitales basado en álgebra Booleana. Sin embargo, hace mucho tiempo que el álgebra Booleana basada en el operador EXOR parece haber sido ignorada completamente de los libros de sistemas digitales;

sin embargo se darán a conocer algunos puntos como:

- Esta álgebra tiene los dos métodos del mapa y los métodos de simplificación algebraica para funciones Booleanas que no son en ningún caso más difíciles al usarse, que los métodos correspondientes del álgebra utilizando OR/AND.
- Que contiene aplicaciones útiles si usted está diseñando con dispositivos de la lógica programables.

Una motivación para estudiar álgebra EXOR es, en general, que una proporción significativa de funciones de la lógica puede algunas veces representarse con menos términos si se usan compuertas EXOR en la representación de las funciones. Muchos CPLDs[1] contienen una compuerta EXOR en su macros celdas, por consiguiente una manera sistemática de fabricación usando compuertas EXOR debe ser parte de las posibilidades de los diseñadores en el área digital.

El operador EXOR mantiene los requisitos matemáticos de linealidad. Las propiedades de este, guían a resultados como aplicaciones, particularmente en sistemas de comunicaciones (corrección y detección de errores). Sin embargo, el énfasis en este documento está en el desarrollo y aplicación del operador EXOR para el diseño de circuitos digitales. Por consiguiente el presente trabajo se concentrará en el desarrollo de diferentes aplicaciones del álgebra con el operador EXOR como el código Reed-Muller en minimización de funciones lógicas, detección y corrección de errores y FPGA[2] con el fin de contribuir a la formación de futuros profesionales en el área de diseño digital.

---

[1] Complex Programmable Logic Device (Dispositivos complejos de lógica programable)

[2] Field Programmable gate array

Las representaciones de las funciones AND/EXOR son parte del área de la lógica digital. Estas pueden ser usadas para procesar imágenes, codificando y reconocimiento de imágenes[3] . También es utilizado para la manipulación de las formas lógica en Diseño Asistido por Computadora (CAD), siendo una base de probadores automáticos y lenguajes de programación, como Prolog (componente de inteligencia artificial desarrollado por IF Computer Japón).

Con la finalidad de no extenderse exageradamente en el desarrollo de los diferentes capítulos se ha decidido remitir a los anexos del documento todo aquel material referente a deducciones, programa de aplicación, flujo gramas, tablas comparativas, otros.

---

[3] *Vision* Reddy B., Pai A., Reed-Muller Transform Image Coding, *Computer, Graphics and Image Processin*g, vol. 42, 1988, pp. 48-61

CAPÍTULO 1

1. MINIMIZACIÓN DE VARIABLES LOGICAS MEDIANTE REED-MULLER

1.1 FUNDAMENTOS DE COMPUERTA OR EXCLUSIVA

El operador EXOR ($\oplus$[4]) se relaciona con el operador OR por:

$$x \oplus y = xy' + x'y = (x+y)(x' +y') \qquad (2.0)$$

Donde X y Y son variables Booleanas

Y la expresión representada por la operación EXOR de la ecuación (2.0) utiliza conexiones AND y OR, donde la ecuación (2.0) puede ser definida por lo siguiente:

$$X \oplus Y = 0 \text{ cuando } X = Y \qquad (2.1)$$

$$X \oplus Y = 1 \text{ cuando } X \neq Y \qquad (2.2)$$

Y a continuación se muestra los teoremas para la identificación EXOR[5]:

Th1 $x \oplus x = 0$

Th2 $x \oplus x' = 1$

Th3 $x \oplus 0 = x$

Th4 $x \oplus 1 = x'$

Th5 $x \oplus y = y \oplus x$

Th6 $x' \oplus y' = x \oplus y$

Th7 $(x \oplus y)' = x' \oplus y = x \oplus y' = xy + x' y'$

---

[4] Símbolo para el operador EXOR

[5] Los teoremas han sido enumerados para facilitar al lector, pero el orden de los teoremas es completamente arbitrario.

Th8 $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$

Th9 $x(y \oplus z) = xy \oplus xz$

Th9' $x(y \oplus z) = (x'+y) \oplus (x'+z)$

Th10 If: $f = g \oplus h$ and $gh = 0$, then $f = g + h$

Th11 If: $f = g \oplus h$, then $g = f \oplus h$ and $h = g \oplus f$

Th12 $x'y \oplus xy' = x \oplus y$

Th13 $f.x' = f \oplus f.x$

Donde f es cualquier expresión Booleana

Th14 $x + y = x \oplus x'y$

Th15 $x \oplus xy = xy'$

Th16 $x + y = x \oplus y$ si $xy = 0$

Th17 $x + y = x \oplus y \oplus xy$ si $xy \neq 0$

Th18 $x' \oplus 1 = x$

Th19 $x' \oplus 0 = x'$

Th20 $x' \oplus x' = 0$

Los teoremas (Th1) - (Th5) pueden ser derivadas directamente de la ecuación (2.0), las deducciones de los teoremas (Th6) - (Th17) son demostradas en el anexo.

El teorema de DeMorgan no es aplicable aquí. Esto produce una interpretación diferente para dualidad en el álgebra EXOR.

Dualidad en los teoremas de EXOR

El principio de dualidad para el álgebra booleana dice "Dado cualquiera de los teoremas básicos del álgebra Booleana, cambiar del operadores OR al operador AND, del operadores AND al operadores OR y cambiando 0s a 1s y 1s a 0s donde éstos conducen o lleva a otro de los teoremas básicos" específicamente aplicables al operador AND y operadores OR.

1.2 SUMA DE PRODUCTOS EXOR (ESOP)

Un producto combinado por términos EXOR se llaman suma de productos EXOR (ESOP). Los ESOP[6] son expresiones genéricas formadas por operadores AND y EXOR.
Las operaciones utilizando ESOP requieren Menos Términos que una suma de productos con OR (SOP[7]) en algunos casos.
Las formas de ESOP se Clasifican por:

a) Su tipo de expansión:

Expansión Shannon:

$F(Xn, Xn-1,...Xi,...Xl) = Xi' Fo \oplus xiFo$

Expansión Davio-positivo:

$F(Xn, Xn-1,...Xi,...Xl) = F0 \oplus XiF2$

Expansión Davio-negativo:

$F(Xn, Xn-1,...Xi,...Xl) = F1 \oplus Xi'F2$

b) Y por la polaridad de las variables:

[6] Exclusive-OR Sum of Products

[7] Sum of Products

b.1) **PPRM** (POSITIVE POLARITY REED-MULLER)

b.2) **FPRM** (FIXED POLARITY REED-MULLER)

b.3) **MPRM** (MIXED POLARITY REED-MULLER)

b.4) **GRM** (GENERALIZED REED-MULLER)

La figura 1 nos muestra como está estructurada esta clasificación:

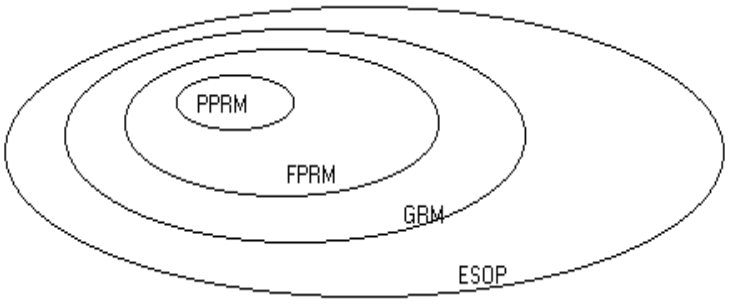


Fig. 1

En la siguiente tabla se observa la relación que existe tanto para SOP y ESOP

OPERACIÓN	SOP	ESOP
MULTIPLICACIÓN	AND	AND
ADICIÓN	OR	EXOR

1.3 POLINOMIOS REED-MULLER

Formas canónicas para la lógica de Reed-Muller

La forma canónica de Reed-Muller para funciones Booleana fue introducido en 1954 por I. S. Reed y D. E. Muller y su expresión es de este tipo:

$$F(X1,.....Xn) = \sum_{i=0}^{2^n-1} (C_i).(x1)^{i1} \cdot x2^{i2} \ldots xn^{in}$$

Donde C_i puede ser 0 ó 1, siendo una constante y el signo " $\sum$ " la función estándar EXOR, el " . " denota la función AND.

Dando lugar a muchas generalidades como Reed-Muller con polaridad positiva, fija, y mixta, cerca de los años 1960 por M. Cohn,

En el diseño de lógica digital ahora existe dos paradigmas, el primero usando compuertas AND, OR, NOT llamada lógica Booleana. La segunda es usando operadores AND, EXOR , NOT y esto es llamado lógica Reed-Muller (RM).

Para cualquier función de dos variables f(x, y), la expresión canónica de suma de productos para estas dos variables es la siguiente:

$$F(x, y) = h0x'y' + h1x'y + h2xy' + h3xy \qquad (2.3)$$

Donde hi Є {0,1} y, x'y' + x'y + xy' + xy son los minterms y pueden ser abreviados como m0, m1, m2, m3. Y aplicando el teorema Th10 o directamente la ecuación del teorema Th16:

$x + y = x \oplus y$ si $xy = 0$

a la ecuación (2.3), el operador OR puede reemplazarse por el operador EXOR quedando transformado así:

$$F(x, y) = h0x'y' \oplus h1x'y \oplus h2xy' \oplus h3xy \qquad (2.4)$$

El teorema Th10 puede ser expresada por una función dado por los términos del operador OR, en términos del operador de EXOR. La función en términos del operador de OR se extiende a una forma minterm, para que todos los términos encajen. El operador + pueden reemplazarse entonces con el operador ⊕ .

Esto significa que cualquier función puede expresarse en forma EXOR directamente de su tabla de verdad o mapa-k[8].

Una forma canónica para el operador de OR para tres variables es:

$$f(abc) = \alpha_0 a'b'c' \oplus \alpha_1 a'b'c \oplus \alpha_2 a'bc' \oplus \alpha_3 a'bc \oplus \alpha_4 ab'c' \oplus \alpha_5 ab'c \oplus \alpha_6 abc' \oplus \alpha_7 abc$$

dónde: αi = 1,0 que dependen si los término de la expresión f(abc) están presente o no.

[8] Mapas de Karnaugt, dispositivo gráfico que se utiliza para simplificar una ecuación lógica o para convertir una tabla de verdad en su circuito lógico

Una forma canónica alternativa es sustituyendo las variables negadas de la ecuación (2.4) por la ecuación del teorema Th4: $x \oplus 1 = x'$

dando lugar a la siguiente ecuación:

$$F(x, y) = h0(x \oplus 1)(y \oplus 1) \oplus h1(x \oplus 1)y \oplus h2x(y \oplus 1) \oplus h3xy \qquad (2.5)$$

Luego aplicando los siguientes teoremas Th8 y Th9

$(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$ (Th8)
$x(y \oplus z) = xy \oplus xz$ (Th9)
tenemos:

$$\mathbf{F(x, y) = g0 \oplus g1x \oplus g2y \oplus g3xy} \qquad \mathbf{(2.6)}$$

Donde:

$g0 = h0$

$g1 = h0 \oplus h2$

$g2 = h0 \oplus h1$

$g3 = h0 \oplus h1 \oplus h2 \oplus h3$

Y la ecuación (2.6) se le conoce como polinomio Reed-Muller para una función Booleana de dos variables.
La ecuación anterior puede ser expresada para una función de n variables:

$$F(x1, \ldots, xn) = g0 \oplus g1x1 \oplus g2\,x2 \oplus \ldots \oplus g(2^n\text{-}1)\, x1x2 \ldots xn \qquad (2.7)$$

Donde $gi \in \{0,1\}$, $i = 0$ hasta $2^n - 1$ y la ecuación (2.7) es la ecuación canónica del polinomio Reed-Muller para una función de n variables.

Dando lugar a que la ecuación (2.7) también pueda ser expresada como:

$$f = \oplus \sum_{i=0}^{2^{n-1}} g_i \, m_i$$

donde el término " mi " es el mintemrs y " $\oplus \sum$ " denota la sumatoria EXOR.

Una variable del polinomio Reed-Muller puede también existir de forma negada. Si cada variable aparece de forma verdadera o de forma negada pero no ambas, el polinomio Reed-Muller esta referido a un polinomio Reed-Muller con polaridad fijo.

1.3.1 POLINOMIO REED-MULLER CON POLARIDAD POSITIVA (RMPP)

Un polinomio Reed-Muller para n variables de polaridad positiva es aquel en el cual todos sus términos están de forma verdadera y no hay ninguna variable negada.

EJEMPLO 1

Siguiendo como ejemplo para una función " f " de 4 variables, expresarla en términos de Reed-Muller.

$F(x1, x2, x3, x4) = \sum(0, 3, 5, 6, 13, 15)$

$= X1' X2' X3' X4' + X1' X2' X3 X4 + X1' X2 X3' X4 + X1' X2 X3 X4' + X1 X2 X3' X4 + X1 X2 X3 X4$ (2.8)

Ahora reemplazando los términos OR por EXOR de la ecuación (2.8) tenemos:

$= X1' X2' X3' X4' \oplus X1' X2' X3 X4 \oplus X1' X2 X3' X4 \oplus X1' X2 X3 X4' \oplus X1 X2 X3' X4 \oplus X1 X2 X3 X4$ (2.9)

Pero sustituyendo Xi' por $1 \oplus$ Xi la ecuación (2.9) puede rescribirse como:

$F(x1, x2, x3, x4)$

$= 1 \oplus X1 \oplus X2 \oplus X3 \oplus X4 \oplus X1X2 \oplus X1X3 \oplus X1X4 \oplus X1X2X4$ (2.10)

Y a esta ecuación se le considera el polinomio de Reed-Muller de polaridad positiva donde todos los términos están en su forma verdadera.

EJEMPLO 2

$F(X3, X2, X1, X0) = \sum(2, 5, 6, 7, 8, 9, 11, 13)$

$=X3'X2'X1X0' \oplus X3'X2X1'X0 \oplus X3'X2X1X0' \oplus X3'X2X1X0 \oplus X3X2'X1'X0'$
$\oplus X3X2'X1'X0 \oplus X3X2'X1X0 \oplus X3X2X1'X0$ (2.11)

Aplicando Th4, Th9 y Th1 a la ecuación (2.11) tenemos:

$F(X3, X2, X1, X0)$
$= X1 \oplus X3 \oplus X1X2 \oplus X2X0 \oplus X3X2 \oplus X3X2X1$ (2.12)

1.3.2 POLINOMIO REED-MULLER CON POLARIDAD FIJA (RMPF)

Las ecuaciones anterior puede ser una función generalizada para cualquier número de variables, y cada variable puede estar en forma verdadera o de forma negada, pero no ambas en la misma función. De tal forma que a este polinomio se le considera con polaridad fija. Por lo tanto el RMPF es una optimización de RMPP.
Ahora considerando el misma ejemplo 2, pero con polaridad fija tenemos:

EJEMPLO 3

$F(X3, X2, X1, X0) = \sum(2, 5, 6, 7, 8, 9, 11, 13)$

$=X3'X2'X1X0' \oplus X3'X2X1'X0 \oplus X3'X2X1X0' \oplus X3'X2X1X0 \oplus X3X2'X1'X0'$
$\oplus X3X2'X1'X0 \oplus X3X2'X1X0 \oplus X3X2X1'X0$ (2.13)
Igualando los siguientes términos tenemos:

$X2 = 1 \oplus X2'$
$X0 = 1 \oplus X0'$
$X3' = 1 \oplus X3$
$X1' = 1 \oplus X1$

Sustituyendo estas igualaciones en la ecuación (2.13) tenemos :

$F = 1 \oplus XO' \oplus X2' \oplus X2'X0' \oplus X1X0' \oplus X3X2' \oplus X3X1 \oplus X3X2'X1$

Note que X3 y X1 solamente tienen forma verdadera y X0 y X2 están de forma negada, y a esta ecuación se le llama polinomio de Reed-Muller con polaridad fija.

EJEMPLO 4

También es posible que las variables X3 y X1 estén de forma negada y las variables X0 y X2 de forma verdadera utilizando siempre los teoremas Th4 y Th18.

Tenemos:

X2' = 1⊕ X2
X0' = 1⊕ X0
X3 = 1 ⊕ X3'
X1 = 1 ⊕ X1'

F(X3, X2, X1, X0) = ∑(2, 5, 6, 7, 8, 9, 11, 13)

=X3'X2'X1X0' ⊕ X3'X2X1'X0 ⊕ X3'X2X1X0' ⊕ X3'X2X1X0 ⊕ X3X2'X1'X0'

⊕ X3X2'X1'X0 ⊕ X3X2'X1X0 ⊕ X3X2X1'X0 (2.14)

Sustituyendo y operando en la ecuación (2.14) se tiene :

F = X3' ⊕ X1' ⊕ X0 ⊕ X2X1' ⊕ X1'X0 ⊕ X2X0 ⊕ X3'X2X1'

Note que tanto X3 y X1 están de forma negada y X2 Y X0 de forma verdadera.

Las expresiones de RMPF son una manera para optimizar las expresiones de RMPP y a la ves son derivadas de esta misma, permitiendo escoger Xi ' ó Xi pero no ambos.

1.3.3 POLINOMIO REED-MULLER CON POLARIDAD MIXTA (RMPM)

Los polinomios con polaridad mixta son aquellos en los cuales sus variables poseen doble polaridad o son bipolares, en la cual están de forma verdadera o de forma negada, ambas y esta generalidad de Reed-Muller es considerada la más optima.

Para poder considerar la polaridad mixta se hará uso de mapas-K, pero de una forma especial y particular, diferente a como se acostumbra a realizar.

Exprese la función: `f = ac + b'c + a'bc'` en términos EXOR .

f = ac + b'c + a'bc'

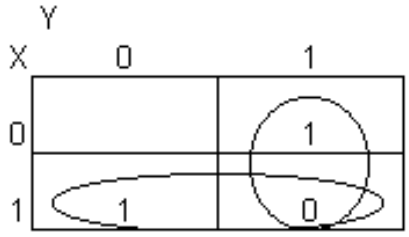
= c(a + b') + a'bc'

= c(a'b)' + (a'b)c'

= c ⊕ a'b

Hay muchos casos para la resolución de funciones de tres variables. Sin embargo, a menudo este resolución no puede ser tan fácil de llevar a cabo. Afortunadamente la técnica de mapa-k puede aplicarse para este propósito.

Primero considere la función EXOR trazada en un mapa-K como es mostrada en Figura 2. La técnica del mapa-K para la función EXOR depende de Th10 y las formas extendidas de Th1 y Th3.



X ⊕ Y = XY' + X'Y

Fig 2

Trazando las variables individualmente x y y en los resultados del mapa en xy del minterm que se traza dos veces, por consiguiente por la forma extendida de Th1, este término desaparecerá de la forma de EXOR.

Generalmente esta observación para Reed-Muller utilizando mapas-k esta dada por las siguientes reglas:

- **Debe trazarse cualquier minterm que será incluido en la función un número impar de veces .**
- **Debe trazarse cualquier minterm que será excluido de la función (o ceros) el número par de veces .**

Se a agrupado un cero con ambos unos, pero según el teorema Th10 en esa casilla XY = 0 y donde tendría que ser F = X + Y es F = X ⊕ Y = X' Y ⊕ X Y' y esto es Reed-Muller con polaridad mixta.

Considere los siguientes ejemplos:

EJEMPLO 1

F(a, b, c) = ∑(3, 5, 6, 7)

Colocados en el mapa tenemos la figura 3

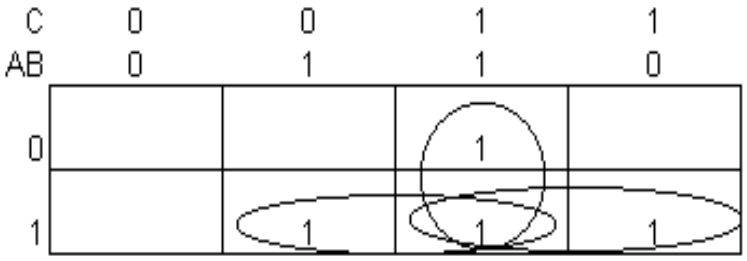


Fig.3

F = ab ⊕ bc ⊕ ac

El segundo ejemplo es una función que tiene la misma forma para los operadores de EXOR y los operadores AND/OR. Aquí un término se ha trazado tres veces y ha sido por consiguiente incluido en la función. Con un poco la práctica las formas de EXOR pueden encontrarse prontamente para tres funciones y a veces para cuatro funciones .

EJEMPLO 2

F = (a, b, c) = ∑(1, 2, 5, 7)

Ploteando los minters en el mapa de la siguiente forma:

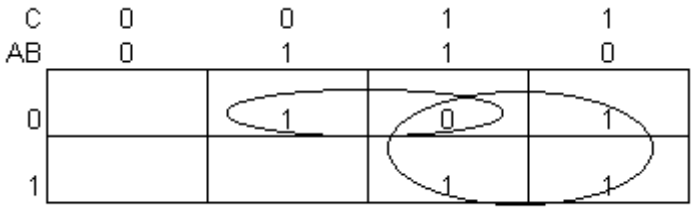


Fig. 4

Usted puede ver que el minterms “ a'bc ” se ha trazado dos veces y se ha excluido por consiguiente de la ecuación.

F = c ⊕ a’b

Y utilizando el método tradicional: mapas K tendríamos:

F = b’c + ac +a’bc’

EJEMPLO 3

F = (a, b, c) = ∑(0, 3, 6, 7)

Colocándolos en el mapa tenemos:

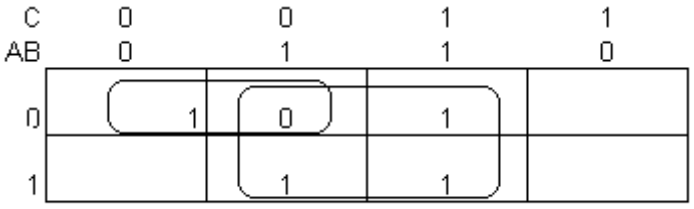


Fig. 5

F = b ⊕ a’c

EJEMPLO 4

Dada la siguiente función F = (a, b, c) = ∑(1, 2, 5, 6)

Colocada en el mapa tenemos agrupados en el mapa de la figura 6

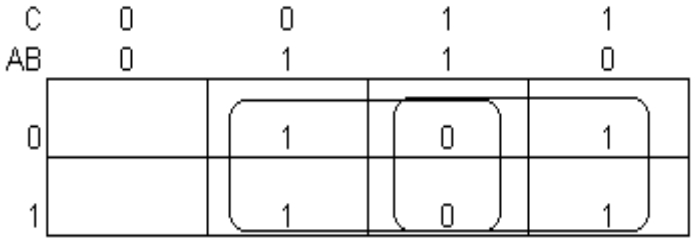


Fig. 6

Y aplicando Reed- Muller con mapas-k llegamos:

F = c ⊕ b

Un procedimiento para usar mapa-K para encontrar la simplificación de una expresión lógica usando una compuerta EXOR (por inspección) es buscar una agrupación de 1s' con 0s' eso podría simplificarse si un cuadrado extra o cuadrados entre ellos están llenos con 1s combinado los términos. Esto es aplicable si sólo una EXOR se necesita representar en una función. Los ejemplos siguientes muestran un acercamiento más general donde más de una EXOR se necesita.

EJEMPLO 5

F(A, B, C, D) = ∑ (2, 3, 5, 7, 8, 12, 13, 14)

Ploteandolos en el mapa de la figura 7:

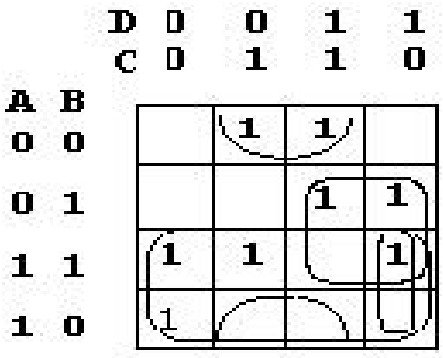


Fig. 7

Se han agrupado los unos con ceros (el cuadro vacío se asume cero).

Donde la función con EXOR para cuatro variables nos queda:

F = A ⊕ B'C ⊕ BD ⊕ AC'D

EJEMPLO 6

F = (A, B, C, D) = ∑(1, 2, 3, 4, 8, 12)

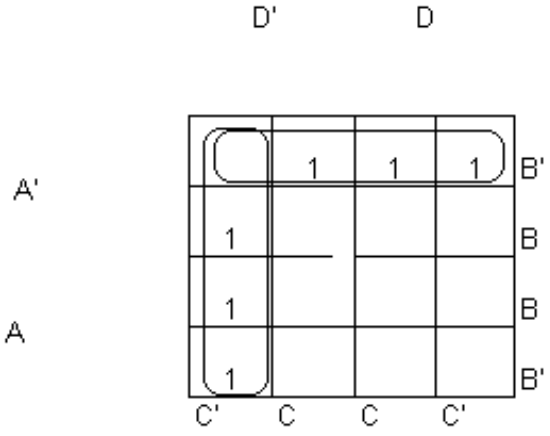


Fig. 8

F = C'D' ⊕ A'B'

Dos ejemplos mas son mostrados en Figuras 9 y 10 .

Considere la función trazada en el K-mapa mostrado en Figura 9a. Nosotros podríamos empezar trazando las variable en un cuadro mostrado a las 9b, adicionando un 0, con esto cubre tres de los 1s en el mapa. Luego agrupamos la variable c como es mostrado a la 9c. Esto completa que el cero se agrupe dos veces como regla, pero el 1 de la posición ab'c se esta agrupando 2 veces, por lo tanto lo agrupamos solo como es mostrado en la figura 9d.

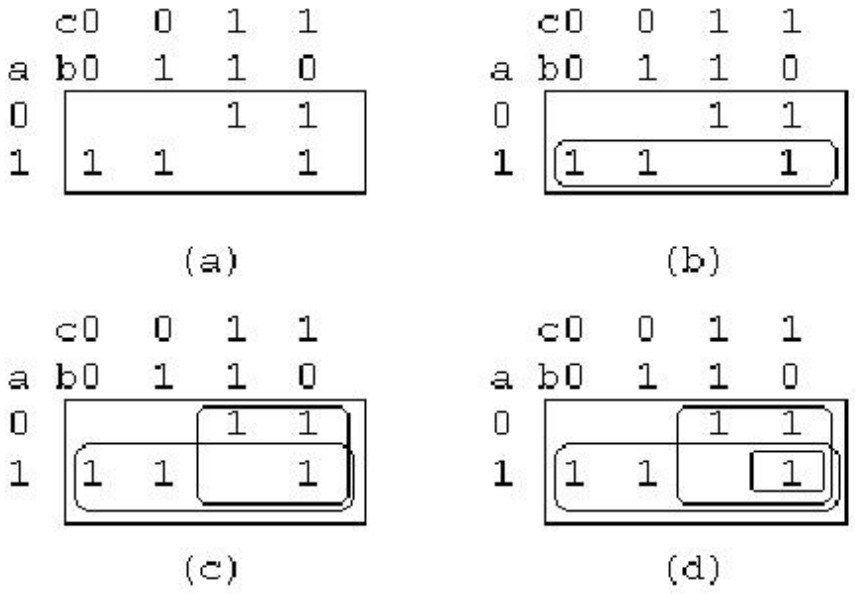


Fig. 9

Y la salida de la expresión booleana en términos de la función EXOR es:

$F = A \oplus C \oplus AB'C$

Otro ejemplo se muestra en Figura 10. Los mismos pasos se toman como se hizo en Figura 9. La función mostrada es trazado en un mapa-K (figura 10a). Se empieza como antes trazando 0 como es mostrado a la figura 10b. Esto cubre dos de los 1s en la función . Luego trazamos la variable b como se muestra en la figura 10c. Finalmente agregamos la porción para la variable c . Cuando usted puede ver que esto restaura que el 1 de los términos de la etapa abc donde debe ser agrupaciones impar, y el 0 del término a'bc es agrupado de forma par.

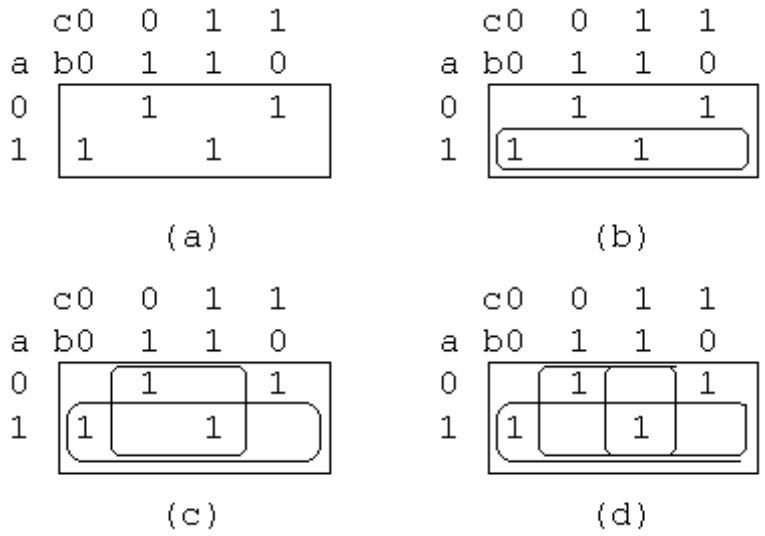


Fig. 10

donde Y = a ⊕ b ⊕ c

CAPITULO II

2. ANÁLISIS PARA EL DESARROLLO DEL PROGRAMA

2.1 POLARIDAD POSITIVA

Partiendo del modelo de Reed-Muller

$Y = k0 \oplus k1a \oplus k2b \oplus k3ab$

Dándole valores a Y

$Y = \sum(1, 2, 3)$

Y para los términos “ba” las 4 posibles combinaciones

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

Sustituyendo cada valor de la tabla en la ecuación de Reed-Muller tenemos:

Caso 1

a = 0
b = 0
Y = 0

$0 = K0$ (1)

Caso 2

a = 1
b = 0
Y = 1

$1 = K0 \oplus K1$ (2)

Caso 3

a = 0
b = 1
Y = 1

$1 = K0 \oplus K2$ (3)

Caso 4

a = 1
b = 1
Y = 1

$1 = K0 \oplus K1 \oplus K2 \oplus K3$ (4)

Y obtenemos 4 ecuaciones

$$0 = K0 \quad (1)$$
$$1 = K0 \oplus K1 \quad (2)$$
$$1 = K0 \oplus K2 \quad (3)$$
$$1 = K0 \oplus K1 \oplus K2 \oplus K3 \quad (4)$$

Pero como el sistema tiene que ser sistemático se utiliza métodos matriciales .
Llevando estas 4 ecuaciones a matrices tenemos:

$$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix}$$

Este arreglo matricial guarda similitudes con los arreglos usados en Álgebra lineal, y gracias a los cuales es posible resolver sistemas de ecuaciones. Sin embargo, no se pueden aplicar exactamente las mismas reglas, porque se trata de expresiones booleanas. Sin embargo, hay al menos 2 funciones que tienen sus paralelos en el álgebra booleana. La multiplicación tiene un comportamiento muy similar al de la función and, y la suma algebraica tiene propiedades de linealidad y superposición similares a los de la función EXOR. Por tanto, si se utilizan en las reglas de resolución de ecuaciones las funciones AND en lugar de la multiplicación, y las funciones EXOR en lugar de la de las sumas, es posible resolver los sistemas de ecuaciones planteados en forma matricial. Las alternativas involucrarían usar métodos como los de Kramer, Gauss o Gauss-Jordan. Aunque sean factibles, dado que la matriz que multiplica a los coeficientes K depende únicamente de las combinaciones de las variables de entrada, y no de los valores de la función de salida, esta matriz será la misma para un número n de variables de entrada, y de ser posible obtener su inversa, esta a su vez debería ser la misma para un número n de variables de entrada.

Así, aplicando el método de inversión de Gauss, se plantearía:

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

El método nos indica que debemos de llevar a cabo sumas entre los términos de las filas de tal forma que se obtenga en le lugar de la matriz original la matriz identidad, y en el lugar de la matriz identidad deberá quedar la inversa de la matriz original. La diferencia fundamental radica en que en lugar de llevar a cabo sumas o restas se llevarán a cabo funciones EXOR entre los términos de cada fila del arreglo matricial.

Una vez obtenida la inversa se obtiene la respuesta por el producto matricial de la inversa por el vector de la tabla de verdad de la función que se está analizando. Las reglas del producto matricial se restringen a usar la AND como producto y la exor en lugar de suma.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A = Matriz original
I = Matriz Identidad

Realizando la operación EXOR del arreglo matricial:

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Realizamos EXOR con fila 1 y el resto:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Exor fila 2 con 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Exor fila 3 y 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Luego después de obtener la matriz identidad, donde estaba la matriz original, también se obtiene la matriz inversa donde estaba la matriz identidad, pero se observa que la matriz inversa es igual a la matriz original.

Por tanto:

$$A = A^{-1}$$

Entonces: $A\,K = Y$

Donde:

$$K = \begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Y " Y " dependerá de los valores de salida

Despejando K

$$K = A^{-1}Y$$

Realizando esta ecuación matricialmente tenemos:

$$\begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Resolviendo la matriz booleanamente :

La primera fila por la columna:

[(1.0) ⊕ (0.1) ⊕ (0.1) ⊕ (0.1)] = 0 = K0

[(1.0) ⊕ (1.1) ⊕ (0.1) ⊕ (0.1)] = 1 = K1

[(1.0) ⊕ (0.1) ⊕ (1.1) ⊕ (0.1)] = 1 = K2

[(1.0) ⊕ (1.1) ⊕ (1.1) ⊕ (1.1)] = 1 = K3

Nuevamente de la ecuación modelo de Reed-Muller tenemos:

Y = k0 ⊕ k1a ⊕ k2b ⊕ k3ab

= 0 ⊕ (1)(a) ⊕ (1)(b) ⊕ (1)(a)(b)

= a ⊕ b ⊕ ab

2.2 POLARIDAD POSITIVA MEDIANTE KRONECKER

Todo lo anterior es valido y confiable en el momento de aplicarlo para RMPP, pero resulta ser un proceso muy largo, haciendo que la máquina se tarde a la hora de ejecutar alguna función, y más cuando hay muchas variables.

Por tal razón se ha implementado RMPP a partir de expansión Kronecker:

Aquí siempre es válido la expresión

$$A = A^{-1}$$

Entonces

AK = Y

Donde

$$K = \begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Si despejamos K

$$K = A^{-1}Y$$
$$= A\ Y$$

Ahora para construir la matriz A

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Partimos mediante expresiones Kronecker, para la expresión más simple que es una variable:

$Y = K0 \oplus K1A$

Si a esta ecuación la representamos matricialmente tenemos

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Y esta sería la matriz para una variable. Para dos variables tendríamos el producto Kronecker de la misma matriz así:

$$A2 = \left(A1 \otimes A1 \right)$$

Y para un valor de n variables tendríamos

$$An = A_{n-1} \otimes A1$$

Tomemos como ejemplo

$Y = \sum(1, 2, 3)$
Aquí tenemos dos variables

El producto Kronecker sería:

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Y multiplicando esta matriz por Y tenemos:

$$K = A^{-1}Y$$

$$K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Resolviendo la matriz llegamos:

$= a \oplus b \oplus ab$

El algoritmo incluido en el programa desarrolla este proceso.

Como ya se había descubierto anteriormente para la matriz en el caso de polaridad positiva, esta es igual a su inversa, por lo tanto, no hay necesidad de obtener la inversa de la matriz, y el resto del procedimiento consiste únicamente en llevar a cabo el producto matricial del vector de la tabla de verdad Y por la matriz A.

2.3 POLARIDAD FIJA

Con la siguiente secuencia:

$1 \oplus a' \oplus b \oplus a'b$

tenemos

$$\left(\begin{array}{cccc|cccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Luego aplicando funciones EXOR filas con filas llegamos a la matriz inversa

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Se observa que para el caso de polaridad fija la matriz original es diferente a la matriz inversa.

$$A^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Y realizando $K = A^{-1} Y$

$$\begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Dando como resultado

K0 = 1

K1 = 1

K2 = 0

K3 = 1

Nuevamente de la ecuación modelo de Reed-Muller tenemos:

Y = k0 ⊕ k1a ⊕ k2b ⊕ k3ab

= 1 ⊕ (1)(a) ⊕ (0)(b) ⊕ (1)(a)(b)

= 1 ⊕ a' ⊕ a' b

2.4 POLARIDAD FIJA MEDIANTE KRONECKER

Partimos de la matriz básica:

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = k0 \oplus k1A$$

Ahora si negamos la variable tendríamos

K0 ⊕ K1A'

$$A' = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Si sacamos la inversa a la matriz básica

$$A = A^{-1}$$

$$A^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Luego sacando la inversa a la matriz negada tenemos:

$$(A')^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Consideremos ahora una función en la cual una variable sea negada:

B A'

$$B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad A' = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Realizando Kronecker

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^{-1} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Observemos que es la misma matriz inversa que se obtuvo en el análisis anterior para polaridad fija.

Luego efectuamos la operación de las matrices:

$$\begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Dando como resultado

$= 1 \oplus a' \oplus a' b$

2.5 POLARIDAD MIXTA

Para una combinación:

1 a b ba'

tenemos las siguientes matrices

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Realizando operaciones EXOR con las filas, para obtener la matriz inversa llegamos:

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

Y obtenemos los valores de las constantes Kn de la ecuación modelo de Reed-Muller: **Y = k0 ⊕ k1a ⊕ k2b ⊕ k3ab**

$$K = A^{-1} Y$$

$$\begin{pmatrix} K0 \\ K1 \\ K2 \\ K3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Efectuando cada fila por la columna tenemos:

F = (b, a) = ∑(1, 2, 3)

Y = a ⊕ ba'

2.6 POLARIDAD MIXTA MEDIANTE KRONECKER

Para este caso se desarrollarán tres posibles métodos de solución :

a) La solución propuesta, en la cual se aprovecha de escoger términos de la polaridad mixta que formen una matriz identidad, entonces, al igual que para el caso de polaridad positiva resulta fácil el desarrollo de la solución.

Ej.

Se tiene una función Y = ∑(1,2,3)

Su matriz identidad será:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

y el vector Y será = [0 1 1 1]

operando tenemos:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Llegando a: K = B'A ⊕ BA` ⊕ BA

B) Una segunda solución en la cual se le permite al usuario introducir los minterminos que considere pertinentes, y el programa se encargará de generar a partir de ellos sus respectivas tablas de verdad, y a partir de estas, las columnas de la matriz A. Se corre el riesgo de que si el usuario no escoge adecuadamente los términos de la matriz A no arroje una solución para el caso.

c) Una tercera alternativa consiste en que se desista de usar métodos matriciales y se aproveche el siguiente teorema:

Si g y h son funciones tales que

$f = g \oplus h$

Entonces también será cierto que

G ⊕ f = h

Así, si se tiene la tabla de verdad de F, y se conoce la tabla de verdad de un posible término de el polinomio de Reed-Muller, entonces h constituirá un residuo de la operación de g y f. De preferencia se buscara un residuo que tenga un menor número de unos que la función original. Si se repite el proceso con h, probando con un término g2, se puede llegar a obtener un h2 que tenga un menor número de 1 que h. De repetirse el proceso indefinidamente, entonces seria posible que el enésima valor de h sea cero. Esto significa que

F = g ⊕ g2 ⊕ g3

Lo cual es un polinomio de Reed-Muller. Los distintos valores de g pueden escogerse en las diferentes combinaciones que resulten de 3^n posibles términos que se pueden generar con n

variables. El programa va probando uno por uno de tal forma que existan altas posibilidades de escoger términos con polaridad mixta, y llegar a una solución válida para estos casos.

Existen algunas sugerencias para instalar el programa y ejecutarlo, el programa contiene 470Kbytes.

Los requerimientos de la PC son: sistema operativo Windows 98 o 2000, el disco duro con una capacidad de 10 Gbytes, monitor VGA , el cpu Pentium como mínimo, se sugiere que sea un Pentium III, una memoria ram alta, aproximadamente 64Mgytes.

Nota: En el caso de la matriz para el arreglo de polaridad positiva siempre la matriz A es igual a su inversa, pero si se usara polaridad fija con alguna de las variables negadas, o polaridad mixta, este principio no se cumple.

CAPITULO III

3 OTRAS APLICACIONES DE POLINOMIO REED-MULLER

3.1 DETECCION Y CORRECCION DE ERRORES

3.1.1 INTRODUCCION

Entre las aplicaciones de polinomios de Reed-Muller se encuentran las que involucran su uso en técnicas de detección y corrección de errores con fines de transmisión, y en la implementación de circuitos para otras áreas, los llamados FPGA, representan una alternativa de implementar funciones lógicas, muchas de las cuales se pueden expresar en la forma de polinomios de Reed-Muller.

Si los datos a la salida de un sistema de comunicación contienen errores que son demasiados frecuentes, es usual que se puedan reducir con el uso de dos técnicas principales:

- Solicitud de repetición automática (ARQ)
- Detección y corrección de errores de transmisión.

Un sistema ARQ consiste cuando en un circuito receptor detecta errores en un bloque de datos, solicitando que se transmita el bloque de datos nuevamente. Y un sistema de corrección, los datos se codifican dé modo que el receptor pueda detectar y corregir los errores.

La elección entre usar la técnica ARQ y corrección depende de la aplicación particular. Con frecuencia se usa la ARQ en sistema de comunicación por computadora porque es relativamente barato ponerlo en práctica y por lo general existe un canal duplex de modo que el extremo receptor pueda transmitir de regreso una confirmación ACK (acknowledgment) de que los datos se recibieron correctamente.

La técnica de corrección se utiliza en sistemas de largas demoras en la transmisión, por que si se utilizara la técnica ARQ, la velocidad de transferencia de datos efectiva seria muy lenta; el transmisor tendría largos periodos de inactividad mientras espera un reconocimiento ACK.

Para poder seleccionar un código para errores útil y eficaz hay que tener presente tres cosas:

a) Que detecte y corrija los errores introducidos en un canal de datos
b) Una transmisión eficaz de datos
c) Una fácil codificación y decodificación del código

El primer punto es el más importante, ya que lo ideal es tener un código que sea capaz de corregir todos los errores de un canal debido al ruido. El segundo es un problema de eficacia, ya que no seria agradable perder tiempo al enviar datos extraños. El tercero tiene razones naturales debido a la facilidad que se presentaría al utilizar un equipo útil y eficaz. Si el código presenta estos tres casos se considera un código equilibrado a la hora de detectar y corregir errores.

3.1.2 FUNDAMENTOS

El origen de la teoría de códigos correctores de errores se encuentra en los trabajos de *Golay, Hamming y Shannon.*

A mediados del presente siglo, tanto en su vertiente probabilista como en la algebraica, y desde su inicio el tema ha sido siempre un problema de ingeniería, con aplicación tanto en la *transmisión de información* (ingeniería en telecomunicaciones) como en el almacenamiento de la misma en *soporte digital* (ingeniería en computación), siendo su finalidad el preservar la calidad de la información y las comunicaciones contra la amenaza del ruido, la distorsión o el deterioro del medio. No obstante, el desarrollo de dicha teoría se ha realizado gracias a la utilización de *técnicas matemáticas* cada vez más sofisticadas; estas técnicas recorren múltiples campos de la matemática, que van desde la teoría de probabilidades, el cálculo combinatorio o el álgebra lineal hasta la aritmética, la teoría de cuerpos o la geometría algebraica.

En un principio, la teoría empezó con un enfoque esencialmente probabilístico con la teoría de Shannon, pero pasó posteriormente a un enfoque más algebraico, surgiendo entonces muchos ejemplos prácticos como los códigos de *Golay* y de *Hamming*, los *códigos cíclicos* y *BCH*, o los códigos de *Reed-Salomón* y posteriormente ***Reed-Muller***.

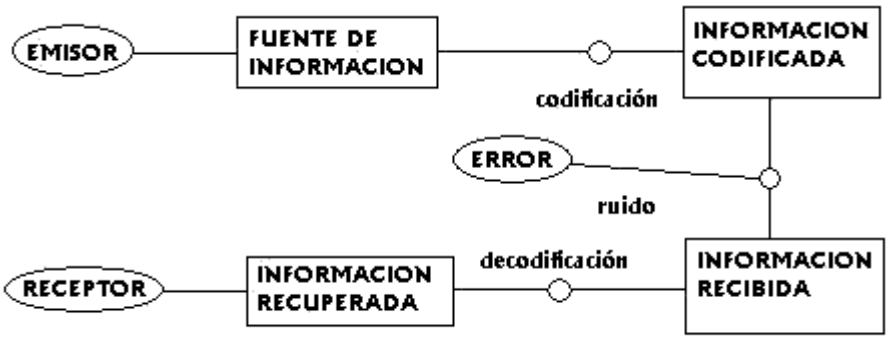
El modelo general de un **sistema de protección contra los errores** producidos en el almacenamiento o la transmisión de datos a través de un *canal sometido a ruido* comprende los siguientes elementos:

1. Una **fuente de información** que genera una cadena o *palabra* de longitud k con símbolos o letras en un *alfabeto.*
2. Un proceso de **codificación** que transforma unívocamente el *mensaje* anterior en otro de longitud $n > k$, sobre el mismo alfabeto u otro diferente, y al que se ha añadido *información redundante* suficiente como para poder detectar y *corregir* un número razonable de *errores* que pudieran producirse durante el proceso de almacenamiento o de transmisión.
3. Un **canal** a través del cual se transmite el mensaje anteriormente codificado o en el cual se almacena dicha información, la cual puede sufrir algunos errores debidos al *ruido* existente en dicho canal, o al *deterioro* del mismo en el caso de almacenamiento en un soporte digital.
4. Un proceso de **decodificación** que asigna al mensaje distorsionado por el canal otro mensaje que, en caso de haberse producido pocos errores, es el mensaje introducido inicialmente en el canal, permitiéndonos así *recuperar la información* transmitida o almacenada, según el caso.

Por lo tanto son cuatro elementos involucrados:

a) La fuente
b) La codificación
c) El canal
d) La decodificación

Todo esto puede ser ilustrado mediante el siguiente esquema:



Un código de detección y corrección de errores es un algoritmo para expresar una secuencia de números tales que cualquier error que se introduzca que se detecte y corrija (dentro de ciertas limitaciones), llamándosele así código de bloque, el código Reed-Muller es un ejemplo de ello y para eso hace uso de operaciones AND/EXOR para la codificación y la decodificación.

La detección de errores es mucho más simple que la corrección.

Un ejemplo de este código lo tenemos en las pruebas de Mariner 4[1].

En 1965 el Mariner 4 envió a la tierra las primeras imágenes del planeta Marte, fue un gran avance, aunque las fotografías recibidas nos parecerían hoy de una mala calidad. Pero después (1969-1972) los mariner 6, 7 y 9 repetían el experimento.

Sin embargo las imágenes fueron en este caso, aunque todavía lejos de las excelentes imágenes que hoy podemos recibir, de mejor calidad. La razón principal de esta mejora fue que para la transmisión de las últimas se había utilizado un código superior de corrección, capaz de corregir hasta 5 bits erróneos de cada secuencia de 32 bits.

Los avances actuales en la transmisión digital no serian posible sin el desarrollo de los códigos correctores, por ejemplo la telefonía móvil, la televisión digital, los sistemas de navegación aérea entre otros.

3.1.3 CODIGO REED-MULLER

Los códigos Reed-Muller son algunos de los correctores más antiguos, con utilidad al enviar información a distancias muy largas, en el cual puedan ocurrir errores en aplicaciones de Telecomunicación.

Se inventaron estos códigos de Reed-Muller en 1954 por D. E. Muller y S. Reed.

[1] Mariner 4, 6, 7, 9: Código Reed-Muller utilizado por el programa Mariner de la NASA

En 1972 este código fue usado por mariner 9 para transmitir fotografías en blanco y negro del planeta Marte, son fáciles de decodificar sobre todo los de primer orden que son especialmente eficaces.

3.1.4 Códigos Reed-Muller de primer orden

Los códigos Reed-Muller binarios de primer orden constituyen una familia de códigos RM[2] (m) donde m $\geq$ 1, cuya característica principal es que tiene una alta capacidad correctora. Para cada entero positivo m y cada entero r satisface que 0 $\leq$ r $\leq$ m, donde " r " indica el orden del código Reed-Muller; RM(r, m), en el cual es un código con parámetros lineales [2^m, (m/ 0) + (m/1) +... +(m/r), 2^(m-r)].
El factor RM(1, 5) fue usado por Mariner 9 de la NASA para transmitir imágenes en blanco y negro del planeta Marte.

3.1.5 Generación de matrices

Definición 4.1
Para m $\geq$0 se define de forma recursiva la matriz G(m):

$$G(0) = (1) \qquad (4.1)$$

Definición 4.2

Y para m $\geq$ 1:

$$G(m) = \begin{Bmatrix} G(m-1) & G(m-1) \\ 0....0 & 1.\ 1 \end{Bmatrix}$$

Para m $\geq$1, el m-ésimo código de Reed-muller de primer orden es el código sobre F2 RM (m) que tiene como matriz generadora.
Por ejemplo, para m = 3, la matriz es:

[2] Reed-Muller

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Donde su longitud esta definida por la cantidad de columnas:

$$\text{Columnas} = 2^{m} \qquad (4.2)$$

Y su dimensión esta definida por la cantidad de filas:

$$\text{Filas} = m + 1 \qquad (4.3)$$

Para generar la matriz del código R(1,1) es:

$$G(m) = \begin{pmatrix} G(m-1) & G(m-1) \\ 0 & 1 \end{pmatrix}$$

Donde, si aplicamos la ecuación (4.1) a los términos dentro del corchete tenemos:

$$G(1) = \begin{pmatrix} G(1-1) & G(1-1) \\ 0 & 1 \end{pmatrix}$$

$$G(1) = \begin{pmatrix} G(0) & G(0) \\ 0 & 1 \end{pmatrix}$$

$$G(1) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Y esta sería la matriz generada para un código Reed-Muller RM(1,1)

Ahora para un código RM(1,2), ya cambia;
Tendía una longitud de cuatro y una dimensión de tres

$$G(2) = \begin{pmatrix} G(m-1) & G(m-1) \\ 0 & 1 \end{pmatrix}$$

$$G(2) = \begin{pmatrix} G(1) & G(1) \\ 0 & 1 \end{pmatrix}$$

$$G(2) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Igual sería para generar una matriz RM(3):

$$G(3) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Para definir la matriz de la codificación de RM (r; m), la primera fila de la codificación será 1, la longitud del vector 2^ m con todas las entradas igual a 1. Si r es igual a 0, entonces esta fila es la única en la matriz de la codificación.

A estas filas se les asocia un vector, por ejemplo a la 1º fila " 1" a la 2º fila X1, etc. Por lo tanto tendríamos una función M de todos los monomios de grado r:

M = (1, v1, v2, ….. , vm, v1v2…. , vm-1,vm, v1v2v3, ….. , v1v2 …. Vm)
Y para el caso de G(3) tenemos:

1 = 11111111
X3 = 01010101
X2 = 00110011
X1 = 00001111

Estos vectores pueden ser considerados como las filas de una matriz generada.

3. 1 .6 Codificación de Reed-Muller

Cuando se va a codificar un mensaje utilizando Reed-Muller para RM(r; m) se operan la función M junto al mensaje (m), dando lugar al mensaje ya codificado Mc.

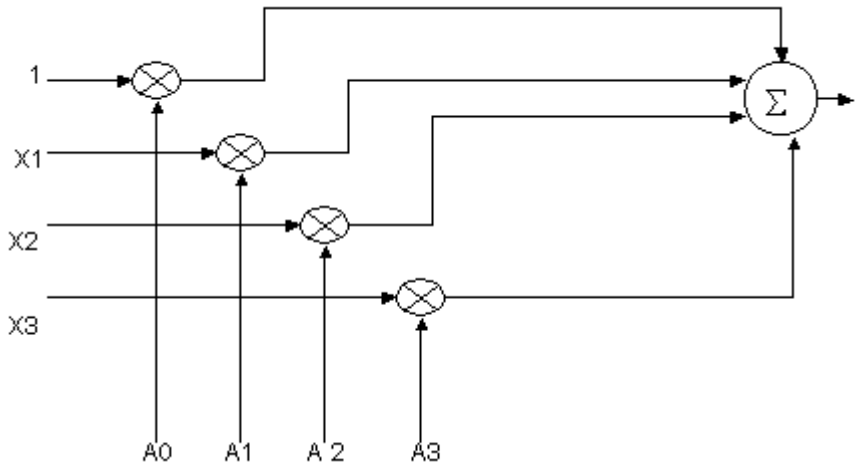
$$Mc = \oplus \sum_{I=1}^{k} mi.\ Xi \qquad (4.4)$$

Por ejemplo utilizando RM(1,3) con un mensaje m = (0110) tenemos

$$(0*1) \oplus (1*X1) \oplus (1*X2) \oplus (0*X3) = Mc$$

0 * (11111111) $\oplus$ 1 * (00001111) $\oplus$ 1* (00110011) $\oplus$ 0* (01010101) = (00111100):

El siguiente bloque muestra un RM de primer orden, donde el arreglo es con varias compuertas AND junto con una compuerta EXOR y este arreglo se vuelve más grande a medida crecen los datos llegando a una necesidad de colocar dispositivos programables.



Donde el mensaje esta comprendida en los bits A0, A1, A2, A3 (A0 es el bits menos significativo), y ∑ denota una sumatoria EXOR.

3.1.7 Decodificación de Reed-Muller

La Decodificación del mensajes es más compleja que la codificación de este . La teoría de la codificación y decodificación esta basado en la distancia entre los vectores. La distancia entre dos vectores cualquiera es el número de veces en los cuales dos vectores tienen valores diferentes. La distancia entre dos códigos para el código R(r, m) es 2 ^ (m-r).

El método de decodificación usado no es muy eficaz, pero es una implementación directa. Este verifica cada fila de la matriz codificada y usando lógica en su mayoría, para determinar si esa fila se usó formando el mensaje de la codificación. Con esto es posible, que el mensaje codificado tenga un error menos al compararse con el código del mensaje original . Este método de decodificación es dado por el siguiente algoritmo :

Aplicando los Pasos 1 y 2 , a cada fila de la matriz, empezando desde la última y hacia arriba.

Paso 1. Se escoge una fila del RM(r; m) de la matriz codificada. Encontramos la característica de 2^(m - r) (este proceso se describe debajo) para esa fila, y entonces se hace la operación AND de cada uno de esas filas con el mensaje puesto en código.

Paso 2. Tomando el valor mayor de la operación AND, y asignando ese valor al coeficiente de la fila.

Paso 3. Después de hacer los pasos 1 y 2 para cada fila excepto la primera fila de la matriz, se multiplica cada coeficiente por su fila correspondiendo y sumando los vectores resultantes para formar My. Sume este resultado al mensaje codificado recibido. Si el vector resultante tiene más unos que ceros, entonces el coeficiente de la fila de arriba es 1, por otra parte es 0 . Sumando la fila de arriba, multiplicado por su coeficiente, da como resultado el mensaje original My. Así, se pueden identificar los errores. El vector formado por la sucesión de coeficientes empezando de la fila de arriba de la matriz codificada y terminando con la última fila, es el mensaje original.

Para encontrar los vectores característicos de cualquier fila de la matriz, tome el monomio r que esta asociado con la fila de la matriz de la codificación. Entonces, tome E para establecer todos las Xi, que no está en el monomio r, pero está en la matriz de la codificación.

EJEMPLO. Si el mensaje original es m = (0110) usando R(1; 3), tenemos la matriz:

$$G(3)=\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Luego, el mensaje lo operamos por cada fila respectiva:

$$Mc = (0*1) \oplus (1*X1) \oplus (1*X2) \oplus (0*X3)$$

$$0 * (11111111) \oplus 1 * (00001111) \oplus 1* (00110011) \oplus 0* (01010101) = (00111100):$$

el mensaje codificado es Mc = (00111100). La distancia es 2 ^ (3–1) = 4, este código puede corregir un error. Permitiendo que el mensaje codificado después del error sea Me

= (10111100): Los vectores característicos de la segunda fila sea x3 = (01010101), y las posibles combinaciones de las filas restantes son x1x2, x1' x2, x1x2', y x1x2. Dando lugar a lo siguiente:

X1 = 00001111 X1'= 11110000

X2= 00110011 X2' = 11001100

Si realizamos operación AND en las posibles combinaciones llegamos:

X1X2 = (11000000)

X1'X2 = (00001100)

X1X2' = (00110000)

X1'X2' = (00001100)

Luego realizando la operación AND entre la respuesta de las combinaciones y el mensaje Me, tenemos:

(11000000) . (10111100) = 10000000

(00001100) . (10111100) = 00001100

(00110000) . (10111100) = 00110000

(00000011) . (10111100) = 00000000

Después a dicha respuesta se le aplica operación EXOR, ella misma:

1 0 0 0 0 0 0 0 = 1

0 0 0 0 1 1 0 0 = 0

0 0 1 1 0 0 0 0 = 0

0 0 0 0 0 0 0 0 = 0

Teniendo como respuesta una cantidad de ceros mucho mayores que los unos, por lo tanto se puede concluir que el coeficiente X3 es 0.

Haciendo lo mismo para la siguiente fila de la matriz, x2 = (00110011), obtenemos los vectores característicos de X1 y X3: x1 x3, x1x3' x1' x3 y X1' X3'. Realizando la

operación AND llegamos: (10100000), (01010000), (00001010), y (00000101), respectivamente. Efectuando ahora la operación AND junto al mensaje Me:

(10100000) . (10111100) = 0; (01010000) . (10111100) = 1;
(00001010) . (10111100) = 1; (00000101) . (10111100) = 1:

Y concluimos que el coeficiente X2 es 1.
Haciendo lo mismo para el coeficiente X1:
X1 = 00001111

(10001000) . (10111100) = 0; (00100010) . (10111100) = 1;
(01000100) . (10111100) = 1; (00010001) . (10111100) = 1:

X1 es también 1
Luego operamos el valor del coeficiente con su fila respectiva:

$$(0 * X3) \oplus (1 * X2) \oplus (1 * X1) =$$

$$(0 * 10101010) \oplus (1 * 11001100) \oplus (1 * 11110000) = My$$

My = (00111100).

Entonces observamos que la suma EXOR de My y Me es igual a:

$$(00111100) \oplus (10111100) = (10000000):$$

Este mensaje tiene más ceros que unos, así que los coeficientes de la primera fila de la matriz codificada es cero. Así nosotros podemos reunir los coeficientes para las cuatro filas de la matriz, 0,1,1, y 0, y ver que el mensaje original era (0110). También podemos ver que el error estaba en la primera posición del mensaje libre de errores Mc = (00111100).
Por lo tanto el código RM tiene una distancia mínima de 2^(m-1), y una capacidad correctora de 2^(m - 2) - 1. (4.5)

Para los casos en los cuales RM sea de r orden, la ecuación (4.5) se cumple siempre y cuando $m \geq r$.

EJEMPLO 2

Considere el mensaje M = 1011, codificando con RM(1,3) tenemos:

Número de filas = m+1 = 4

Número de columnas = 2 ^m = 8

Su matriz generadora es:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Donde:

X3 = 01010101 ; X2 = 00110011 ; X1 = 00001111

1(11111111) $\oplus$ 1(00001111) $\oplus$ 0(00110011) $\oplus$ 1(01010101) = Mc

10100101 = Mc

Mc: Mensaje codificado

Ahora, asumiendo que el mensaje lleva un error (Me), antes de llegar al receptor

Me = 10100111

Decodificamos el mensaje:

X3 = 01010101

Las combinaciones de los otros vectores restantes con su producto es:

X1 . X2 = 00000011

X1 . X2' = 00001100

X1' . X2 = 00110000

X1' . X2' = 11000000

(00000011).(10100111) = 0

(00001100).(10100111) = 1

(00110000).(10100111) = 1

(11000000).(10100111) = 1

= X3 = 1

Ahora para X2 tenemos

X2 = 00110011

X1. X3 = 00000101
X1. X3' = 00001010
X1'.X3 = 01010000
X1'.X3' = 10100000

Realizando nuevamente operación AND luego a su respuesta la aplicamos operación EXOR:

(00000101).(10100111) = 0

(00001010).(10100111) = 1

(01010000).(10100111) = 0

(10100000).(10100111) = 0

= X2 = 0

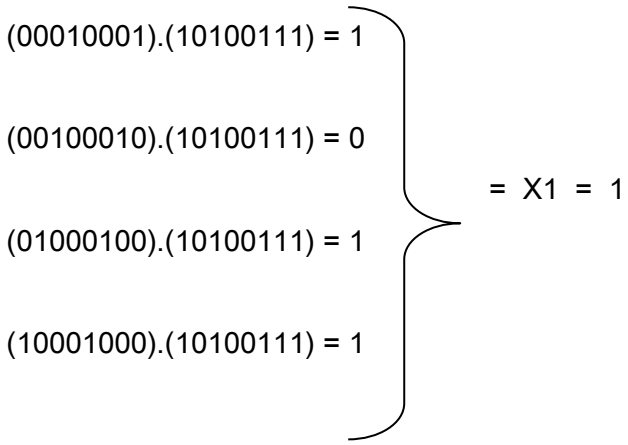
X1 = 00001111

X2 . X3 = 00010001

X2 . X3' = 00100010

X2' . X3 = 01000100

X2' . X3' = 10001000



Por tanto tenemos:

X3 = 1

X2 = 0

X1 = 1

Multiplicamos este valor por su valor matricial:

1.(01010101) ⊕ 0.(00110011) ⊕ 1.(00001111)

= 01010101 ⊕ 00001111

= 01011010 = My

My ⊕ Me = 01011010 ⊕ 10100111 = 11111101

Por lo tanto, observamos que hay más unos que ceros y esto no determina que el primer valor del mensaje es " 1 " .

El mensaje es : X3, X2, X1, 1 = 1, 0, 1, 1

Y se detectó un error en la segunda posición del mensaje codificado :

1111110<u>1</u>

EJEMPLO 3

Consideremos el siguiente ejemplo en el cual se introduce dos errores

Mensaje = 1010

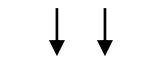
RM(1,3)

Codificando el mensaje obtenemos lo siguiente :

Mc = 01011010

Considerando dos errores:

Me = 11001010



X3 = 01010101

X1 . X2 = 00000011

X1 . X2' = 00001100

X1' . X2 = 00110000

X1' . X2' = 11000000

(00000011).(11001010) = 1

(00001100).(11001010) = 1

(00110000).(11001010) = 0

(11000000).(11001010) = 0

= X3 = ?

No se sabe que valor toma X3

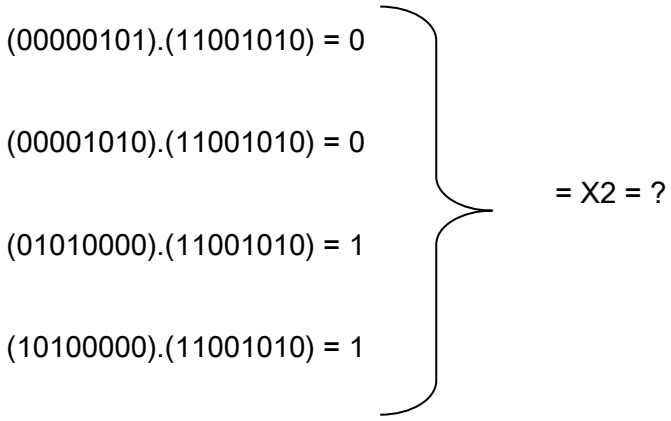
X2 = 00110011

X1. X3 = 00000101

X1. X3’ = 00001010

X1’.X3 = 01010000

X1’.X3’ = 10100000



Nuevamente no se sabe el valor acertado de X2 debido a que hay un mismo valor de ceros y unos. Por lo tanto RM(1,3), **no puede corregir más de un error.**

EJEMPLO 4

Considere un mensaje = 00001, para RM(1, 4).

Tenemos:

m = 4

r = 1

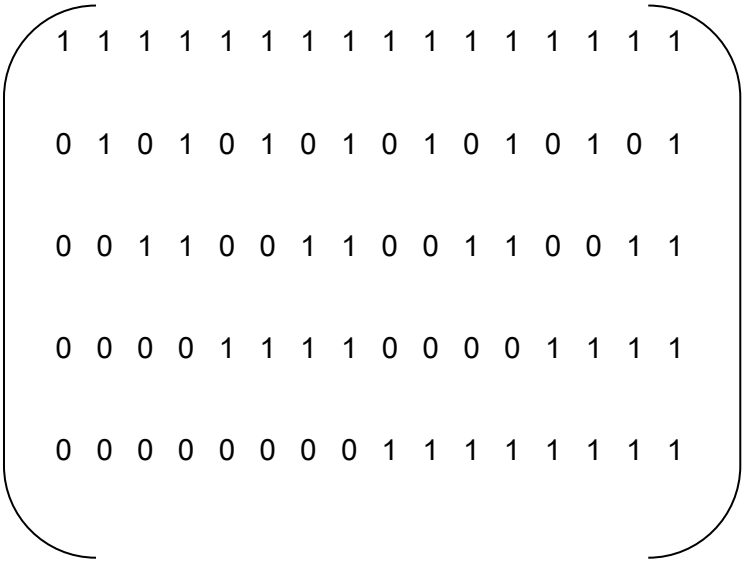
Filas = m + 1 = 5

Columnas = 2^ m = 16

Distancia = 2^(m-r) = 2^(4-1) = 8

Corrección = 2^(m-2) – 1 = 3

Su matriz es la siguiente:



Multiplicando los bits del mensaje por los vectores de la matriz tenemos:

1.(1111111111111111) ⊕ 0 ⊕ ....0 = 1111111111111111 = Mc

Consideremos el mensaje Mc con tres errores:

Me = 1<u>0</u>111111111<u>0</u>1111<u>0</u>

X4 = 01010101010101010

X1.X2.X3 = 0000000000000011

X1.X2.X3’ = 0000000000001100

X1.X2’.X3 = 0000000000110000

X1.X2’.X3’ = 0000000011000000

X1’.X2.X3 = 0000001100000000

X1’.X2.X3’ = 0000110000000000

X1’.X2’.X3 = 0011000000000000

X1’.X2’.X3’ = 1100000000000000

Realizando el producto punto, luego la operación EXOR; tenemos:

X4= 0

X3 = 0

X2 = 0

X1 = 0

Multiplicando los valores de los coeficientes con sus respectiva fila llegamos

0000000000000000

después

0000000000000000 ⊕ 1011111111011110 = 1011111111011110

Observamos que la cantidad de unos es mayor, por lo tanto el primer dato del mensaje es 1.

El mensaje recuperado es X4, X3, X2, X1, 1 = 00001

Y los errores están en el resultado de la operación EXOR de la función My y Me:

1011111111011110

En la tabla 4.1 se muestra algunas características para RM de diferente orden.

RM	DISTANCIA	CAPACIDAD CORRECTORA (BITS)
(1, 3)	4	1
(1,4)	8	3
(1, 6)	32	15
(2,5)	8	3
(2,8)	64	31
(3, 5)	4	1
(3, 9)	64	31
(5,8)	8	3
(6,10)	16	7

TABLA 4.1

Donde:

Distancia = 2^(m-r)

Capacidad correctora = 2^(m – (r +1)) - 1.

3.2 SISTEMA FPGA

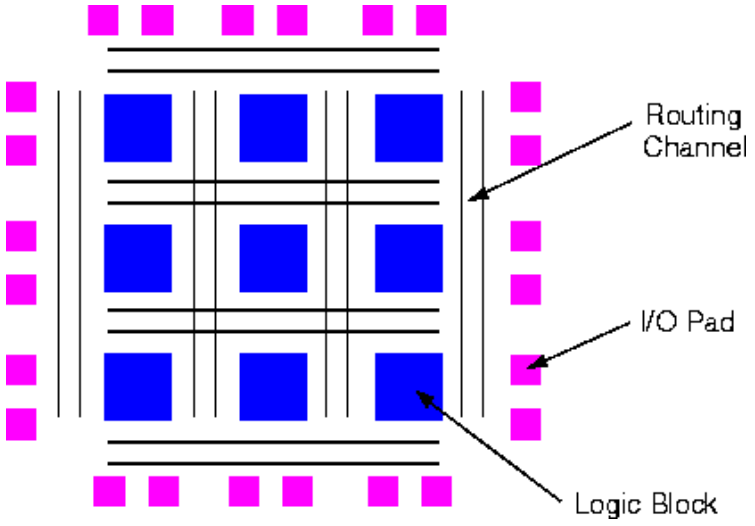
3.2.1 FUNDAMENTOS

Que es la lógica programable:

La lógica programable, como el nombre lo dice, es una familia de componentes que contiene conjunto de elementos lógicos (AND, OR, NOT, LATCH, FLIP-FLOP) que pueden configurarse en cualquier función lógica que el usuario desee y que el componente soporta. Hay varias clases de dispositivos lógicos programables: ASIC, PLA, PAL, GAL, PLD y por supuestos los FPGA[3].

Los FPGA, es una clase de circuitos integrados pionera de la Xilinx[4], en el que la función lógica es definida por el usuario usando un software desarrollado por Xiling. El arreglo de compuertas es otro tipo de circuitos integrados (IC), cuya función lógica se define durante el proceso industrial.

Los FPGA simplemente son matrices de puertas eléctricamente programables que contiene múltiples niveles de lógica. Las FPGA se caracterizan por altas densidades de puerta, alto rendimiento, un número grande de entrada y salida definidas por el usuario. Consisten en una serie de bloques, canales para vías y dos bloques como entrada y salida como se muestra en la figura 5.1.



[3] Field Programmable Gate Array

[4] Empresa Industrial dedicada a la fabricación de FPGA

Figura 5.1. Estructura de FPGA

La localización de un bloque lógico FPGA con sus pines se muestra en la figura 5.2 en donde cada entrada es accesible desde uno de los lados del bloque, mientras que los pines de salida pueden conectarse en ambos alambres del canal y en el canal de abajo del bloque lógico.

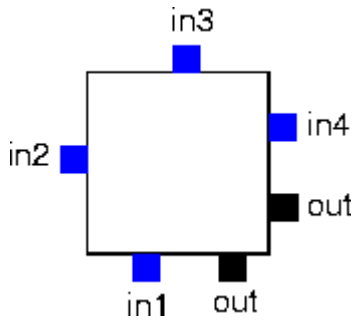


Figura 5.2 Localización de pines en un bloque lógico

Cada pin de entrada del bloque lógico puede conectarse a cualquier alambre de un canal adyacente a este, igual para los pines de salida.

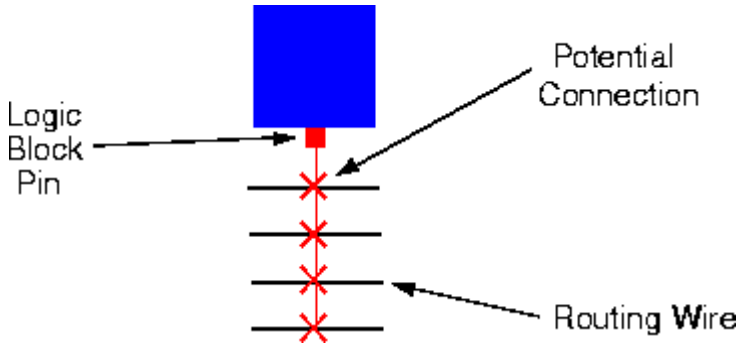


Figura 5.3 Interconexión del canal con un pín de entrada
de un bloque lógico

Cada segmento de línea es separado por un único bloque lógico antes que termine en una caja de interrupción. Encendiendo algunos de los interruptores programables dentro de una caja del interruptor, los caminos más largos pueden ser construidos.

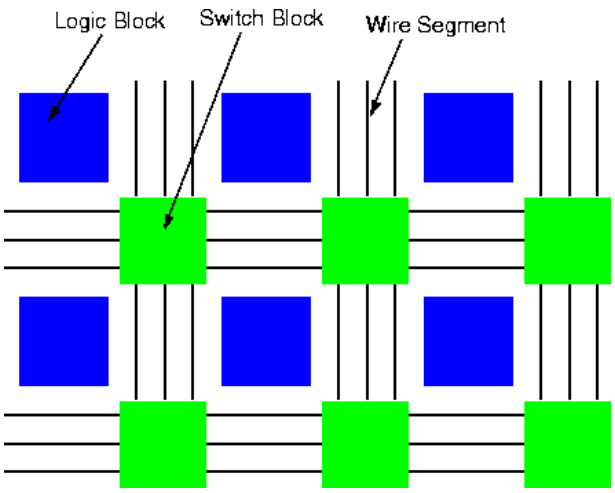


Figura 5.4

Siempre en la intersección de un canal vertical y horizontal hay una caja de interrupción (*switch box).* En esta arquitectura cuando las líneas entran a un *switch box,* hay tres interruptores programables que le permiten conectarse a otros tres alambres de un canal adyacente (Donde se aplica la terminología de W = 3). El modelo de los interruptores usado en esta arquitectura es plana. La figura 5.5 muestra la conexión de un *switch box.*

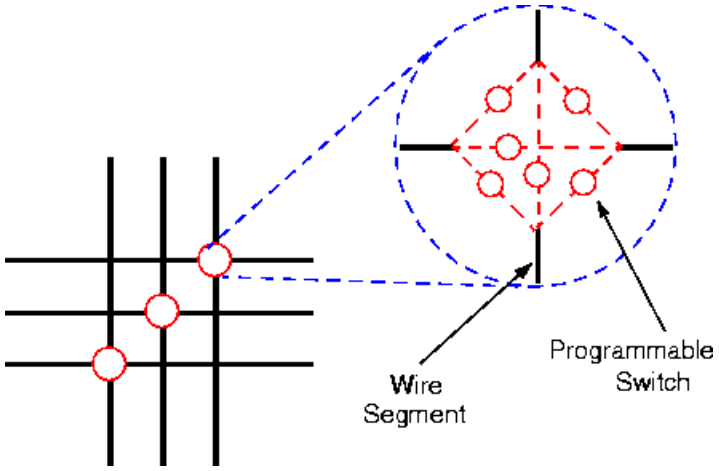


Figura 5.5

Las FPGA no están limitados a la típica matriz de un PLA[5] (AND-OR), al contrario, contiene una matriz interna configurable de relojes lógicos (CLB) y un anillo de circunvalación de bloques de I/O (IOB).

[5] Matriz lógica programable

Cada CLB contiene lógica programable combinacional y registros de almacenamiento. La sección de lógica combinacional es capaz de implementar cualquier función booleana en sus variables de entrada.
Cada IOB puede programarse independientemente para ser una entrada, y salida con control de tres estados o un pin bidireccional.

Como ejemplo tenemos una función muy simple (una EXOR de 3 entradas) llevándose a cabo en un FPGA como se muestra en la figura 5.6. Se configuran tres bloques de la izquierda como entradas, un bloque lógico se usa para crear la compuerta EXOR de 3 entradas y un cuarto bloque a la derecho se usa como salida.

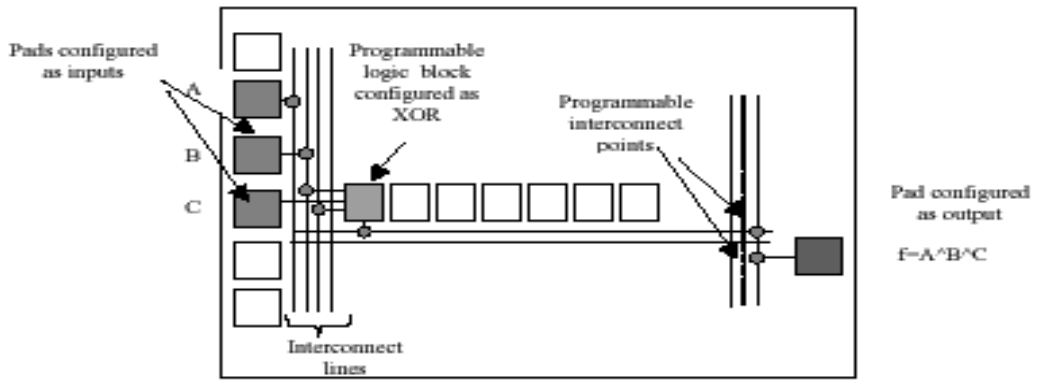


Figura 5.6

Se configuran tres bloques como entradas y representan la información lógica de las variables A, B y C (Figura 5.7). Un camino o ruta interior se crea para establecer una conexión eléctrico entre la región de I/O y el bloque lógica llamado (Internal routing).

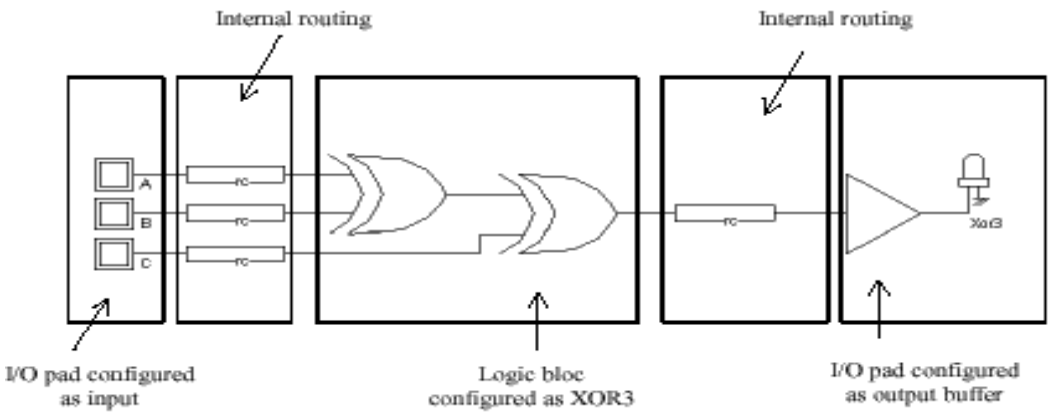


Figura 5.7

Internamente, el bloque lógico puede configurarse en cualquier combinación de una función básica. Cada bloque lógico normalmente soporta 3 a 8 entradas, en el ejemplo el bloque se a configurado como un EXOR de 3 entradas. Entonces, se configuran otros alambres para la ruta interna para llevar a cabo el bloque de I/O .

Note que los FPGA no solamente existen en componentes simples, también están los macro bloques (Figura 5.8).

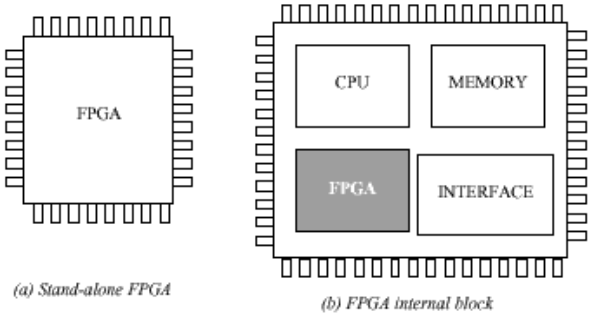


Figura 5.8

Los bloques lógicas programables deben poder llevar a cabo todas las funciones lógica básicas que son NOT, AND, NAND, OR, NOR, EXOR, EXNOR, etc.

Para desarrollar el diseño se hace uso de herramientas como el lenguaje VHDL[6], diseño por computadora (CAD) para realizar los diagramas o algún editor de bajo nivel (XDE). Pero también se necesita la estructura como el hardware, y esto lo proporcionan las empresas como Xilinx, Actel, Altera, Almel entre otras. Una vez teniendo lo necesario se

[6] Lenguaje de descripción para hardware

puede disponer a diseñar aplicaciones industriales siendo una de ellas dispositivos implementados en la corrección y detección de errores (código Reed-Muller) ya sea primitivas o macros.

3.2.2 TIPOS DE FPGA

Algunos FPGA comerciales disponibles son:

- Xilinx (XC2000 - XC8100)
- Altera (MAX, FLEX, Classic)
- Actel (ACT 1- 4, 3200DX, 1200XL)
- Cross Point (CP20)
- Concurrent Logic (CLi6000)
- Quick Logic (pASIC)
- Intel (iFX780)
- AMD (MACH1,2,3,4,5)
- ATMEL (ATV)
- Pilkingston (TS Series)
- Zycad Gatefied (GF Series)

Ahora bien, observemos las familias de Xilinx que dispones de FPGA:

Familia Spartan-II[7]

Dispositivo	Paquete
XC2S15	CS144
	TQ144
	VQ100
XC2S30	CS144
	TQ144
	PQ208
	VQ100

[7] Ver anexos

XC2S50	TQ144
	FG256
	PQ208
XC2S100	TQ144
	FG256
	FG456
	PQ208
XC2S150	FG456
	FG256
	PQ208
XC2S200	FG256
	FG456

Familia Spartan-IIE

Dispositivo	Paquete
XC2S50E	TQ144
	FT256
	PQ208
XC2S100E	TQ144
	FT256
	FG456
	PQ208
XC2S150E	FG456
	FG256
	PQ208

XC2S200E	FT256
	FG456
	PQ208
XC2S300E	FG256
	FG456
	PQ208

Familia VirtexE

XCV50E	CS144
	PQ240
	FG256
XCV100E	CS144
	FG256
	PQ240
	BG352
XCV200E	CS144
	FG256
	PQ240
	FG456
	BG352
XCV300E	BG352
	BG432
	FG256

	FG456
	PQ240

Familia Virtex-II

XC2V40	CS144
	FG256
XC2V80	CS144
	FG256
XC2V250	CS144
	FG456
	FG256

Otros dispositivos FPGA

Spartan
SpartanXL
Virtex
VirtexE
Virtex-II
Virtex-II Pro
XC1800
XC4000[8]
XC4000E
XC4000EX

[8] Ver anexos

XC4000L
XC4000XL
XC4000XLA
XC4000XV

3.2.3 FUNCION EXOR EN FPGA

De todas las familias la que interesa es aquellas que dispongan de compuertas EXOR y estas son :

ELEMENTO				**FAMILIAS**		
	Spartan-II, Spartan-IIE	**Virtex Virtex-E**	**Virtex-II, Virtex-IIPRO**	**XC9500/XV/XL**	**CoolRunner XPLA3**	**CoolRunner-II**
XOR2 XOR3 XOR4	Primitivo	Primitivo	Primitivo	Primitivo	Primitivo	Primitivo
XOR5	Primitivo	Primitivo	Primitivo	Macro	Macro	Macro
XOR6 XOR7 XOR8 XOR9	Macro	Macro	Macro	Macro	Macro	Macro

Y las compuertas disponibles de 2 hasta 9 entradas se muestran en la figura 5.10

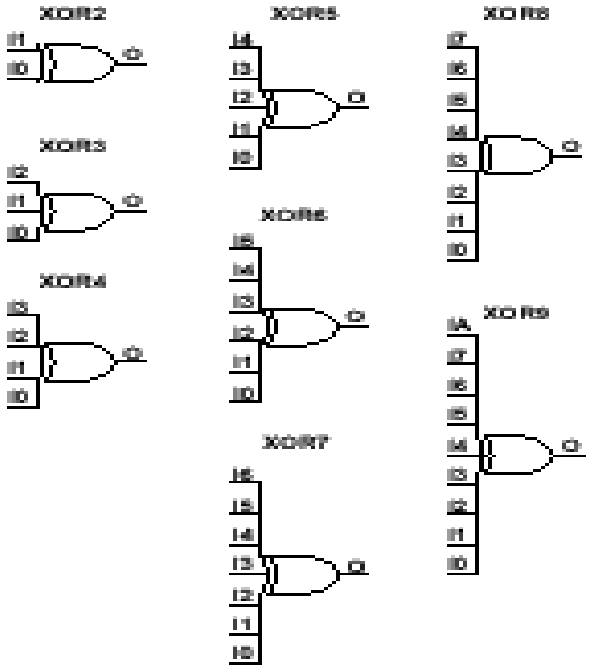


Figura 5.10 Representación de la compuerta EXOR.

Se dispone de diferentes entradas para una determinada familia:

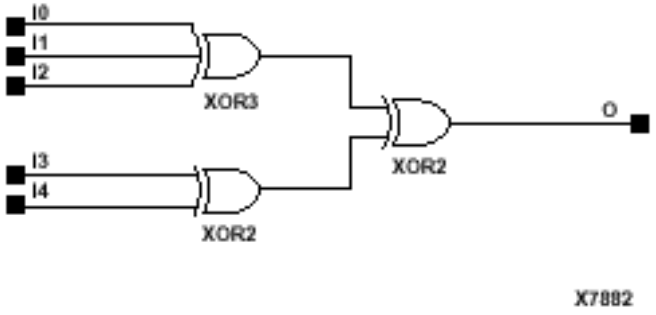


Figura 5.11 Implementacion de EXOR5 XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

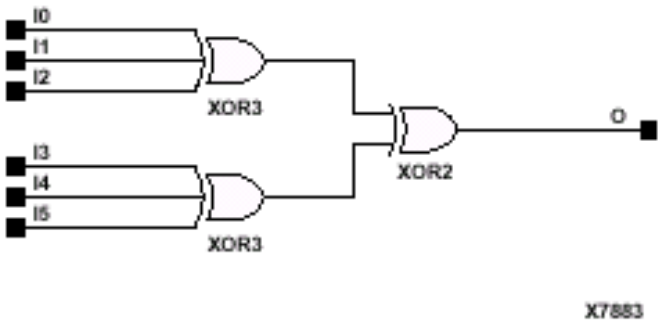


Figura 5.12 Implementación de una XOR6 XC9500/XV/XL, CoolRunner XPLA3.

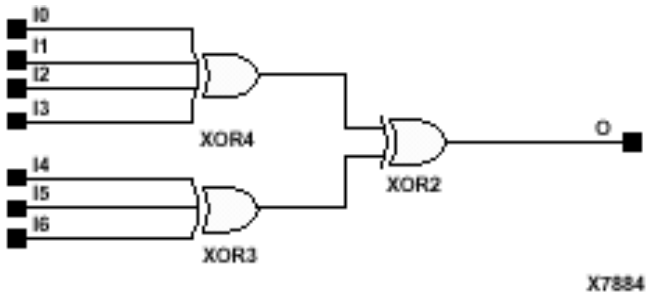


Figura 5.13 Implementacion de una **XOR7 XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II**

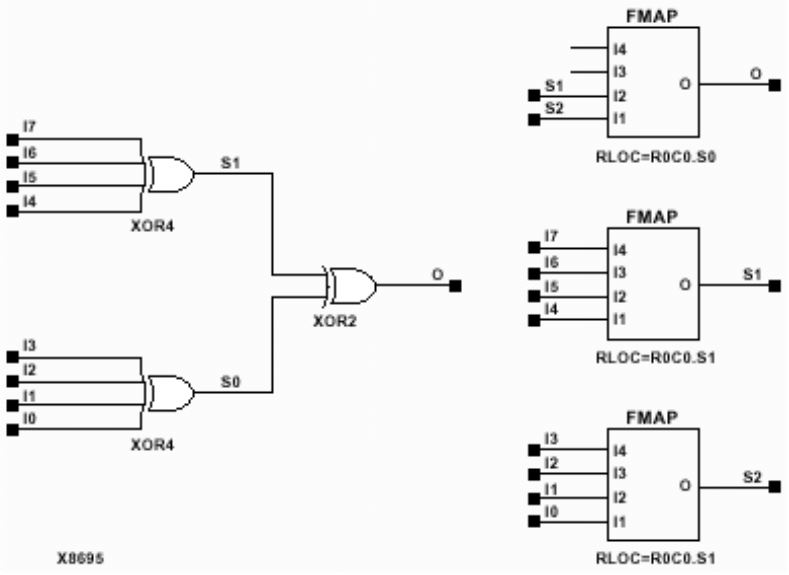


Figura 5.13 V Implementación de la XOR8 Spartan-II, Spartan-IIE, Virtex, Virtex-E

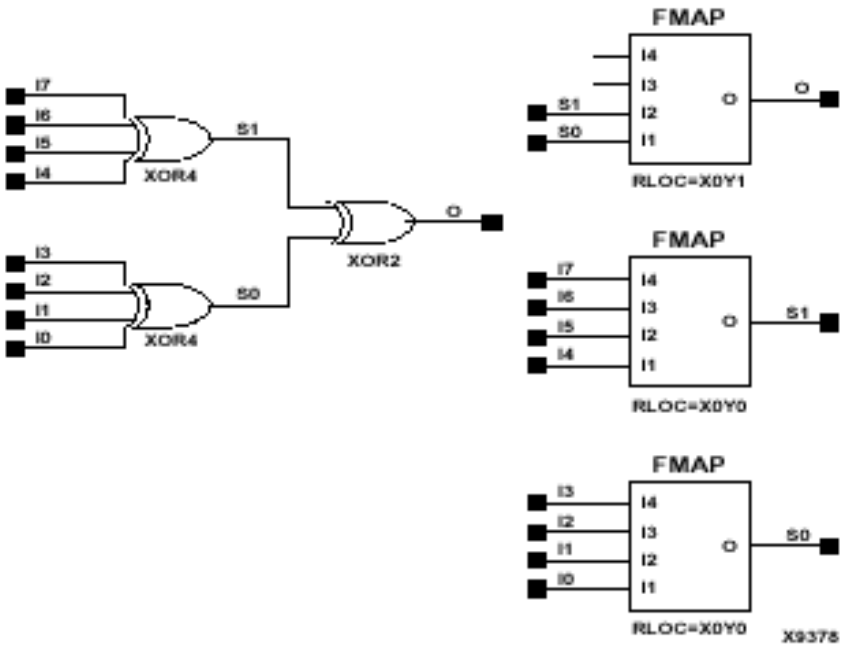


Figura 5.14 Implementación de la operación XOR8 Virtex-II, Virtex-II PRO

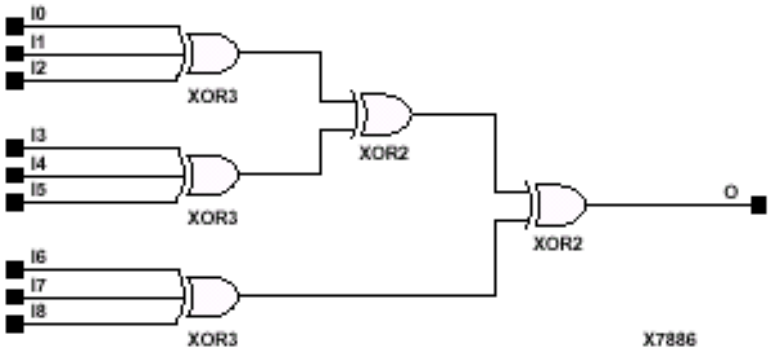


Figura 5.13 Funciones EXOR9 XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

CONCLUSIONES

- El código rm resulta ser otra alternativa para la minimización de funciones lógicas.
- El código rm con polaridad mixta se presenta como el método mas reductivo, en algunos casos, en comparación con las otras polaridades.
- La polaridad mixta presenta una solución mínima, pero no única.
- En la corrección de errores de primer orden presenta una alta capacidad correctora, a pesar de eso, el ruido, las interferencias y las señales parásitas siempre están presentes.
- Con fpga se desarrollan circuitos a pequeña escala utilizando el álgebra exor.
- Con fpga es posible desarrollar compuertas exor de 2 o mas entradas.
- El código reed-muller no se limita a aplicaciones en un área (sistemas digitales, telecomunicaciones, teoría de la información)

SUGERENCIAS

- Para la ejecución del programa donde involucre una cantidad de variables muy grande, utilizar una PC rápida con suficiente memoria.

- Se pueden diseñar sistemas utilizando fpga, siempre y cuando se tengan los recursos necesarios

- Existen nuevas investigaciones orientada a diferentes aplicaciones utilizando Reed-Muller, seria recomendable otras investigaciones.

REFERENCIAS

Easily Testable Realizations for Generalized Reed-Muller Expressions

Tsutomu Sasao, Fellow, IEEE TRANSACTIONS ON COMPUTERS, VOL. 46, NO. 6, JUNE 1997 709

Decomposition Method for Minimisation of Reed-Muller Polynomials in Mixed Polarity

A. TRAN / J.WANG
IEEE vol. 140 Nº1, January 1993

Evaluation of -valued Fixed Polarity Generalizations of . Reed-Muller Canonical Form

Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic, pages 92-98, Freiburg, Germany, May 1999.

Reed-Muller Universal Logic Module and its Application to Function Realization.

TA-CHENG LIN, ANH TRAN and CHIEH OUYANG
INT. J ELECTRONICS 1993, VOL 75 Nº 4, 675-684

SISTEMA DE COMUNICACION DIGITAL Y ANALOGICOS

Prentice may (Quinta Edición), LEON W. COUCH II

http://wmatem.eis.uva.es/~ignfar/TC.html

http://www.cs.columbia.edu/~theobald/pubs/acm_eurodac_fprm.pdf.

www.sigda.org/Archives/ProceedingArchives/Dac/Dac96/papers/1996/dac96/pdffiles/19_1.pdf

www.sigda.org/Archives/ProceedingArchives/Dac/Dac94/papers/1994/dac94/pdffiles/22_4.pdf

http://web.syr.edu/~rrosenqu/ecc/concept.html

http://www.cbel.com/applications_math / ?order=alpha

http://wmatem.eis.uva.es/~ignfar/TC.html

http://mmc.et.tudelft.nl/~richie/ecc.html

http://www.mac.cie.uva.es/~cureemm/codigos/codigos.htm

www.rose-hulman.edu/~langley/MA416/reedmuller.pdf

http://www.stanford.edu/class/ee387/handouts/rm.pdf

www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html

www.xilinx.com

www.dcs.gla.ac.uk/research/fpga/papers/pdf/fccm95.pdf

www.eetasia.com/ARTICLES/2000SEP/2000SEP01_PL_AN1.PDF

www.intrage.insa-tlse.fr/~etienne/Microwind/bookch9.pdf

www.ei.cs.vt.edu/~csei/project/sf/Overhead/fpgas.pdf

http://support.xilinx.com/techdocs/12209.htm

www.ece.gmu.edu/courses/ECE449/Lectures/449_lecture6.pdf

www.nalanda.nitc.ac.in/industry/appnotes/xilinx/documents/partinfo/databook.htm

http://www.cedcc.psu.edu/ee497i/xilinx/day1/sld011.htm

http://www.cs.washington.edu/research/projects/lis/triptych/www/

http://www.xilinx.com/products/tables/fpga.htm

ANEXO A

A. Comprobación de teoremas

T1 $x \oplus x = 0$

Partiendo de la ecuación fundamental de EXOR:

$$X \oplus Y = X'Y + XY'$$

Si X = X, y Y = X:

Entonces tenemos

$$X \oplus Y = X'X + XX'$$
$$= 0 + 0$$
$$= 0$$

T2 $x \oplus x' = 1$

Si X = X; Y = X':

$$X \oplus Y = X'Y + XY'$$
$$= X'X' + XX$$
$$= X' + X$$
$$= 1$$

T3 $x \oplus 0 = x$

Si X = X; Y=0:

$$X \oplus Y = X'Y + XY'$$
$$= X'. 0 + X.0'$$
$$= 0 + X.1$$
$$= X$$

T4 $x \oplus 1 = x'$

X = X; Y = 1:

$$X \oplus Y = X'Y + XY'$$

$$= X'.1 + X.1'$$

$$= X' + X.0$$

$$= X'$$

T5 $x \oplus y = y \oplus x$

Si X = Y; Y = X

$$X \oplus Y = X'Y + XY'$$

$$= Y'X + YX'$$

$$= Y \oplus X$$

T6 $x' \oplus y' = x \oplus y$

Si X = X'; Y = Y'

$$X \oplus Y = X'Y + XY'$$

$$= XY' + X'Y$$

$$= X \oplus Y$$

T7 $(x \oplus y)' = x' \oplus y = x \oplus y' = xy + x'y'$

$$(X \oplus Y)' = (XY' + X'Y)'$$

$$= (XY')' (X'Y)'$$

$$= (X' + Y)(X + Y')$$

$$= X'Y' + XY$$

$$= X' \oplus Y = X \oplus Y'$$

T8 **$(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$**

$(x \oplus y) \oplus z = (X \oplus Y)Z' + (X \oplus Y)'Z$

$= (XY' + X'Y)Z' + (XY + X'Y')Z$

$= XY'Z' + X'YZ' + XYZ + X'Y'Z$

$= X(Y'Z' + YZ) + X'(YZ' + Y'Z)$

$= X(Y \oplus Z)' + X'(Y \oplus Z)$

$= X \oplus Y \oplus Z$

T9 **$x(y \oplus z) = xy \oplus xz$**

$= XY \oplus XZ = XY(XZ)' + (XY)'XZ$

$= XY(X' + Z') + (X' + Y')XZ$

$= XYZ' + XY'Z$

$= X(YZ' + Y'Z)$

$= X(Y \oplus Z)$

T9' **$x(y \oplus z) = (x'+y) \oplus (x'+z)$**

$(x'+y) \oplus (x'+z) = (X' +Y)(X' + Z)' + (X' +Y)'(X' + Z)$

$= (X' +Y)(XZ') + (XY')(X' +Z)$

$= XYZ' + XY'Z$

$= X(YZ' + Y'Z)$

$= X(Y \oplus Z)$

T10 **If: $f = g \oplus h$ and $gh = 0$, then $f = g + h$**

Véase el teorema T16

T11 **If: $f = g \oplus h$, then $g = f \oplus h$ and $h = g \oplus f$**

Véase el teorema T17

T13 f.x ′ = f ⊕ f.x

$$F.X' = F (1 \oplus X)$$

$$= F \oplus F.X$$

T14 x + y = x ⊕ x′y

$$X \oplus X'Y = X'(X'Y) + X(X'Y)'$$

$$= X'Y + X(X + Y')$$

$$= X'Y + X + XY'$$

$$= X'Y + X(1 + Y')$$

$$= X'Y + X$$

$$= X + Y$$

T15 x ⊕ xy = xy‘

$$X \oplus XY = X'(XY) + X(XY)'$$

$$= 0 + X(X' + Y')$$

$$= XY'$$

T16 x + y = x ⊕ y si xy =0

Si XY = 0, X’ + Y’ = (XY)’ = 1

$$X + Y = X(X' + Y') + Y(X' + Y')$$

$$= XY' + X'Y$$

$$= X \oplus Y$$

T17 x + y = x ⊕ y ⊕ xy si xy ≠0

Tomando la identidad de:

X +X’ = 1 , y Y + Y’ =1

Y como

$$X + Y = X.1 + Y.1$$

Sustituyendo las identidades llegamos:

$$X + Y = X(Y + Y') + Y(X + X')$$

$$= XY + XY' + X'Y$$

Ahora desarrollando:

$$X \oplus Y \oplus XY = (XY' + X'Y) \oplus XY$$

$$= (XY' + X'Y)(XY)' + (XY' + X'Y)'(XY)$$

$$= XY + XY' + X'Y$$

Y debido a que ambas identidades llegan a la misma respuesta, el teorema T17 es verdadero.

ANEXO B

MANUAL DEL USUARIO

OBJETIVO:

El siguiente programa tiene como utilidad minimizar funciones lógicas, mediante el método de Reed-Muller, dando lugar a tener tres formas de hacerlo, ya sea mediante polaridad positiva, fija o mixta.

El polinomio de Reed-Muller tiene su principio en el operador EXOR, por lo tanto la minimización que resulte ya sea de cualquiera de los tres métodos, proporcionara su respuesta con operador AND/EXOR y no como AND/OR.

COMO INTRODUCIR LOS DATOS:

Los datos se introducen en forma decimal uno después de otro separados por una coma.

Ejemplo: 1,2,3

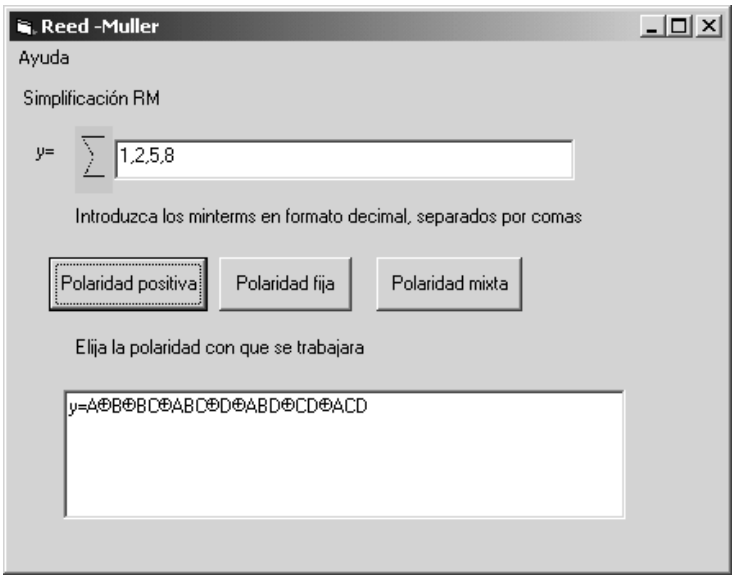
Esto da a entender que es la sumatoria de los minterms de una tabla de verdad.

Índice

POLARIDAD POSITIVA:

Para tener acceso ha este método se logra mediante el botón que dice " POLARIDAD POSITIVA ". Pero antes de presionarlo tiene que haber introducido ya los datos.

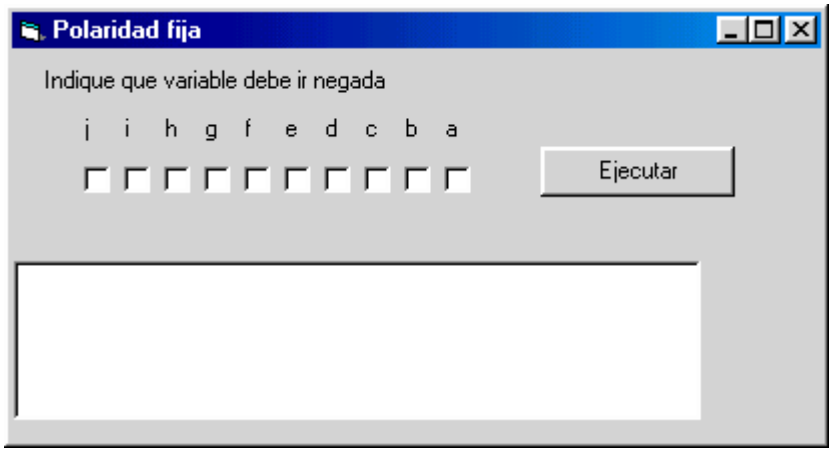
La respuesta de la polaridad Positiva se presentara en la parte de abajo de la pantalla.



La figura anterior muestra la forma en la cual se dará la respuesta.

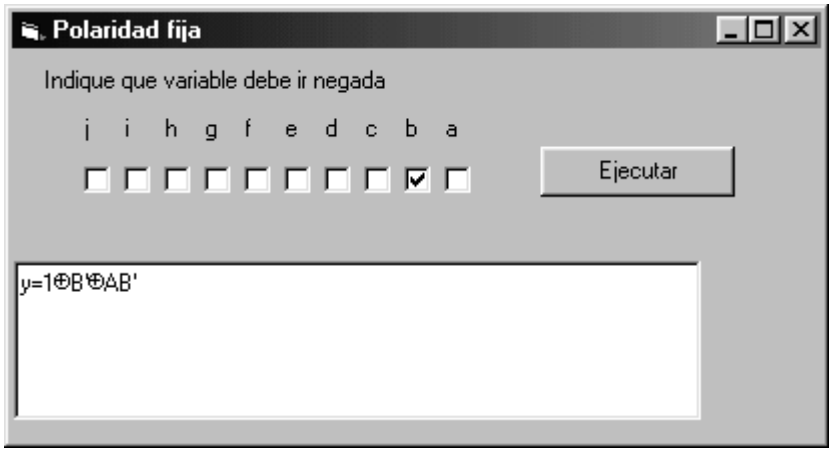
POLARIDAD FIJA:

La polariza fija es parecida a la positiva, se introduce los datos de forma decimal y se presiona la tecla “Polaridad fija”, luego se presenta otra pantalla en la cual nos da la opción de negar las variables posibles, como se presenta en la siguiente figura



Se puede observar las variables posibles que se pueden negar, solo colocando una marca debajo de cada variable.

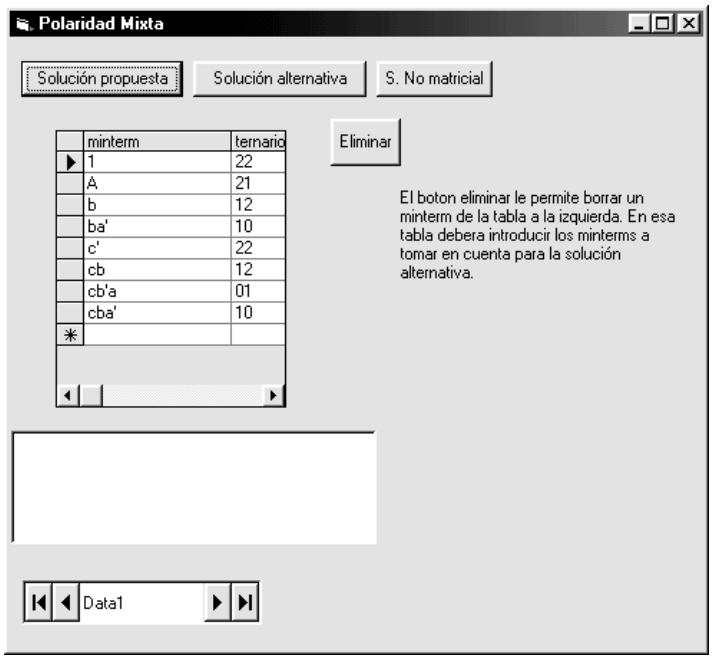
Luego que se tiene marcadas las variables se presiona el botón “Ejecutar” para el resultado en polaridad fija.



Para regresar a la pantalla principal cierre la polaridad fija.

POLARIDAD MIXTA:

Luego de haber introducido los datos y presionado el botón respectivo, se presenta una pantalla que muestra tres formas de realizar la polaridad mixta.

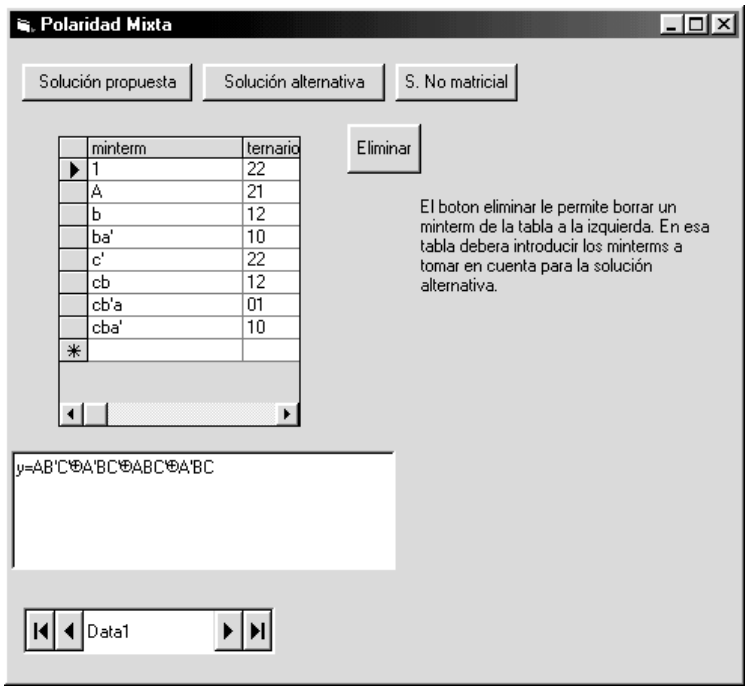


Y esas tres formas mencionadas son:

- Solución Propuesta
- Solución Alternativa
- Solución no Matricial

Solución Propuesta

Para la solución Propuesta, la respuesta que proporcionara será para cada minsterms que se coloque al principio, si son 4 minsterms dará como resultado 4 expresiones AND/EXOR.

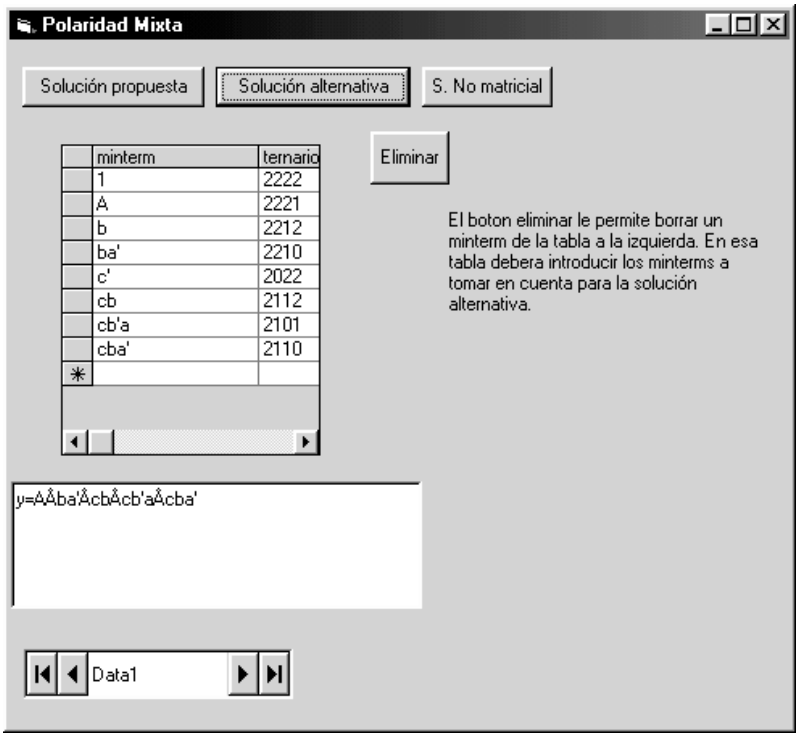


Solución Alternativa

Para la Solución Alternativa la lógica cambia, aquí el usuario tiene la posibilidad de escoger su respuesta, en base a que minsterms desea que la expresión de salida aparezca, pero corre el riesgo de que si no escoge adecuadamente los términos no arroje una solución para el caso, ya que los minterminos colocados no pertenecen a la respuesta o son insuficientes

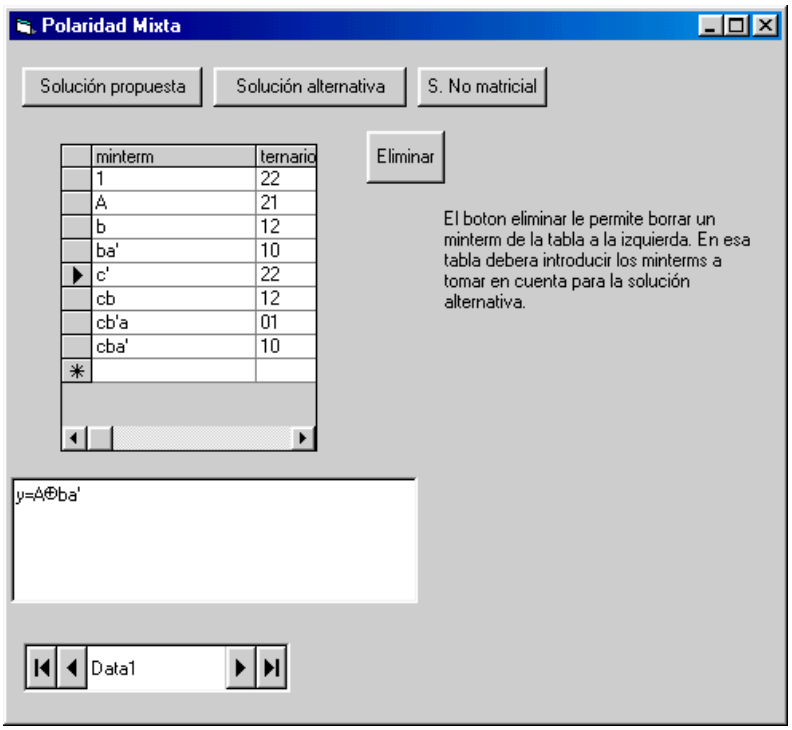
Puede dar mensajes como el siguiente

Como resultado tendríamos datos errados.



Pero también se da la posibilidad de que el usuario coloque los minsterms correctos y de una respuesta valida como la siguiente:

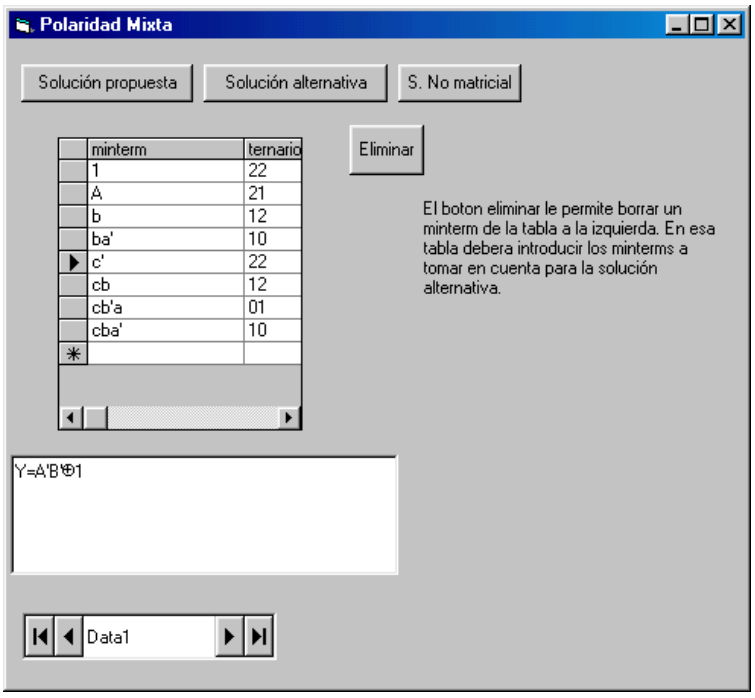
El botón “Eliminar” elimina el minsterms que usted indique en la tabla de la izquierda.



Solución no Matricial

En este método se trata de lograr una respuesta lo mas mínima posible y mejor que los caso anteriores, por lo tanto se presenta como la mejor alternativa para obtener una expresión lo mas minimizada posible mediante Reed-Muller.

Y esto se logra presionando el botón " S no Matricial", siguiente pantalla.


Polaridad Mixta
Solución propuesta
Solución alternativa
S. No matricial
minterm
ternario
1
22
A
21
b
12
ba'
10
c'
22
cb
12
cb'a
01
cba'
10
Eliminar
El boton eliminar le permite borrar un minterm de la tabla a la izquierda. En esa tabla debera introducir los minterms a tomar en cuenta para la solución alternativa.
Y=A'B'⊕1
Data1

ANEXO C

MANUAL DEL PROGRAMADOR

A continuación se definen los diferentes campos utilizados en la implementación del programa:

- Definición de la base de datos y descripción
- Descripción de formularios
- Procedimientos
- Variables publicas

DEFINICIÓN DE BASE DE DATOS Y DESCRIPCION

Tipo de base de datos : Microsoft Access
Nombre del archivo: rm.mdb

	TABLA	
CAMPOS	**MINTERMS**	**TERNARIO**
TIPO	TEXTO	TEXTO
TAMAÑO	20	12

DESCRIPCIÓN DE FORMULARIO

Lenguaje de programación: Visual Basic

Nombre del archivo del proyecto: prm.vbp

NOMBRE	Frmrm
ARCHIVO	Frmrm.frm
DESCRIPCION	Muestra los tres métodos disponibles para Reed-Muller, también se muestra la entrada de datos, y la ventana donde se muestra la salida para la polaridad positiva.

NOMBRE	Frmfija
ARCHIVO	Frmfija.frm
DESCRIPCIÓN	Muestra las posibles variables que pueden ser negadas, luego un ventana para visualizar la respuesta para la polaridad fija.

NOMBRE	Frmmixta
ARCHIVO	Frmmixta.frm
DESCRIPCIÓN	Nos muestra las tres soluciones con que cuenta la polaridad mixta que son: la propuesta, Alternativa, y no matricial. También la tabla que sirve como interfaz con el usuario para la introducción de los minterms.

PROCEDIMIENTO

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND1 FRMRM	TEXT1	Recibe los datos del usuario que estén en tex1, lo procesa y saca los datos a través de R1 del formulario

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
KRONE	-	Realiza el producto Kronecker para la polaridad positiva

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND3	-	Llama el formulario de la polaridad fija

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND4	-	Llama el formulario de la polaridad mixta

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
MNUHELP	-	Permite la ayuda al usuario a conocer el entorno grafico del programa

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
FORM_LOAD DE FRMFIJA	-	Genera los checkbox para las variables

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
CMD2_CLICK	TEXT1 DE FRMRM	Realiza el calculo para polaridad fija

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
KRONE2	-	Realiza la expansión kronecker para polaridad fija

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
FORM_LOAD DEFRMMIXTA	FRMRM.TEXT1	Genera la tabla de verdad en base a los datos del primer formulario

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND1 PARA FRM MIXTA	Y	Genera el procedimiento para la solución propuesta

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND2 PARA FRMMIXTA	VARIABLE “Y”	Genera el código ternario de los minsterms de la base de datos

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND3 PARA FRMMIXTA	-	Elimina los minsterms de la base de datos

NOMBRE DEL PROCEDIMIENTO	PARAMETRO	DESCRIPCIÓN DE SU FUNCION
COMMAND4 PARA FRMMIXTA	VECTOR PUBLICO “Y”	Es la solución para no matricial

VARIABLES PUBLICAS

NOMBRE DE CONSTANTE	TIPO DE DATO	FUNCION
Y	BOOLEANA	ALMACENA LA TABLA DE VERDAD SEGÚN LOS DATOS INTRODUCIDOS
t	STRING	GUARDA EL EQUIVALENTE DEL TERNARIO DE CADA MINTERSM
mt	STRING	GUARDA LOS MINTERSM
g	BOOLEANA	ES EL VECTOR DE LA SALIDA CON QUE SE CONSTRULLEN LOS RESULTADOS

CODIGO

PROGRAMA

POLARIDAD POSITIVA

```
Dim y(512)
Private Sub Command1_Click()
Dim t(512)
Dim v(512) As Boolean
Dim tl As String
Dim rv, bn, mmax
rv = Text1
n = 1
mmax = 0
For i = 1 To 512
   y(i) = False
Next
Do While True
   i = InStr(1, rv, ",")
   If i <> 0 Then
      t(n) = Left(rv, i - 1)
      rv = Mid(rv, i + 1)
      n = n + 1
   Else
      t(n) = rv
      Exit Do
   End If
Loop
For i = 1 To n

   y(Val(t(i)) + 1) = True
   If Val(t(i)) > mmax Then mmax = Val(t(i))
Next

bn = Int(Log(mmax) / Log(2)) + 1
b2 = bn * bn
'MsgBox (CStr(bn))
'bn = 3
Dim mt(512)
'Label3.Caption = "y="
R1.Text = "y="
mt(1) = "1"
For i = 1 To b2
   ta = i - 1
   j = 1
   Do While True
      u = Int(ta / 2)
      z = ta - 2 * u
      If z = 1 Then
         mt(i) = mt(i) + Chr(64 + j)
```

```
        End If
        j = j + 1
        ta = u
        If u = 0 Then
            Exit Do
        End If
    Loop
Next
Dim a(512, 512), b(2, 2), c(512, 512), g(512)
a(1, 1) = True
a(1, 2) = False
a(2, 1) = True
a(2, 2) = True
b(1, 1) = True
b(1, 2) = False
b(2, 1) = True
b(2, 2) = True
m = 2
For h = 1 To bn
    GoSub krone
Next
j = 1
For i = 1 To b2
    g(i) = False
    For k = 1 To m
            g(i) = g(i) Xor (a(i, k) And y(k))
    Next

    If g(i) Then
        If j > 1 Then
            'Label3.Caption = Label3.Caption + "+"
            R1.Text = R1.Text + Chr(197)
        End If
        'Label3.Caption = Label3.Caption + mt(I)
        R1.Text = R1.Text + mt(i)
        j = j + 1
    End If
Next
u = R1.Text
k = 1
For i = 1 To m
    j = InStr(k, u, Chr(197))
    If j = 0 Then
        Exit For
    End If
    k = j + 1
    R1.SelStart = j - 1
    R1.SelLength = 1
    R1.SelFontName = "Symbol"

Next
```

```
Exit Sub
krone:
For i = 1 To 2
   For j = 1 To 2
      For k = 1 To m
         For l = 1 To m
            p = k + i * m - m
            q = l + j * m - m
            c(p, q) = a(k, l) And b(i, j)
         Next
      Next
   Next
Next
m = 2 * m
For i = 1 To m
   For j = 1 To m
      'MsgBox (CStr(c(i, j)))
      a(i, j) = c(i, j)
   Next
Next
Return
End Sub
Private Sub Command2_Click()
'ingreso de datos
End Sub

Private Sub Command3_Click()
frmfija.Show

End Sub

Private Sub Command4_Click()
frmmixta.Show

End Sub

Private Sub Command5_Click()
R1.Font = "Arial"

'R1.Font = "Arial"
R1.TextRTF = "ABCDEF" + Chr(197) + "A'B"' + Chr(197) + "AB"
u = R1.Text
k = 1
For i = 1 To 10
   j = InStr(k, u, Chr(197))
   If j = 0 Then
      Exit For
   End If
   k = j + 1
R1.SelStart = j - 1
R1.SelLength = 1
```

```
R1.SelFontName = "Symbol"

Next

R1.Refresh

'R1.SelText = Chr(197)
End Sub
```

POLARIDAD FIJA

```
Dim y(512)
Private Sub Cmd2_Click()
Dim a(512, 512), d(2, 2), b(2, 2), c(512, 512), g(512)
Dim t(512)
Dim mt(512) As String
Dim v(512) As Boolean
Dim tl As String
Dim rv, mmax, bn
rv = frmrm.Text1
n = 1
R1.Text = "y="
Do While True
   i = InStr(1, rv, ",")
   If i <> 0 Then
      t(n) = Left(rv, i - 1)
      rv = Mid(rv, i + 1)
      n = n + 1
   Else
      t(n) = rv
      Exit Do
   End If
Loop
mmax = 0
For i = 1 To n
   If Val(t(i)) > mmax Then mmax = Val(t(i))
   y(Val(t(i)) + 1) = True
Next
'crear los minterms
bn = Int(Log(mmax) / Log(2)) + 1
b2 = bn * bn
'MsgBox (CStr(bn))
mt(1) = "1"
For i = 1 To b2
   w = 0

   ta = i - 1
   j = 1
   Do While True
      u = Int(ta / 2)
      z = ta - 2 * u
```

```
        If z = 1 Then
            mt(i) = mt(i) + Chr(64 + j)
            If Check1(w) = 1 Then
                mt(i) = mt(i) + """
            End If
        End If
        j = j + 1
        ta = u
        If u = 0 Then
            Exit Do
        End If
        w = w + 1
    Loop
    'MsgBox (mt(i))
Next


b(1, 1) = True
b(1, 2) = False
b(2, 1) = True
b(2, 2) = True
d(1, 1) = False
d(1, 2) = True
d(2, 1) = True
d(2, 2) = True
For i = 1 To 2
    For j = 1 To 2
        If Check1(0) = 0 Then
            a(i, j) = b(i, j)
        Else
            a(i, j) = d(i, j)
        End If
    Next
Next
m = 2
z = 1
For h = 1 To bn - 1
    GoSub krone2
Next
j = 1
For i = 1 To m
    g(i) = False
        For k = 1 To m
            g(i) = g(i) Xor (a(i, k) And y(k))
        Next

    If g(i) Then
        If j > 1 Then
            'Label3.Caption = Label3.Caption + "+"
            R1.Text = R1.Text + Chr(197)
        End If
        'Label3.Caption = Label3.Caption + mt(i)
```

```
        R1.Text = R1.Text + mt(i)

        j = j + 1
    End If
Next
'nuevo
u = R1.Text
k = 1
For i = 1 To 10
    j = InStr(k, u, Chr(197))
    If j = 0 Then
        Exit For
    End If
    k = j + 1
R1.SelStart = j - 1
R1.SelLength = 1
R1.SelFontName = "Symbol"

Next



Exit Sub
krone2:
For i = 1 To 2
    For j = 1 To 2
        For k = 1 To m
            For l = 1 To m
                p = k + i * m - m
                q = l + j * m - m
                If Check1(z) = 0 Then
                    c(p, q) = a(k, l) And b(i, j)
                Else
                    c(p, q) = a(k, l) And d(i, j)
                End If
            Next
        Next
    Next
Next
m = 2 * m
For i = 1 To m
    For j = 1 To m
        'MsgBox (CStr(c(i, j)))
        a(i, j) = c(i, j)
    Next
Next
Return

End Sub


Private Sub Form_Load()
```

```
For i = 1 To 9
Load Check1(i)
Check1(i).Top = Check1(0).Top
Check1(i).Left = Check1(i - 1).Left - 300
Check1(i).Visible = True
Load Label1(i)
Label1(i).Top = Label1(0).Top
Label1(i).Left = Label1(i - 1).Left - 300
Label1(i).Caption = Chr(97 + i)
Label1(i).Visible = True



Next
MsgBox ("hecho")
End Sub
```

POLARIDAD MIXTA

```
Dim y(512)
Dim n As Integer
Dim t(64)
Dim mt(64), g(64)

Private Sub Command1_Click()
R1.Text = "y="
For i = 1 To n

   y(Val(t(i)) + 1) = True
   If Val(t(i)) > mmax Then mmax = Val(t(i))
Next

bn = Int(Log(mmax) / Log(2)) + 1
b2 = bn * bn

For i = 0 To b2
   mt(i + 1) = ""
   r = i
   j = 1
   Do While True
      u = Int(r / 2)
      x = r - 2 * u
      If x = 0 Then
         mt(i + 1) = mt(i + 1) + Chr(64 + j) + "'"
      Else
         mt(i + 1) = mt(i + 1) + Chr(64 + j)
      End If
      r = u
      If j = bn Then
         Exit Do
      End If
      j = j + 1
```

```
    Loop
Next
j = 1
m = b2
For i = 1 To m
    If y(i) Then
        If j > 1 Then
            'Label3.Caption = Label3.Caption + "+"
            R1.Text = R1.Text + Chr(197)
        End If
        'Label3.Caption = Label3.Caption + mt(i)
        R1.Text = R1.Text + mt(i)
        j = j + 1
    End If
Next
u = R1.Text
k = 1
For i = 1 To m
    j = InStr(k, u, Chr(197))
    If j = 0 Then
        Exit For
    End If
    k = j + 1
    R1.SelStart = j - 1
    R1.SelLength = 1
    R1.SelFontName = "Symbol"

Next


End Sub

Private Sub Command2_Click()
'On Error GoTo alarma
Dim cz As Integer
Dim c(64, 64) As Boolean

R1.Text = "y="
Dim cv As String
Dim a(512, 512)
Dim v(16), z(16)
For i = 1 To n

    y(Val(t(i)) + 1) = True
    If Val(t(i)) > mmax Then mmax = Val(t(i))
Next

bn = Int(Log(mmax) / Log(2)) + 1
b2 = 2 ^ bn
Dim w As Boolean
```

```
'n = 8
'b = 3
'b2 = 8
'b3 = 27
For i = 1 To b2
Load Label1(i)
Label1(i).Visible = True
Label1(i).Top = Label1(i - 1).Top + Label1(i).Height

Label1(i).Left = Label1(i - 1).Left

Next

With Data1.Recordset
   cz = .RecordCount
   cz = Int(Sqr(cz))
   'MsgBox (cz)
   If cz < bn Then
      MsgBox "no hay suficientes minterms"
   End If
   For i = 1 To cz
      v(i) = False
      z(i) = False

   Next
   .MoveFirst
   l = 1
   Do While Not .EOF
      cv = UCase(!minterm)
      cq = ""

      'i maneja el bit
      For i = bn To 1 Step -1
         xm = Chr(64 + i)
         j = InStr(1, cv, xm)
         If j = 0 Then
            cq = cq + "2"
            'z(i) = True
            'Beep
         Else
            z(i) = False
            If Mid(cv, j + 1, 1) = "'" Then
               cq = cq + "0"
              ' v(i) = True
            Else
               cq = cq + "1"
               'v(i) = False
            End If
         End If


      Next
```

```
    .Edit
    !ternario = cq
    .Update
    .MoveNext

Loop

.MoveFirst
Do While Not .EOF
    cq = Trim(!ternario)
   ' MsgBox !minterm
    st = ""
    For j = 1 To bn
        z(j) = False
        v(j) = False
        If Mid(!ternario, j, 1) = "2" Then
           ' MsgBox (Mid(!ternario, j, 1))
            z(j) = True
        Else
            z(j) = False
            If Mid(cq, j, 1) = "1" Then
                v(j) = True
            Else
                v(j) = False
            End If
        End If
    st = st + CStr(z(j))
    Next
    .MoveNext

    'MsgBox st
    For k = 0 To b2 - 1
        f = True
        'MsgBox j
        ta = k
        'MsgBox k
        For n = 1 To bn
            p = Int(ta / 2)
            q = ta - p * 2
            If z(bn + 1 - n) Then
                'MsgBox CStr(n) '!minterm
            Else
             f = f And ((Not (v(bn + 1 - n) Xor CBool(q))))
            'MsgBox CBool(f)
            End If

            ta = p
            'If k = 0 Then Exit For
        Next
         '   MsgBox CStr(f), vbApplicationModal, cv
```

```
            'If cv = "1" Then f = True
            a(k + 1, l) = f
            'Label1(l - 1).Caption = Label1(l - 1).Caption + CStr(CInt(a(j + 1, l)))

        Next
        l = l + 1
        '.MoveNext
    Loop

        'If cv = "1" Then cq = "x"
    For k = 0 To b2 - 1
    For j = 1 To b2
        Label1(k).Caption = Label1(k).Caption + CStr(Abs(CInt(a(k + 1, j))))
    Next
    Next
For i = 1 To b2
    g(i) = y(i)
Next
For i = 1 To b2
    If a(i, i) Then
    Else
        Beep
        For j = 1 To b2
            If a(j, i) Then
                For k = 1 To b2
                    a(i, k) = a(i, k) Xor a(j, k)
                Next
                g(i) = g(i) Xor g(j)
                Exit For
            End If
        Next
    End If
    'Label1(i - 1).Caption = ""

Next
For i = 1 To b2
    For j = 1 To b2
        c(i, j) = a(i, j)
    Next
Next
MsgBox "espere"
For k = 0 To b2 - 1
    Label1(k).Caption = ""
    For j = 1 To b2
        Label1(k).Caption = Label1(k).Caption + CStr(Abs(CInt(c(k + 1, j))))
    Next
Next
f = True
For i = 1 To b2 - 1
    f = f And c(i, i)
    For j = i + 1 To b2
        If i <> j And c(j, i) Then
```

```
            For k = 1 To b2
                c(j, k) = c(j, k) Xor c(i, k)
            Next
        End If
    Next
Next
f = f And c(b2, b2)
If Not (f) Then MsgBox ("no va funcionar")
For i = 1 To b2
For j = 1 To b2

        If (j <> i) Then
            f = a(j, i) And a(i, i)
            For k = 1 To b2
                a(j, k) = a(j, k) Xor (f And a(i, k))
            Next
            g(j) = g(j) Xor (f And g(i))
        End If

Next
Next
j = 1
m = b2
.MoveFirst

For i = 1 To m
    If g(i) Then
        If j > 1 Then
            'Label3.Caption = Label3.Caption + "+"
            R1.Text = R1.Text + Chr(197)
        End If
        'Label3.Caption = Label3.Caption + !minterm
        R1.Text = R1.Text + !minterm
        j = j + 1
    End If
    If .EOF Then
        MsgBox ("minternm insuficientes")
        Exit Sub
    End If
    .MoveNext

Next
End With
u = R1.Text
k = 1
For i = 1 To m
    j = InStr(k, u, Chr(197))
    If j = 0 Then
        Exit For
    End If
    k = j + 1
    R1.SelStart = j - 1
```

```
    R1.SelLength = 1
    R1.SelFontName = "Symbol"

Next
Exit Sub
alarma:
MsgBox ("imposible con estos minterms")
End Sub

Private Sub Command3_Click()
With Data1.Recordset
    .Delete
End With
End Sub

Private Sub Command4_Click()
Randomize
For i = 1 To n

    y(Val(t(i)) + 1) = True
    If Val(t(i)) > mmax Then mmax = Val(t(i))
Next

bn = Int(Log(mmax) / Log(2)) + 1
b2 = 2 ^ bn
b3 = 3 ^ bn
Dim mt(6561), r(6561), tri(6561), p(6561)
Dim nv(6561), c(6561)
ti = 0
For i = 0 To b3 - 1
    j = 1
    ta = ti
    nv(i) = 1
    For k = 1 To bn
        x = Int(ta / 3)
        u = ta - 3 * x
        If u = 0 Then
            mt(i) = mt(i) + Chr(64 + j) + """
            tri(i) = "0" + tri(i)
        End If
        If u = 1 Then
            mt(i) = mt(i) + Chr(64 + j)
            tri(i) = "1" + tri(i)
        End If
        If u = 2 Then
            tri(i) = "2" + tri(i)
            nv(i) = nv(i) * 2
        End If
        j = j + 1
        ta = x
    Next
    If mt(i) = "" Then mt(i) = "1"
```

```
    'MsgBox mt(i)
    'MsgBox tri(i)
    p(i) = False
    ti = ti + 1
    If ti >= b3 Then
        ti = ti - b3
    End If
Next
s = n
'MsgBox s, vbCritical, "s"
 i = 0
 kin = 0
 If s > (b2 / 2) Then
    i = b3 - 1
End If
Do While s > 0
    bmin = n
    'For j = 0 To b3 - 1
     '   If p(j) Then
      '  Else
       '     If s >= nv(j) Then
        '        If (s - nv(j)) < bmin Then
         '           i = j
          '          bmin = s - nv(j)
           '     End If
      '      End If
    '    End If
    'Next
    'MsgBox mt(i)
    bmin = s
    'MsgBox s
    If Not (p(i)) And ((s > (b2 / 2)) Or (s >= nv(i))) Then
      'and s >= nv(i)g
        sa = 0
        For j = 0 To b2 - 1
            f = True
            ta = j
            For k = 1 To bn
                x = Int(ta / 2)
                u = ta - 2 * x
                z = Mid(tri(i), bn + 1 - k, 1)
                If z = "2" Then
                Else
                    If z = "0" Then
                        f = f And (Not (CBool(u)))
                    Else
                        f = f And (CBool(u))
                    End If
                End If
                ta = x
            Next
            r(j) = y(j + 1) Xor f
```

```
        If r(j) Then sa = sa + 1
    Next
    If sa < bmin Then
        kin = i
        'MsgBox kin
        bmin = sa
 '       s = sa
'       For j = 0 To b2 - 1
  '         y(j + 1) = r(j)
   '     Next
    '    c(i) = True
    End If
    'p(i) = True
End If

i = i + 1
If i >= b3 Then
    'MsgBox kin, vbApplicationModal, kin
    i = 0 ' ki
    sa = 0
    For j = 0 To b2 - 1
        f = True
        ta = j
        For k = 1 To bn
            x = Int(ta / 2)
            u = ta - 2 * x
            z = Mid(tri(kin), bn + 1 - k, 1)
            If z = "2" Then
            Else
                If z = "0" Then
                    f = f And (Not (CBool(u)))
                Else
                    f = f And (CBool(u))
                End If
            End If
            ta = x
        Next
        r(j) = y(j + 1) Xor f
        If r(j) Then sa = sa + 1
    Next
     s = sa
     For j = 0 To b2 - 1
          y(j + 1) = r(j)
     Next
     c(kin) = True
    ' MsgBox mt(kin)
     'p(i) = True
    For j = 0 To b3 - 1
        If c(j) Then
            p(j) = True
        Else
            p(j) = False
```

```
            End If
        Next
        If s = 0 Then
            Exit Do
        Else
            i = 0
        End If
    End If

Loop
j = 1
m = b3
For i = 0 To m - 1
    If c(i) Then
        If j > 1 Then
            'Label3.Caption = Label3.Caption + "+"
            R1.Text = R1.Text + Chr(197)
        End If
        'Label3.Caption = Label3.Caption + mt(i)
        R1.Text = R1.Text + mt(i)
        j = j + 1
    End If
Next
u = R1.Text
k = 1
For i = 1 To m
    j = InStr(k, u, Chr(197))
    If j = 0 Then
        Exit For
    End If
    k = j + 1
    R1.SelStart = j - 1
    R1.SelLength = 1
    R1.SelFontName = "Symbol"

Next


End Sub

Private Sub Form_Load()
Dim a(64, 64), d(2, 2), b(2, 2), c(64, 64)
Dim mt(64)
Dim v(64) As Boolean
Dim tl As String
Dim rv
rv = frmrm.Text1
n = 1

Do While True
    i = InStr(1, rv, ",")
    If i <> 0 Then
```

```
        t(n) = Left(rv, i - 1)
        rv = Mid(rv, i + 1)
        n = n + 1
    Else
        t(n) = rv
        Exit Do
    End If
Loop
For i = 1 To n

    y(Val(t(i)) + 1) = True
Next

End Sub
```