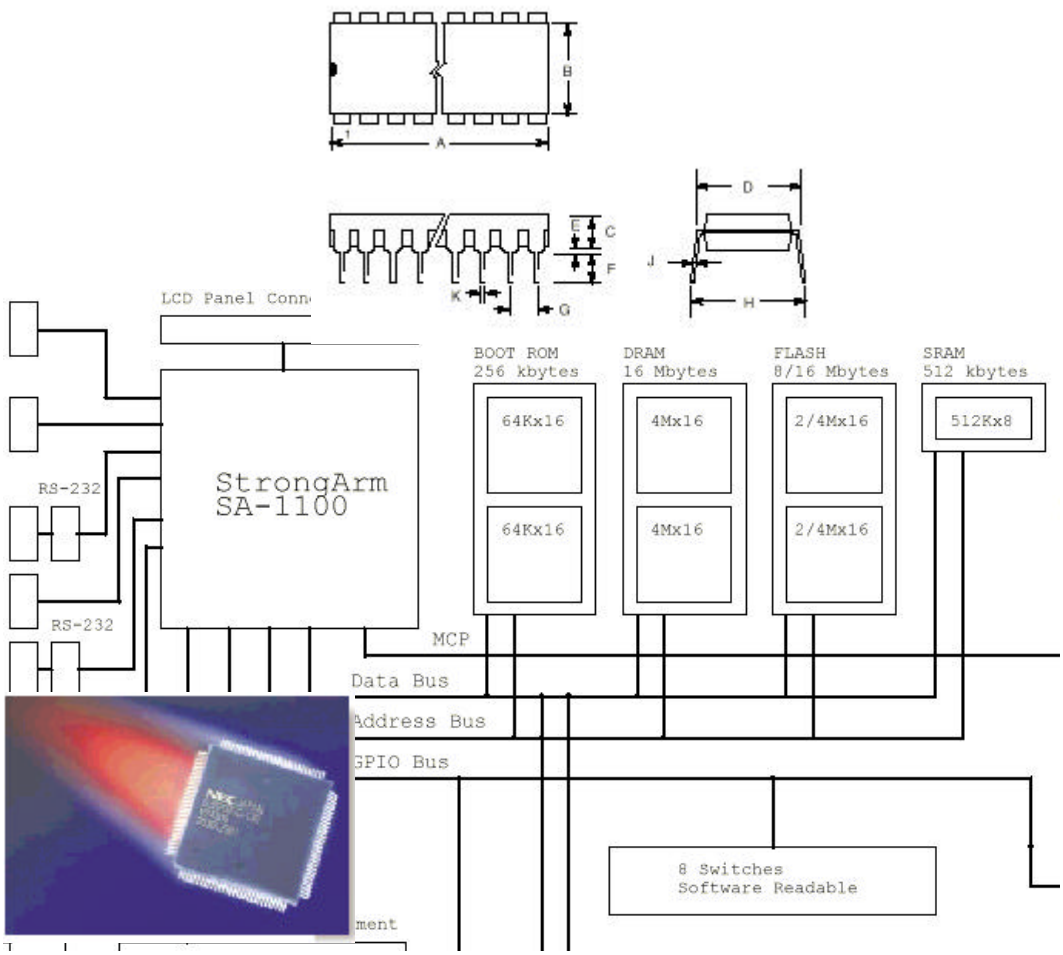
Universidad de Huelva

Escuela Politécnica Superior
Universidad de Huelva

TERCER CURSO. ELECTRÓNICA DIGITAL

CIRCUITOS COMBINACIONALES MSI, PROGRAMABLES Y ARITMÉTICA



Manuel Sánchez Raya
Versión 1.1
1 de Octubre de 2000

ÍNDICE

1.- CIRCUITOS COMBINACIONALES.

En este tema se realizará un repaso de conceptos vistos en cursos anteriores, haciendo hincapié en las bases de la lógica combinacional.

1.1.- Sistemas numéricos. Códigos.

Un **alfabeto** es el conjunto de símbolos cuya combinación produce los distintos valores de entrada y salida. Este proceso se denomina **codificación**. Los sistemas de numeración pueden ser de dos tipos:

- **Simbólico**: cada símbolo tiene un valor independientemente del lugar que ocupa. Por ejemplo, los números romanos.
- **Ponderado**: Cada símbolo tiene un valor que es función de su posición. Cada número tendrá un valor resultante como combinación del número asignado al símbolo y la posición que ocupe dentro del número. Por ejemplo, todos los sistemas de numeración posicionales.

Un número se puede representar en cualquier base por una parte entera y otra fraccionaria.

$N_x = d_j\ d_{j-1} \ldots d_2\ d_1\ d_0\ d_{-1}\ d_{-2} \ldots d_{-k}$

$N_{x\,(r} = d_j \cdot r^j + d_{j-1} \cdot r^{j-1} + \ldots + d_1 \cdot r^1 + d_0 \cdot r^0 + d_{-1} \cdot r^{-1} + \ldots + d_{-k} \cdot r^{-k} = \sum_{-k}^{j} d_i \cdot r^i$

El sistema binario esta formado por un conjunto de símbolos: 0 y 1. Es un código ponderado como el decimal. Cada dígito se denomina **bit**.

Nibble	: código de 4 símbolos o bits
Byte	: código de 8 símbolos o bits
Word	: Dos bytes, 16 bits.
Double-Word	: Cuatro bytes, 32 bits.
Quadruple-Word	: Ocho bytes, 64 bits.

Ejemplos:

$N_{(2} = 1101.011$
$N_{(10} = 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} = 13.375_{(10}$

MSB: Most Significative Bit, bit más significativo.
LSB: Least Significative Bit, bit menos significativo.

Hay varios códigos potencia de dos, que son los más útiles:

Decimal	Binario	Octal	Hexadecimal
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Ejemplos:

$271_{(8} = 185_{(10}$ $271_{(8} = 010\ 111\ 001_{(2} = 0000\ 1011\ 1001_{(2} = 0B9_{(16}$

$1D2_{(16} = 466 = 0001\ 1101\ 0011_{(2} = 722_{(8}$

1.1.1.- Representación de números negativos.

- **Magnitud signo**: se añade un bit al comienzo que será el bit de signo, codificando el número como positivo si este vale cero o como negativo si vale uno.

-7.11 = 1 0000111 . 00011 con patrón byte. El bit subrayado es el bit de signo y el resto de bits representa la **mantisa**.

- **Complemento a 1** (a la base menos la unidad). No hay bit de signo, el número negativo se obtiene complementando el positivo a la base. En la práctica se obtiene invirtiendo el número positivo y sumando uno.

$$Comp_{B-1}(N) = B^e - B^{-f} - N_0 = \{Para\ enteros\} = B^e - 1 - N_0$$

- **Complemento a 2** (a la base). Se obtiene invirtiendo lógicamente la representación binaria del número.

$$Comp_B(N) = B^e - N_0 = \{Para\ enteros\} = B^e - N_0$$

Número decimal	Representación signo-magnitud	Representación complemento a 1	Representación complemento a 2
+127	01111111	01111111	01111111
+31	00011111	00011111	00011111
+3	00000011	00000011	00000011
+0	00000000	00000000	00000000
-0	10000000	11111111	00000000
-3	10000011	11111100	11111101
-31	10011111	11100000	11100001
-127	11111111	10000000	10000001
-128	-	-	10000000

Ejemplo:

Complemento (101.0011) = $2^3 - 2^{-4} - 101.011 = 8 - 1/16 - 101.011 = 1000 - 0.0001 - 101.0011 = 1000 - 101.0100 = 010.0100$

1.1.2.- Códigos Binarios.

Un código binario es **continuo** si las combinaciones correspondientes a números decimales consecutivos son adyacentes, es decir, se diferencian solo en un bit. Un código continuo en que la última combinación es adyacente a la primera se denomina cíclico. Los códigos continuos más usados son el código Gray y el Johnson.

El código Gray también se denomina código binario reflejado. Presenta la propiedad de variar solo en un bit para cada posición y por ello se emplea para discos codificados, eliminándose la posibilidad de proporcionar un código incorrecto. El código Johnson se utiliza para la conversión analógica/digital.

	Código Gray	Código Johnson
1	0000	00000
2	0001	00001
3	0011	00011
4	0010	00111
5	0110	01111
6	0111	11111
7	0101	11110
8	0100	11100
9	1100	11000
10	1101	10000

Los códigos decimales codificados en binario se emplean para representar los decimales mediante la combinación de los signos de base dos. Los códigos ponderados son los que cada posición se le asigna un peso.

- **BCD con pesos 8421**: cada número decimal se codifica directamente en un código binario de cuatro bits, siendo inválidas las combinaciones mayores de 9.
- **BCD Aiken con pesos 2421**: es un código autocomplementario, porque el complemento a 9 se obtiene mediante inversión lógica de cada bit.
- **5421** : es un código ponderado pero no es autocomplementario.
- **BCD en exceso 3**: cada número decimal N se representa como N+3, no es un código ponderado pero si es autocomplementario.

	BCD	BCD Aiken	5421	BCD exceso3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0010	0101
3	0011	0011	0011	0110
4	0100	0100	0100	0111
5	0101	1011	1000	1000
6	0110	1100	1001	1001
7	0111	1101	1010	1010
8	1000	1110	1011	1011
9	1001	1111	1100	1100

1.1.3.- Códigos alfanuméricos.

El código **ASCII** (American Standard Code for Information Interchange) define códigos de ocho bits para letras, números y símbolos especiales. El código ASCII de seis bits es el siguiente:

Bits			6	0	0	1	1
4	3	2	1 \ 5	0	1	0	1
0	0	0	0	@	P	Ð	0
0	0	0	1	A	Q	!	1
0	0	1	0	B	R	»	2
0	0	1	1	C	S	#	3
0	1	0	0	D	T	$	4
0	1	0	1	E	U	%	5
0	1	1	0	F	V	&	6
0	1	1	1	G	W	‘	7
1	0	0	0	H	X	(	8
1	0	0	1	I	Y	)	9
1	0	1	0	J	Z	*	:
1	0	1	1	K	[	+	;
1	1	0	0	L	\	,	<
1	1	0	1	M	]	-	=
1	1	1	0	N	?	.	>
1	1	1	1	O	?	/	?

1.2.1.- Teoremas básicos del álgebra de boole.

El Algebra de Boole **{B}** es un conjunto finito compuesto de dos elementos (el nulo y el universal) y tres operaciones (dos binarias y una unaria).

Elemento nulo:	0
Elemento universal:	1
Operación or (+):	$? x, y ? \ B \ \ x ? \ y ? \ B$
Operación and (·):	$? x, y ? \ B \ \ x \cdot y ? \ B$
Complemento:	$? x ? \ B, ? \overline{x} ? \ B$

Cumple además los siguientes axiomas:

A1: Idempotencia	$x \cdot x = x$	$x + x = x$
A2: Conmutatividad	$x \cdot y = y \cdot x$	$x + y = y + x$
A3: Asociabilidad	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	
A4: Absorción	$x \cdot (x + y) = x$	$x + (x \cdot y) = x$
A5: Distributibidad	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	
	$x + (y \cdot z) = (x + y) \cdot (x + z)$	
A6: Elemento neutro	$x + 0 = x$	$x \cdot 0 = 0$
	$x + 1 = 1$	$x \cdot 1 = x$
A7: Elemento inverso	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$

A partir de estos axiomas podemos obtener los siguientes teoremas:

- **Teorema de dualidad**:

Si en una identidad se sustituye cada (+, · , 0, 1) por (· , +, 1, 0) se obtiene una función dual de la anterior. Todo teorema tiene dos formas, la enunciada y su dual.

- **Leyes de simplificación**:

$x + (\overline{x} \cdot y) = x+y \qquad x \cdot (\overline{x} + y) = x \cdot y$

- **Teorema de De Morgan**:

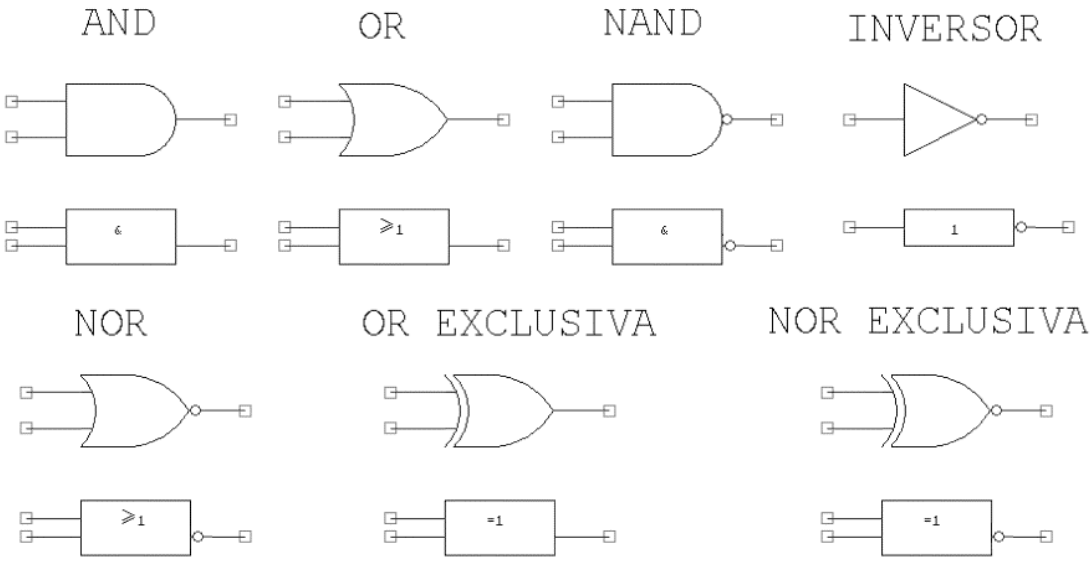
El complemento de la suma es el producto de los complementos. Su dual también es cierto.

$\overline{x + y + z \ + ...} = \ \overline{x} \cdot \overline{y} \cdot \overline{z} \cdot$

$\overline{x \ \cdot \ y \ \cdot \ z \ \cdot \ ...} = \ \overline{x} ? \overline{y} ? \overline{z} \cdot$

2.2.- Funciones de conmutación. Puertas lógicas.

Las más usadas son las siguientes:



Cuyas tablas de verdad son las siguientes:

B	A	OR(+)	AND(·)	NOR	NAND	OR EX(?)	NOR EX
0	0	0	0	1	1	0	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	1	0	0	0	1

A	NOT
0	1
1	0

2.3.- Formas canónicas. Mintérminos y Maxtérminos.

- **Término producto**: producto lógico de literales donde cada variable aparece como mucho una vez.
- **Término suma**: suma lógica de literales donde cada variable aparece una sola vez.
- **Suma canónica o maxtérmino**: término suma donde aparecen todas las variables una sola vez complementadas o no.
- **Producto canónico o mintérmino**: término producto donde aparecen todas las variables una sola vez complementadas o no.

Propiedades de MINTÉRM.(m) y MAXTÉRM.(M):

- El número de mintérminos o maxtérminos que se pueden construir con n-variables es 2^n.
- Dados los 2^n mintérminos y asignando a cada variable un valor {0,1}, solo hay un mintérmino que toma el valor uno para dicha asignación. Igual para maxtérminos
- La suma de los 2^n mintérminos que se pueden construir con n-variables es idénticamente igual a uno. Igual para maxtérminos.

Para Mintérminos: $\sum_{i=0}^{2^n-1} m_i(x_1, x_2, \dots, x_n) = 1$

Para Maxtérminos: $\prod_{i=0}^{2^n-1} M_i(x_1, x_2, \dots, x_n) = 1$

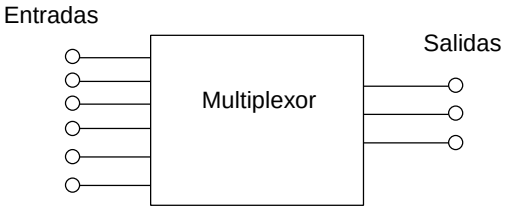
X_1	X_2	X_3	MINTÉRMINOS	MAXTÉRMINOS
0	0	0	$m_0 = \overline{X_1}\,\overline{X_2}\,\overline{X_3}$	$M_0 = X_1 + X_2 + X_3$
0	0	1	$m_1 = \overline{X_1}\,\overline{X_2}\,X_3$	$M_1 = X_1 + X_2 + \overline{X_3}$
0	1	0	$m_2 = \overline{X_1}\,X_2\,\overline{X_3}$	$M_2 = X_1 + \overline{X_2} + X_3$
0	1	1	$m_3 = \overline{X_1}\,X_2\,X_3$	$M_3 = X_1 + \overline{X_2} + \overline{X_3}$
1	0	0	$m_4 = X_1\,\overline{X_2}\,\overline{X_3}$	$M_4 = \overline{X_1} + X_2 + X_3$
1	0	1	$m_5 = X_1\,\overline{X_2}\,X_3$	$M_5 = \overline{X_1} + X_2 + \overline{X_3}$
1	1	0	$m_6 = X_1\,X_2\,\overline{X_3}$	$M_6 = \overline{X_1} + \overline{X_2} + X_3$
1	1	1	$m_7 = X_1\,X_2\,X_3$	$M_7 = \overline{X_1} + \overline{X_2} + \overline{X_3}$

Las funciones lógicas combinacionales se pueden especificar de varias formas:

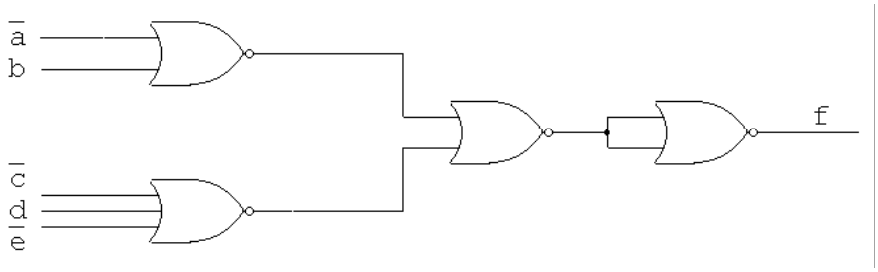
- **Tabla de Verdad:**

	c	b	a	f_3
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

- **Esquema del circuito.** Bloque o caja negra con entradas y salidas y descripción de la función que realiza.



- **Diagrama de puertas lógicas**:



- **Expresión algebraica:**

Puede ser suma de mintérminos: $f_3 = abc + a\bar{b}c + \bar{a}\bar{b}c$, o producto de maxtérminos: $f_4 = (a + \bar{b} + \bar{c})\cdot(\bar{a} + \bar{b} + \bar{c})\cdot(a + b + \bar{c})$.

En *forma canónica*: $f_3 = abc + a\bar{b}c + \bar{a}\bar{b}c$

En *forma no canónica*: $f_5 = a + b\bar{c} + de$

Para pasar a forma canónica mintérminos se multiplica por $(x + \overline{x})$, donde x es el factor que falta al término producto. Para pasar a forma canónica maxtérminos se suma $(x \cdot \overline{x})$, donde x es el factor que falta al término suma.

Se puede pasar de suma de productos a productos de suma y viceversa:

Forma canónica PDS de f = $\overline{\text{Forma estándar SDP de } \overline{f}}$

Forma canónica SDP de f = $\overline{\text{Forma estándar PDS de } \overline{f}}$

Ejemplo:

$$f_3 = \sum_3 m(4,5,7) \quad = \quad \bar{f}_3 = \sum_3 m(0,1,2,3,6)$$

$$\bar{\bar{f}}_3 = \prod_3 M(0,1,2,3,6)$$

- **Mapas de Karnaugh:**

Tablas formadas por celdas donde cada celda adyacente varía en un solo bit.

cd \ ab	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

1.2.4.- Simplificación de Funciones. Mapas de Karnaugh

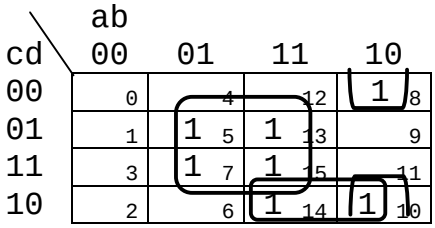
Los procedimientos de simplificación de funciones booleanas se desarrollan con el propósito de obtener una expresión booleana mínima que permita construir un circuito con el menor número posible de puertas lógicas.

El criterio que vamos a utilizar es el de obtener un circuito mínimo en dos niveles. Esto se consigue a partir de una expresión booleana normalizada, bien como suma de productos, bien como producto de sumas que sea mínima en cuanto al número de términos utilizados, (producto o suma) y mínima en cuanto al número de literales de los mismos.

Los mapas de Karnaugh son tablas de verdad construidas de tal manera que mintérminos (o maxtérminos) adyacentes ocupan una posición adyacente en la tabla.

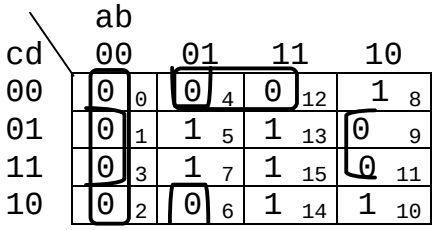
Son especialmente útiles para la simplificación manual de funciones booleanas de hasta seis variables, porque permiten encontrar los implicantes primos, e identificar los esenciales, por simple inspección ocular.

Ejemplo: Simplificar la función $f_4 = \sum_4 m(5,7,8,10,13,14,15)$



$f = ac + \bar{a}\bar{c}d + \bar{a}bd$

También se puede obtener a partir del diagrama de Karnaugh la función como producto de sumas agrupando ceros en lugar de unos, en algunas ocasiones nos puede interesar hacerlo de esta forma.



$f = (\bar{a} + c) \cdot (c + d) \cdot (a + d) \cdot (\bar{c} + b + a)$

Para simplificar funciones de más variables se emplean los otros métodos. Para cinco variables se pueden hacer dos tablas de cuatro variables y para seis cuatro tablas.

Funciones incompletamente definidas.

Se llaman así a las funciones booleanas que no están definidas para todas las combinaciones de entrada, esto es, su dominio es un subconjunto de todas las posibles combinaciones de las variables de entrada.

En los mapas de Karnaugh, las combinaciones no definidas suelen representarse mediante el símbolo **X**, (indiferente o indeterminación) y será utilizado como 1 o 0, según convenga, a la hora de buscar los grupos de la función, durante el proceso de simplificación.

Para obtener funciones booleanas mínimas utilizando los mapas de Karnaugh se siguen los siguientes pasos:

1- Se construye el mapa de Karnaugh adecuado según el número de variables de la función dada.
2- Se representan sobre él los mintérminos (maxtérminos) que hacen uno (cero) a la función para la simplificación mediante suma de productos (producto de sumas).
3- Se representan las posibles indefiniciones de la función (si las hay).
4- Se obtienen los grupos de la función, buscando mintérminos (maxtérminos) adyacentes, empezando por las de mayor orden. En esta operación las indefiniciones se toman como unos o ceros según convenga.
5- De todos los grupos se selecciona un conjunto mínimo que cubra todos los mintérminos (maxtérminos) para formar la expresión mínima en forma suma de productos (producto de sumas):
    - En primer lugar se escogen aquellos grupos que sean esenciales, esto es, aquellos que son los únicos que cubren a algún mintérmino (maxtérmino).
    - Se verifica si con estos se cubren todos los mintérminos (maxtérminos) de la función. Si es así, ya se tiene la función mínima.
    - En caso contrario se añaden grupos no esenciales hasta conseguir la cobertura. En esta elección se escogen en primer lugar aquellos grupos que cubran a más mintérminos (maxtérminos) y que contengan el menor número de literales.

2.4.1.- Realización de funciones con puertas lógicas.

Las redes o sistemas generados por interconexión de elementos lógicos se denominan circuitos lógicos. Los diagramas de bloques que representan la estructura de estos sistemas, y que utilizan los símbolos gráficos asociados a las puertas lógicas, se denominan diagramas lógicos.

Los diagramas lógicos de los sistemas combinacionales dan lugar a grafos acíclicos, esto es, que no contienen bucles cerrados o lazos de realimentación.

En un diagrama lógico bien construido la salida de una puerta lógica se conecta a la entrada de una o más puertas lógicas sin formar bucles de realimentación. Tampoco se permite conectar entre sí las salidas de distintas puertas lógicas.

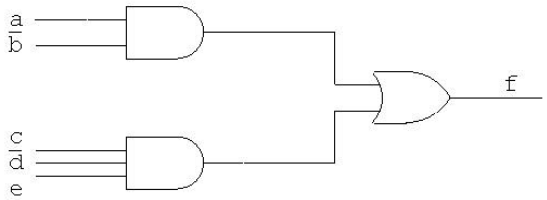
NAND y NOR forman cada uno de ellos un conjunto completo de operadores, por tanto cualquier función booleana puede implementarse utilizando exclusivamente uno de estos dos tipos de puertas lógicas. Este tipo de implementaciones ofrece ciertas ventajas debido a que:

- **NAND y NOR** son tecnológicamente más simples de construir que los operadores AND y OR
- Los circuitos lógicos así construidos son más uniformes y de más fácil mantenimiento.
- Las características anteriores permiten una mejor integración para construir circuitos integrados MSI, LSI, VLSI.

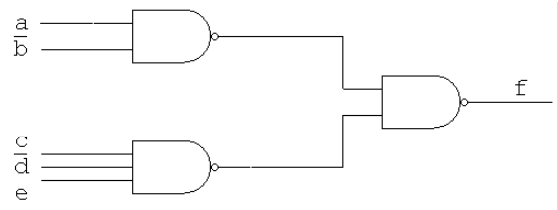
Para diseñar un circuito con puertas NAND o NOR, se parte de una expresión normalizada y mínima, y se transforma utilizando los teoremas del álgebra de Boole, y especialmente las leyes de De Morgan:

- Se aplican a la función canónica dos inversiones, con lo cual la misma queda invariable.
- Si la operación más externa es una suma (producto) lógica, se opera una de las dos inversiones aplicando De Morgan y si es producto (suma) no se opera ninguna de las dos.
- Si en el interior existen sumas (productos) lógicos, se aplican a cada uno de ellos dos inversiones y se opera uno de ellos para convertirle en el inverso de un producto (suma).
- Se continua realizando esta operación hasta que todas las sumas (productos) queden convertidas en inversos de productos (sumas).

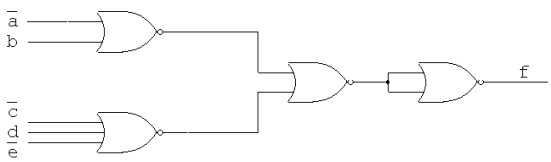
Ejemplo: $f = a\bar{b} + c\bar{d}e$



Realización mediante puertas NAND: $f = \overline{\overline{a\bar{b}} \cdot \overline{c\bar{d}e}} = \overline{\overline{a\bar{b}} \cdot \overline{c\bar{d}e}}$



Realización mediante puertas NOR: $f = \overline{\overline{a\bar{b}} + \overline{c\bar{d}e}} = \overline{\overline{a\bar{b}}} + \overline{\overline{c\bar{d}e}} = \overline{(\bar{a} + b)} + \overline{(c + \bar{d} + \bar{e})}$



1.2.5.- Bloques funcionales combinacionales.

Para diseñar utilizamos bloques integrados que existen en el mercado y así construimos un diseño modular. Estos montajes modulares dieron paso a los Circuitos Integrados. Escalas de integración:

- **SSI** (5 puertas) Small Scale Integration. Puertas lógicas.
- **MSI** (100 puertas) Medium Scale Integration. Multiplexores y decodificadores.
- **LSI** (2.000 puertas) Large Scale Integration. Procesadores, ROM.
- **VLSI** (500.000 puertas) Very Large Scale Integration. 486 – Pentium.

Disponemos de:
- **Bloques universales**: aquellos mediante los cuales podemos diseñar cualquier circuito combinacional. Ej: multiplexores, demultiplexores.
- **Bloques específicos**: orientados a realizar funciones concretas. Ej: sumador, comparador.

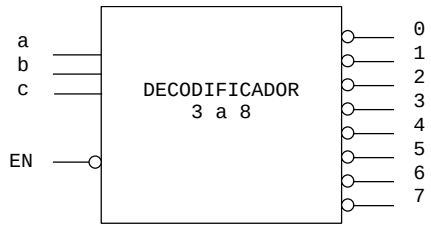
Estos dispositivos suelen incorporar:

- Entradas de **habilitación** (enable): si la entrada enable se pone a uno y la salida del dispositivo es activa a nivel alto todas las salidas pasan a valer cero.
- Entradas de control de **tercer estado**: cuando se activa, todas las salidas pasan a tercer estado, esto es no están a cero ni a uno, sino como si estuvieran desconectadas.
- Entradas de **modo de operación**: selecciona una de varias operaciones lógicas a realizar con las entradas.
- Salidas en **colector abierto**: la etapa de salida esta formada por un transistor conectado como interruptor de corriente.

1.2.5.1.- Decodificadores.

Los circuitos decodificadores son sistemas combinacionales que generan los productos canónicos de un conjunto de variables binarias aplicadas a sus entradas.

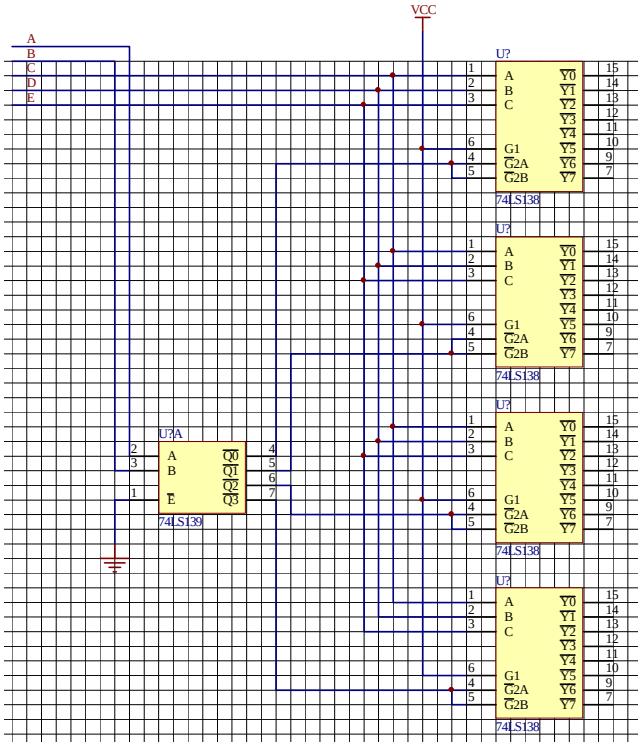
Ej: Decodificador 3 a 8, entradas a nivel alto, salidas a nivel bajo.



E	a	b	c	0	1	2	3	4	5	6	7
1	X	X	X	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1
.											
.											
.											
0	1	1	1	1	1	1	1	1	1	1	0

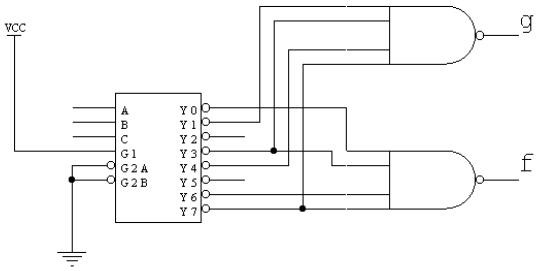
Se pueden agrupar para formar decodificadores más grandes.

Ejemplo: cuatro decodificadores 2 a 4 forman un decodificador 5 a 32.

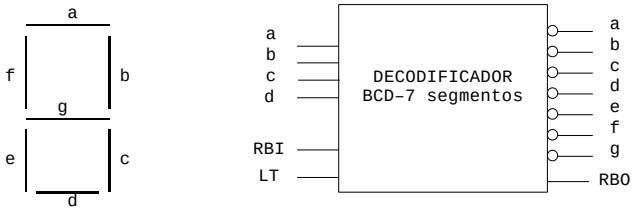


Para realizar funciones se emplean junto con una puerta OR para sumar los productos generados por el decodificador que intervienen en la función a realizar.

Ejemplo: $f ? ?_{3} (0,3,6,7)$ $g ? ?_{3} (1,3,4,7)$



Los decodificadores/excitadores son simples decodificadores empleados para encender displays y que, por tanto, disponen de alta corriente de salida.



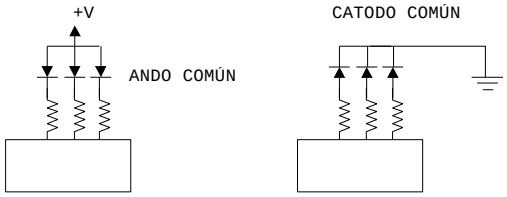


Tabla de verdad de un decodificador BCD a 7 segmentos:

LT	RBI	D	C	B	A	a	b	c	d	e	f	g	RBO
1	1	0	0	0	0	S	S	S	S	S	S	C	1
1	X	0	0	0	1	C	S	S	C	C	C	C	1
1	X	0	0	1	0	S	S	C	S	S	C	S	1
1	X	0	0	1	1	S	S	S	S	C	C	S	1
1	X	0	1	0	0	C	S	S	C	C	S	S	1
1	X	0	1	0	1	S	C	S	S	C	S	S	1
1	X	0	1	1	0	C	C	S	S	S	S	S	1
1	X	0	1	1	1	S	S	S	C	C	C	C	1
1	X	1	0	0	0	S	S	S	S	S	S	S	1
1	X	1	0	0	1	S	S	S	C	C	S	S	1
1	0	0	0	0	0	C	C	C	C	C	C	C	0
0	X	X	X	X	X	S	S	S	S	S	S	S	1

C significa que el transistor a la salida se encuentra en corte y S en saturación. La señal RBI se emplea para apagar el display y la LT para encender todos los segmentos del display. La señal de salida RBO se hace cero cuando el dígito a mostrar vale cero, de esta forma si conectamos cada salida RBO con la entrada RBI del dígito siguiente menos significativo hace que un determinado dígito solamente se active si el número presente a la entrada de su visualizador es distinto de cero o bien si siendo igual a cero, es distinto de cero alguno de los que le preceden.

1.2.5.2.- Codificadores.

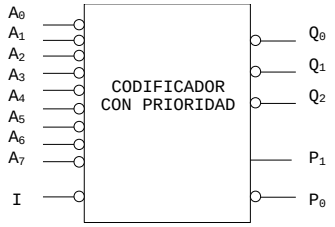
Son sistemas combinacionales de 2^n entradas y n salidas realizados de tal forma que cuando una sola de las entradas adopta un estado determinado cero o uno, a la salida aparece una combinación binaria correspondiente al número decimal asignado a dicha entrada.

Un codificador puede ser sin prioridad si podemos garantizar para su buen funcionamiento que en cada instante está a nivel activo una y solo una de las entradas. Un codificador con prioridad es aquel que permite que varias de sus entradas puedan estar activas a la vez en cualquier momento, mostrando a la salida un código correcto; para ello se establece una prioridad entre las entradas y se muestra a la salida el código asignado a la entrada más prioritaria de las que en cada instante estén activas. La prioridad se establece según el orden creciente o decreciente de los números de orden asignados a las entradas. Este sistema obedece a una función booleana más compleja.

Atendiendo a esta característica los codificadores de prioridad se clasifican en:

- Codificadores de **prioridad creciente**, aquellos para los que la entrada m es menos prioritaria que la m+1,
- Codificadores de **prioridad decreciente**, aquellos para los que la entrada m es más prioritaria que la m+1.

Ejemplo: codificador de prioridad 8 a tres con entrada de habilitación (I) y salida de detección de entrada activa.

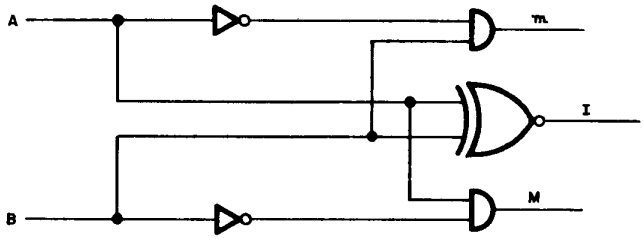


I	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Q_2	Q_1	Q_0	P_0
1	X	X	X	X	X	X	X	X	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1
0	X	X	X	X	X	X	X	0	1	1	1	0
0	X	X	X	X	X	X	0	1	1	1	0	0
0	X	X	X	X	X	0	1	1	1	0	1	0
0	X	X	X	X	0	1	1	1	1	0	0	0
0	X	X	X	0	1	1	1	1	0	1	1	0
0	X	X	0	1	1	1	1	1	0	1	0	0
0	X	0	1	1	1	1	1	1	0	0	1	0
0	0	1	1	1	1	1	1	1	0	0	0	0

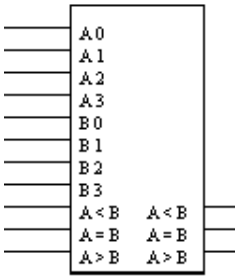
1.2.5.3.- Comparadores.

Son sistemas combinacionales que detectan si dos combinaciones binarias de n bits en el sistema binario natural son iguales o no y si no son iguales cual de ellas es mayor.

Comparador de un bit:

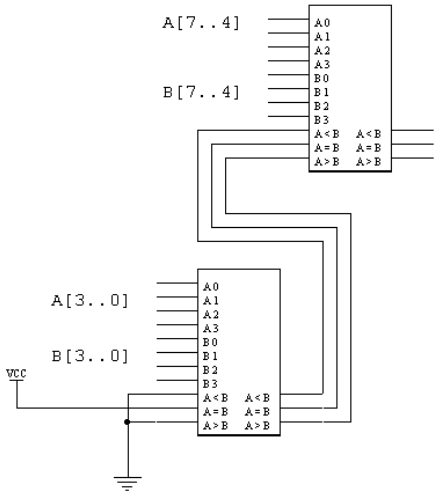


Comparador de cuatro bits:



A B	>	=	<	A<B	A=B	A>B
A > B	X	X	X	0	0	1
A < B	X	X	X	1	0	0
A = B	0	1	0	0	1	0
A = B	0	0	1	1	0	0
A = B	1	0	0	0	0	1

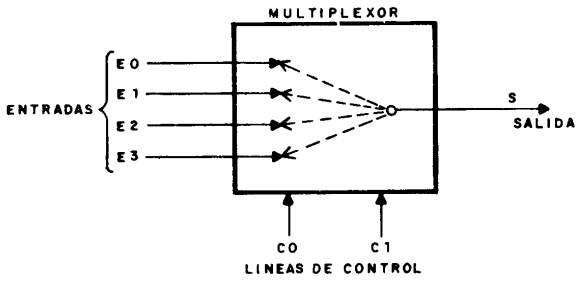
Conexión en cascada de dos comparadores:

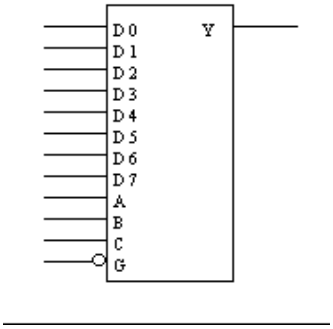


1.2.5.4.- Multiplexor.

Es un conmutador electrónico que en función de las señales de control determina cual de las entradas al circuito va a ser conectada a la salida. Dispone de 2^n señales de entrada y una de salida para n señales de control. Las señales de control especifican en código binario natural que entrada es la que se conecta a la salida.

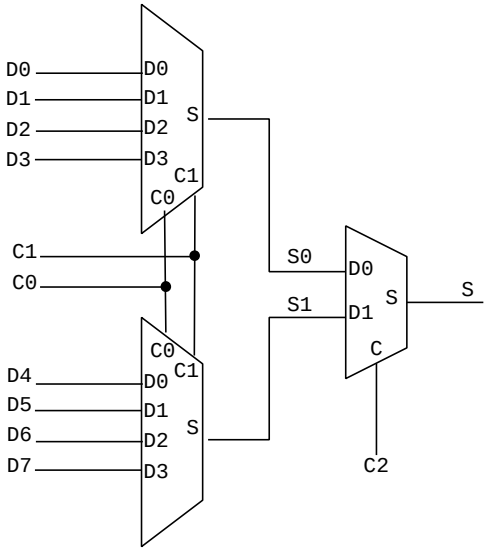
Multiplexores 4x1 y 8x1:





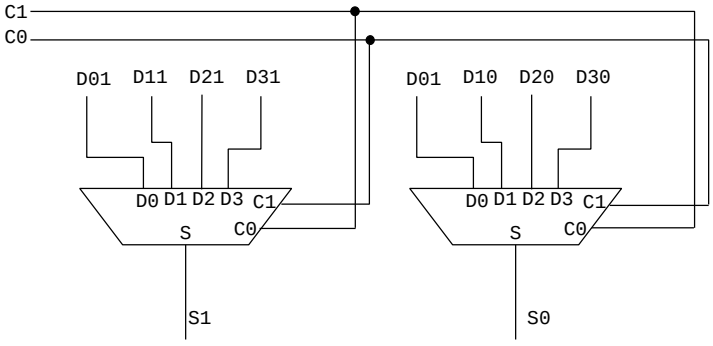
A) **Asociación en serie**. Permite aumentar el número de entradas a seleccionar.

Ejemplo: Asociación de multiplexores 4x1 y multiplexores 2x1 para formar un multiplexor 8x1.



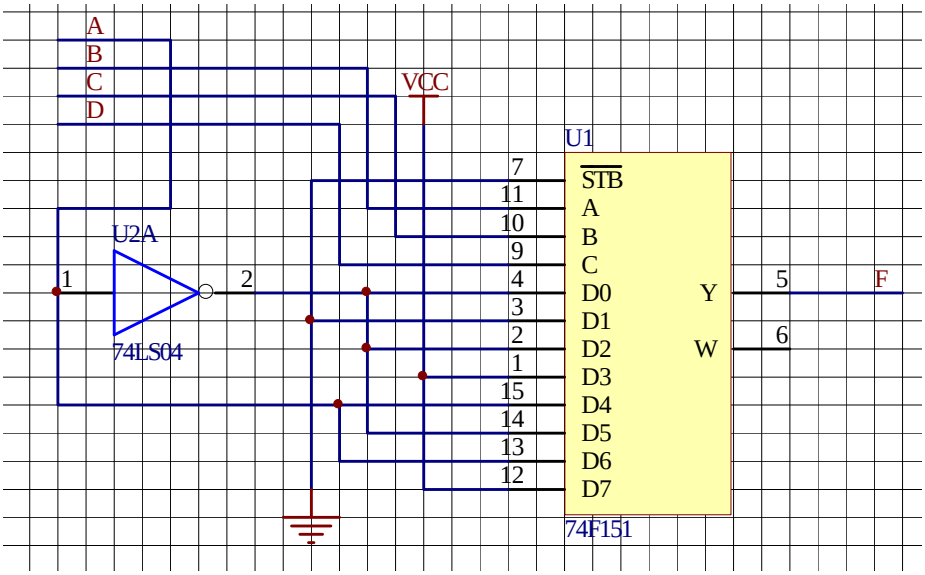
B) **Asociación en paralelo**. Permiten aumentar el número de bits de las entradas.

Ejemplo: Asociación de multiplexores 4x1 (MUX2) para formar un MUX2 para palabras de dos bits.



Con multiplexores de n entradas de selección podemos realizar funciones de n+1 variables. Ejemplo: $f ? ?_{4} m(0,2,3,5,7,11,12,14,15)$

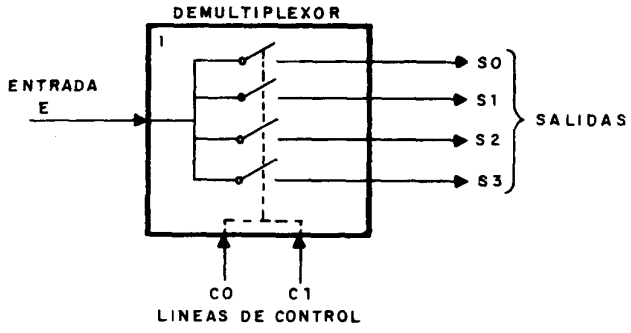
a \ bcd	C_0 000	C_1 001	C_2 010	C_3 011	C_4 100	C_5 101	C_6 110	C_7 111
0	1 0	1	1 2	1 3	4	1 5	6	1 7
1	8	9	10	1 11	1 12	13	1 14	1 15
	$\bar{a}$	0	$\bar{a}$	1	a	$\bar{a}$	a	1



1.2.5.5.- Demultiplexor.

Un demultiplexor es un circuito combinacional que posee una entrada, m salidas y c entradas de control, tal que $2^{C} ? m ? 2^{C?1}$, y que seleccionan una línea de salida para presentar el valor de la entrada. Un demultiplexor realiza la operación contraria al multiplexor encauzando los datos de una fuente común a diversos destinos.

Se nombran demultiplexor (1 x m) o bien DEMUX.



Las asociaciones de módulos demultiplexores son similares a las de módulos multiplexores, puede ser en serie o en paralelo.

1.2.6.- Análisis de circuitos combinacionales.

El análisis consiste en la determinación del comportamiento de un sistema dada su estructura y el comportamiento de sus componentes básicos. El resultado del análisis puede ser utilizado para comprender el funcionamiento de sistema y, en su caso, intentar mejorar el diseño, bien mejorando prestaciones o reduciendo costo; o bien corrigiendo deficiencias.

En los sistemas combinacionales la estructura la proporciona su diagrama lógico. En éste los componentes son las puertas lógicas básicas cuyo comportamiento es conocido. El comportamiento del sistema combinacional puede modelarse mediante multifunciones booleanas. La tarea de análisis consiste pues en obtener la función booleana correspondiente a cada salida del sistema, dependiente de las entradas del mismo. Para analizar un diagrama lógico se parte de las entradas del sistema y se avanza hacia las salidas, traduciendo en expresiones booleanas cada una de las puertas lógicas que encontramos en el camino.

Un circuito como el de la figura se dice que es multinivel, porque posee varios niveles de puertas entre las entradas y las salidas. Mientras mayor sea el número de niveles de un circuito, mayor será el retardo de propagación entrada-salida. Se denomina número de niveles de un circuito lógico al número de puertas lógicas que intervienen en el camino más largo desde la entrada a la salida. Los circuitos multinivel pueden presentar fenómenos aleatorios (riesgos) en la propagación de los cambios de las entradas debido al diferente retardo de propagación de las señales por los distintos caminos entrada-salida.

La respuesta en la salida solo depende de los valores que tenemos en la entrada del circuito una vez que este se ha estabilizado cuando no hay lazos de realimentación ni elementos de memoria. El análisis puede ser:

- **Análisis lógico**: Tabla o expresión algebraica que descubre el comportamiento de la salida del circuito en función de la entrada.
- **Análisis físico o dinámico**: estudio de cada uno de los elementos del circuito y su respuesta real en el tiempo.

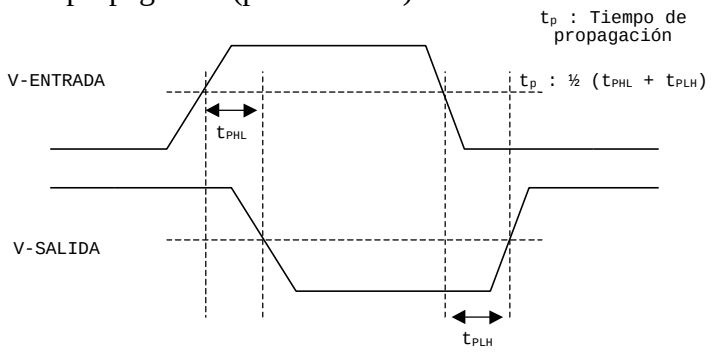
Para un análisis dinámico tendremos en cuenta el retraso del circuito, su carga y la estructura de niveles que lo forma.

Proceso:
- Se identifican los distintos niveles del circuito o puertas.
- Se identifican las entradas, las salidas y las variables intermedias.
- Se escribe la expresión de salida en función de las entradas.
- Repetir para todos los niveles hasta que las variables de salida estén en función de las entradas.

1.2.6.1.- Características técnicas de los circuitos combinacionales.

- **FAN-IN**. Número máximo de puertas del mismo tipo que la dada que pueden conectarse a ésta, de manera que el circuito funcione correctamente.
- **FAN-OUT**. Número máximo de puertas del mismo tipo que la dada que pueden conectarse como carga de ésta, de manera que el circuito funcione correctamente.
- **Coste**: El diseño puede ser más sencillo si empleamos más circuitos, pero el coste es mayor y viceversa.
- **Respuesta Temporal**. La respuesta de una puerta lógica no es instantánea, requiere un cierto tiempo para que un cambio a la entrada produzca un posible cambio a la salida. Este retraso puede originar carreras o azares e incluso la no aparición de respuesta para una posible entrada.

  t_d. Tiempo de retraso (para puerta)
  t_p. Tiempo de propagación (para circuito)



- **Retraso total** = número de niveles · tiempo de retraso de puerta
- **Retraso efectivo** = tiempo real que tarda en responder el circuito para una determinadas entradas.
- **Retraso neto** = media de los retrasos para cada una de las entradas.

1.2.6.2.- Azares lógicos

Las puertas lógicas introducen retardo en la propagación de las señales digitales que representan a las variables lógicas que procesan. Este retardo no es constante, sino que puede variar dentro de un intervalo acotado entre dos valores máximo y mínimo, especificados por el fabricante, y que es consecuencia de la dispersión de los parámetros de fabricación. Por ello el tiempo de propagación de una señal en un circuito combinacional es difícil de predecir con exactitud.

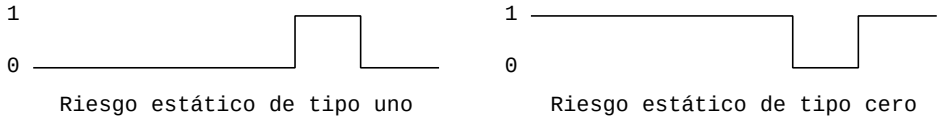
Por otro lado, una salida de un sistema combinacional puede depender de una misma variable de entrada de diversas formas que dan lugar a una multiplicidad de caminos de entrada/salida, cada uno de los cuales puede llevar asociado diferentes tiempos de propagación. Como consecuencia de esta diferencia, las transiciones de las salidas de un sistema, como consecuencia de cambios en las entradas, pueden no ser limpias y presentar ciertas fluctuaciones, de carácter aleatorio, hasta que alcanzan su valor final estable esperado. Estos fenómenos aleatorios reciben el nombre de “riesgos” (hazards). Dependiendo del uso que se haga del sistema estos fenómenos pueden ser más o menos peligrosos.

Definición: Un riesgo es una breve excursión o fluctuación de la señal de salida de un circuito digital, que produce una transición momentánea a una nivel lógico no esperado.

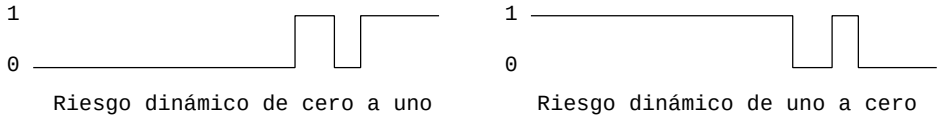
Clasificación:

Atendiendo a la forma de la fluctuación de la señal los riesgos se clasifican en riesgos estáticos y riesgos dinámicos.

- **Riesgo estático**. Se llama así al riesgo en el que la excursión consiste en que la señal pasa por un estado transitorio de nivel lógico contrario al esperado. Así se habla de riesgo estático de tipo uno y riesgo estático de tipo cero según la señal presente de forma transitoria un nivel lógico (1 o 0).



- **Riesgo dinámico**. Se llama así al riesgo que se produce cuando una señal que debe transitar de un nivel lógico a otro (0->1 o 1->0), no lo hace directamente sino que realiza una o más oscilaciones entre ambos niveles antes de alcanzar su nivel estable final.



Atendiendo a la causa específica que produce el riesgo se clasifican en riesgos lógicos y riesgos funcionales.

- **Riesgo lógico**. Se llama así al riesgo que aparece en un circuito como consecuencias de una implementación física particular de una función booleana. Puede ser eliminado rediseñando el circuito a partir de otra expresión booleana para dicha función.
- **Riesgo funcional**. Se llama así al riesgo que aparece en un circuito inducido por la propia función, independientemente de la expresión booleana que sirve como base para su implementación física. Por tanto, no puede ser eliminado rediseñando el circuito a partir de otra expresión booleana para dicha función.

Diseño de circuitos libres de riesgos.

El problema de los riesgos suele abordarse desde tres puntos de vista:

a) Se intenta eliminar el riesgo **equilibrando el tiempo de propagación** de las señales por los diferentes caminos introduciendo elementos de retardo. Es poco usado por la gran dependencia de los retardos de las puertas lógicas, e incluso de los propios elementos de retardo introducidos, respecto a la temperatura, variaciones de la tensión de alimentación, etc.

b) Se asume que existen riesgos y se trata de eludir su efecto nocivo **sincronizando los cambios** de las señales con una señal maestra y definiendo en base a ésta instantes activos, de manera que en un entorno de estos instantes esté garantizada la estabilidad de todas las señales que debían conmutar. La velocidad de funcionamiento de estos

sistemas está pues limitada por la separación entre instantes activos, que ha de coincidir con el máximo tiempo necesario para que se estabilicen todas las señales que pueden conmutar en el sistema, teniendo en cuenta todas las situaciones posibles. Una medida estimada de ese tiempo se obtiene calculando el tiempo de retardo del circuito para todos los caminos posibles desde cada una de las entradas a cada una de las salidas del sistema. Esta es la técnica más usada.

c) En algunos casos es posible aplicar ciertas reglas para rediseñar el circuito y obtener **expresiones algebraicas** que producen circuitos libres de riesgos.

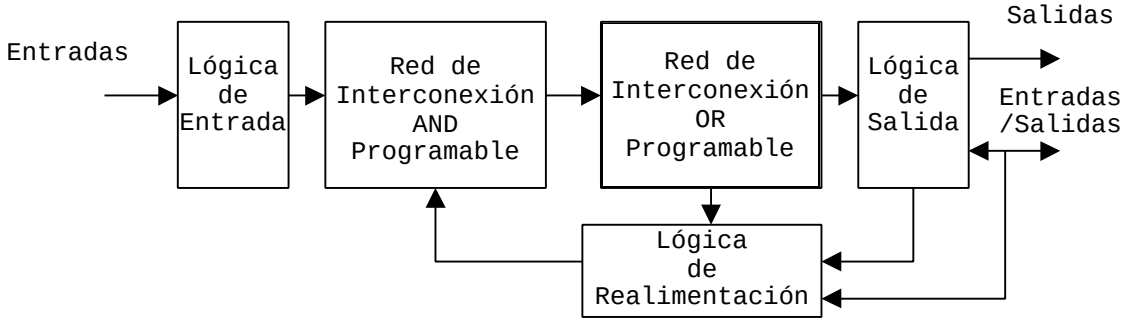
- Los **riesgos lógicos estáticos** se suelen producir cuando las entradas del sistema transitan entre combinaciones adyacentes que producen la misma salida en la función considerada y que en la expresión que implementa el circuito no están cubiertas por el mismo implicante primo. Por tanto para estos riesgos la solución está en añadir a la expresión que presenta el riesgo la adyacencia involucrada en él. Esta adyacencia se llama término de consenso. Como procedimiento general suficiente, aunque no necesario, los diseños libres de riesgos lógicos estáticos se obtiene de expresiones que contienen todos los términos de consenso de la función.

- Los **riesgos lógicos dinámicos** pueden evitarse obteniendo las expresiones booleanas de la función simplificando según el método de los mapas de Karnaugh. Así se garantiza que la expresión obtenida no contiene relaciones del tipo $A \cdot \overline{A}$ o $A + \overline{A}$.

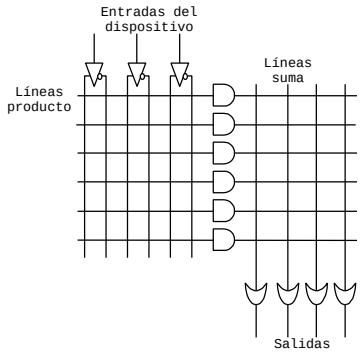
2.- SISTEMAS COMBINACIONALES PROGRAMABLES.

2.1.- Introducción.

Los sistemas digitales combinacionales que hemos visto hasta ahora son cableados, es decir, su funcionalidad se obtiene conectando los elementos lógicos de una forma fija. Sin embargo, también pueden ser programables, los cuales son más flexibles que los segundos. Los sistemas programables pueden desarrollar varias funciones usando los mismos recursos hardware. Se basan la mayoría de los casos en una matriz de puertas AND seguida de una matriz de puertas OR. De forma que desarrollan una suma de productos.

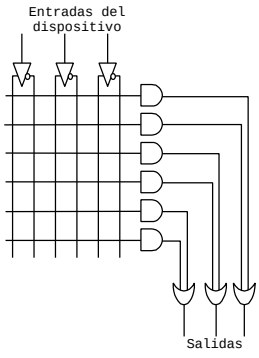


MATRIZ AND	MATRIZ OR	DISPOSITIVO
programable	programable	PLA
programable	fija	PAL
fija	programable	ROM



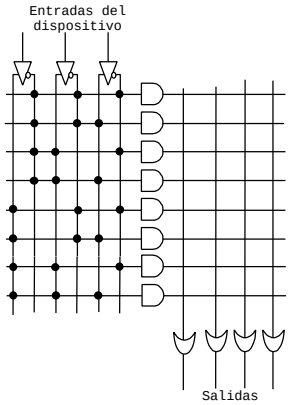
AND PROGRAMABLE – OR PROGRAMABLE

PLA (Programmable Logic Array)
Matriz Lógica Programable



AND PROGRAMABLE – OR FIJA

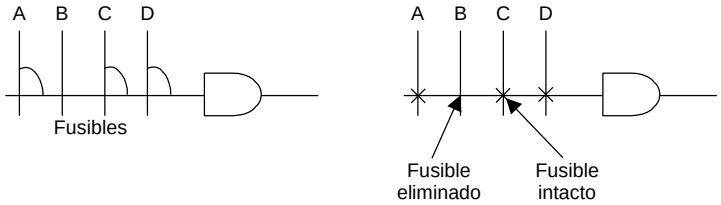
PAL (Programmable 'AND' Array Logic)
Matriz Lógica 'Y' Programable



AND FIJA – OR PROGRAMABLE

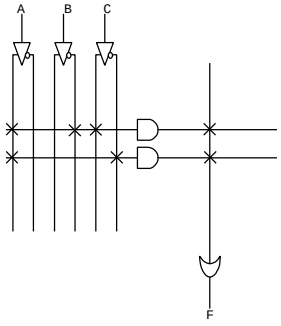
PROM (Programmable Read Only Memory)
Memoria programable de sólo lectura

En dispositivos bipolares (basados en transistores bipolares), se emplean fusibles que fundimos para programarlo. En dispositivos CMOS (basados en transistores MOS) se trata de transistores especiales que cuando se hace circular una fuerte corriente dejan de conducir, pero pueden volver a conducir aplicando una diferencia de potencial en el sustrato del circuito, borrando de esta forma toda la programación. Son eléctricamente borrables.

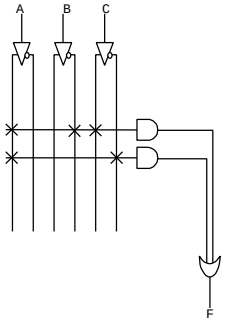


Definiciones:

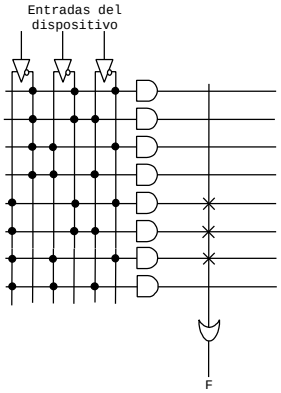
- **Línea producto**: los términos productos se representan como una línea que llega a la puerta AND.
- **Línea suma**: es una línea que representa a todos los productos que llegan a la puerta OR.
- **Sistemas Incompletos**: son aquellos para los que el número de puertas AND es mucho menor de 2 elevado al número de entradas. $P << 2^n$, por ejemplo PLA y PAL.
- Sistemas con **estructura completa**: $P = 2^n$, por ejemplo las ROM.



Realización de la función:
F=A&B'&C#A&C'
en una estructura PLA

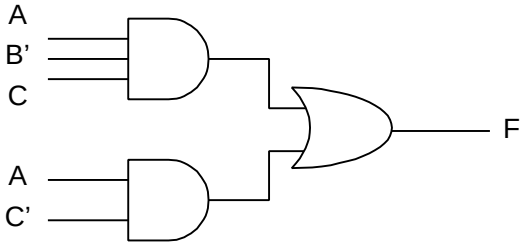


Realización de la función:
F=A&B'&C#A&C'
en una estructura PAL



Realización de la función:
F=A&B'&C#A&C'=A&B'&C#A&B&C'#A&B'&C'
en una estructura PROM

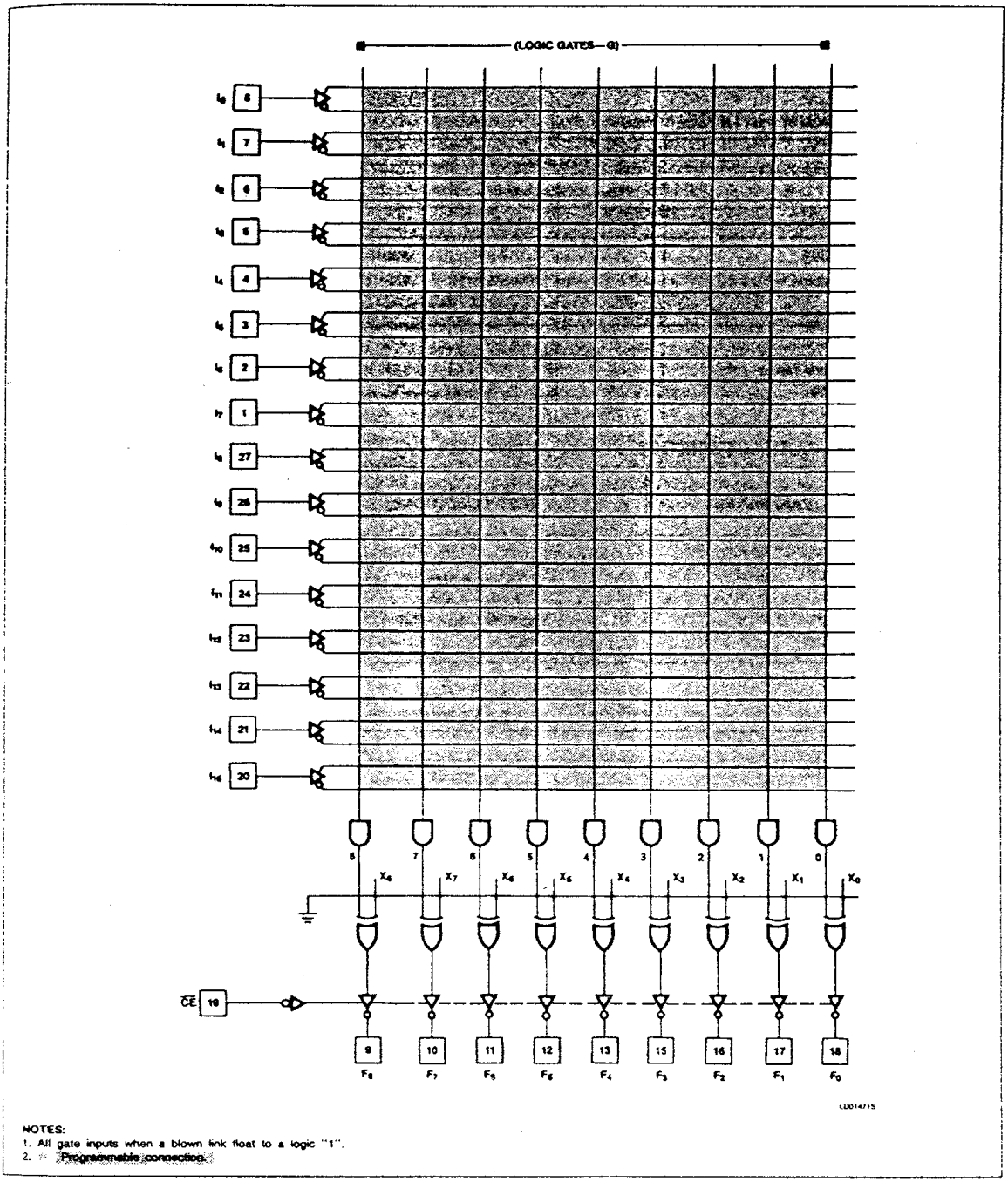
Circuito de ejemplo:



Field-Programmable Gate Array (16 × 9 × 9)

PLS103

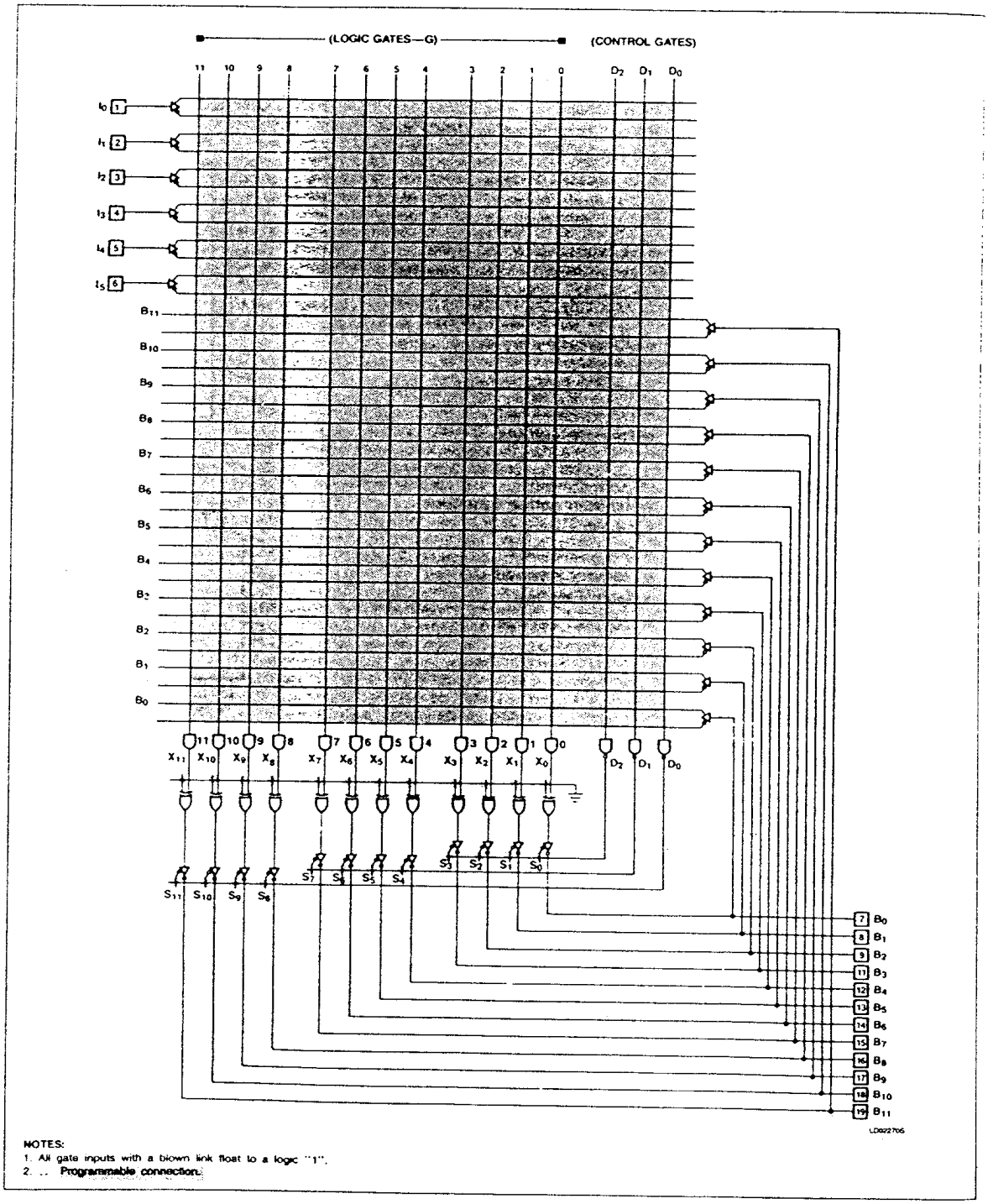
FPGA LOGIC DIAGRAM



Field-Programmable Gate Array (18 × 15 × 12)

PLS151

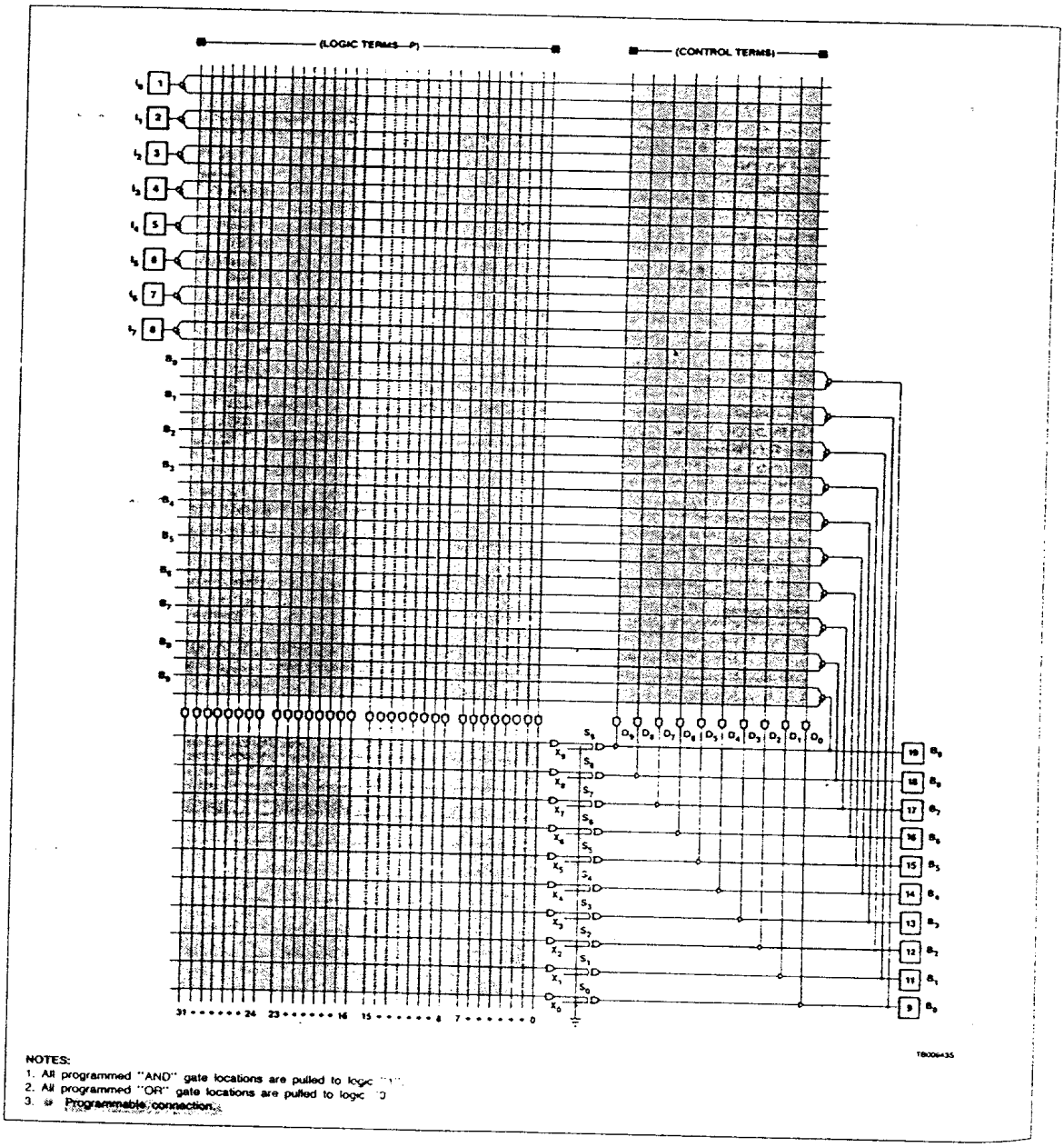
FPGA LOGIC DIAGRAM



NOTES:
1. All gate inputs with a blown link float to a logic "1".
2. ⋅ Programmable connection.

Field-Programmable Logic Array (18 × 42 × 10) PLS153

FPLA LOGIC DIAGRAM



NOTES:
1. All programmed "AND" gate locations are pulled to logic "1".
2. All programmed "OR" gate locations are pulled to logic "0".
3. ⊗ Programmable connection.

Signetics

PLHS18P8A

Programmable AND Array Logic (18 × 72 × 8)

Signetics Programmable Logic
Product Specification

Application Specific Products
- **Series 20**

DESCRIPTION

The PLHS18P8A is a two-level logic element consisting of 72 AND gates and 8 OR gates with fusible connections for programming I/O polarity and direction.

All AND gates are linked to 10 inputs (I) and 8 bidirectional I/O lines (B). These yield variable I/O gate configurations via 8 direction control gates, ranging from 18 inputs to 8 outputs.

On-chip T/C buffers couple either True (I, B) or Complement ($\overline{I}$, $\overline{B}$) input polarities to all AND gates. The 72 AND gates are separated into 8 groups of 9 each. Each group of 9 is associated with one bidirectional pin. In each group, eight of the AND terms are ORed together, while the ninth is used to establish I/O direction. All outputs are individually programmable via an EX-OR gate to allow implementation of AND/OR or NAND/NOR logic functions.

In the virgin state, the AND array fuses are back-to-back CB-EB diode pairs which will act as open connections. Current is avalanched across individual diode pairs during fusing, which essentially short circuits the EB diode and provides the connection for the associated product term.

The PLHS18P8A is field-programmable, allowing the user to quickly generate custom pattern using standard programming equipment.

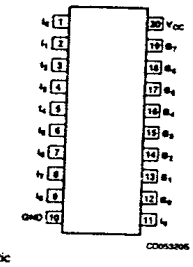
Order codes are contained in the pages following.

FEATURES

- **100% functionally compatible with AmPAL18P8A**
- **Field-Programmable**
- **10 inputs**
- **8 bidirectional I/O lines**
- **72 AND gates/product terms**
  - **configured into eight groups of nine**
- **Programmable output polarity (Tri-state output)**
- **I/O propagation delay: 20ns (max.)**
- **Power dissipation: 750mW (nominal)**
- **TTL compatible**
- **Verify Lock Fuse**
- **On-chip test features for extensive AC and DC parametric testing**

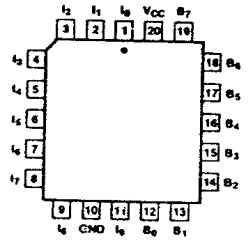
PIN CONFIGURATIONS

N Package



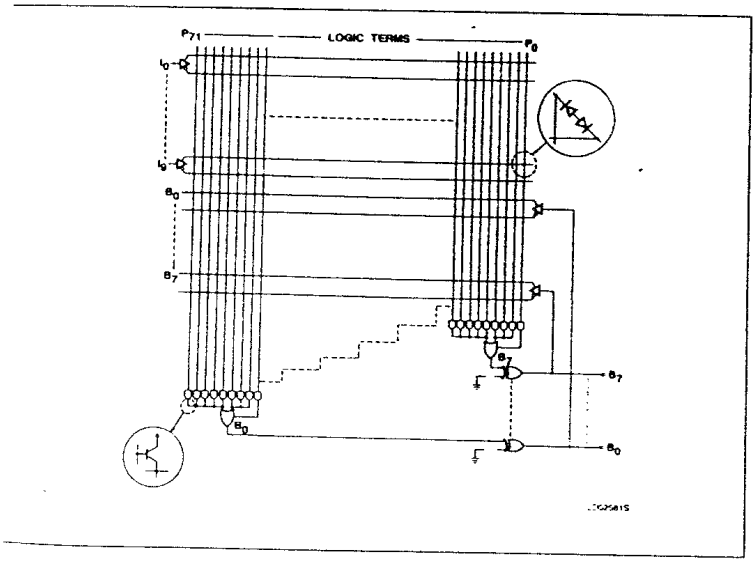
N = Plastic

A Package



A = Plastic Leaded Chip Carrier

FUNCTIONAL DIAGRAM



LOGIC FUNCTION

TYPICAL PRODUCT TERM:
$Pn = A \cdot \overline{B} \cdot C \cdot D \cdot \ldots$

TYPICAL LOGIC FUNCTION:

AT OUTPUT POLARITY = H
$Z = P0 + P1 + P2 \ldots$

AT OUTPUT POLARITY = L
$\overline{Z} = \overline{P0 + P1 + P2 + \ldots}$
$Z = \overline{P0} \cdot \overline{P1} \cdot \overline{P2} \cdot \ldots$

NOTES:
1. For each of the 8 outputs, either function Z (Active-High) or $\overline{Z}$ (Active-Low) is available, but not both. The desired output polarity is programmed via the EX-OR gates.
2. Z, A, B, C, etc. are user defined connections to fixed inputs (I) and bidirectional pins (B).

APPLICATIONS

- 100% functional replacement for all 20-pin combinatorial PALs*
- Random logic
- Code converters
- Fault detectors
- Function generators
- Address mapping
- Multiplexing

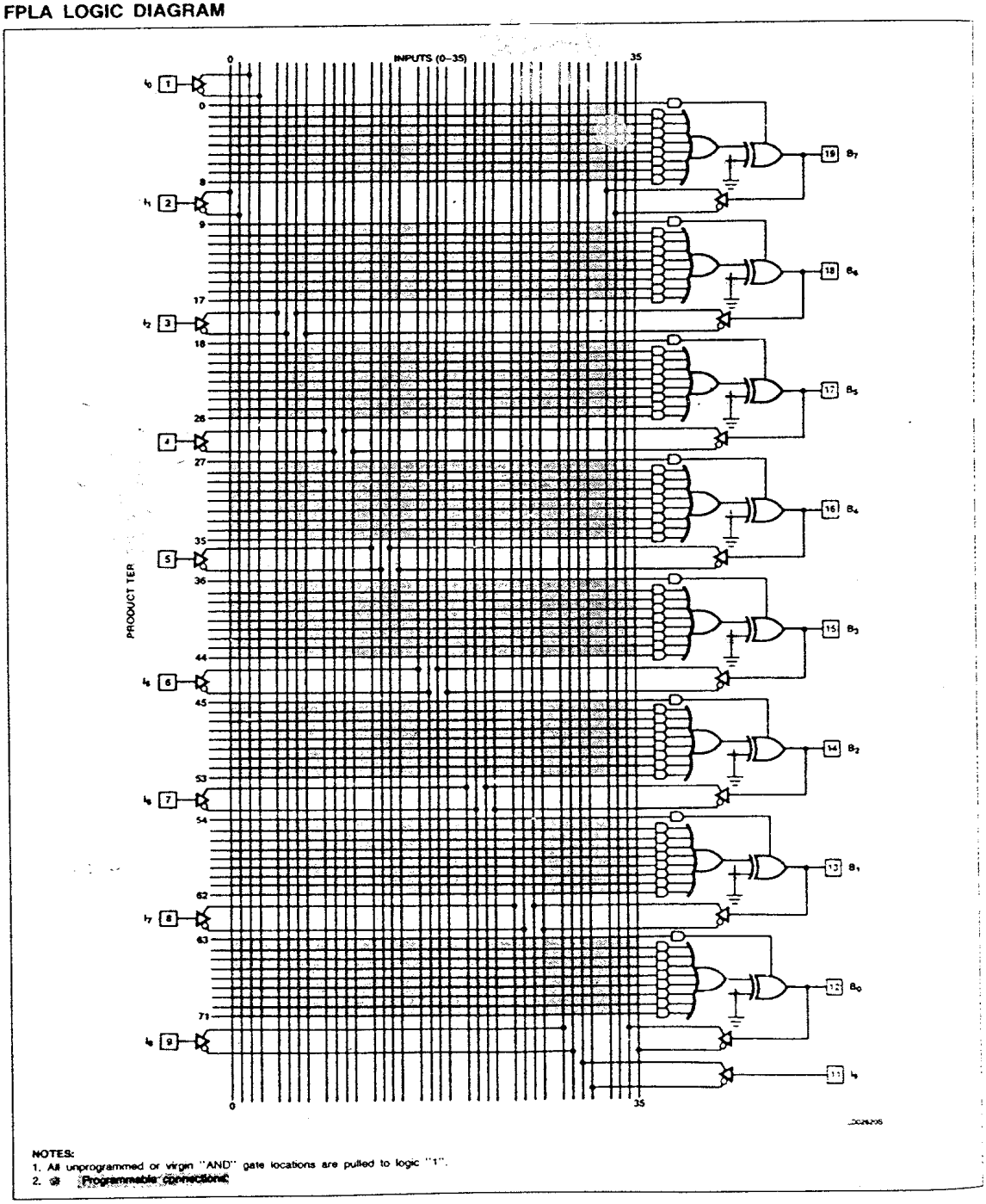
*PAL is a registered trademark of Monolithic Memories, Inc.

November 19, 1986

853-0683 86602

Programmable AND Array Logic (18 × 72 × 8) PLHS18P8A

FPLA LOGIC DIAGRAM



NOTES:
1. All unprogrammed or virgin "AND" gate locations are pulled to logic "1".
2. ⊗ Programmable connections.

2.- Memorias pasivas (ROM).

Una memoria ROM (Memoria de solo lectura) de 2^n palabras de m bits (2^n x m bits) es un bloque combinacional que agrupa un decodificador de n entradas y un conjunto de m puertas OR, una para cada salida.

La conexión entre las salidas del decodificador y las entradas de las puertas OR pueden ser especificadas de diferente manera de modo que el sistema resultante queda configurado como un elemento de almacenamiento de información de modo permanente.

Una ROM 2^n x m bits programada implementa m funciones booleanas de n variables. Cada combinación de entrada es una dirección de memoria y cada salida una palabra de m bits.

Tipos de memoria ROM:

- **ROM**, propiamente dicha. Es un dispositivo programable por máscara: es programable una sola vez durante el proceso de fabricación. Una vez fabricada no puede ser modificada.
- **PROM** o ROM programable: ROM usada en prototipos hasta conseguir una versión definitiva de la función requerida, tras lo cual se fabrica como ROM. Existe programas comerciales que permiten definir su tabla de verdad y simular su comportamiento antes de realizar su programación. Una vez programada la situación es irreversible y no puede ser modificada.
- **EPROM**: ROM programable y borrable.
- **FLASH y EEPROM**: ROM programable y borrable eléctricamente.

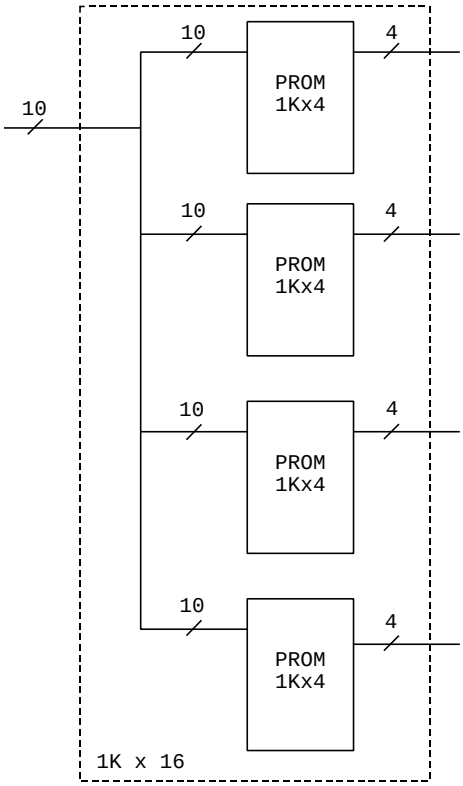
Tipo	Fabricante	Tamaño	T. de acceso	T. borrado y escritura	Vida en ciclos	Borrado mínimo
FLASH	INTEL	1 Mbit	120 ns	Total 1s byte 10?S	10^5	Completo
FLASH	TEXAS	256 Kbit	170 ns	Total 15ms Pág. 15ms	10^3	Completo
FLASH	SEEQ Tech	1 Mbit	200 ns	Total 12s Pág. 525?S	10^3	Completo o sector
EEPROM	SIMTEK	256 Kbit	120 ns	Total 10ms byte 160?S	10^5	Byte
EEPROM	XICOR	1 Mbit	200 ns	Total 5ms	10^5	Byte

Una PROM se representa por el número de posiciones que coincide con el número de puertas AND y por el número de salidas. Ej: PROM (8x4).

Ej: Una PROM de 512K x 8 tendrá 8 salidas, $2^8 = 512 \cdot 1024$ posiciones y el mismo número de puertas AND.

2.2.1.- Aumento del número de bits/posición

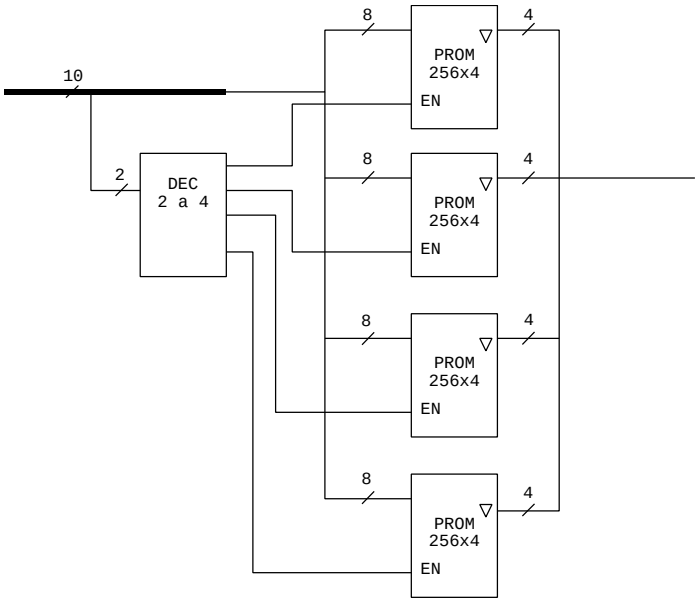
Consiste en la conexión de memorias en serie. Ej: memoria de 1K x 16 a partir de circuitos integrados de 1K x 4.



2.2.2.- Aumento del número de posiciones.

Consiste en la conexión de memorias en paralelo. Ej: Obtener una memoria de 1K x 4 a partir de circuitos integrados de 256 x 4.

Se necesitan 4 integrados de 256 para formar 1K posiciones. Además es necesario un decodificador para distribuir las 4 páginas formadas por los 4 integrados de 256 posiciones:



2.3.- Matrices lógicas programables (PLA).

Un circuito PLA es un bloque combinacional con ***n*** entradas y ***m*** salidas con una estructura AND-OR programable. Para especificar completamente un circuito PLA es necesario describir la estructura AND-OR indicando cuantos términos producto puede albergar (***k***), el número de entradas de cada término producto (en general ***n***), y el número de términos suma (en general ***m***).

Mientras que una ROM implementa todos los mintérminos de n variables, y resulta por tanto adecuada para implementar funciones a partir de su tabla de verdad; un circuito PLA permite implementar funciones booleanas a partir de expresiones normalizadas, con el ahorro que ello supone en número de elementos a integrar en un chip, pero con la limitación, claro está, de disponer de un número predeterminado (k) de términos producto.

Para diseñar una ROM de n entradas y m salidas son necesarios 2^n x m valores de programación. Para diseñar una PAL de n entradas, m salidas y k términos producto son necesarios 2(n x k) + (k x m) + m valores de programación.

Para realizar funciones booleanas con PLA es importante partir de una expresión que contenga el menor número de términos producto y que cada uno de ellos tenga el menor número de literales. Los programas de minimización de multifunciones booleanas tienen especial importancia en el diseño con PLA.

Los datos para programar una PLA se muestran en forma tabular en la que se indica la composición de cada uno de los términos producto y que términos producto forman parte de cada término suma.

Ejemplo: Tabla que describe una PLA de tres entradas y dos salidas.

Términos producto			Términos suma	
Entradas			Salidas	
A	B	C	F_1	F_2
H	L	-	?	·
H	-	H	?	?
-	H	H	·	?

La tabla refleja la información que permite realizar las funciones:

$$F_1 = A\overline{B} + AC \quad \text{y} \quad F_2 = AC + BC$$

Una PLA se representa por el número de entradas, número de términos producto y por el número de salidas: PLA (3x3x2).

2.4.- Matrices lógicas programables de puerta OR fija (PAL).

Un circuito PAL es un bloque combinacional con n entradas y m salidas con una estructura AND programable y OR fija. Para especificar completamente un circuito PAL es necesario describir la estructura AND-OR indicando cuantos términos producto puede albergar (***k***), el número de entradas de cada término producto (***n***), el número de términos suma (en general ***m***) y que puertas AND están asociadas a cada término suma.

Al igual que una PLA, un circuito PAL permite implementar funciones booleanas a partir de expresiones normalizadas, pero con las restricciones que impone el hecho de la no programabilidad de la estructura OR. Un inconveniente adicional es la imposibilidad de compartir términos producto comunes a varias funciones.

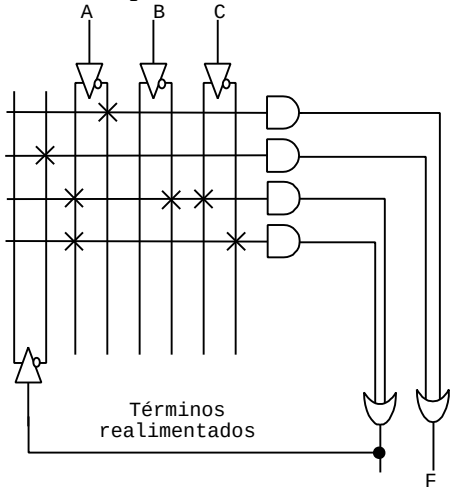
La ventaja principal de la PAL está en la mayor simplicidad de programación al ser necesario especificar un menor número de valores de programación. Para diseñar una PAL de n entradas, m salidas y k términos producto solo son necesarios 2(n+k) + m valores de programación.

Una PAL se representa también por el número de entradas, número de términos producto y por el número de salidas: PAL (3x3x2).

2.5.- Lógica multinivel.

La mayoría de los dispositivos programables se estructuran en dos niveles, y se usan mejor de esta manera. Sin embargo, algunas veces resulta necesario emplear más de dos niveles de lógica por razones de limitación de términos producto, pero con el coste de retardo adicional.

Se puede colocar en cascada dos unidades para ampliar el número de términos producto, esta conexión solo es posible en dispositivos PAL porque en una PLA estarían presentes todos los términos producto para sumarlos. El costo de hacer esto es un aumento del retardo y desperdiciar dos patillas, una salida y una entrada, a no ser que dispongamos de realimentación interna en la PAL. En este caso evitamos el desperdicio de patillas de entrada y reducimos el retardo al no pasar estas señales por los buffers de entrada y salida. Varias funciones aritméticas se pueden implementar de forma económica de esta forma, especialmente funciones de paridad que requieren muchos términos productos.



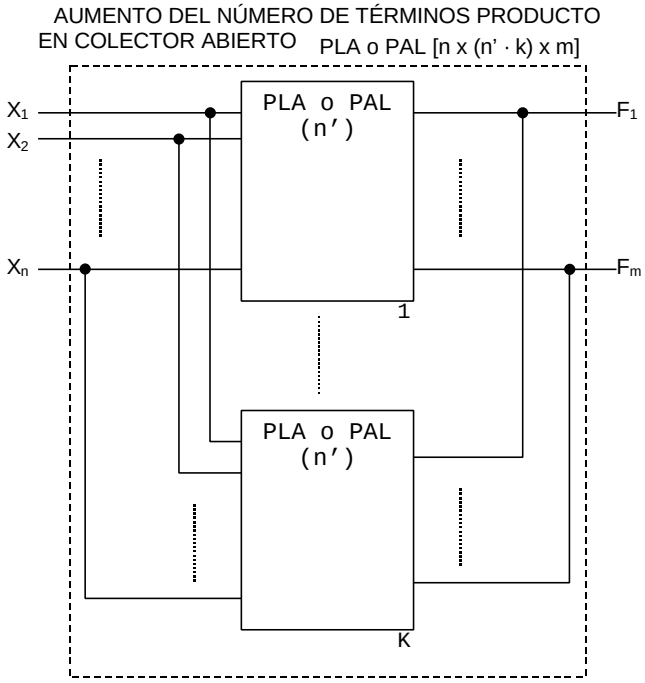
2.6.- Uso de múltiples dispositivos.

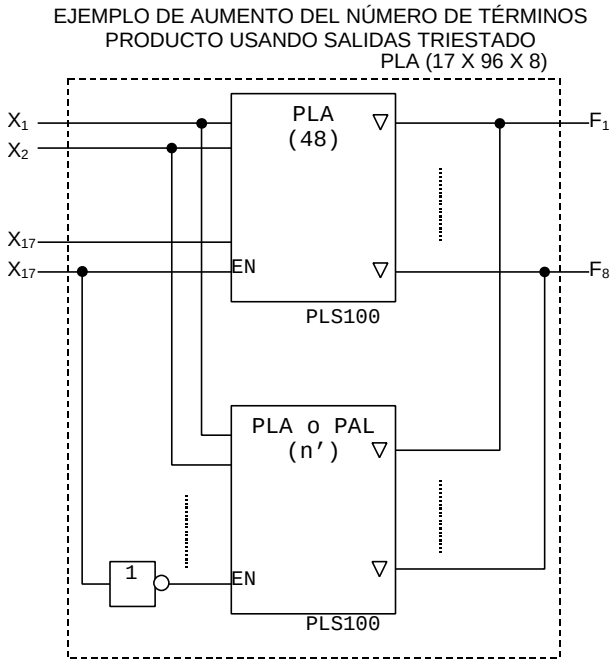
En algunas ocasiones un conjunto de funciones combinacionales no se pueden hacer con un único dispositivo. La solución en este caso consiste en descomponer el problema de tal forma que los requisitos de lógica se puedan conseguir con una red de dos o más PLDs, implementando cada una parte del problema.

El problema general va a consistir en diseñar una red para implementar un bloque lógico con **M** entradas, **N** salidas y **R** términos producto mediante dispositivos con un máximo de **m** entradas, **n** salidas y **r** productos, donde al menos un de lo siguiente es cierto: **m<M**, **n<N**, **r<R**.

2.6.1.- Expansión de términos producto.

En algunos casos tenemos un PLD que dispone de suficientes patillas de entrada y salida pero con insuficientes términos producto para resolver el problema. Si en un dispositivo PAL solo unas pocas salidas tienen este problema podemos conectar las salidas de varios dispositivos directamente si disponen de salidas en colector abierto, para que este sistema funcione, las polaridades de salida deben ser a nivel bajo y se debe conectar una resistencia externa de PULL-UP.



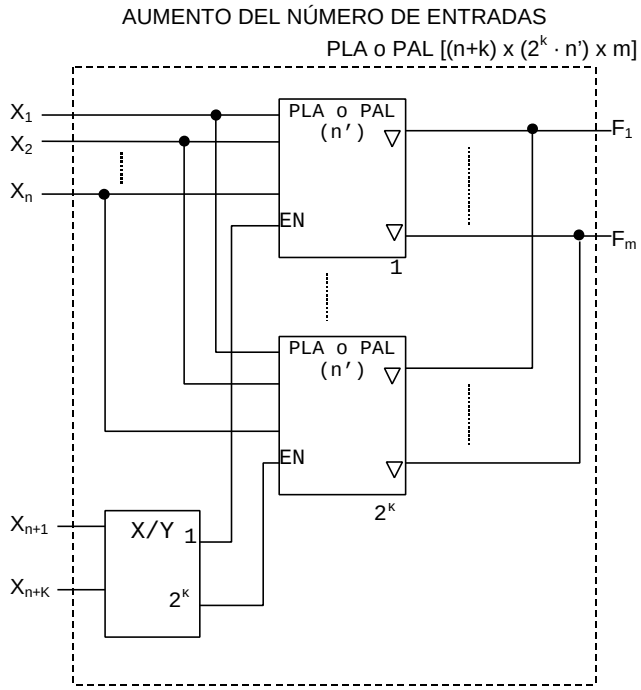


Si disponemos de dispositivos con salida triestado es posible partir la implementación con una entrada como decisión en dos PALs o PLAs de forma que solo esté activa la correspondiente a la entrada que se usa como decisión. Este método se puede extender eligiendo más variables para segmentar y decodificar estas para generar las señales de enable.

2.6.2.- Expansión de entradas.

El problema aquí es que el conjunto de funciones a implementar demanda un PLD con más pines de entrada que los disponibles. Una solución es idéntica a la última mencionada en la

sección anterior. Se retiran un conjunto de variables para alimentar a un decodificador que genera señales enable para PLDs en paralelo.

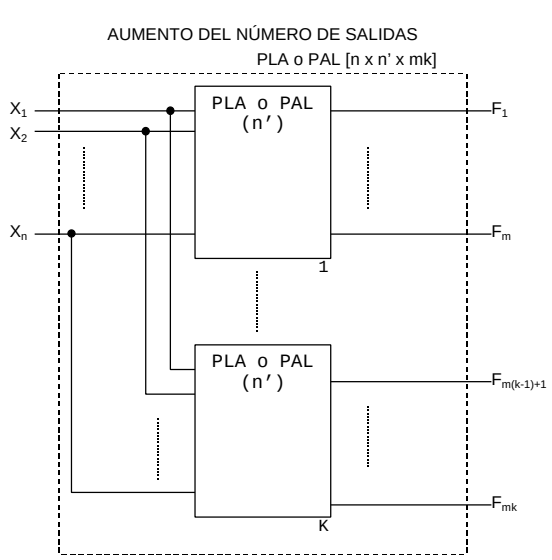


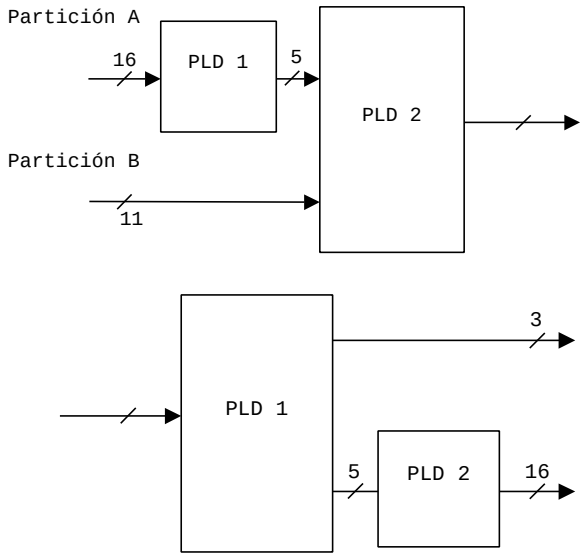
Un segundo método de reducir el número de entradas necesarias es codificación de entrada. Un subconjunto de variables de entrada se codifican en un bloque para reducir su tamaño. Esto es posible si solo se necesitan menos del número máximo de combinaciones de estas variables para definir la función a realizar.

2.6.3.- Expansión de salidas.

El problema a resolver aquí es el de implementar un conjunto de funciones mayor que el número de salidas proporcionadas por el dispositivo disponible. Este problema se soluciona de forma simple conectando en paralelo el número necesario de dispositivos. Esto es equivalente a un particionado arbitrario de las columnas de salida de la tabla de verdad.

Otra técnica que puede ser económica, si el conjunto de funciones lo permite, es la codificación de salida. Si podemos constatar de la tabla de verdad que las salidas se repiten formando un patrón, este se puede codificar en unos pocos bits generados por un dispositivo, los cuales luego se expanden mediante otro dispositivo.





La codificación de entrada y salida también puede usarse para reducir términos producto reduciendo también algunas veces pines de entrada o salida.

2.7.- Procedimiento de diseño con dispositivos programables.

El procedimiento es el siguiente:

1. Especificar la función que el circuito deseado debe realizar.
2. Generación de las ecuaciones booleanas requeridas para implementar esa función.
3. Simplificación de las ecuaciones booleanas.
4. Generación de un mapa de fusibles desde las ecuaciones booleanas.
5. Simulación lógica (opcional).
6. Programación del dispositivo seleccionado.
7. Chequeo o test del dispositivo ya programado con el programador.

Ejemplo: Funciones a realizar en una pal tipo 18P8 :

$$F1 = \overline{ab} + c\bar{d}$$

$$F2 = a + e$$

$$F3(d,c,b,a) = \sum_{4}(4,5,6,7,9,11)$$

$$F4(d,c,b,a) = \begin{cases} 1 & \text{Si } 7 \geq m_i \geq 11 \\ 0 & \text{En caso contrario} \end{cases}$$

Fichero fuente en formato PROTEL/CUPL:

```
Name         PLDDesign             ;
Partno       18P8                  ;
Revision     1                     ;
Date         27/10/99              ;
Designer     Manuel Sanchez        ;
Company      Universidad de Huelva ;
Assembly                           ;
Location     Huelva                ;
Device       p18p8;
Format                             ;

/** Entradas  **/
Pin 1  =  a ;
Pin 2  =  b ;
Pin 3  =  c ;
Pin 4  =  d ;
Pin 5  =  e ;

/** Salidas **/
Pin 12  =  F1;
Pin 13  =  F2;
Pin 14  =  F3;
Pin 15  =  F4;

/** Declaraciones y Variables Intermedias **/
```

```
/** Ecuaciones Logicas **/

F1 = !(a & b) # (c & !d);   /* suma de productos */
F2 = a $ e;                 /* operador xor */

TABLE [d,c,b,a] => F3 {
     [4..7] => 1;
      9 => 1;
      B => 1;
     [0..3] => 0;
     8 => 0;
     A => 0;
     [C..F] => 0;
}

F4 = [d,c,b,a]:[7..B];
```

Listado de salida que muestra las ecuaciones simplificadas por el compilador:

```
*****************************************************************************
                                    PLDDesign
*****************************************************************************
ADVANCED PLD    4.0 Serial# MW-67999999
Device          p18p8  Library DLIB-h-36-3
Created         mar nov 02 08.44.00 1999
Name            PLDDesign
Partno          18P8
Revision        1
Date            27/10/99
Designer        Manuel Sanchez
Company         Universidad de Huelva
Assembly
Location        Huelva

=============================================================================
                            Expanded Product Terms
=============================================================================

F1 =>
    !a
  # !b
  # c & !d

F2 =>
    a & !e
  # !a & e

F3 =>
    c & !d
  # a & !c & d

F4 =>
    a & b & c & !d
  # !c & d

F1.oe  =>
    1

F2.oe  =>
```

```
    1

F3.oe  =>
    1

F4.oe  =>
    1

================================================================================
                                  Symbol Table
================================================================================

Pin Variable                                          Pterms   Max     Min
Pol   Name              Ext     Pin     Type          Used   Pterms  Level
--- --------            ---     ---     ----          ------ ------  -----

    F1                          12       V             3       8       1
    F2                          13       V             2       8       1
    F3                          14       V             2       8       1
    F4                          15       V             2       8       1
    a                           1        V             -       -       -
    b                           2        V             -       -       -
    c                           3        V             -       -       -
    d                           4        V             -       -       -
    e                           5        V             -       -       -
    F1                  oe      12       D             1       1       0
    F2                  oe      13       D             1       1       0
    F3                  oe      14       D             1       1       0
    F4                  oe      15       D             1       1       0


LEGEND    D : default variable         F : field      G : group
          I : intermediate variable  N : node       M : extended node
          U : undefined        V : variable   X : extended variable
          T : function
================================================================================
                                    Fuse Plot
================================================================================

Pin #19  02592  Pol x
 00000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00036 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00072 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00108 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00144 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00180 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00252 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00288 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18  02593  Pol x
 00324 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00360 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00396 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00432 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00468 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00540 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00612 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17  02594  Pol x
 00648 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
 00684 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00720 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00756 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00792 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00828 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00900 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00936 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16  02595  Pol x
 00972 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01008 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01044 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01080 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01116 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01188 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01260 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #15  02596  Pol -
 01296 ------------------------------------
 01332 x-x-x----x-------------------------
 01368 -----x--x--------------------------
 01404 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01476 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01512 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01548 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01584 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14  02597  Pol -
 01620 ------------------------------------
 01656 ----x----x-------------------------
 01692 --x--x--x--------------------------
 01728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01764 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01836 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01872 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01908 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13  02598  Pol -
 01944 ------------------------------------
 01980 --x----------x---------------------
 02016 ---x--------x----------------------
 02052 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02124 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02196 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02232 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12  02599  Pol -
 02268 ------------------------------------
 02304 ---x--------------------------------
 02340 -x---------------------------------
 02376 ----x----x-------------------------
 02412 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02484 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02520 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02556 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

LEGEND    X : fuse not blown
          - : fuse blown
```

```
================================================================================
                                Chip Diagram
================================================================================

                              _______________
                             |  PLDDesign    |
                        a x--|1            20|--x Vcc
                        b x--|2            19|--x
                        c x--|3            18|--x
                        d x--|4            17|--x
                        e x--|5            16|--x
                          x--|6            15|--x F4
                          x--|7            14|--x F3
                          x--|8            13|--x F2
                          x--|9            12|--x F1
                      GND x--|10           11|--x
                             |________________|
```

El fichero JEDEC de salida será el siguiente:

```
ADVANCED PLD    4.0  Serial# MW-67999999
Device          p18p8  Library DLIB-h-36-3
Created         mar nov 02 09.13.57 1999
Name            PLDDesign
Partno          18P8
Revision        1
Date            27/10/99
Designer        Manuel Sanchez
Company         Universidad de Huelva
Assembly
Location        Huelva
*QP20
*QF2600
*G0
*F0
*L01280 00000000000000001111111111111111
*L01312 11111111111111111111010101111011
*L01344 11111111111111111111111111111011
*L01376 01111111111111111111111111110000
*L01600 00000000000000000000111111111111
*L01632 11111111111111111111111111110111
*L01664 10111111111111111111111111111101
*L01696 10110111111111111111111111111111
*L01920 00000000000000000000000011111111
*L01952 11111111111111111111111111111101
*L01984 11111111101111111111111111111111
*L02016 11101111111101111111111111111111
*L02048 11110000000000000000000000000000
*L02240 00000000000000000000000000001111
*L02272 11111111111111111111111111111111
*L02304 11101111111111111111111111111111
*L02336 11111011111111111111111111111111
*L02368 11111111111101111101111111111111
*L02400 11111111111100000000000000000000
*L02592 00001111
*C393B
* FC7E
```

3.- ARITMÉTICA BINARIA.

Un elemento aritmético es cualquier circuito que pueda sumar, restar, multiplicar, dividir o realizar alguna otra función aritmética con números binarios.

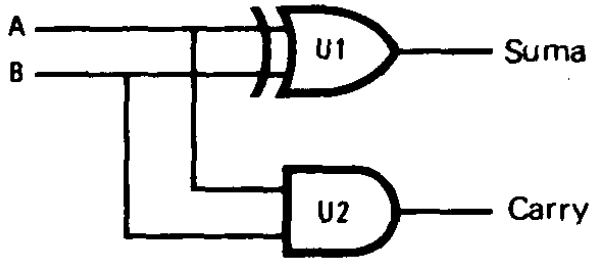
3.1.- Circuitos sumadores.

Un circuito para sumar dos dígitos de un bit sin tener en cuenta el acarreo (carry) de otras etapas constituye un semisumador o Half Adder (H.A.).

b	a	s (suma)	c (acarreo)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S ? a ? b

C ? a·b



3.1.1.- Implementación de un Sumador completo (Full-Adder) binario.

El semisumador debe modificarse más para que resulte útil en la mayoría de las aplicaciones. En general, los números constan de más de un bit, por tanto, se precisa un circuito sumador independiente para sumar cada par de bits y poderse realizar la suma en paralelo. Sin embargo, cuando se suma en paralelo, la posición de cada bit afecta a la suma del bit de la izquierda.

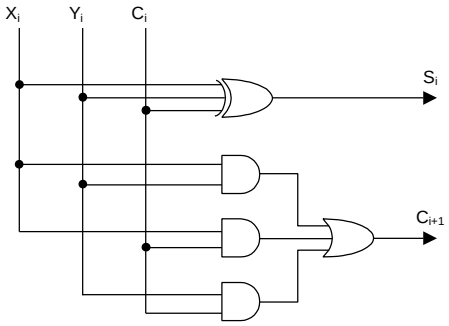
La tabla de valores de un F.A. (full-adder) de cuatro entradas x_i, y_i, c_i y salidas c_{i+1} y s_i es:

x_i	y_i	c_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

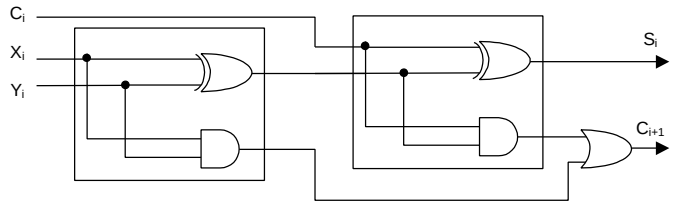
Así: $S_i = x_i \oplus y_i \oplus c_i$

$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$

la implementación sería:



También es posible una implementación con semisumadores (half-adder) :

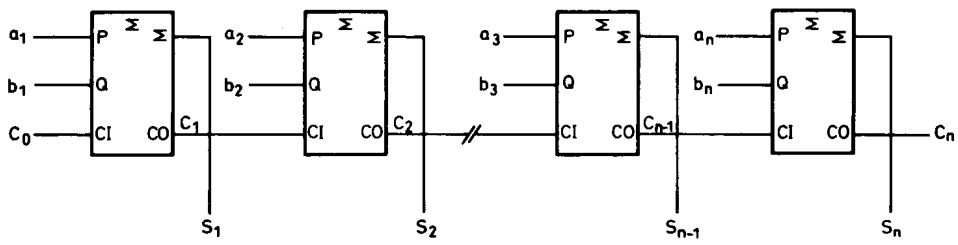


$$S_i = (x_i \oplus y_i) \oplus c_i$$
$$C_{i+1} = x_i y_i + (x_i \oplus y_i) c_i =$$
$$= x_i y_i + \bar{x}_i y_i c_i + \bar{y}_i x_i c_i$$
$$= y_i (x_i + \bar{x}_i c_i) + \bar{y}_i x_i c_i$$
$$= x_i y_i + y_i c_i + \bar{y}_i x_i c_i$$
$$= x_i y_i + y_i c_i + x_i c_i$$

Si bien el sumador completo puede construirse de muchas formas diferentes, siempre realiza la misma función: suma 2 bits y el arrastre previo y genera el resultado de la suma y el acarreo si se ha producido. Este circuito se emplea en calculadoras, ordenadores y muchos otros circuitos aritméticos, no solamente para hacer sumas, sino también multiplicación, división y resta. Todas estas operaciones se pueden realizar con el sumador completo, empleando algoritmos especiales para cada caso.

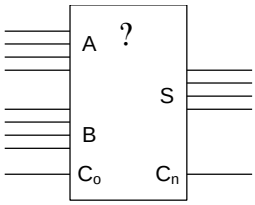
3.1.2.- Sumadores binarios de n-bits.

Para realizar sumadores de mayor número de bits, se pasan los acarreos de cada sumador formando un sumador de Rizado (Ripple-Adder) o sumador en paralelo. El sumador paralelo consta de varios pasos de circuitos de sumador completos que están conectados entre sí, de forma que la salida de acarreo de un paso es la entrada de acarreo del paso siguiente. Por lo tanto, un sumador paralelo de 4 pasos realizará la propagación de todos los acarreos y puede usarse para sumar dos números cualesquiera de cuatro bit.



Los acarreos pasan en secuencia a través de todos los pasos del sumador paralelo y la última salida de la suma no es correcta hasta que se genere el último acarreo. Este sumador se denomina sumador en paralelo con arrastre en serie. Cada uno de los pasos del sumador paralelo realiza una suma tan pronto como están presentes las entradas A y B, pero las salidas

de los pasos no son correctas hasta tanto no se procesen todos los acarreos. Por consiguiente, se genera una salida de suma provisional incorrecta, lo cual, en general, no lo permitirán todos los circuitos del sistema. Un sumador paralelo con arrastre en serie requiere un tiempo relativamente largo para que todos los acarreos sean generados y procesados. Se considera este tipo de sumadores como lentos, si bien son suficientes para la mayoría de las aplicaciones. Si el tiempo de la suma de cada elemento es 30 ns, el tiempo máximo de suma de los cuatro bits será 4 x 30 ns = 120 ns.

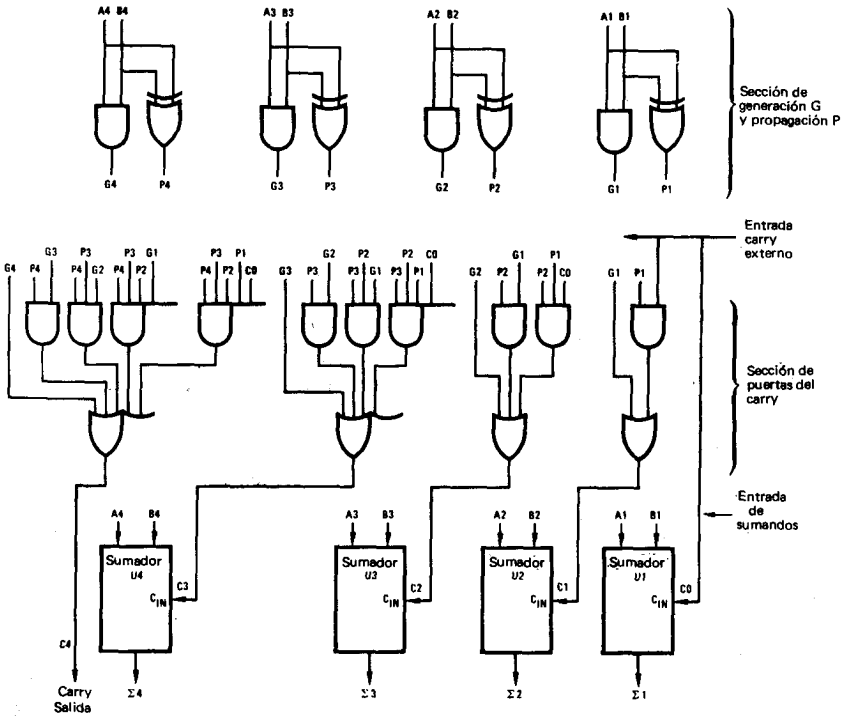


3.1.3.- Sumadores de acarreo anticipado.

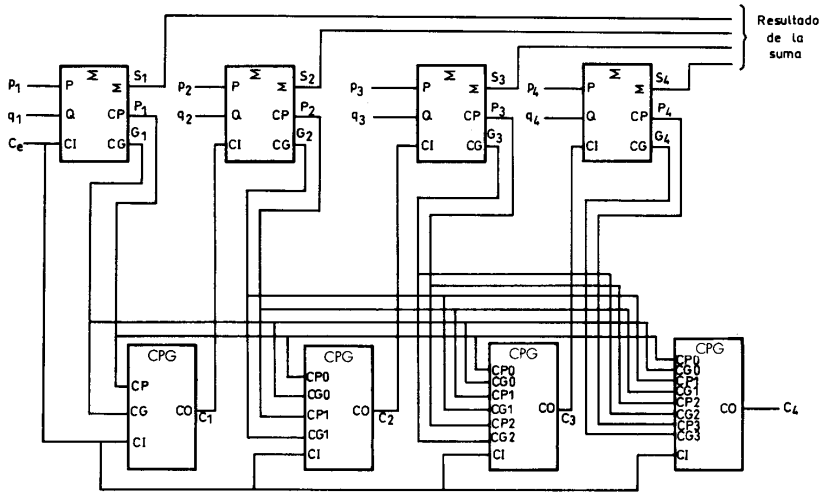
El sumador de arrastre anticipado es un sumador paralelo que no tiene que esperar a que se originen los acarreos para pasar de un paso sumador completo al siguiente. En realidad, puede anticipar todas las señales de acarreo andes de que se generen, utilizando cierta lógica auxiliar, la cual proporciona simultáneamente las entradas de acarreo a todos los pasos del sumador, antes de que sean desarrollados por cada etapa independientemente y puede también anticipar cuál será la salida de arrastre del último sumador completo. Estos sumadores, son más rápidos que los sumadores paralelos de arrastre en serie y se emplean en los ordenadores rápidos.

Denominamos acarreo generador G_1 y lo representamos como: $G_1 = A_1 \cdot B_1$, para expresar la segunda condición, definimos P_1, acarreo propagador, como : $P_1 \; ? \; \overline{A_1}B_1 \; ? \; A_1\overline{B_1} \; ? \; A_1 \; ? \; B_1$. Con estas definiciones, la ecuación lógica completa para el primer acarreo puede escribirse de la siguiente forma: $C_1 = G_1 + P_1 C_0$. En donde cada uno de los términos representa una de las dos condiciones, y los subíndices 1 se refieren al primer paso sumador. Puede escribirse una ecuación similar para el segundo paso del sumador paralelo: $C_2 = G_2 + P_2 C_1$.

Sustituimos el término C_1 por su valor de la ecuación anterior: $C_2 = G_2 + P_2 (G_1 + P_1C_0) = G_2 + P_2G_1 + P_2P_1C_0$. La expresión de C_2 se representa en términos de G y P solamente, ahora C_2 depende únicamente de las señales de entrada A y B hasta los dos primeros pasos sumadores. Esto significa que, con el fin de desarrollar el arrastre de C_2, sólo es necesario conocer el estado de A_1, A_2, B_1 y B_2. La salida C_1 del primer paso sumador ya no es necesaria. De forma similar, la ecuación de C_3 se puede escribir únicamente con los términos G y P: $C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0$. Lo mismo se puede hacer con C_4.



Las entradas de arrastre para todas las etapas de un sumador pueden desarrollarse tan pronto como se reciben las señales de entrada A y B. No es necesario con este procedimiento esperar a que se propaguen los arrastres por todas las etapas.



El sumador de arrastre anticipado tiene ventajas e inconvenientes. Es mucho más rápido que un sumador de arrastre en serie. Si suponemos que la lógica de anticipación del arrastre consume un tiempo de 30 ns y que cada paso sumador realiza la suma en 30 ns, el tiempo total será de 30 + 30 = 60 ns, o sea, la mitad de tiempo que precisa un sumador de arrastre en serie. Para sumador de 5 o más bits, la diferencia se hace cada vez mayor.

La limitación más importante del sumador de arrastre anticipado es que a medida que aumenta el número de etapas, la ecuación precedente y cada uno de sus términos se hacen demasiado largos. Cada vez se precisan más puertas para llevar a cabo la ecuación, y dichas puertas necesitan cada vez mayor número de líneas de entrada (mayor fan-in). Como

consecuencia de todo esto, un sumador de arrastre anticipado debe promediar la rapidez y la complejidad.

Hay varias formas de hacer sumadores de 8 bits. Una de las posibilidades es utilizar dos sumadores de arrastre anticipado de 4 bit con un arrastre en serie entre éstos; otra consiste en utilizar dos sumadores de arrastre en serie de 4 bit, pero desarrollar únicamente el arrastre C_4 mediante la lógica de anticipación de arrastre y permitir que ambas etapas de 4 bit realicen las adiciones simultáneamente, en lugar de hacerlo en secuencia.

3.2.- Resta binaria.

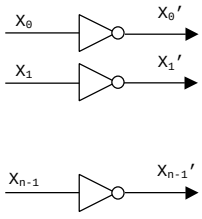
La resta puede efectuarse de varias formas distintas. En primer lugar, para hacer una resta de un número binario a otro, pueden escribirse sus reglas, que son similares a las de la suma:

- 0 – 0 = 0
- 0 – 1 = 1 y 'llevamos' 1
- 1 – 0 = 1
- 1 – 1 = 0

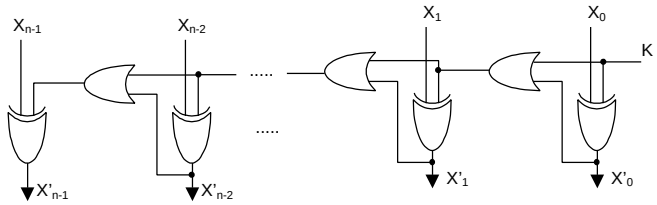
Estos algoritmos permiten la resta de dos números binarios cualesquiera, siempre que el minuendo sea mayor que el sustraendo. No obstante existe otro método de resta más utilizado, que se basa en sumar el complemento de un número a otro, en lugar de restar los números directamente.

Al igual que en el sistema de números decimales existen complementos a 9 y a 10, en el sistema de números binarios existen complementos a 1 y a 2. Por ejemplo, en el sistema decimal, el complemento a 9 del número 5 es 4 (porque 5 + 4 = 9) y el complemento a 10 de 5 es 5 (porque 5 + 5 = 10). En el sistema binario el complemento a 1 del número 0 es 1 (porque 0 + 1 = 1) y el complemento a 1 de 0100 es 1011, porque 0100 + 1011 = 1111.

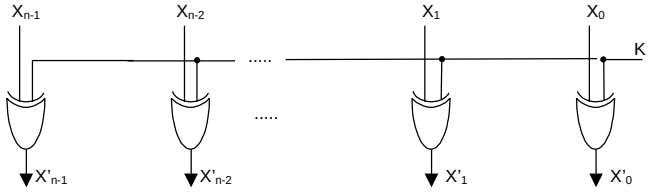
Para hallar el complemento a 1 de cualquier número binario, todos sus 1 se cambian en 0 y viceversa. Para hallar el complemento a 2 en el sistema binario, primero se halla el complemento a 1 y luego al resultado se le suma 1. Un circuito para obtener el complemento a 1 puede ser:



El siguiente circuito calcula el complemento a uno si k=1 y el complemento a dos si k=0:



Este otro circuito calcula el complemento a uno si k=1, y efectúa la transferencia sin modificación si k=0:



3.3.- Representación de números negativos.

Debido a que la resta puede realizarse mediante la suma de un complemento, en la lógica típica de ordenadores la diferencia entre la suma y la resta pierde su clara distinción. Hasta ahora, todos los números tenían que ser positivos y en la resta el minuendo tenía que ser mayor que el sustraendo. Si se permite el uso de números negativos debemos escoger alguna representación para ellos.

Para distinguir entre números positivos y negativos, generalmente se usa un símbolo simple: si es un número positivo el signo + se representa con 0 y si es negativo con 1 en el dígito más a la izquierda del número.

Complemento a uno. Los números binarios positivos se almacenan en código binario, precediendo un 0 al número que indica el signo positivo. Los números negativos se almacenan en forma de complemento a 1, precedidos por 1, que representa el signo negativo. La suma de dos números tiene lugar directamente, con independencia del signo de los números. El acarreo, cuando se genera, debe añadirse a la posición del bit MSB del resultado de la suma. La resta de dos números requiere que el sustraendo se complemente antes, con independencia de su signo.

- Si A>B, la resta es positiva, se produce un acarreo en el bit más significativo que sumamos para obtener el resultado.
- Si A ? B, la resta es negativa, no se produce acarreo, el resultado se obtiene complementando la suma obtenida.

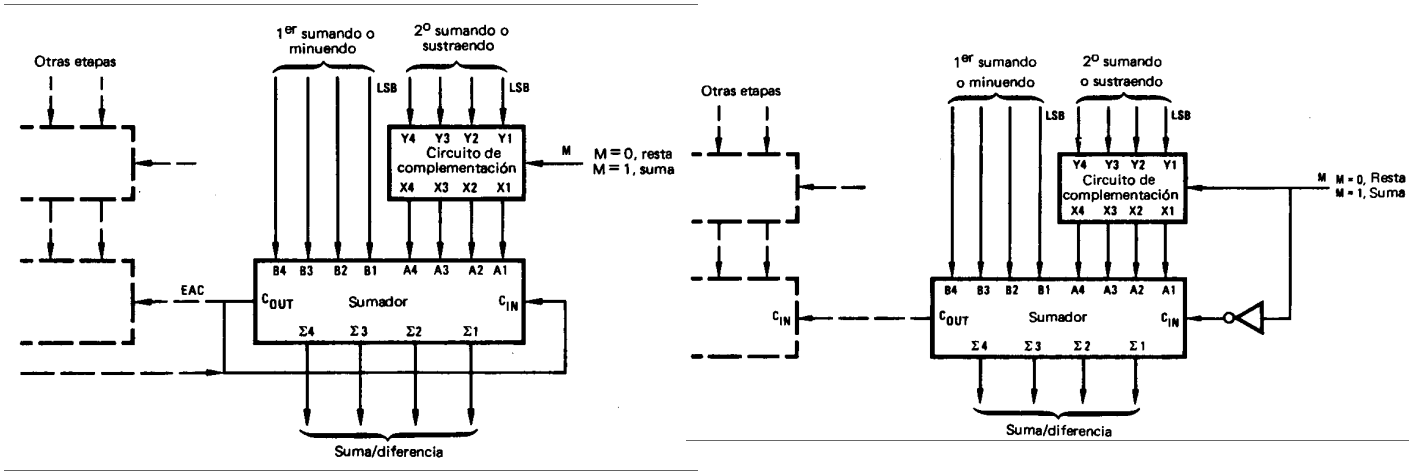
Complemento a dos. Los números binarios positivos se almacenan en la memoria del ordenador en código binario, con un 0 en la posición del signo. Los números negativos se almacenan en forma de complemento a dos, indicando el signo negativo, un bit 1 a la izquierda. La suma tiene lugar directamente y la resta requiere complementar el sustraendo antes de que tenga lugar la suma. Las operaciones con complemento a dos nunca generan acarreo.

- Si A ? B, la resta es positiva o nula, despreciamos el acarreo en el bit más significativo.
- Si A < B, la resta es negativa. La resta se obtiene del complemento a dos del resultado de la suma.

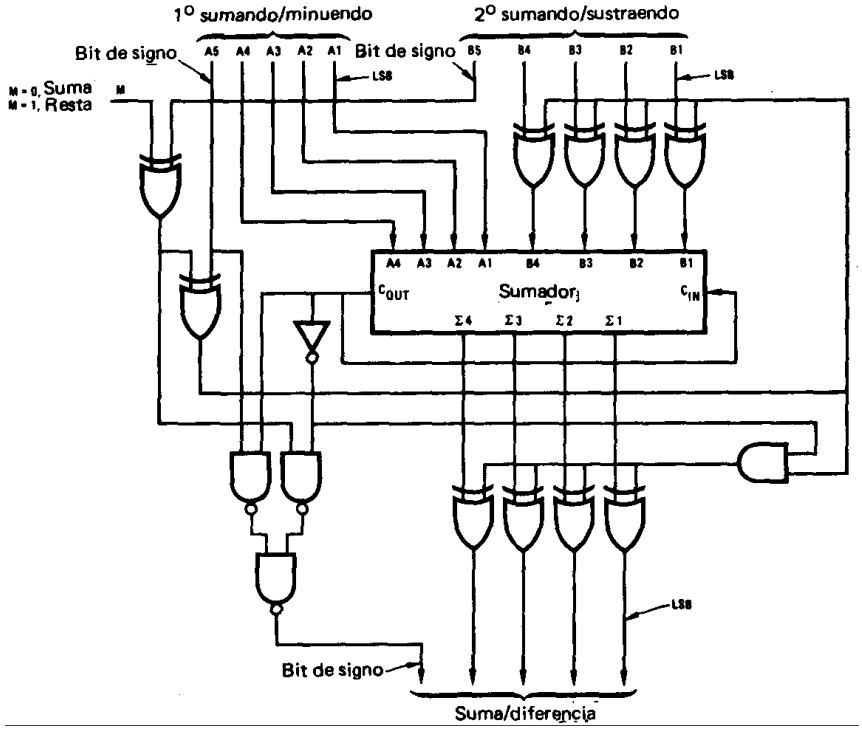
Signo y magnitud. Los números, tanto positivos como negativos, se almacenan en la memoria del ordenador en forma no complementada. La única diferencia entre números negativos y positivos es el bit de signo. Contrariamente a lo que ocurre con los dos métodos anteriores de los complementos, las operaciones aritméticas de suma y resta, en el formato de magnitud y signo, no son iguales para todas las combinaciones de números. Para saber cuál de los dos números debe complementarse en una operación de suma o resta, y para conocer el

signo de la suma o diferencia, es necesario saber los signos de los números que intervienen, así como cuál de los números es el mayor.

- Si A positivo y A>B o bien A negativo y B>A, se suma el sustraendo en complemento a 1 y luego se suma el acarreo generado al resultado.
- Si A positivo y A<B o bien A negativo y B<A, la resta es negativa, se suma el sustraendo en complemento a 1 y luego se complementa el resultado y se añade el bit de signo.



Circuito típico de sumador-restador de complemento a uno y de complemento a dos.



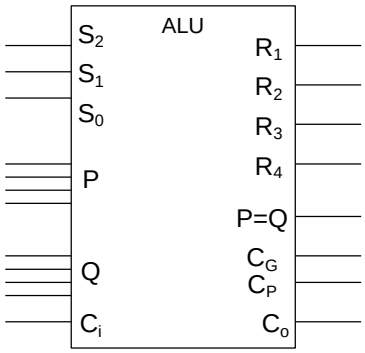
Circuito típico de un sumador-restador de signo y magnitud.

3.4.- Unidades aritmético – lógicas.

La ALU es un elemento polivalente capaz de realizar operaciones lógicas y aritméticas. Para la ejecución de las operaciones de tipo aritmético y de tipo lógico, se puede emplear la ALU de 4 bit que responde a la siguiente tabla de verdad:

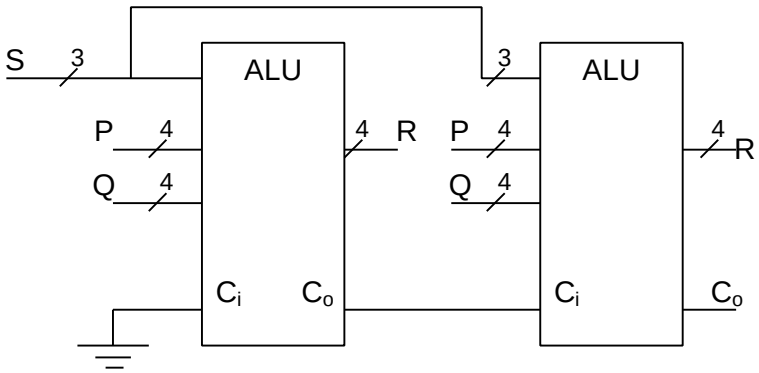
S: selección de operación.

S_1	S_0	Op. Lógicas (S_2=0)	Op. Aritmética (S_2=1)
0	0	P and Q	P mas Q
0	1	P or Q	P menos Q
1	0	P nand Q	P mas 1
1	1	P or excl Q	P menos 1



Las entradas S_2, S_1 y S_0 seleccionan el tipo de operación, a realizar con los dos operandos de 4 bits P y Q. C_i es la entrada de acarreo y C_0 la salida. Las unidades aritmético-lógicas suelen incorporar una salida que detecta cuando los dos operandos son iguales.

Para aumentar el número de bits de procesamiento de la ALU se pueden conectar en serie propagando el acarreo.



El rebosamiento (overflow) se produce cuando no hay suficientes bits para representar el número obtenido tras una suma o resta. Por ejemplo, si con una representación signo-magnitud de 8 bits, sumo 128 obtengo erróneamente un número negativo. En realidad se ha producido un rebosamiento porque el resultado no se puede representar con 8 bits, necesita un bit más.

Se puede realizar un circuito detector de rebosamiento comprobando los signos de los operandos y del resultado:

$$\text{OVERFLOW} ? \; BSP \; BSQ \; \overline{BSR} \; ? \; \overline{BSP} \; \overline{BSQ} \; BSR$$

3.5.- Multiplicación binaria.

La multiplicación consiste en la repetición de una suma; es decir, 4 x 10 significa sumar cuatro veces el 10. Se puede multiplicar de esta forma mediante un sistema secuencial que sume n veces el número.

El desplazamiento hacia la izquierda de un número binario equivale a multiplicarlo por 2. En el sistema decimal un desplazamiento similar equivale a multiplicarlo por 10. La multiplicación binaria, en su forma más simple, es una serie de sumas del multiplicando, con un desplazamiento después de cada adición. Por lo tanto, puede realizarse una multiplicación con un sistema secuencial que conste de sumadores y registros de desplazamiento.

El multiplicador así obtenido tendrá un tiempo de respuesta función del número de veces que se repite la suma, y generalmente largo. Otra técnica de respuesta más rápida para construir multiplicadores consiste en implementar su tabla de verdad en ROM. Por ejemplo, un multiplicador 2x2:

b_2	b_1	a_2	a_1	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
.....							

3.6.- Representación de números fraccionarios.

La representación de números fraccionarios usando dígitos binarios se puede realizar de dos formas:

- Manteniendo la coma que separa la parte fraccionaria de la entera fija. **Coma Fija**.
- Manteniendo el número normalizado y por tanto, la **coma flotante** puede variar de posición. Por esto el número dispondrá además de un exponente para determinar la posición de la coma, similar a la notación exponencial de base diez, pero ahora en base dos.

El proceso de normalización consiste en correr la coma hasta la izquierda y actualizar el exponente hasta obtener la mantisa en la forma O.XXX = 0.M. El rango de valores disponible en la notación empleada es la cantidad de números representables desde el mínimo al máximo. La precisión es función del número de dígitos significativos.

Básicamente un número normalizado en coma flotante tiene la siguiente forma:

S	Exponente	Mantisa

La mantisa dispone generalmente de bit de signo S, un 0 indica número positivo, un 1 lógico indica número negativo. Si el exponente dispone de q bits y la mantisa de p-1 bits, el exponente viene representado en exceso a 2^{q-1}. El valor del número se extrae de la siguiente forma:

$$\text{Valor} = (-1)^{S} \cdot 0.M \cdot 2^{E-\text{Exceso}} \quad ; \quad \text{Exceso} = 2^{q-1}$$

El número máximo y mínimo representable será:

Mínimo = 0.10..0 = $1 \cdot 2^{-1} = 0.5$

Máximo = 0.11..1 = $1 \cdot 2^{-1} + 1 \cdot 2^{-2} + \ldots + 1 \cdot 2^{-(p-1)} = 1$

Norma Ansi/IEEE 754

Las características de la representación de número en coma flotante de la norma son las siguientes:

- Emplea mantisa fraccionaria normalizada, en Signo/Magnitud y bit implícito entero. Puesto que todas las mantisas fraccionarias en S/M son de la forma 0.1XXX, se puede considerar el 1 como implícito, ganando un bit de precisión y además se desplaza el punto decimal hasta la derecha una posición, quedando de la forma 1.XXX.
- Emplea exponente en exceso 2q-1 – 1 = 28-1 –1 = 127 para simple precisión.
- Por lo tanto el valor normalizado que se representa es:

$$(-1)^{S} \cdot 1.M \cdot 2^{E-127}$$

- Existen cuatro casos especiales que facilitan el manejo de números en coma flotante, que son:

  a) Si el exponente E=255 y M <> 0, el resultado no tiene sentido. Se produce esta situación, por ejemplo, en una operación 0/0 -> Nan.
  b) Si el exponente E=255 y M = 0, el resultado es infinito, indicándose con el bit S si es positivo o negativo.
  c) Si el exponente E=0 y M=0, el resultado es cero.
  d) Si el exponente E=0 y M <> 0, se esta representando datos próximos a cero, en un formato no normalizado:

$$(-1)^{S} \cdot 0.M \cdot 2^{E-126}$$

El estándar dispone de tres niveles de precisión:

	SINGLE	DOUBLE	QUADRUPLE
S : SIGNO	1	1	1
E: EXPONENTE	8	11	16
F: FRACTION	23	52	111
ANCHO TOTAL	32	64	128
BIT SIGNO	0+ 1-	0+ 1-	0+ 1-
EX. MAXIMO	126	1022	16382
EX. MINIMO	-126	-1022	-16382
EXCESO	127	1023	16383

3.7.- Circuitos comerciales que contienen elementos aritméticos.

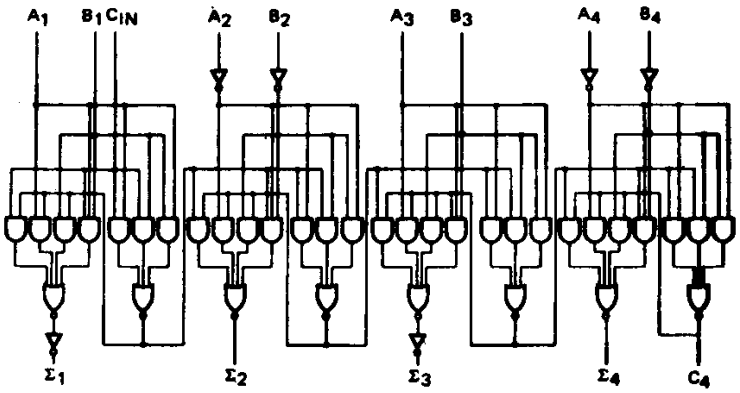
A continuación veremos con detalle componentes de tecnológica TTL clásica, entre los que destacan :

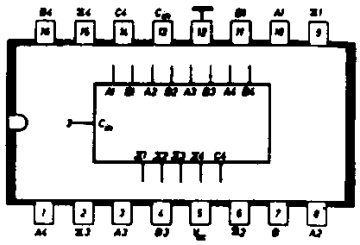
7480:	Sumador completo 1 bit.
7482:	Sumador completo 2 bit.
7483,74283:	Sumador completo 4 bit.
74181:	ALU de 4 bits.
74182:	Generador de acarreo adelantado.
74284,74285:	Multiplicador binario de 4 x 4 bit.
74167:	Multiplicador decimal discreto.

Sumador completo binario de 4 bits

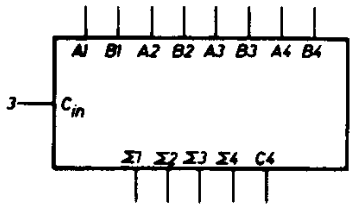
1 - SN 7483 AN	2 - F 7483 PC	3 - F 9383 PC
4 - MC 7483 P	5 - DM 7483 N	6 - DM 8283 N
7 - ZN 7483 AE	8 - N 7483 B	9 -
10 - FJH 211	11 - FLH 241 (7483A)	12 - MIC 7483 N
13 - TL 7483 N	14 - SF C 483 E	15 - SW 7483 N

diagrama lógico





símbolo lógico



DESCRIPCION.–Este dispositivo es un Sumador Completo que ejecuta la adición de dos números binarios de cuatro bits. Hay salidas de sumas (Σ),por cada bit y el acarreo resultante (C4) se obtiene del cuarto bit. Está diseñado para velocidad media a alta y aplicaciones, con bits múltiples, de suma en paralelo/acarreo serie. El circuito utiliza TTL de alta velocidad y elevada cargabilidad de salida. La inclusión de un circuito Darlington de arrastre serie, alta velocidad y una sola inversión minimiza la necesidad de tantos circuitos "lookahead" y de arrastre en cascada.

PATILLAS		CARGA
A1, B1, A3, B3	Entradas de datos	4 U.L.
A2, B2, A4, B4	Entradas de datos	1 U.L.
C_{IN}	Entrada de acarreo	4 U.L.
Σ_1, Σ_2, Σ_3, Σ_4	Salida de suma	10 U.L.
C4	Salida de acarreo bit 4	5 U.L.

Nota: 1 Unidad de carga (U.L.) = 40 μA ALTO/1,6 mA BAJO.

CONDICIONES DE FUNCIONAMIENTO RECOMENDADAS

PARAMETRO	MIN.	TIP.	MAX.	UNIDADES
Tensión de alimentación V_{CC} (V.nota 10)	4.75	5.0	5.25	Volts
Temperatura ambiente de funcionamiento	0	25	70	°C
Cargabilidad de salida normalizada C_4			5.0	U.L.
de las salidas Σ_1, Σ_2, Σ_3 o Σ_4			10	U.L.

CARACTERISTICAS ELECTRICAS EN EL MARGEN DE TEMPERATURA DE FUNCIONAMIENTO
(si no se especifica otra cosa)

SIMBOL.	PARAMETRO	MIN.	TIP.(2)	MAX.	UNIDADES	CONDCNES.DE PRUEBA(1)
V_{IH}	Tensión de entrada ALTA	2.0			Volts	Garantizada V_{IH}
V_{IL}	Tensión de entrada BAJA			0.8	Volts	Garantizada V_{IL}
V_{OH}	Tensión de salida ALTA	2.4			Volts	V_{CC} = MIN., I_{OH} = Σ -400 μA / C -200 μA
V_{OL}	Tensión de salida BAJA			0.4	Volts	V_{CC} = MIN., I_{OL} = Σ 16 mA / C 8 mA

Unidad lógica aritmética de 4 bits

1 - SN 74181 N	2 - F 74181 PC	3 - F 9341 PC
4 - MC 74181 P	5 - DM 74181 N	6 -
7 - ZN 74181 E	8 - N 74181 N	9 -
10 -	11 - FLH 401	12 -
13 - TL 74181 N	14 - SF C 4181 E	15 - SW 74181 N

diagrama lógico

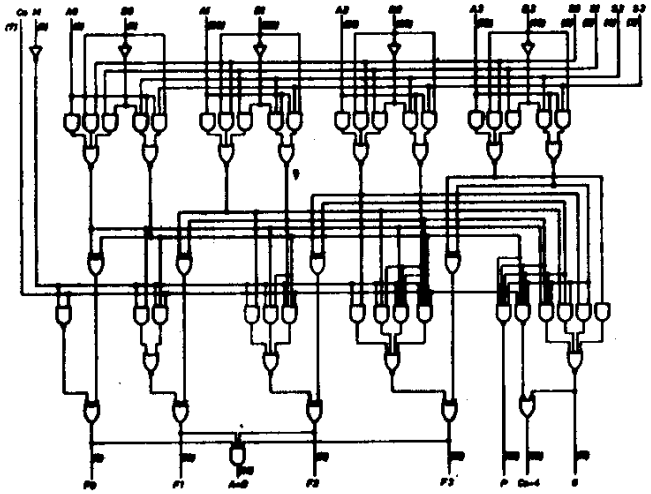
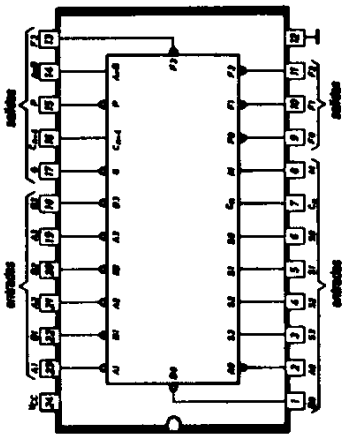
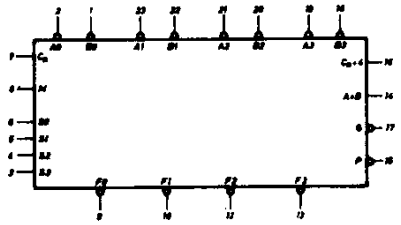


diagrama de conexión

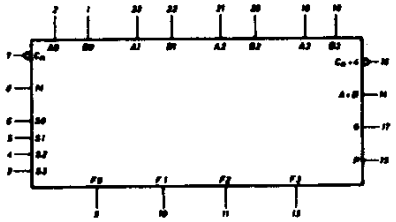


SIMBOLOS LOGICOS

OPERANDOS BAJO ACTIVO



OPERANDOS ALTO ACTIVO



DESCRIPCION. Este dispositivo es una unidad lógica aritmética (Arithmetic Logic Unit) (ALU) paralela de alta velocidad de 4 bits. Controlado por las cuatro entradas de selección de función (SO ... S3) y la entrada de control de modo (M), puede ejecutar todas las 16 posibles operaciones lógicas o 16 operaciones aritméticas diferentes con operandos de bajo activo o alto activo. La tabla de funciones relaciona estas operaciones.

Cuando la entrada de control de modo (M) está alta, todos los acarreos internos están inhibidos y el dispositivo ejecuta operaciones lógicas en cada uno de los bits individuales, tal como se indica en la lista. Cuando la entrada de control de modo está baja, los acarreos quedan habilitados y el dispositivo ejecuta operaciones aritméticas sobre las dos palabras de 4 bits. El dispositivo incorpora un acarreo «adelantado» (look-ahead) interno total y lo proporciona a cualquiera de los dos (ripple carry) acarreo «ripple» entre dispositivo que utilizan la salida $C_n + 4$, o acarreo «adelantado» entre paquetes que utiliza la señal P (carry propagate) y G (carry generate). P y G no quedan afectados por la entrada de acarreo. Cuando las exigencias de velocidad no son muy severas, el dispositivo puede utilizarse en un modo sencillo de ripple carry conectando la señal de salida de acarreo ($C_n + 4$) a la entrada de acarreo (C_n) de la siguiente unidad. Para funcionamiento en alta velocidad el dispositivo se utiliza en unión del circuito de acarreo «adelantado» 74812 o equivalente. Se requiere un paquete de acarreo «adelantado» por cada grupo de cuatro dispositivos. El acarreo adelantado puede suministrarse en varios niveles y ofrece posibilidad de alta velocidad en longitudes de palabra extremadamente grande.

La salida A = B del dispositivo se pone ALTA cuando las cuatro salidas F están altas y puede utilizarse para indicar equivalencia lógica en los 4 bits cuando la unidad está en el modo de sustración. La salida A = B es de colector abierto y puede cablearse AND con otras salidas A = B para conseguir una comparación para más de 4 bits. La señal A = B puede utilizarse con la señal de salida de acarreo para indicar A > B y A < B.

La tabla de funciones relaciona las operaciones aritméticas ejecutadas sin una entrada de acarreo. Un acarreo de entrada añade un uno a cada operación. De aquí que el código LHHL genere A menos D menos 1 (notación de complemento a 2) sin una entrada de acarreo y genere A menos B cuando se aplique acarreo. Como la resta se ejecuta realmente mediante adición complementaria (complemento a 1), una SALIDA DE ACARREO significa DEBE (BORROW); de aquí que sea generado un acarreo cuando haya exceso (overflow) y no se genere cuando no lo hay (underflow).

Como se ha indicado, el dispositivo puede utilizarse con entradas de bajo activo, produciendo salidas de bajo activo o con entradas de alto activo produciendo salidas de alto activo. Para cada caso la tabla relaciona las operaciones que deben hacer los operandos indicados dentro del símbolo lógico.