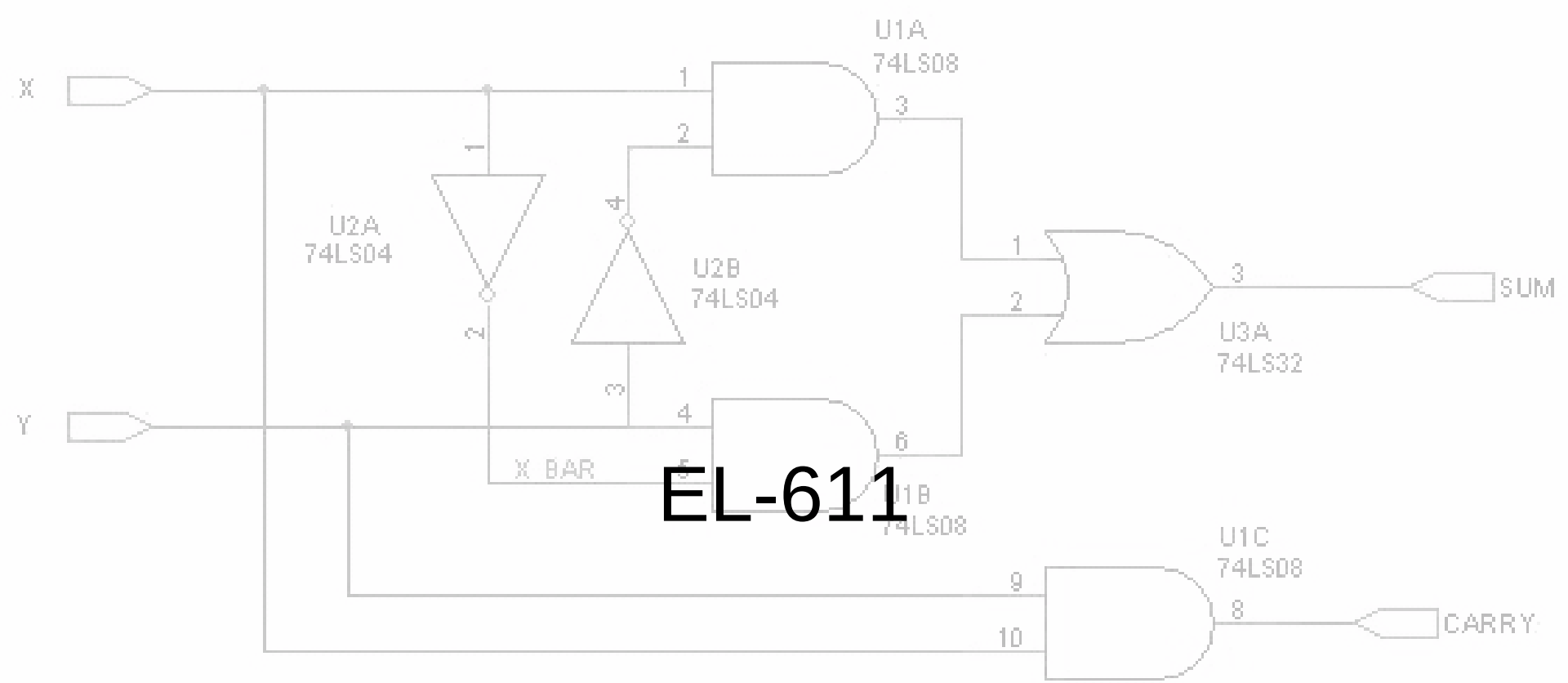
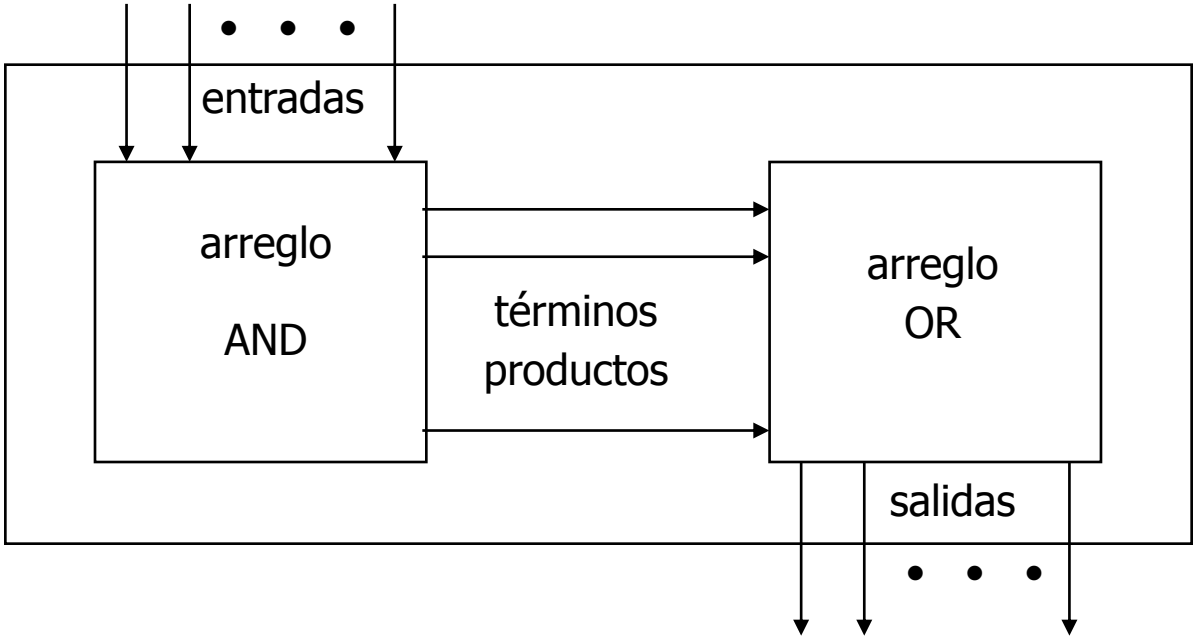


EL-611

ROM y PLA: Arreglos Lógicos Programables

Programmable Logic Arrays

- Bloques pre-fabricados de muchas compuertas AND/OR
  - en realidad NOR o NAND
  - "personalizado" haciendo/deshaciendo conexiones entre las compuertas
  - diagrama arreglos programables para la expresión suma de productos



Conceptos básicos

- Términos productos compartidos entre las salidas

ejemplo:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

matriz personalizada

término producto	entrada A	B	C	salidas F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

en la entrada:

1 = no complementado en el término
0 = complementado en el término
– = no participa

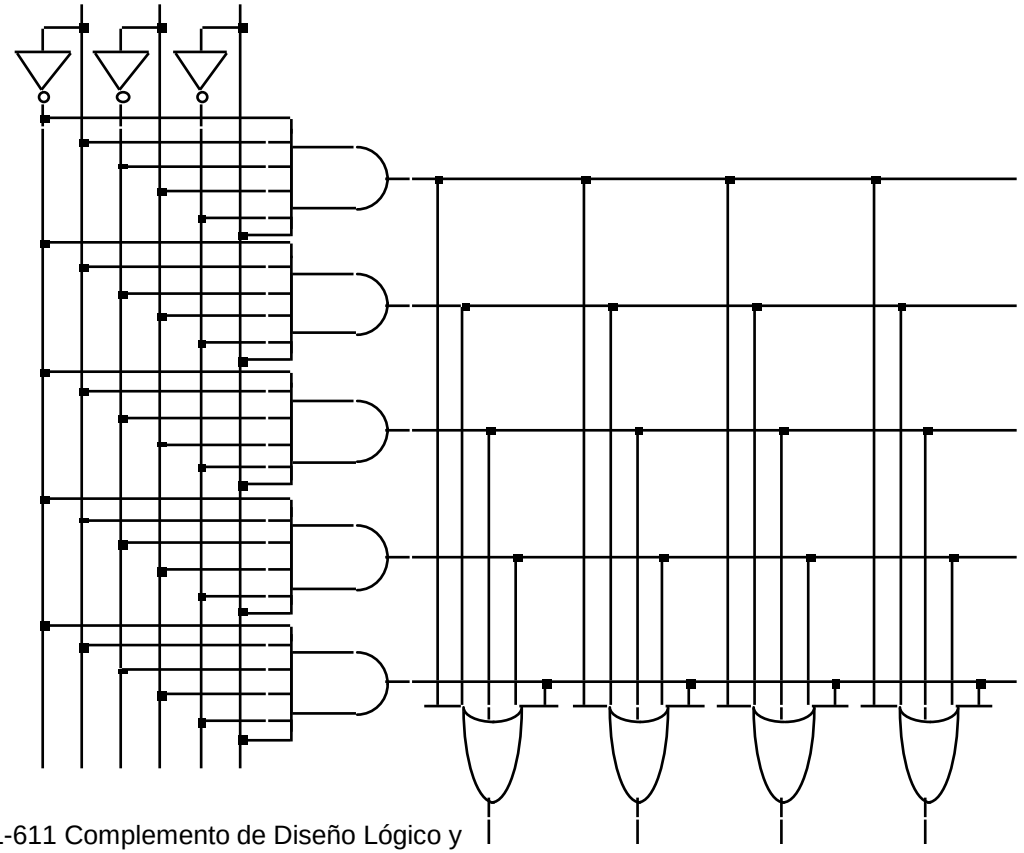
en la salida:

1 = término conectado a la salida
0 = no conectado a la salida

re-uso de los términos

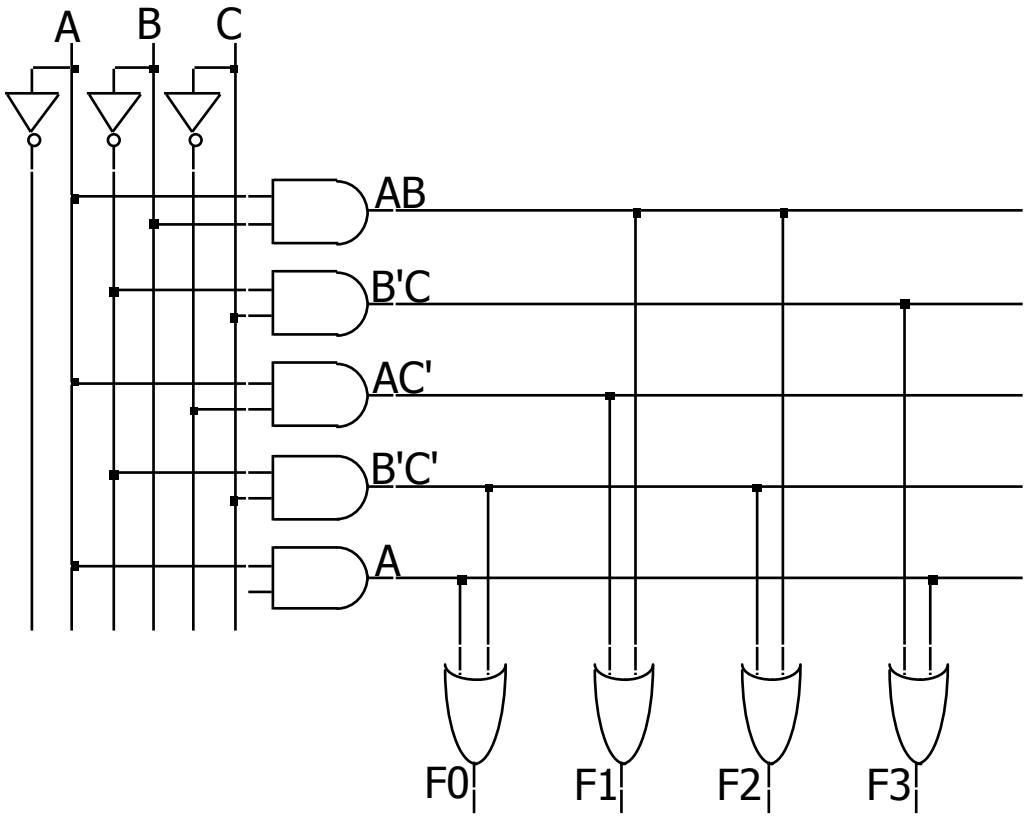
Antes de programar

- Todas las conexiones están disponibles antes de la “programación”
  - en la realidad, todas las compuertas AND y OR son NAND’s



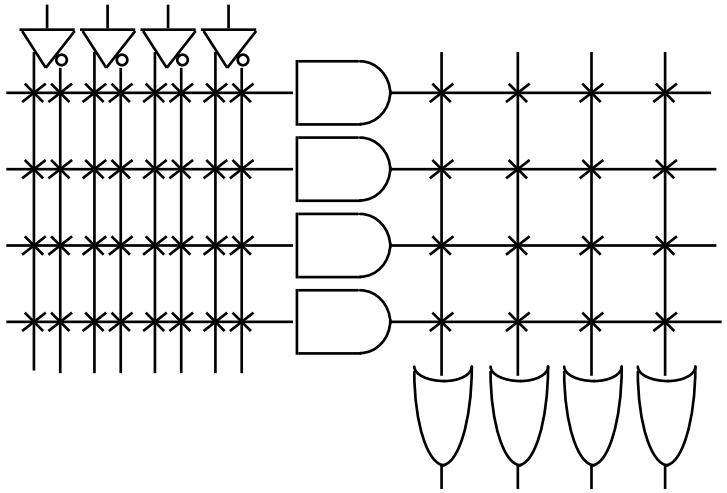
Después de la programación

- Conexiones indeseadas son “quemadas”
  - fusible (normalmente conectado, romper los no deseados)
  - anti-fusible (normalmente desconectado, hacer las conexiones deseadas)



Representación alternativa para estructuras de alto “fan-in”

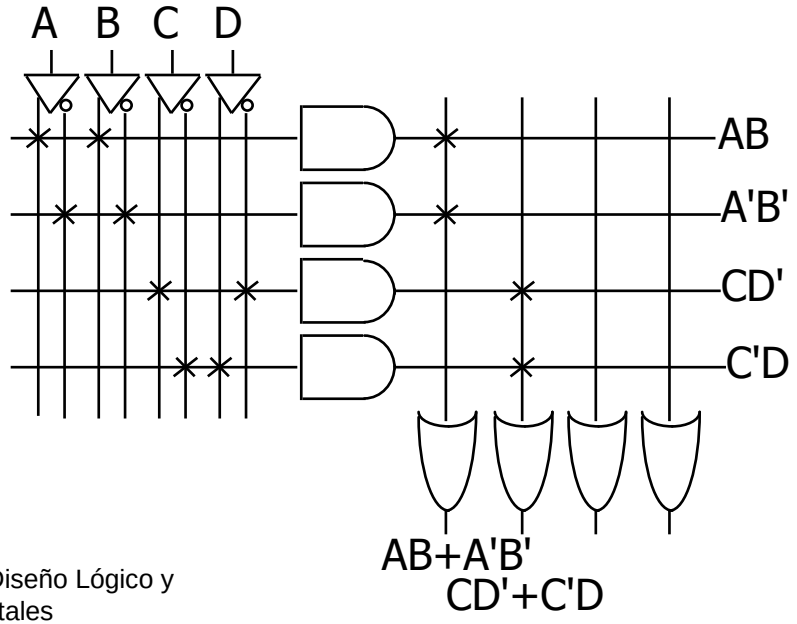
- Notación simplificada de tal manera de no dibujar todas las conexiones (“alambrado”)
  - × significa que existe una conexión y líneas perpendiculares son entradas a la compuerta



notación para la implementación

F0 = A B + A' B'

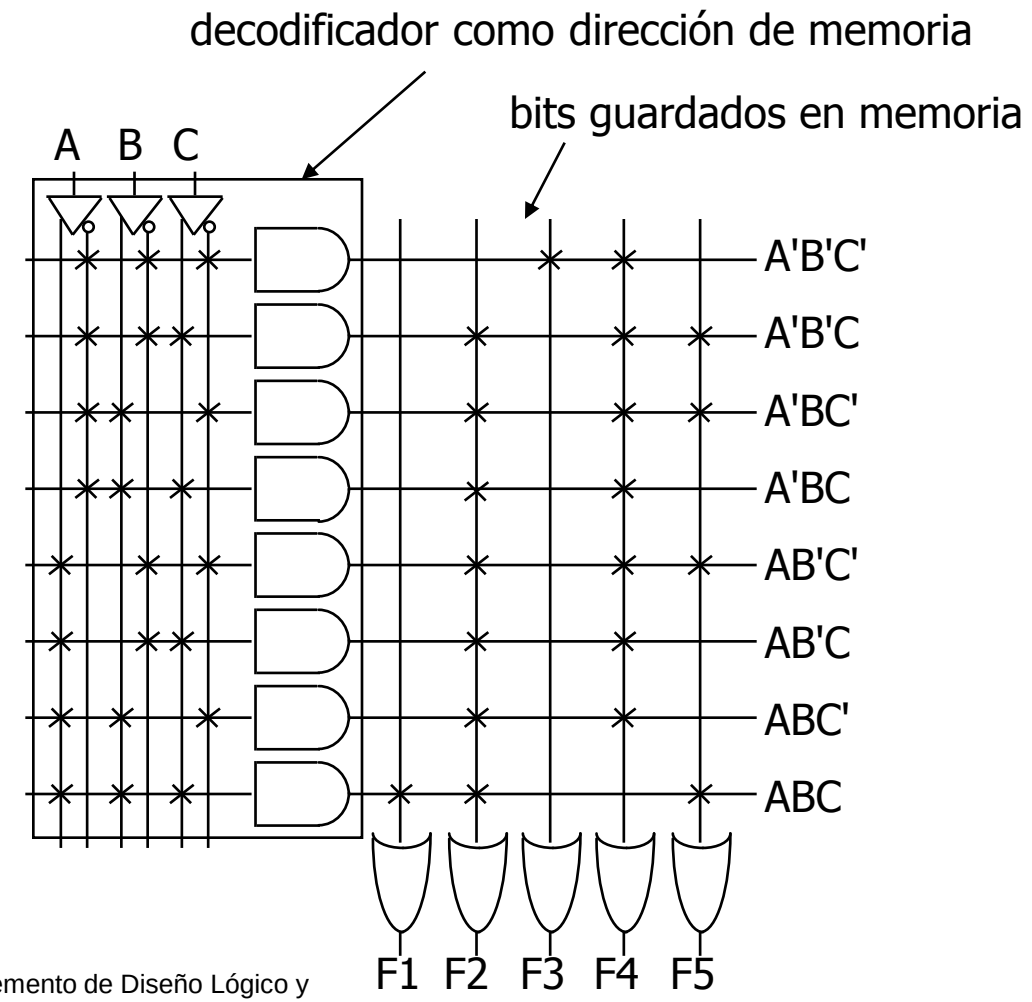
F1 = C D' + C' D



Ejemplo Programmable Logic Array

- Funciones múltiples de A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A' B' C'
  - F4 = A' + B' + C'
  - F5 = A xor B xor C
  - F6 = A xnor B xnor C

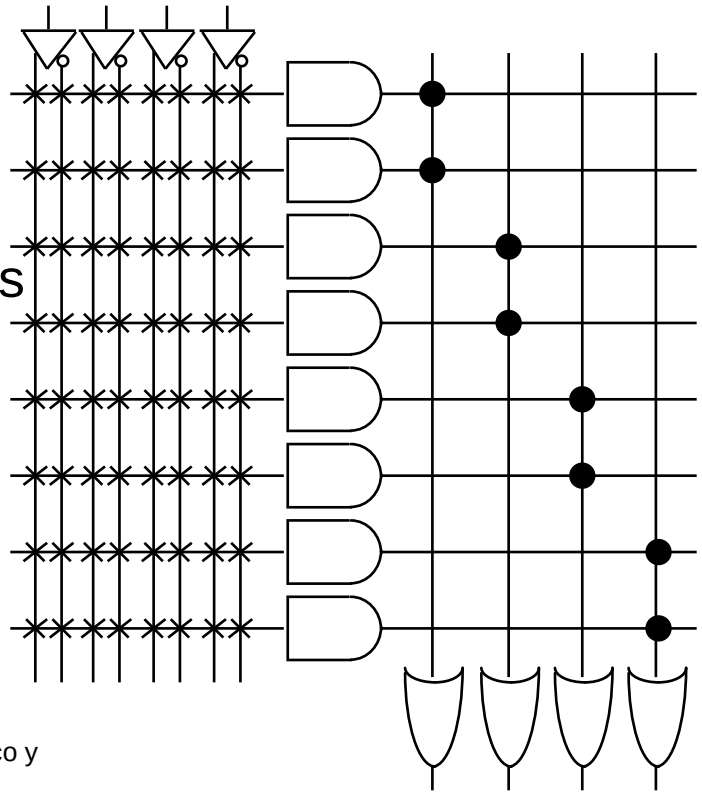
A	B	C	F1	F2	F3	F4	F5	F6
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	1	1



PAL's y PLA's

- Programmable Logic Array (PLA)
  - lo que hemos visto hasta ahora
  - arreglos de AND y OR completamente generales y sin restricciones
- Programmable Array Logic (PAL)
  - topología restringida del arreglo OR
  - innovación de Monolithic Memories
  - planos OR más rápidos y más pequeños

una columna del arreglo OR
tiene sólo acceso a un subconjunto
de los posibles términos productos



PALs y PLAs: ejemplo de diseño

- conversor código BCD a Gray

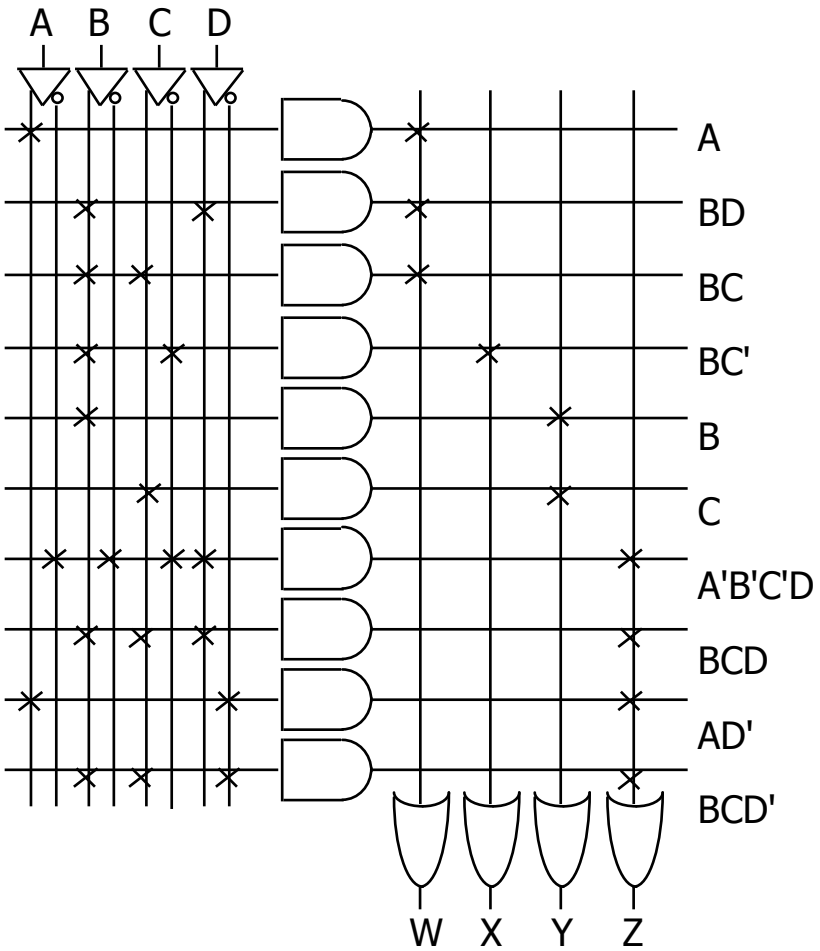
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	–	–	–	–	–
1	1	–	–	–	–	–	–

funciones minimizadas:

W = A + BD + BC
X = BC'
Y = B + C
Z = A'B'C'D + BCD + AD' + B'CD'

PALs y PLAs: ejemplo de diseño (cont.)

- Conversor de código: PLA programado



función minimizada:

W = A + BD + BC
X = B C'
Y = B + C
Z = A'B'C'D + BCD + AD' + B'CD'

no es particularmente un buen candidata para una implementación con PAL/PLA ya que no hay términos para compartir con las salidas

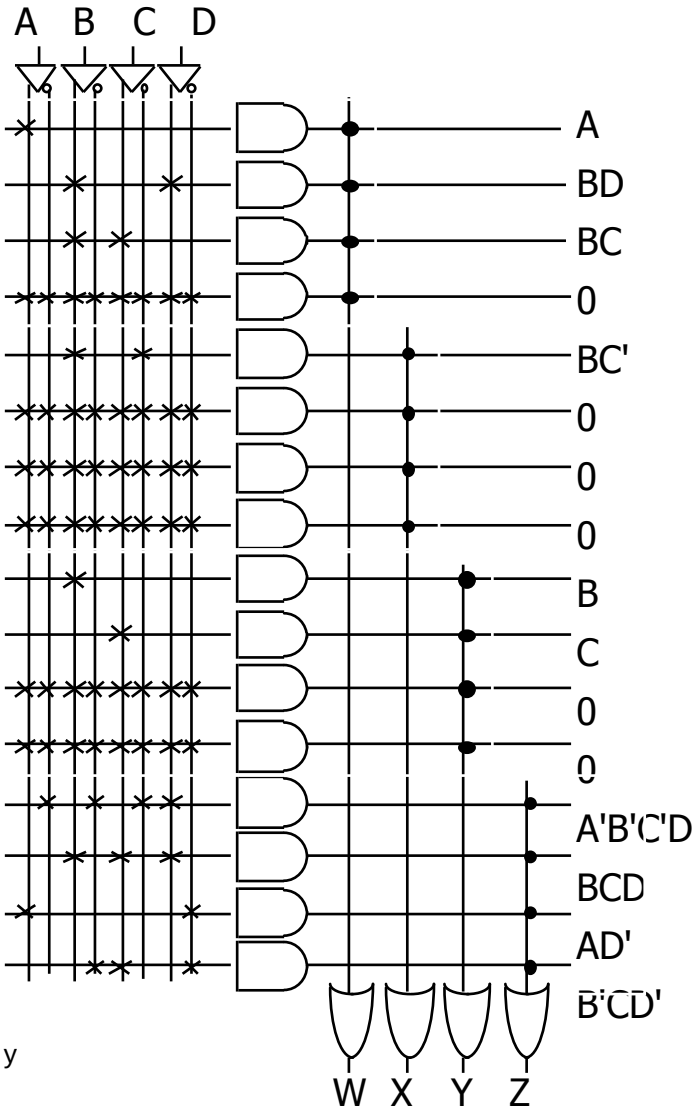
sin embargo, se logra una implementación más compacta y regular cuando se compara con compuertas discretas AND y OR

PALs y PLAs: ejemplo de diseño (cont.)

- Conversor de código:

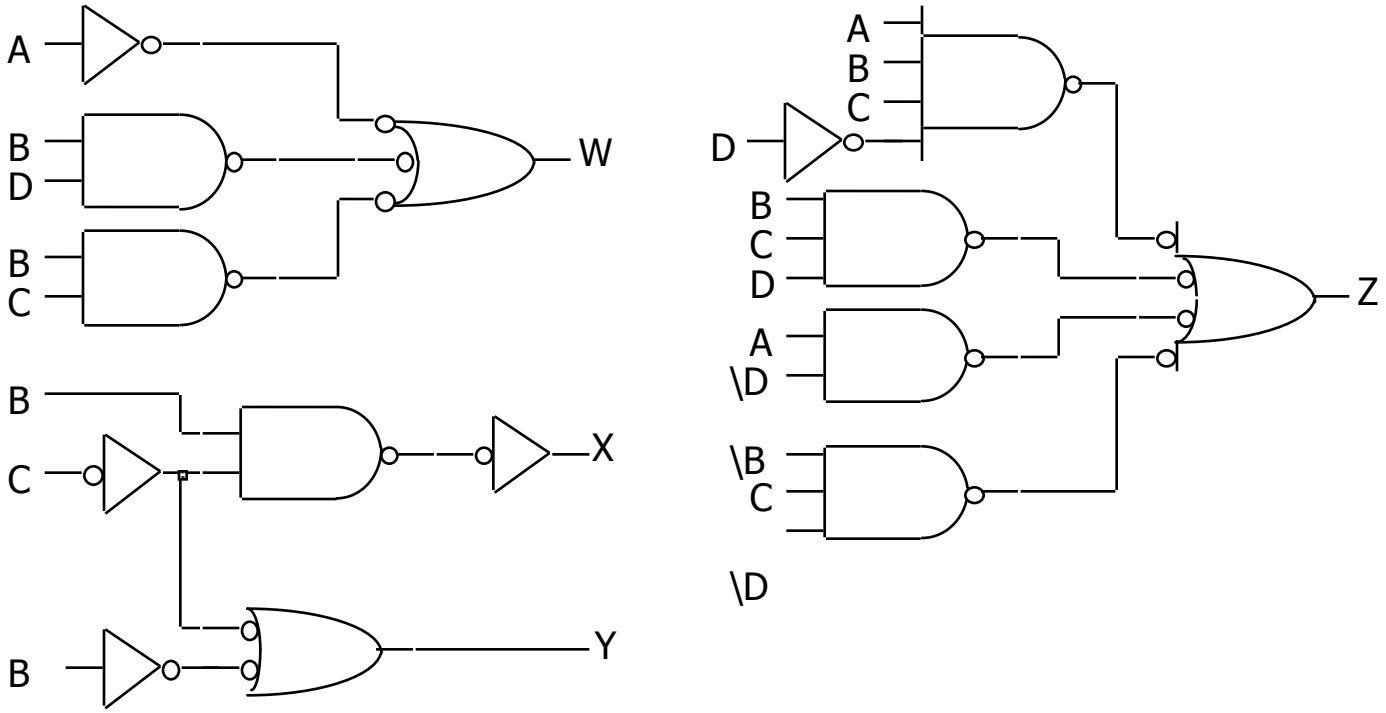
  PAL programado

4 términos productos
por cada OR



PALs y PLAs: ejemplo de diseño (cont.)

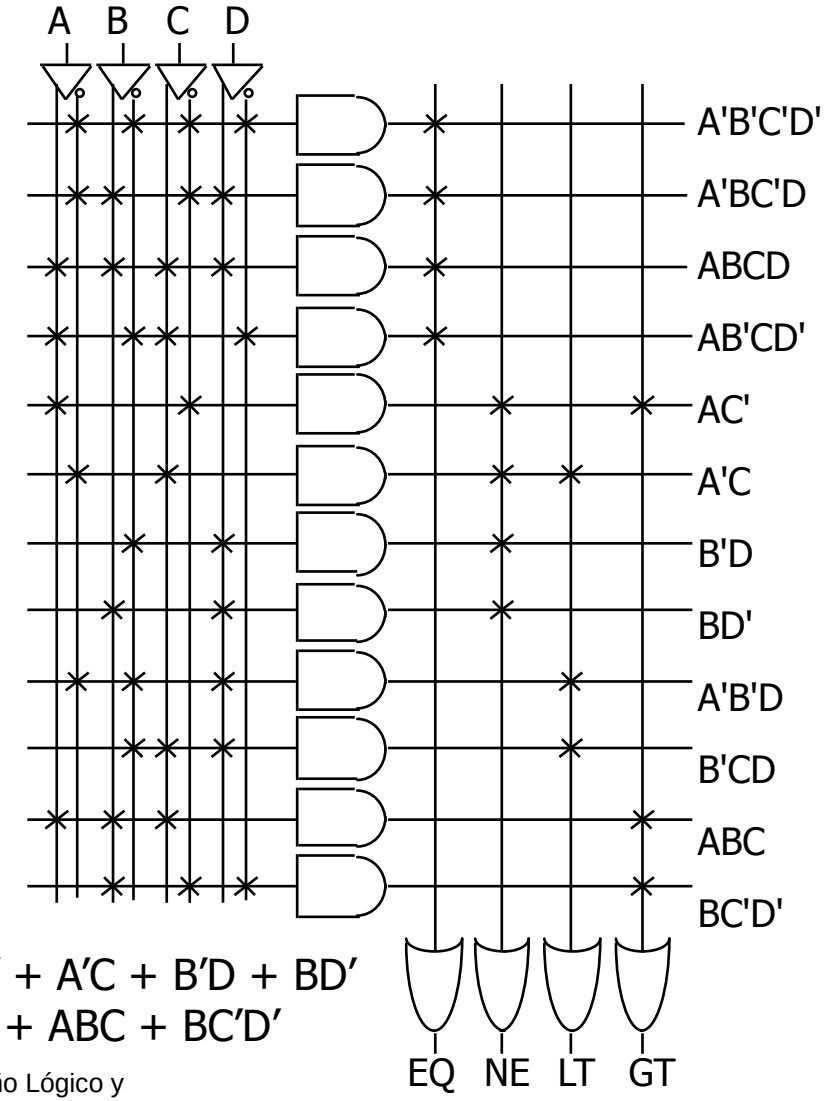
- Conversor de código: implementación con NAND
  - no es regular, difícil de entender
  - difícil para hacer cambios



PALs y PLAs: otro ejemplo de diseño

- Comparador de magnitudes

A	B	C	D	EQ	NE	LT	GT
0	0	0	0	1	0	0	0
0	0	0	1	0	1	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	0	1
1	0	1	0	1	0	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	0	0	1	0	1
1	1	1	1	1	0	0	0



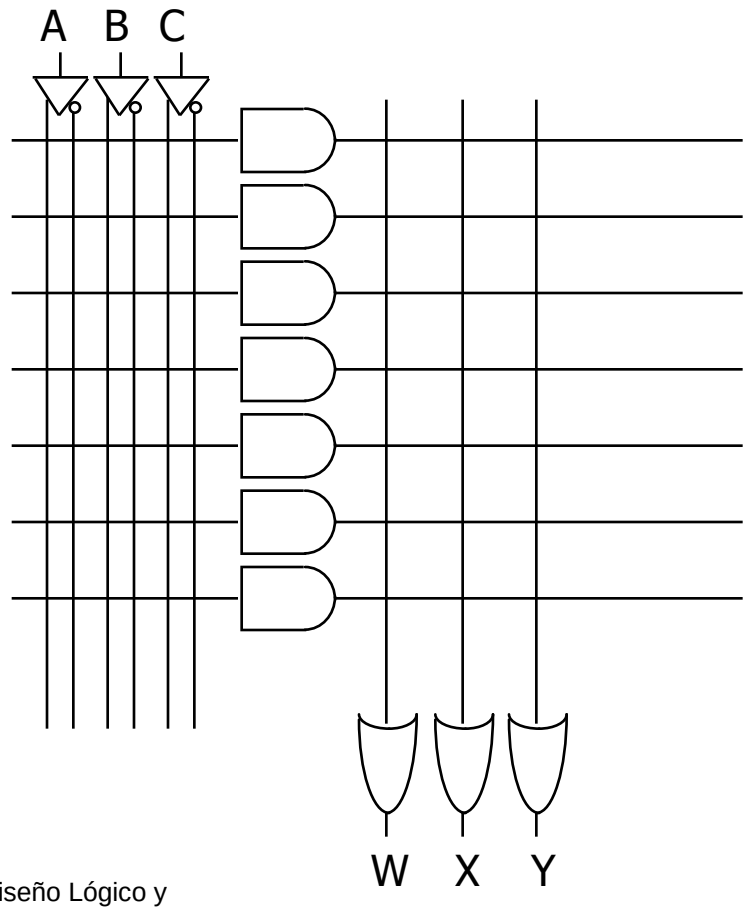
funciones minimizadas:

EQ = A′B′C′D′ + A′BC′D + ABCD + AB′CD′ NE = AC′ + A′C + B′D + BD′

LT = A′C + A′B′D + B′CD GT = AC′ + ABC + BC′D′

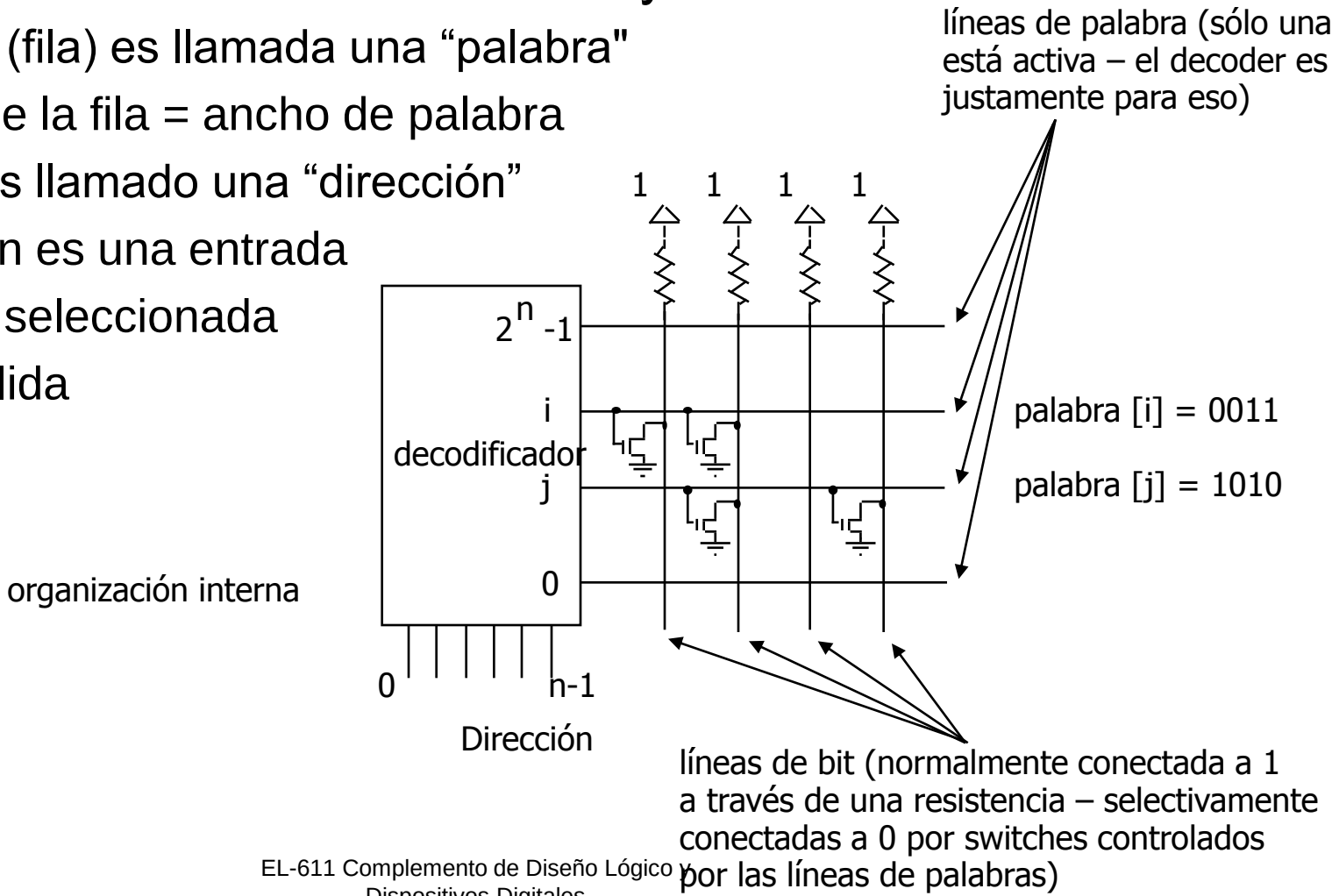
Tarea

- Implementar las siguientes funciones en el PLA siguiente:
  - W = AB + A'C' + BC'
  - X = ABC + AB' + A'B
  - Y = ABC' + BC + B'C'



Read-Only Memories (ROM)

- Arreglo de dos dimensiones de 1's y 0's
  - entrada (fila) es llamada una "palabra"
  - ancho de la fila = ancho de palabra
  - índice es llamado una "dirección"
  - dirección es una entrada
  - palabra seleccionada es la salida



organización interna

ROMs y lógica combinacional

- Implementación lógica combinacional (forma canónica de dos niveles) usando una ROM

$$F0 = A' B' C + A B' C' + A B' C$$

$$F1 = A' B' C + A' B C' + A B C$$

$$F2 = A' B' C' + A' B' C + A B' C'$$

$$F3 = A' B C + A B' C' + A B C'$$

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

tabla de verdad

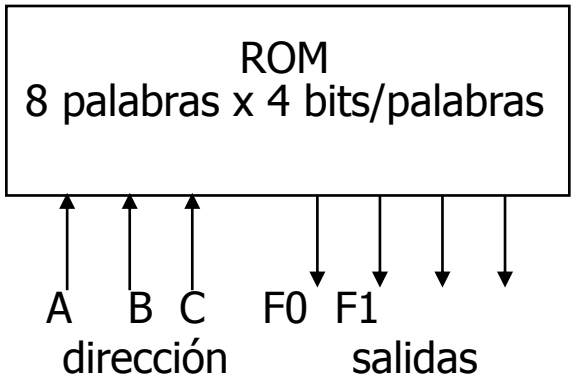
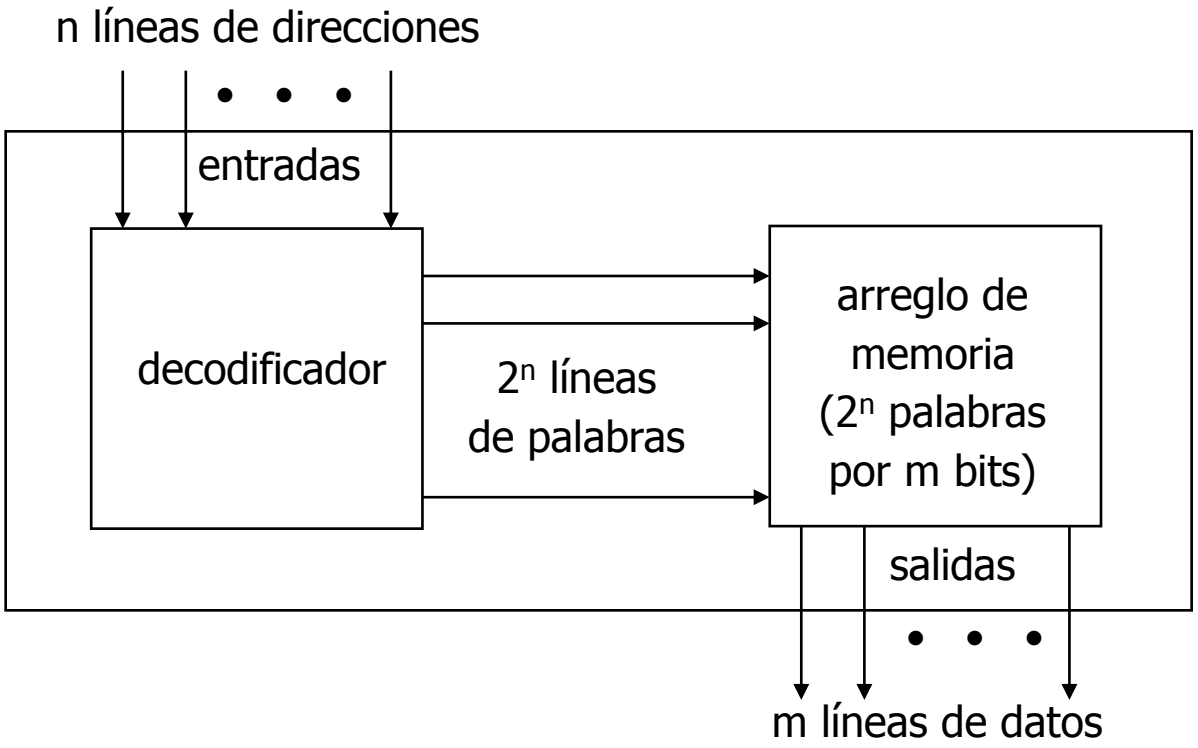


diagrama en bloques

Estructura de una ROM

- Similar a la estructura de un PLA pero con un arreglo AND completamente decodificado
  - arreglo OR completamente flexible (no como el PAL)

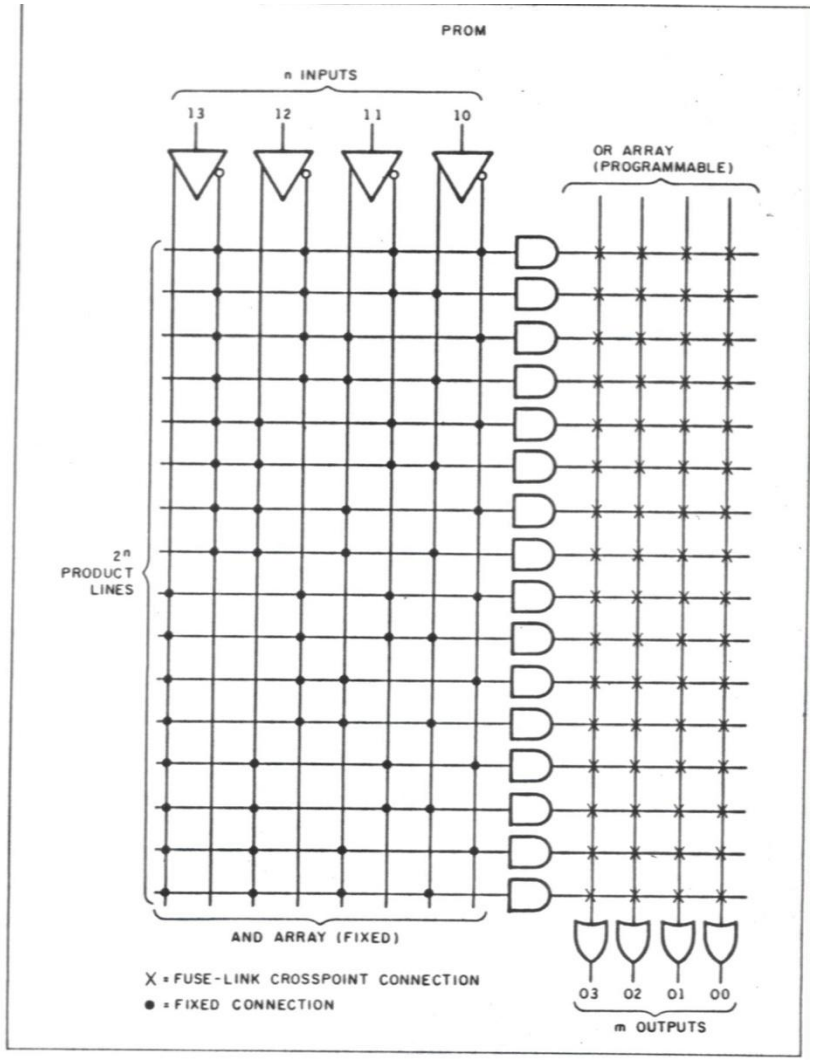


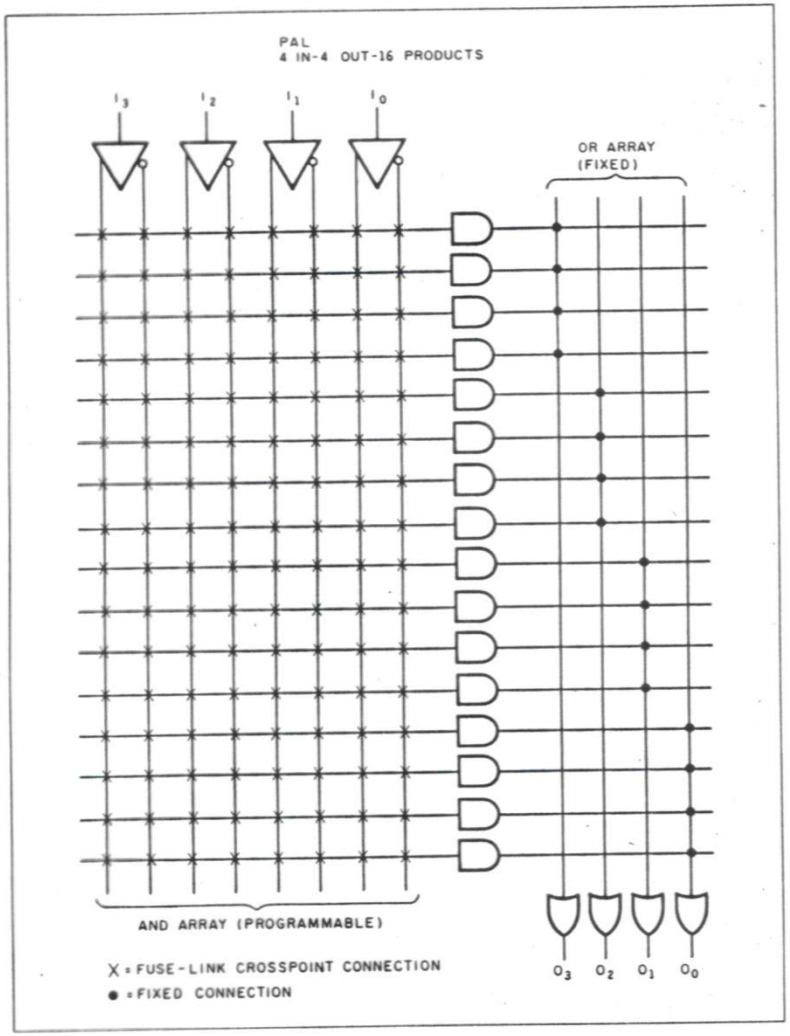
ROM vs. PLA

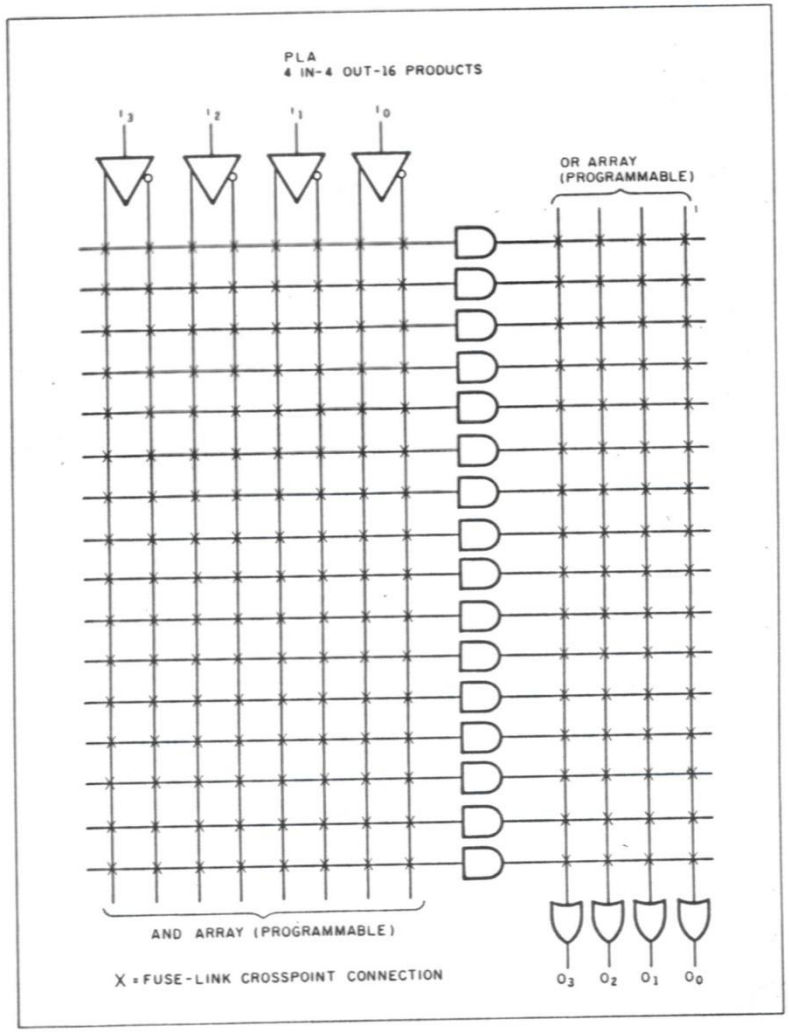
- ROM se ve ventajoso cuando
  - tiempo de diseño es corto (no necesita minimizar funciones de salida)
  - se necesitan la mayoría de las combinaciones de entrada (ej., conversores de códigos)
  - poco compartir de términos productos entre las funciones de salida
- Problemas con las ROM
  - el tamaño se va al doble por cada entrada adicional
  - no puede aprovechar los “no importa”
- PLA es ventajoso cuando
  - hay disponibles herramientas de minimización para multi-salidas
  - hay relativamente pocas combinaciones únicas de minitérminos
  - muchos minitérminos son compartidos entre las funciones de salidas
- Problemas con los PAL
  - restricciones de “fan-in” en el plano OR

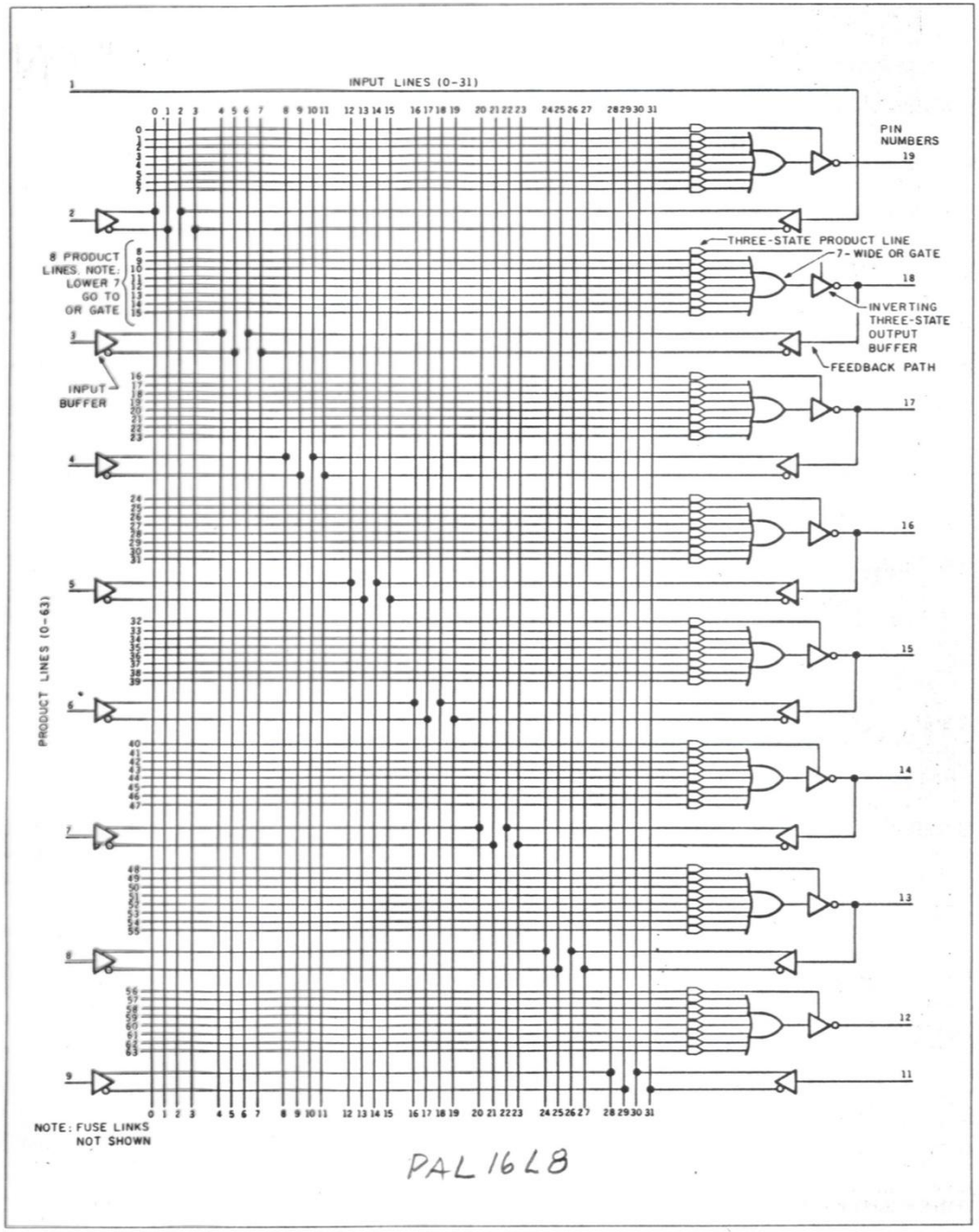
Estructuras lógicas regulares para lógica de dos niveles

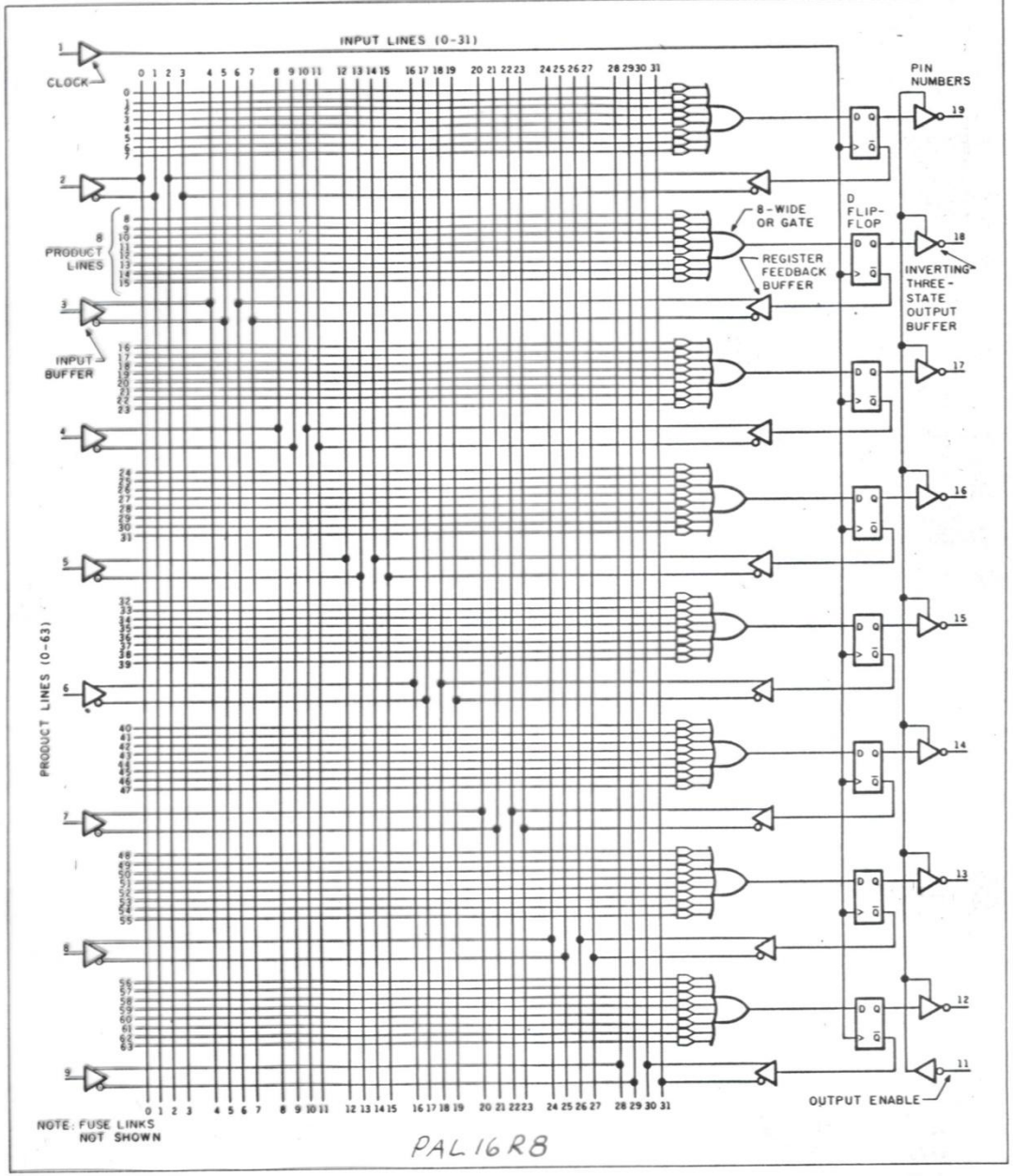
- ROM – plano AND completo, plano OR general
  - barato (componente de alto volumen)
  - puede implementar cualquier función de n entradas
  - velocidad media
- PAL – plano AND programable, plano OR fijo
  - costo intermedio
  - puede implementar funciones limitado por el número de términos productos
  - alta velocidad (sólo un plano programable que es mucho más pequeño que el decodificador de la ROM)
- PLA – planos AND y OR programables
  - más caro (más complejo en el diseño, necesita más herramientas sofisticadas)
  - puede implementar cualquier función hasta un límite de términos productos
  - lento (dos planos programables)


PROM
n INPUTS
13
12
11
10
OR ARRAY
(PROGRAMMABLE)
2ⁿ
PRODUCT
LINES
AND ARRAY (FIXED)
X = FUSE-LINK CROSSPOINT CONNECTION
● = FIXED CONNECTION
O3
O2
O1
O0
m OUTPUTS


PAL
4 IN-4 OUT-16 PRODUCTS
I_3
I_2
I_1
I_0
OR ARRAY
(FIXED)
AND ARRAY (PROGRAMMABLE)
X = FUSE-LINK CROSSPOINT CONNECTION
● = FIXED CONNECTION
O_3
O_2
O_1
O_0


PLA
4 IN-4 OUT-16 PRODUCTS
I_3
I_2
I_1
I_0
OR ARRAY
(PROGRAMMABLE)
AND ARRAY (PROGRAMMABLE)
X = FUSE-LINK CROSSPOINT CONNECTION
O_3
O_2
O_1
O_0


INPUT LINES (0-31)
PIN NUMBERS
8 PRODUCT LINES. NOTE: LOWER 7 GO TO OR GATE
THREE-STATE PRODUCT LINE
7-WIDE OR GATE
INVERTING THREE-STATE OUTPUT BUFFER
FEEDBACK PATH
INPUT BUFFER
PRODUCT LINES (0-63)
NOTE: FUSE LINKS NOT SHOWN
PAL 16L8

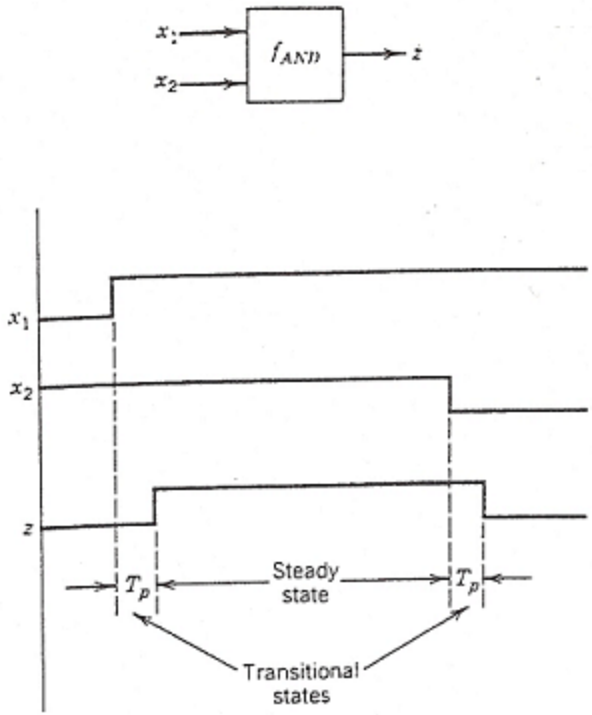

INPUT LINES (0-31)
CLOCK
PIN NUMBERS
8-WIDE OR GATE
D FLIP-FLOP
8 PRODUCT LINES
REGISTER FEEDBACK BUFFER
INVERTING THREE-STATE OUTPUT BUFFER
INPUT BUFFER
PRODUCT LINES (0-63)
OUTPUT ENABLE
NOTE: FUSE LINKS NOT SHOWN
PAL16R8

Módulo Combinacional

- Parámetros
  - tecnología (TTL, MOS, etc.)
  - factor de carga L de entrada (“fan-in”)
  - factor de carga F de salida (“fan-out”)
  - retardo T_p (“delay”): comportamiento dinámico

$$z(t) = F(x(t - T_p))$$

Comportamiento Dinámico



Interconexión de Módulos