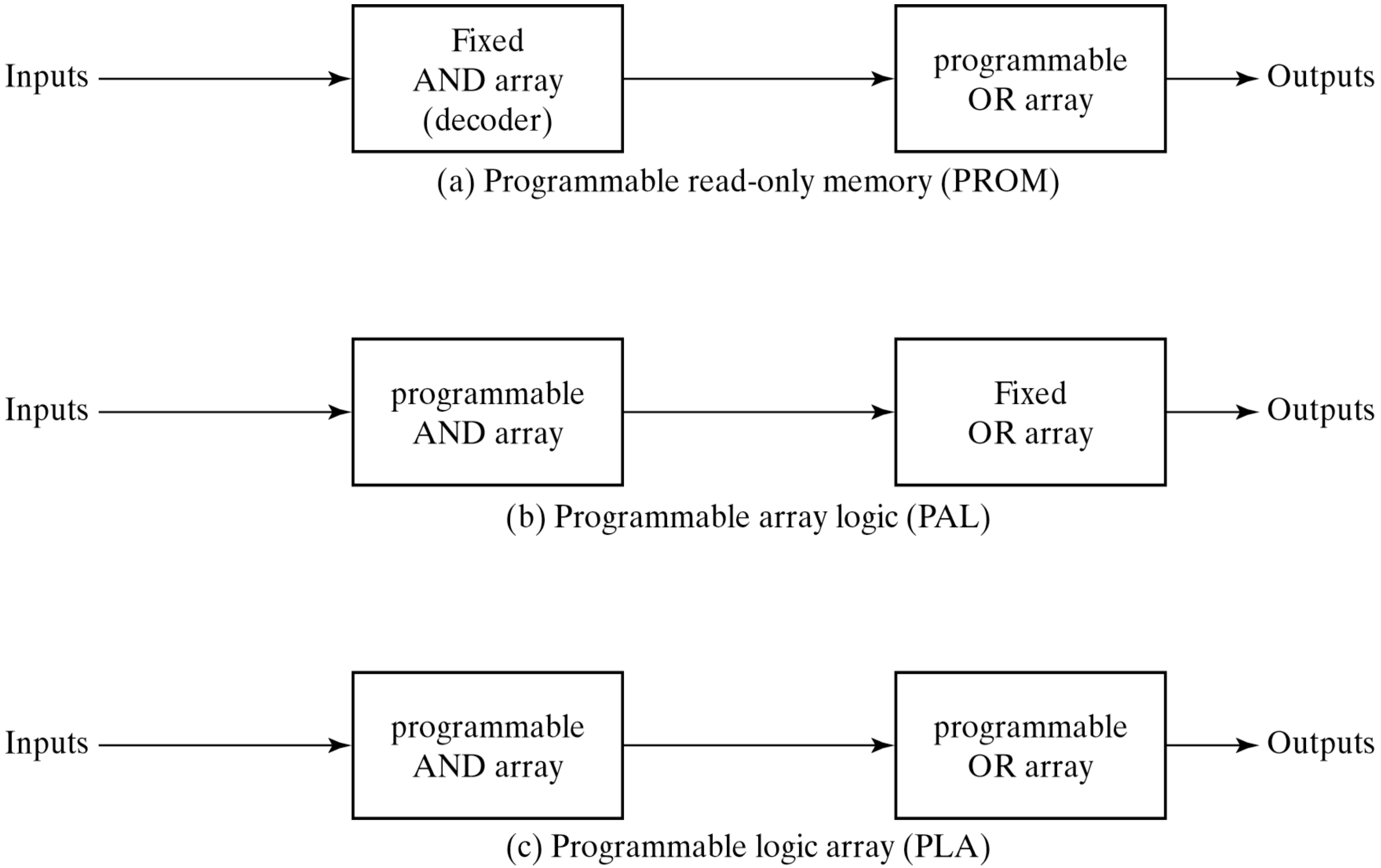
PROGRAMMABLE

LOGIC

DEVICES



(a) Programmable read-only memory (PROM)

(b) Programmable array logic (PAL)

(c) Programmable logic array (PLA)

Fig. 7-13 Basic Configuration of Three PLDs

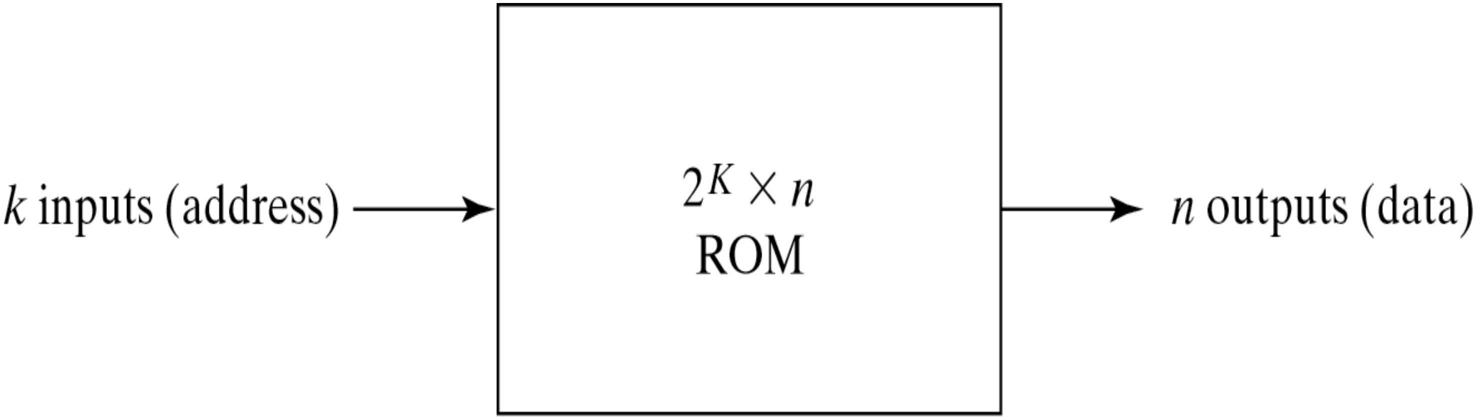
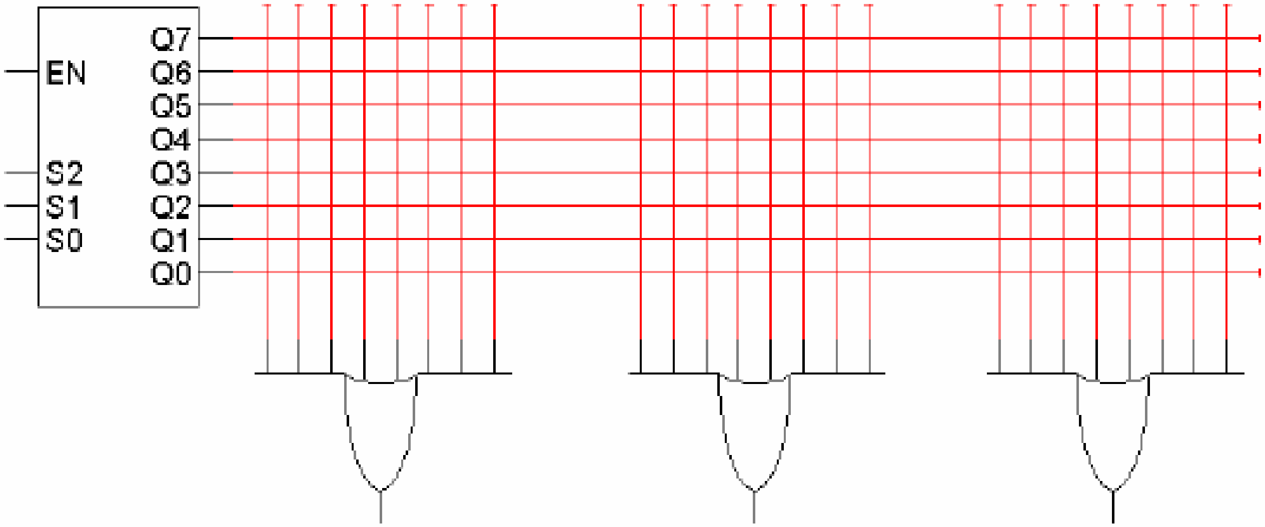


Fig. 7-9 ROM Block Diagram

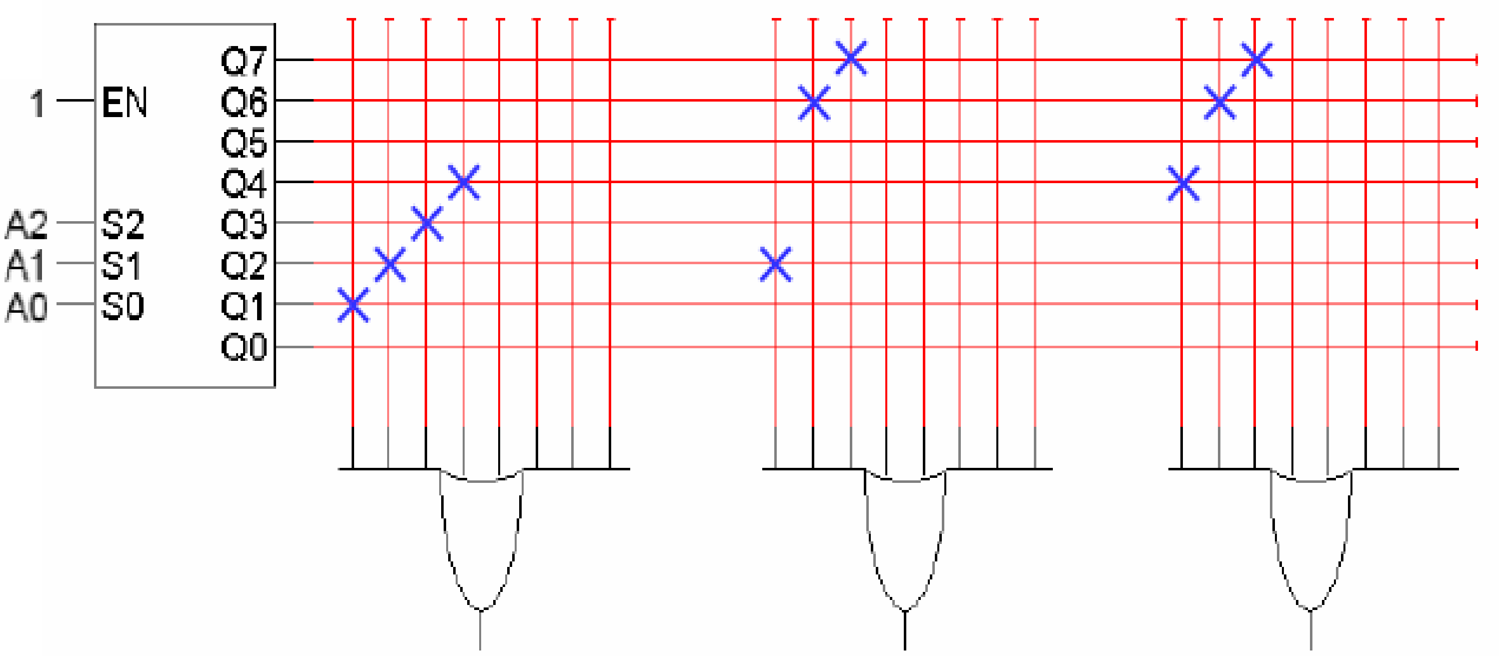
Programmable logic devices

- Building circuits with decoders is so easy that programmable logic devices are often based around decoders.
- The diagram below shows a blank read-only memory, or ROM, device.
  - It's just a decoder whose outputs may be sent to several OR gates.
  - The connections between the decoder outputs and the OR gate inputs are "programmable," so different functions can be implemented.
- To program a ROM for some specific functions, you just have to make the right connections.

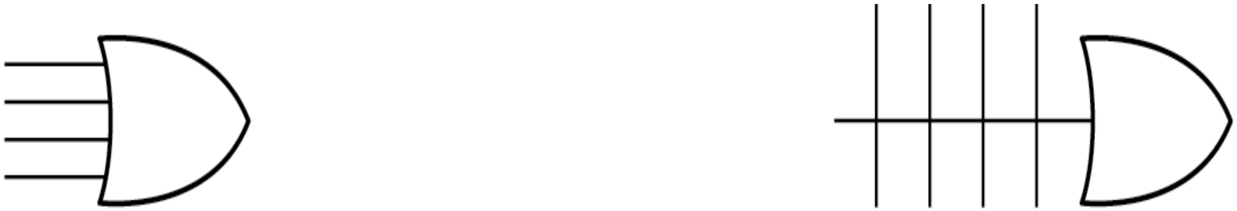


ROM example

- Here are three functions V2, V1 and V0, implemented with our ROM.
- Blue crosses (X) indicate connections between decoder outputs and OR gates. Empty intersections are not connected.
- This is an 8 × 3 ROM since there are 8 decoder outputs and 3 OR gates.



V2 = Σm(1,2,3,4) V1 = Σm(2,6,7) V0 = Σm(4,6,7)



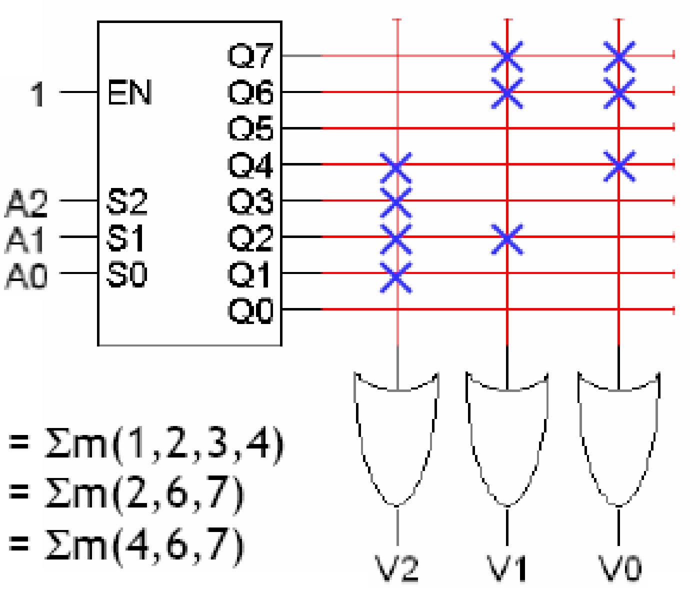
(a) Conventional symbol

(b) Array logic symbol

Fig. 7-1 Conventional and Array Logic Diagrams for OR Gate

The same example again

- Here is an alternative presentation of the same 8 × 3 ROM, but with some simplified OR gates just to make the diagram neater.



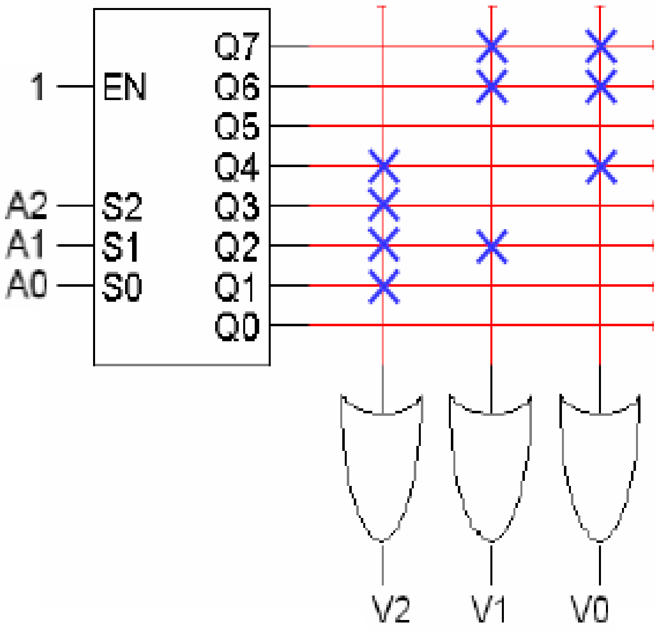
$V2 = \Sigma m(1,2,3,4)$

$V1 = \Sigma m(2,6,7)$

$V0 = \Sigma m(4,6,7)$

Why is this called a memory?

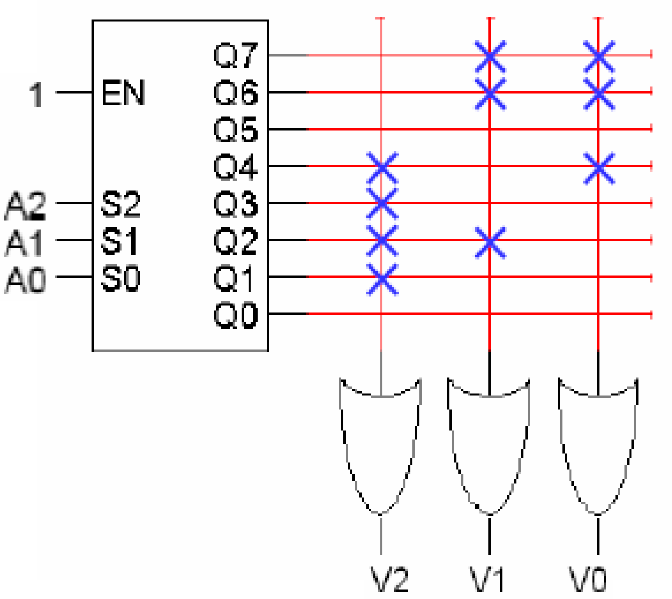
- You can think of this circuit as performing a computation on the inputs A2A1A0, to produce the outputs V2V1V0.
- Note that if the same inputs are chosen again, this circuit would perform the same calculation again, and produce the same output.
- In a sense, the circuit "remembers" the output for each possible input.



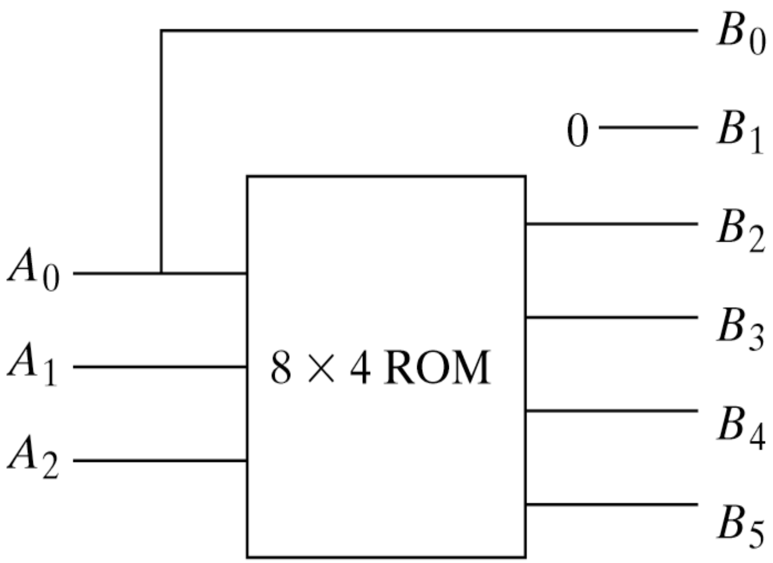
A2	A1	A0	V2	V1	V0
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	0	1	1

Viewing circuits as memories

- So you can also think of this circuit as a memory.
  - It stores eight values of data, each consisting of three bits V2V1V0.
  - A2A1A0 form an address that refers to one of the eight stored values.
- This memory is read-only since you can't modify the data without going through the time-consuming process of re-programming the ROM.



Address			Data		
A2	A1	A0	V2	V1	V0
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	0	1	1



(a) Block diagram

A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

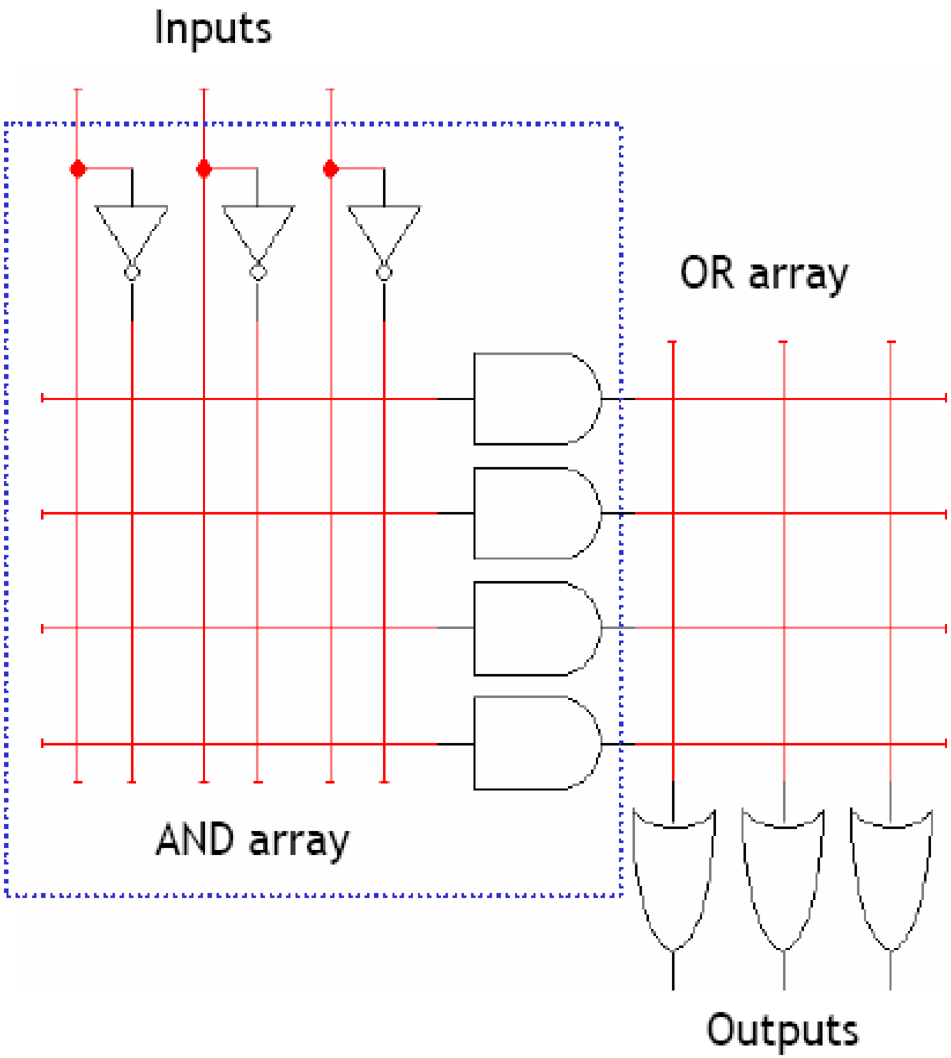
Fig. 7-12 ROM Implementation of Example 7-1

Programmable logic arrays

- A ROM is potentially inefficient since it uses a decoder, which generates *all* possible minterms. No circuit minimization is done.
- Using a ROM to implement an n-input function requires many gates.
  - An n-to-2^n decoder has n inverters and 2^n n-input AND gates.
  - We also need an OR gate with up to 2^n inputs.
  - The number of gates roughly doubles for each additional ROM input.
- A programmable logic array or PLA makes the decoder part of the ROM programmable too. Instead of generating all possible minterms, you can choose which products (not necessarily minterms) to generate.

A blank 3 × 4 × 3 PLA

- This is a 3 × 4 × 3 PLA (3 inputs, up to 4 product terms, and 3 outputs), ready to be programmed.
- The left part of this diagram replaces the decoder used in a ROM.
- Connections are made within the AND array to produce four arbitrary products.
- Those products are then summed in the OR array.



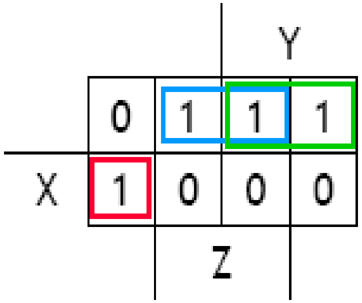
Regular K-map minimization

- The normal K-map approach is to minimize the number of product terms for each *individual* function.
- For our three sample functions, this would result in a total of six different product terms.

$$V2 = \Sigma m(1,2,3,4)$$

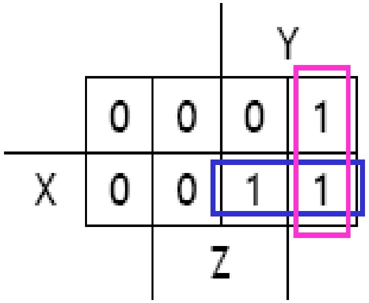
$$V1 = \Sigma m(2,6,7)$$

$$V0 = \Sigma m(4,6,7)$$



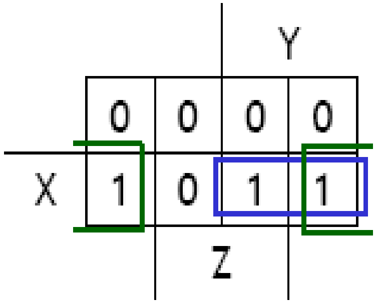
	0	1	1	1
X	1	0	0	0

$V2 = xy'z' + x'z + x'y$



	0	0	0	1
X	0	0	1	1

$V1 = yz' + xy$



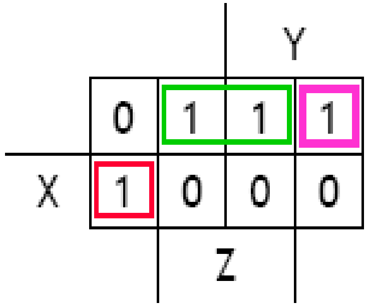
	0	0	0	0
X	1	0	1	1

$V0 = xz' + xy$

PLA minimization

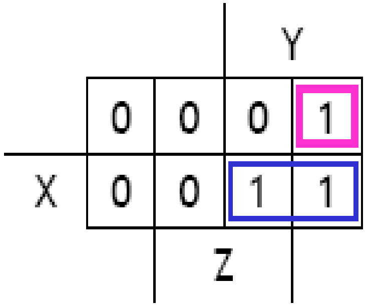
- But for a PLA, what we really want is to minimize the number of product terms in *all* of the functions.
- We could express V2, V1 and V0 with just *four* total products instead.

$$V2 = \Sigma m(1,2,3,4)$$
$$V1 = \Sigma m(2,6,7)$$
$$V0 = \Sigma m(4,6,7)$$



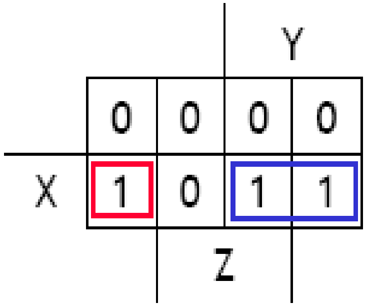
	0	1	1	1
X	1	0	0	0

$V2 = xy'z' + x'z + x'yz'$



	0	0	0	1
X	0	0	1	1

$V1 = x'yz' + xy$

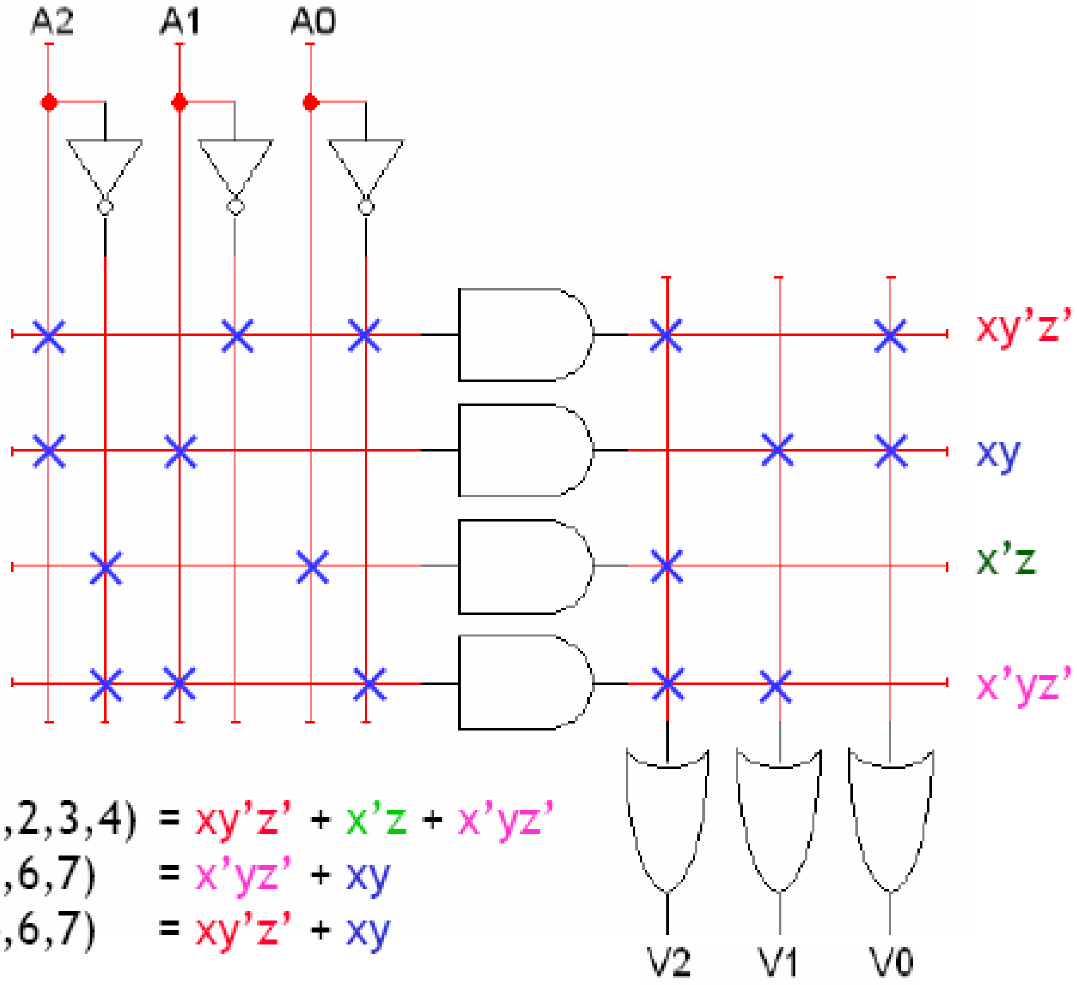


	0	0	0	0
X	1	0	1	1

$V0 = xy'z' + xy$

PLA example

- So we can implement these three functions using a 3 × 4 × 3 PLA.



V2 = Σm(1,2,3,4) = xy'z' + x'z + x'yz'
V1 = Σm(2,6,7) = x'yz' + xy
V0 = Σm(4,6,7) = xy'z' + xy

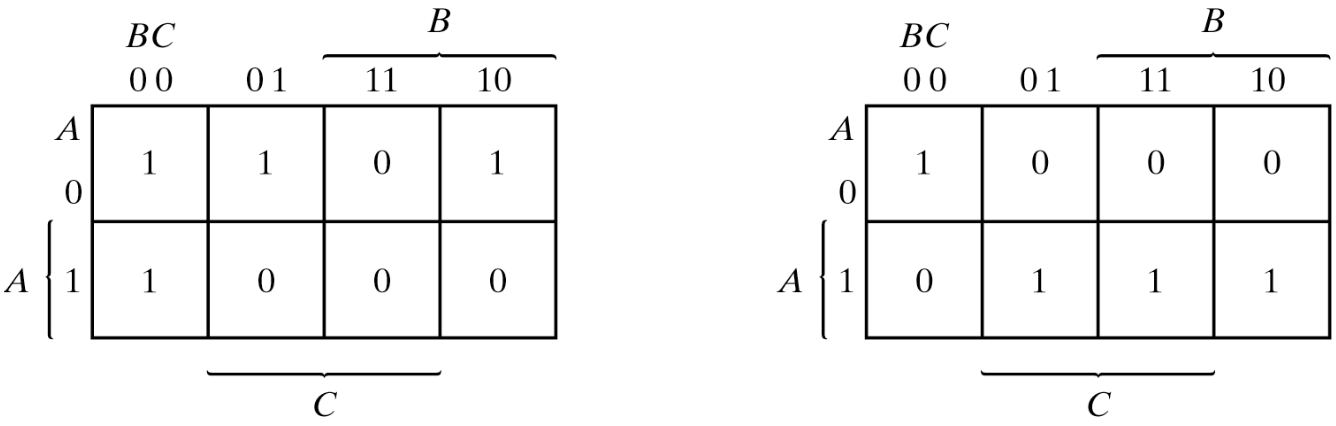
PLA Example

Implement the two functions:

$F1=\Sigma(0,1,2,4)$

$F2= \Sigma(0,5,6,7)$

Using a PLA of size 3x4x2



A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$$F_1 = A'B' + A'C' + B'C'$$
$$F_1 = (AB + AC + BC)'$$

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$$F_2 = AB + AC + A'B'C'$$
$$F_2 = (A'C + A'B + AB'C')'$$

PLA programming table

					Outputs	
	Product term	Inputs			(C)	(T)
		A	B	C	F_1	F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

Fig. 7-15 Solution to Example 7-2

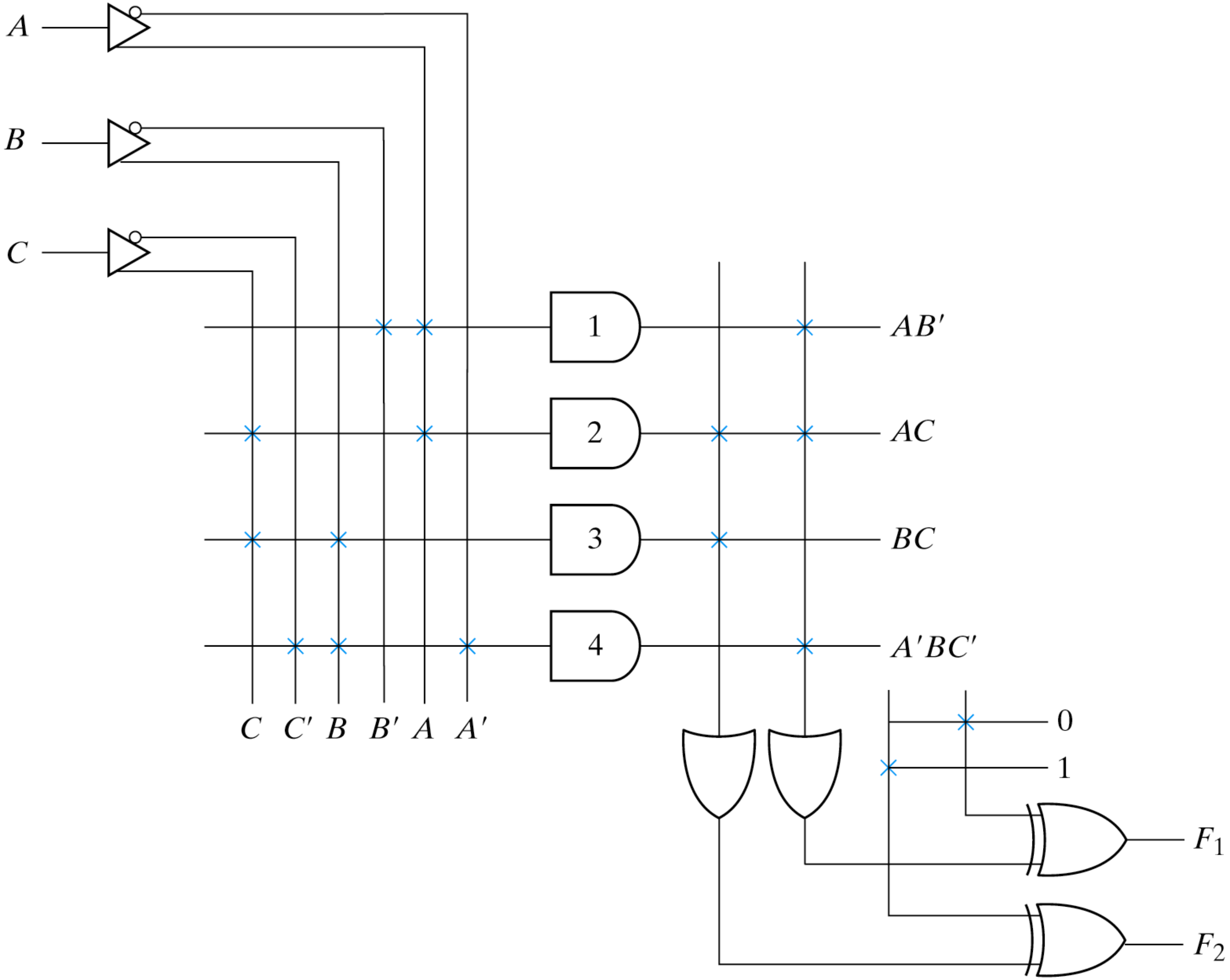


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

PLA evaluation

- A $k \times m \times n$ PLA can implement as many as n functions of k inputs, each of which must be expressible with no more than m product terms.
- Unlike ROMs, PLAs allow you to choose which products are generated.
  - This can significantly reduce the fan-in (number of inputs) of gates, as well as the total number of gates.
  - However, a PLA is less general than a ROM. Not every function may be expressible with the limited number of AND gates in a given PLA.
- You can think of PLAs as memories too.
  - A $k \times m \times n$ PLA has k "address" lines, and each of those 2^k addresses references an n-bit data value.
  - But again, not all possible data values can be stored.

PROGRAMMABLE

ARRAY

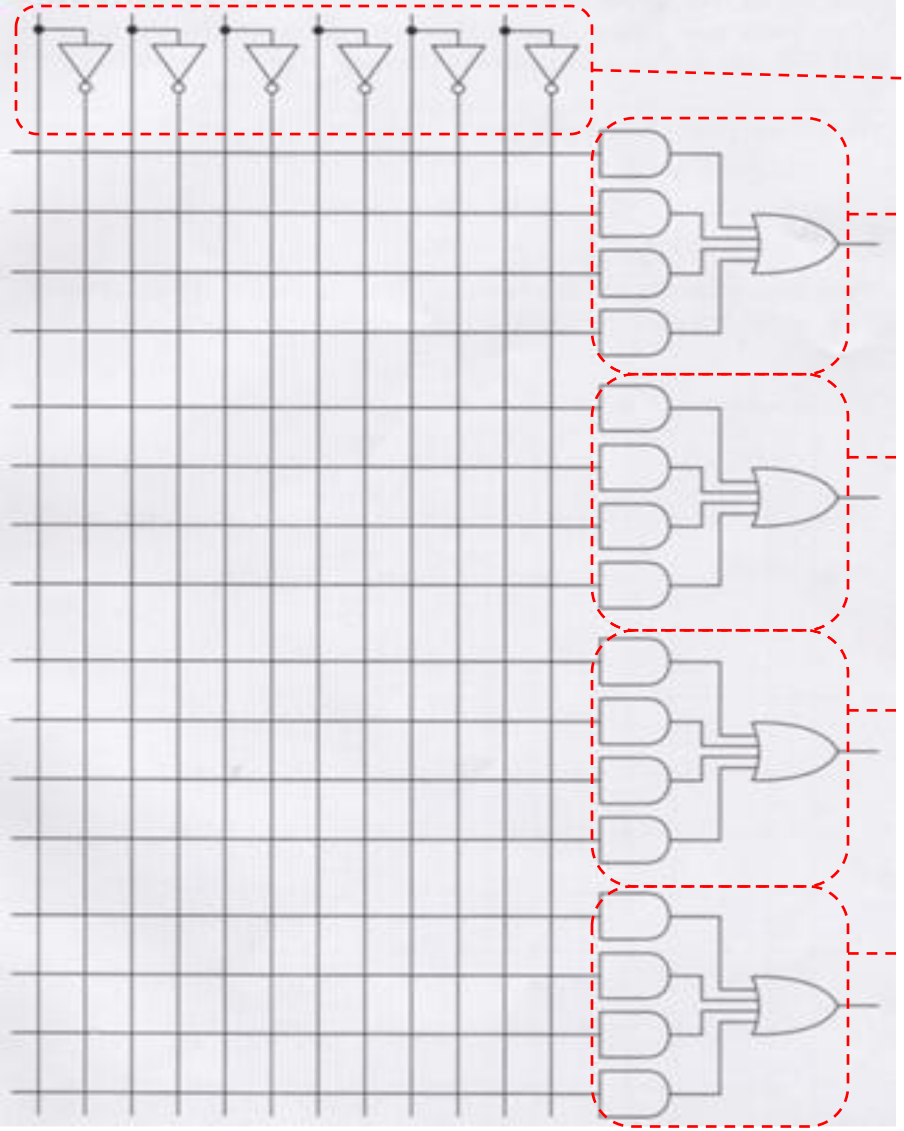
LOGIC

Programmable Array Logic (PAL)

- OR plane (array) is fixed, AND plane can be programmed
- Less flexible than PLA
- # of product terms available per function (OR outputs) is limited

PAL Example

Figure 4.15 A PAL



inputs

1st output section

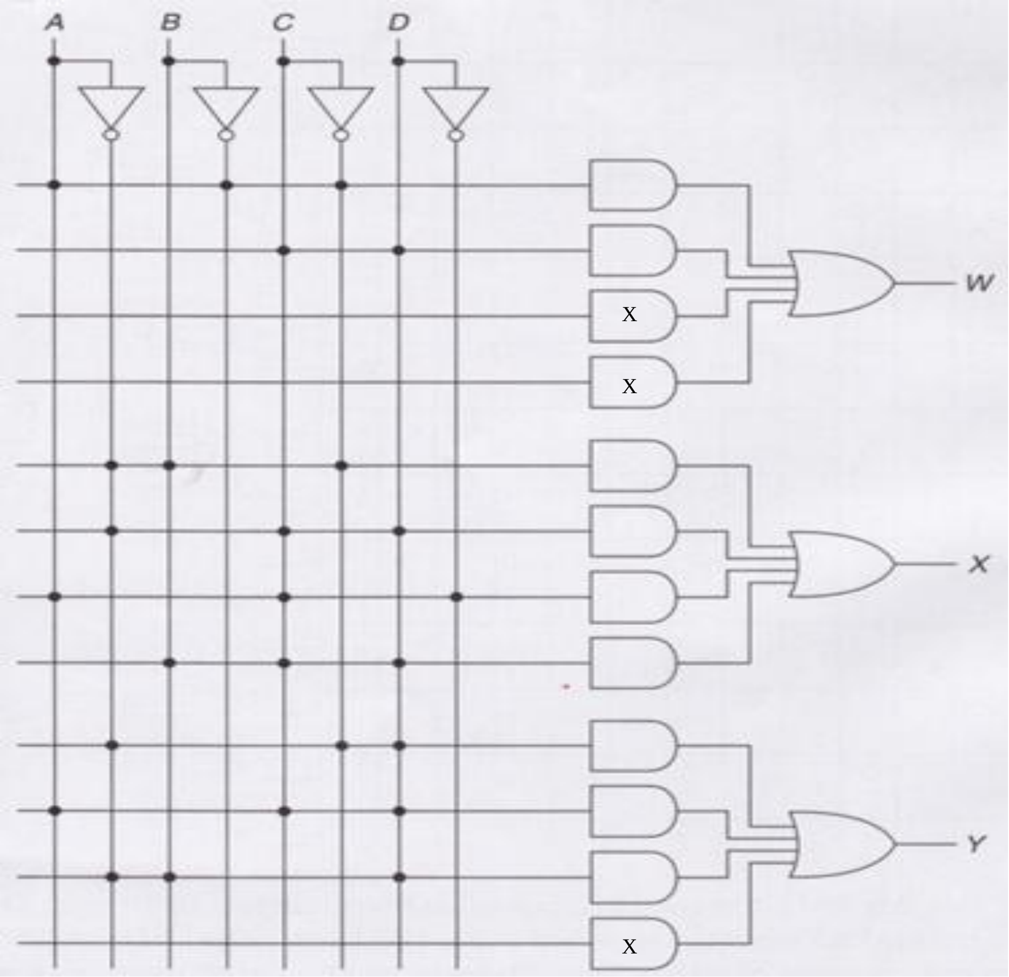
2nd output section

3rd output section

4th output section

Only functions with at most four products can be implemented

PAL-based circuit implementation



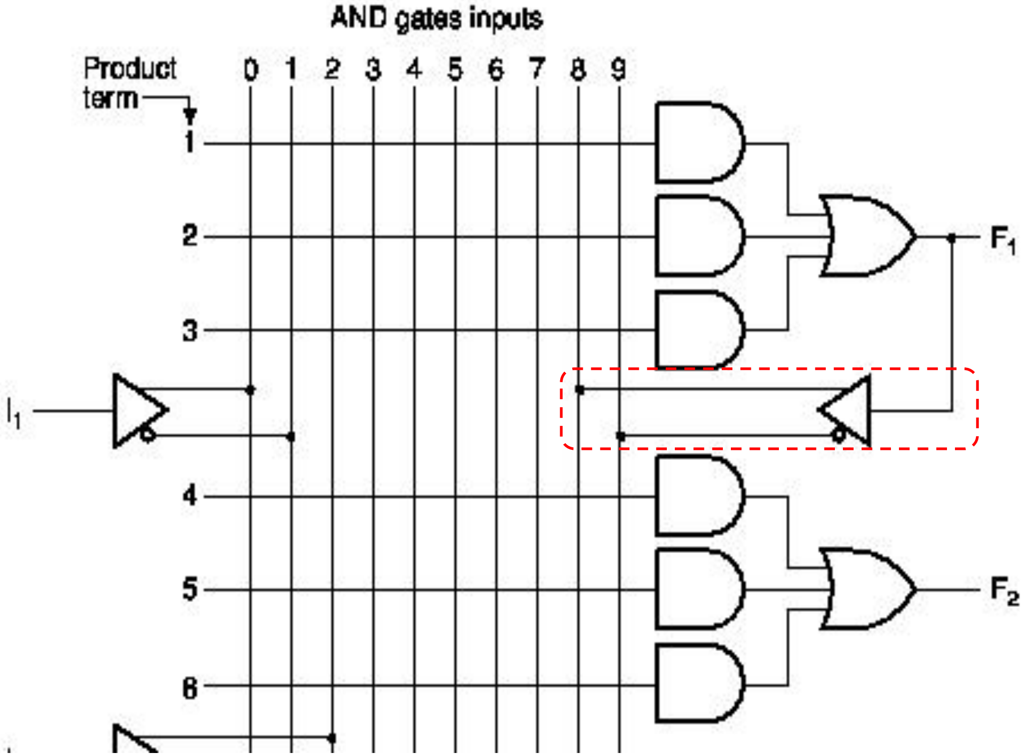
W = AB ' C'   + CD

X = A' BC'   + A' CD + ACD' + BCD

Y = A' C D + ACD + A' BD

Can we implement more complex functions using PALs?

- Yes, by allowing output lines to also serve as input lines in the AND plane.



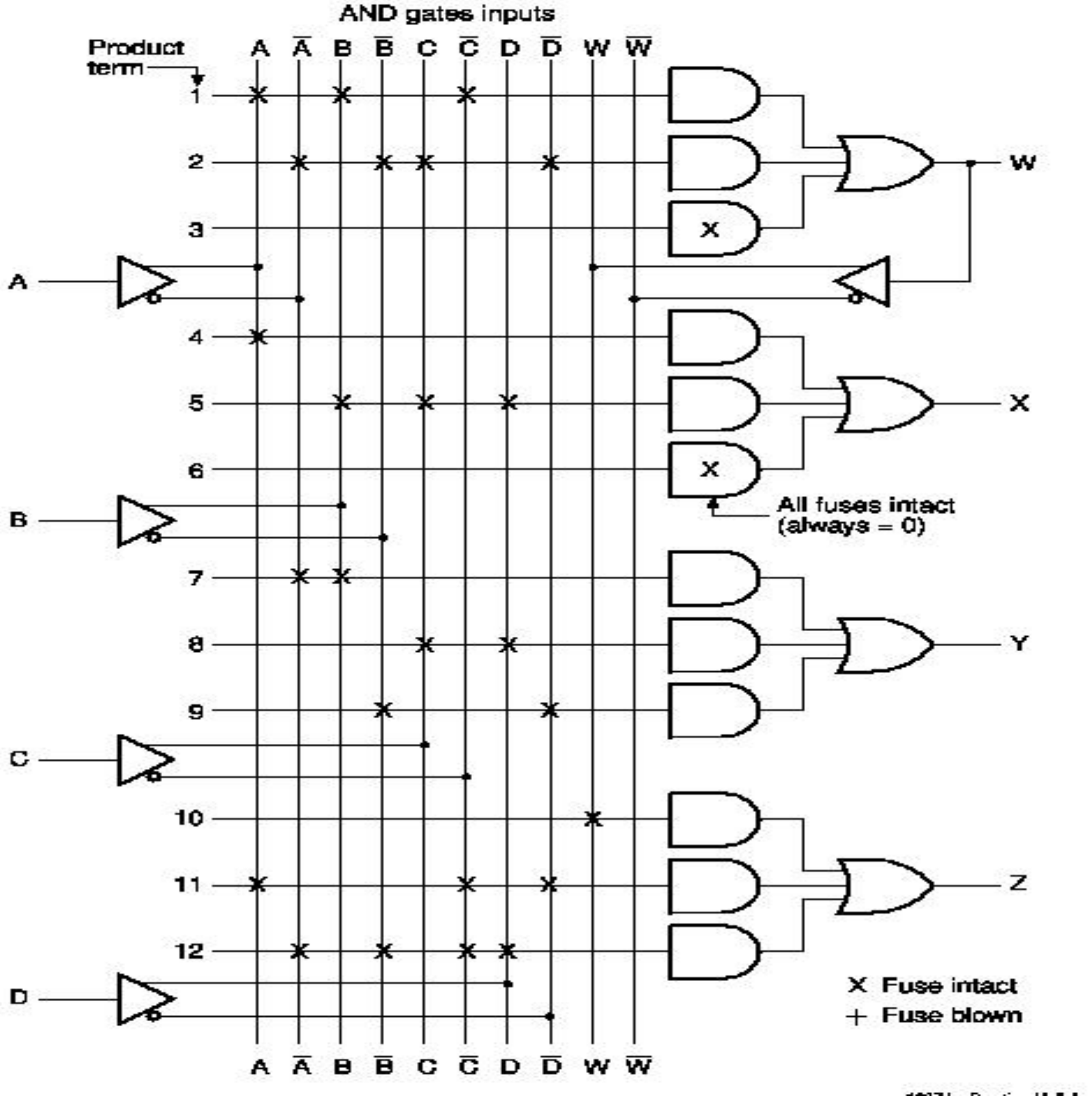
Example

- Implement the combinational circuit described by the following equations, using a PAL with 4 inputs, 4 outputs, and 3-wide AND-OR structure.
  - $W(A,B,C,D) = \sum m(2,12,13)$
  - $X(A,B,C,D) = \sum m(7,8,9,10,11,12,13,14,15)$
  - $Y(A,B,C,D) = \sum m(0,2,3,4,5,6,7,8,10,11,15)$
  - $Z(A,B,C,D) = \sum m(1,2,8,12,13)$

Example (cont.)

- Use function simplification techniques to derive:
  - $W = ABC'+A'B'CD'$
  - $X = A+BCD$
  - $Y=A'B+CD+B'D'$
  - $Z=ABC'+A'B'CD'+AC'D'+A'B'C'D$
    $= W + AC'D'+A'B'C'D$

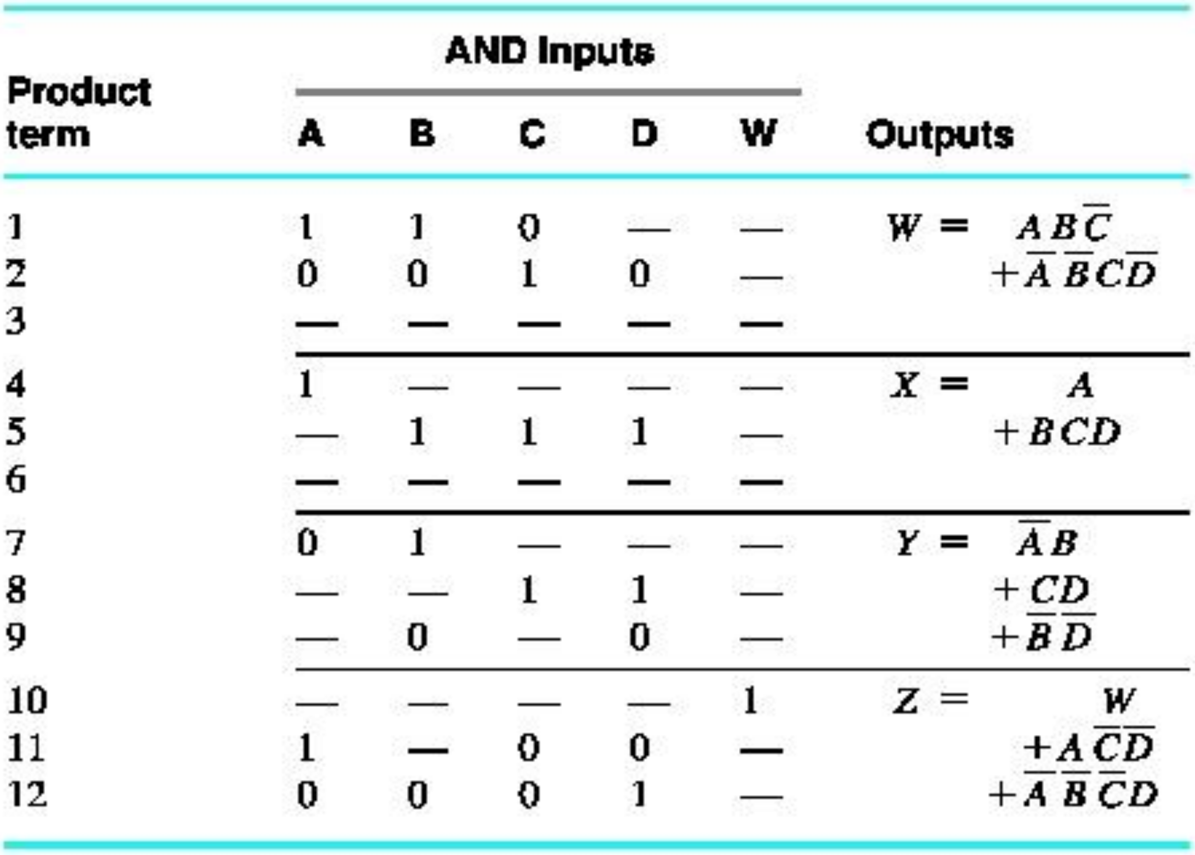
Example (cont.)



Example (cont.)

Tabular Form Specification

of interconnection programming



Product term	AND Inputs					Outputs
	A	B	C	D	W	
1	1	1	0	—	—	$W = A B \overline{C}$
2	0	0	1	0	—	$+ \overline{A}\, \overline{B} C \overline{D}$
3	—	—	—	—	—	
4	1	—	—	—	—	$X = A$
5	—	1	1	1	—	$+ B C D$
6	—	—	—	—	—	
7	0	1	—	—	—	$Y = \overline{A} B$
8	—	—	1	1	—	$+ C D$
9	—	0	—	0	—	$+ \overline{B}\, \overline{D}$
10	—	—	—	—	1	$Z = W$
11	1	—	0	0	—	$+ A \overline{C}\, \overline{D}$
12	0	0	0	1	—	$+ \overline{A}\, \overline{B}\, \overline{C} D$

Implement the functions given below

Using a PLA

- $F1 (a,b,c) = \sum (6,7)$
- $F2 (a,b,c) = \sum (4,5,7)$
- $F3 (a,b,c) = \sum (3,5)$
- $F4 (a,b,c) = \sum (2,6)$