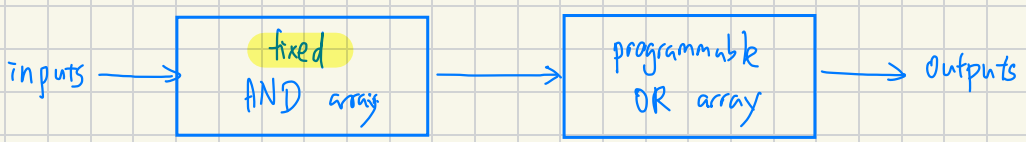
3.2 SPLDs

1 Common SPLDs

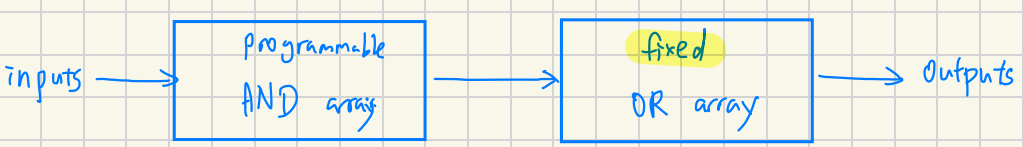
- ROM
- Programmable Logic Array (PLA)
- Programmable Array Logic (PAL)
- Generic Array Logic (GAL)

2 PROM vs PAL vs PLA

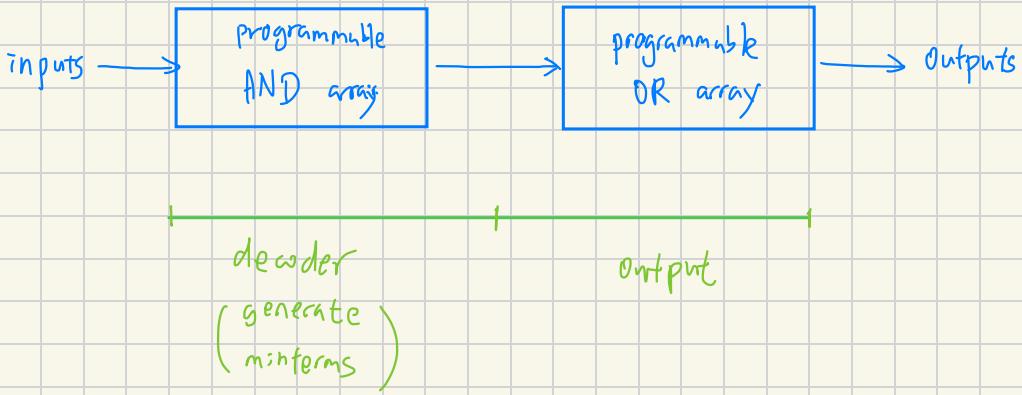
① PROM:



② PAL: (many also have FFs)



③ PLA:



3 ROM

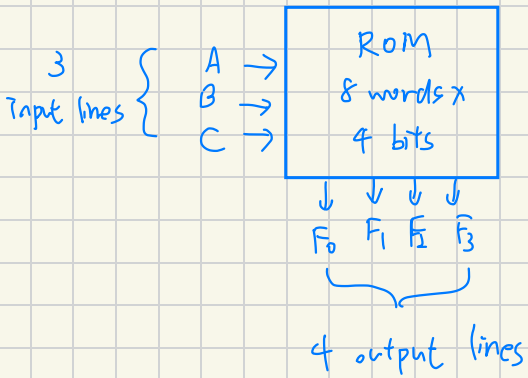
- Outputs can be accessed: LUT
- typical sizes for commercially available:
  $(2^5 \sim 2^{19})$ words x (4 or 8) bits
- capacity of a ROM: $2^n \times m$

  { n: input lines (addresses)
  { m: output lines (data)

  - can store a truth table with 2^n (rows) x m (columns).

- can realize m functions of n input variables.

Ex 1:



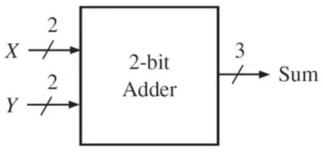
A	B	C	F_0	F_1	F_2	F_3
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1				
1	0	0		⋮		
1	0	1				
1	1	0				
1	1	1	0	1	0	1

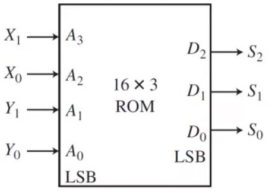
typical data stored in ROM

Ex 2:

Example: a 2-Bit Adder

- ROM implementation of a 2-bit adder:



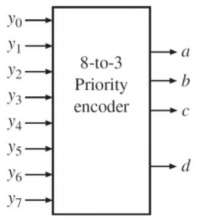


X_1	X_0	Y_1	Y_0	S_2	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Ex 3:

Example: an 8-to-3 priority encoder

- Compute the size of the ROM required to implement an 8-to-3 priority encoder:
  - The 8-to-3 priority encoder has 8 inputs and 4 outputs. ⇒ It needs a $2^8 \times 4$ bit ROM.



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

* Priority: $y_7 > y_6 > \ldots > y_0$

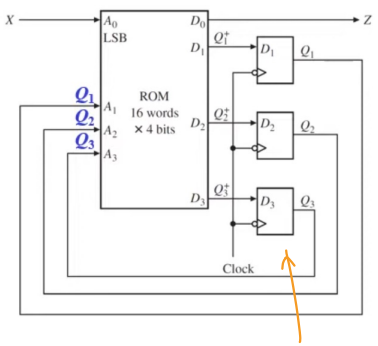
* d: valid-output indicator

Ex 4: A seq ckt

Implement a BCD to excess-3-code converter

<Ans.>

PS	NS X=0	NS X=1	Z X=0	Z X=1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	—	1	—



Q_3	Q_2	Q_1	X	Q_3^+	Q_2^+	Q_1^+	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	1
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	1	0	0	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	1
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

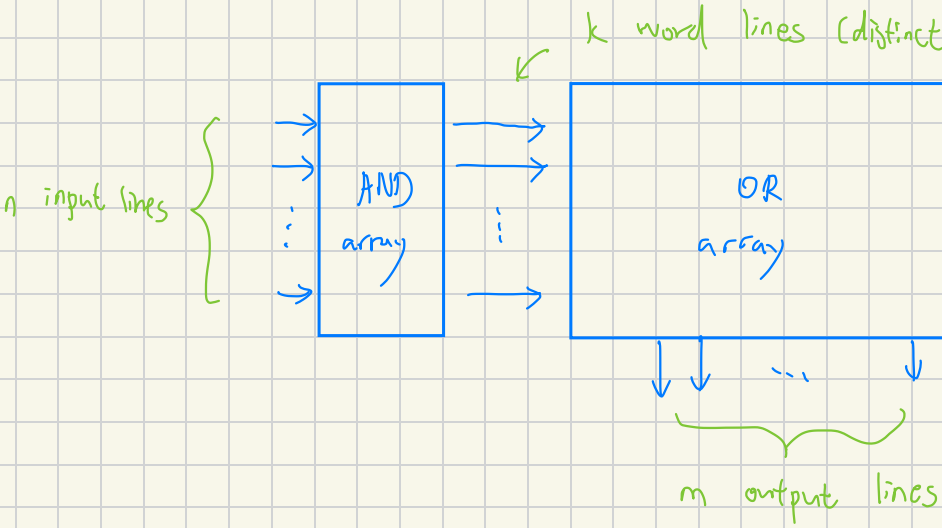
J.J. Shann 3-26

We need FFs to implement seq. ckts.

don't care's

4 Programmable Logic Arrays (PLAs)

- Decoder is replaced with a programmable AND array that realizes selected product terms of the input variables. ⇒ SoP



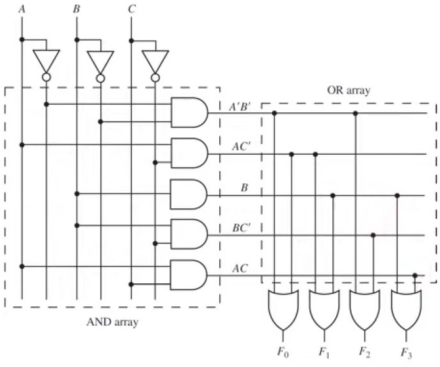
Ex 1:

Example

- Use PLA to realize the following functions:

$$F_0 = \sum m(0,1,4,6) = A'B' + AC$$
$$F_1 = \sum m(2,3,4,6,7) = B + AC'$$
$$F_2 = \sum m(0,1,2,6) = A'B' + BC'$$
$$F_3 = \sum m(2,3,5,6,7) = AC + B$$

$\Rightarrow$ 5 distinct AND terms



PLA table: 3 × 5 × 4

Product Term	Inputs A	B	C	Outputs F_0	F_1	F_2	F_3
$A'B'$	0	0	–	1	0	1	0
AC'	1	–	0	0	1	0	0
B	–	1	–	0	1	0	1
BC'	–	1	0	0	0	1	0
AC	1	–	1	1	0	0	1

3-29

Ex 2 :

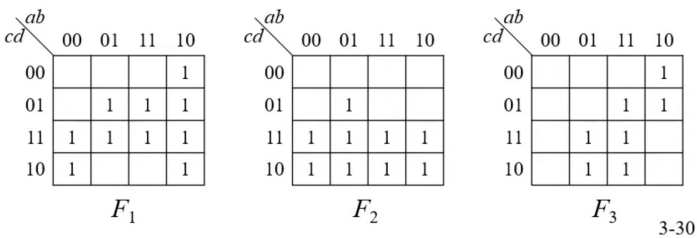
Example

- Realize the following functions using a PLA:

$F_1(a, b, c, d) = \Sigma m(2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

$F_2(a, b, c, d) = \Sigma m(2, 3, 5, 6, 7, 10, 11, 14, 15)$

$F_3(a, b, c, d) = \Sigma m(6, 7, 8, 9, 13, 14, 15)$



3-30

< Approach 1 > minimize each function

<Ans.> Approach 1

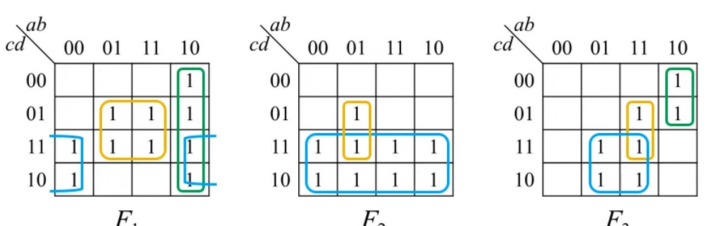
- Minimize each function:

$F_1 = bd + b'c + ab'$

$F_2 = c + a'bd$

$F_3 = bc + ab'c' + abd$

$\Rightarrow$ 8 different product terms $\Rightarrow$ 4×8×3 PLA



3-31

< Approach 2 > Minimize total # of rows in the PLA table (better)

$F_1 = bd + b'c + ab'$
$F_2 = c + a'bd$
$F_3 = bc + ab'c' + abd$

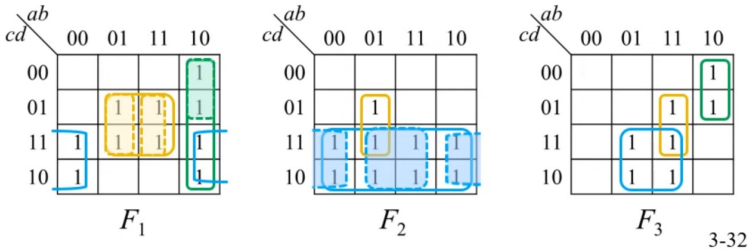
<Ans.> Approach 2

- Minimize the total # of rows in the PLA table

$F_1 = a'bd + abd + ab'c' + b'c$
$F_2 = a'bd + b'c + bc$
$F_3 = abd + ab'c' + bc$

$a'bd$, abd , $ab'c'$, $b'c$, $b'c$

$\Rightarrow$ 5 different product terms $\Rightarrow$ 4×5×3 PLA



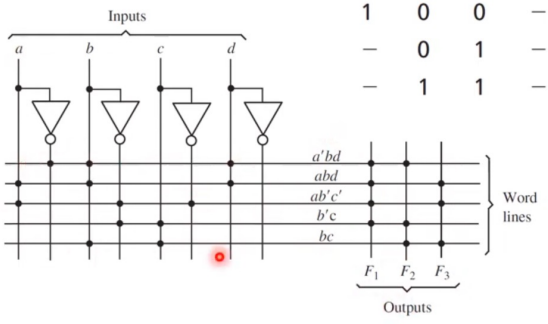
$F_1 = a'bd + abd + ab'c' + b'c$
$F_2 = a'bd + b'c + bc$
$F_3 = abd + ab'c' + bc$

$a'bd$, abd , $ab'c'$, $b'c$, $b'c$

- Reduced PLA table:
  - Each row represents a product term.

a	b	c	d	F_1	F_2	F_3
0	1	–	1	1	1	0
1	1	–	1	1	0	1
1	0	0	–	1	0	1
–	0	1	–	1	1	0
–	1	1	–	0	1	1

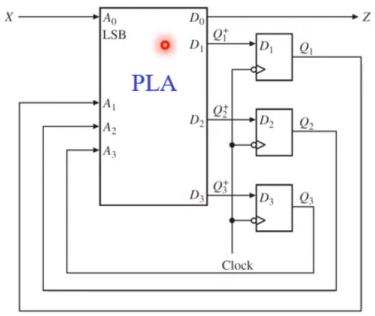
- PLA realization:



Ex 3 :

Example: a Sequential Circuit

- Realize the seq machine BCD to Excess-3-code converter using a PLA and three D flip-flops:

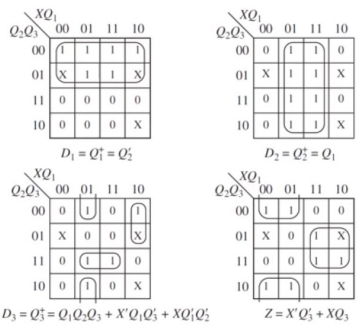


PS	NS X=0	NS X=1	Z X=0	Z X=1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	—	1	—

<Ans.>

PS	NS X=0	NS X=1	Z X=0	Z X=1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	—	1	—

$Q_1Q_2Q_3$	$Q_1^+Q_2^+Q_3^+$ X=0	$Q_1^+Q_2^+Q_3^+$ X=1	Z X=0	Z X=1
S_0 000	100	101	1	0
S_1 100	111	110	1	0
S_2 101	110	110	0	1
S_3 111	011	011	0	1
S_4 110	011	010	1	0
S_5 011	000	000	0	1
S_6 010	000	xxx	1	x
× 001	xxx	xxx	x	x



PLA table:

Product Term	Q_1	Q_2	Q_3	X	Q_1^+	Q_2^+	Q_3^+	Z
Q_2'	–	0	–	–	1	0	0	0
Q_1	1	–	–	–	0	1	0	0
$Q_1Q_2Q_3$	1	1	1	–	0	0	1	0
$Q_1Q_3'X'$	1	–	0	0	0	0	1	0
$Q_1'Q_2'X$	0	0	–	1	0	0	1	0
$Q_3'X'$	–	–	0	0	0	0	0	1
Q_3X	–	–	1	1	0	0	0	1

5 Programmable Array Logic (PAL)

- Less expensive than PLA as only AND array is programmable.
- # (AND terms) that feeds each output OR gate is fixed and limited.
  - AND terms cannot be shared among two or more OR gates.
- Design Process
  1° Simplify logic equations.
  2° If # (AND terms) in a simplified function is too large, may be forced to choose a PAL w/ more gate inputs and less outputs.

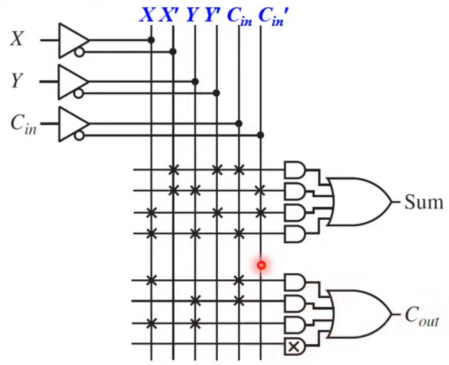
Ex 1.

Example: a Full Adder

- Implement a full adder using PAL:

$$\text{Sum} = X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in}$$

$$C_{out} = XC_{in} + YC_{in} + XY$$

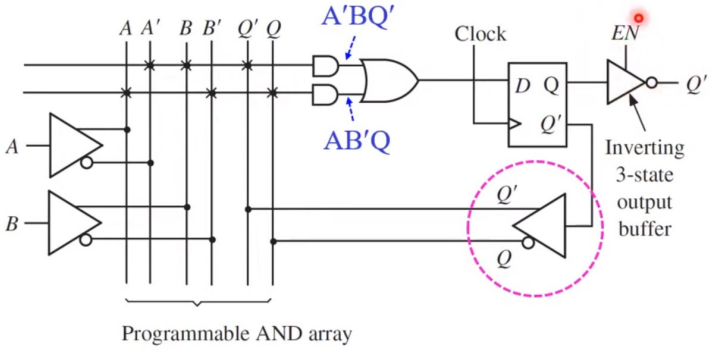


Ex 2.

Example

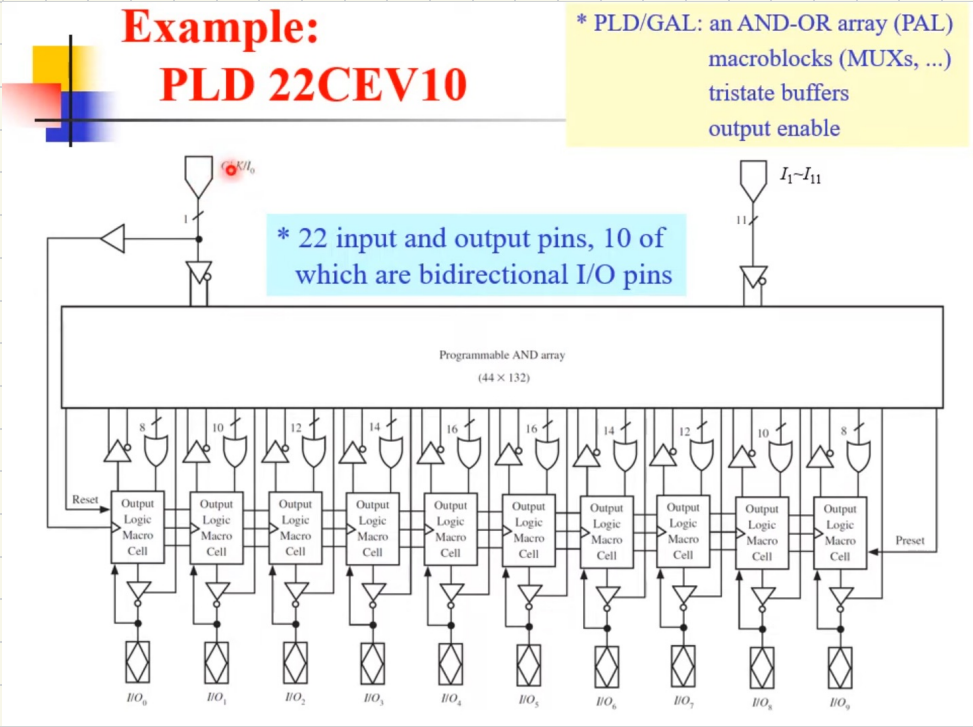
- Realize the following next-state equation:

$$Q^+ = D = A'BQ' + AB'Q$$

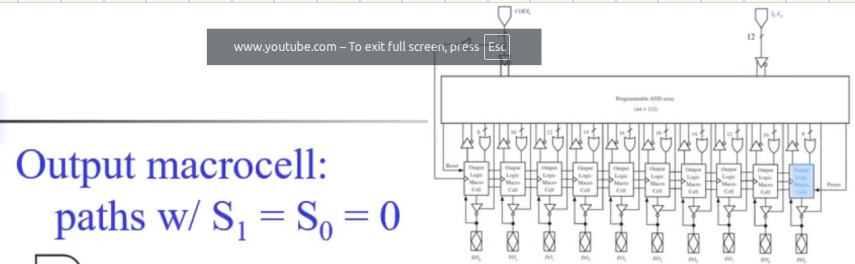


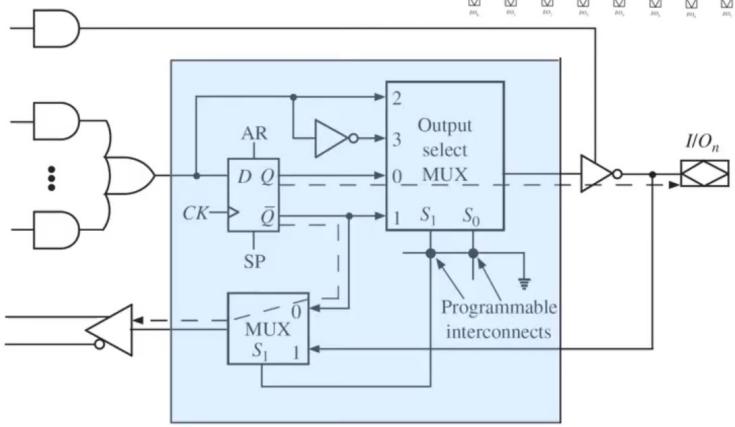
6 Generic Array Logic (GAL)

- flash erasable / reprogrammable PAL
  - consists of:
    - (i) PAL
    - (ii) macroblocks: contains muxes and some additional programmable logic
- tristate buffers at the output, output enable



- Output macrocell: paths w/ $S_1 = S_0 = 0$





(a) Paths with $S_1 = S_0 = 0$