UNIT 9
MULTIPLEXERS, DECODERS & PROGRAMMABLE DEVICES

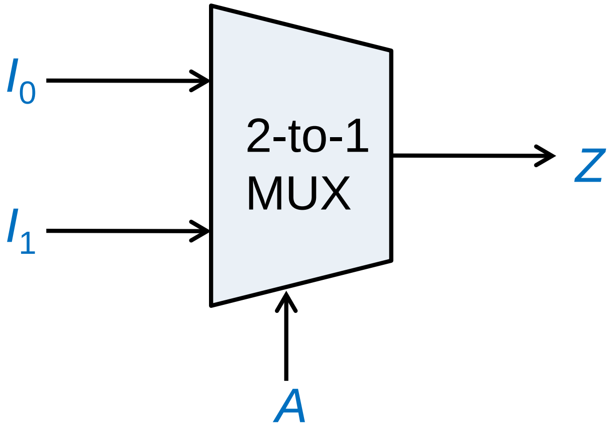
Spring 2011

Multiplexers, Decoders & PLDs

- **Contents**
  - Multiplexers
  - Three-state buffers
  - Decoders and encoders
  - Read-only memories
  - Programmable logic devices
  - Complex programmable logic devices
  - Field programmable gate arrays
- **Reading**
  - Unit 9

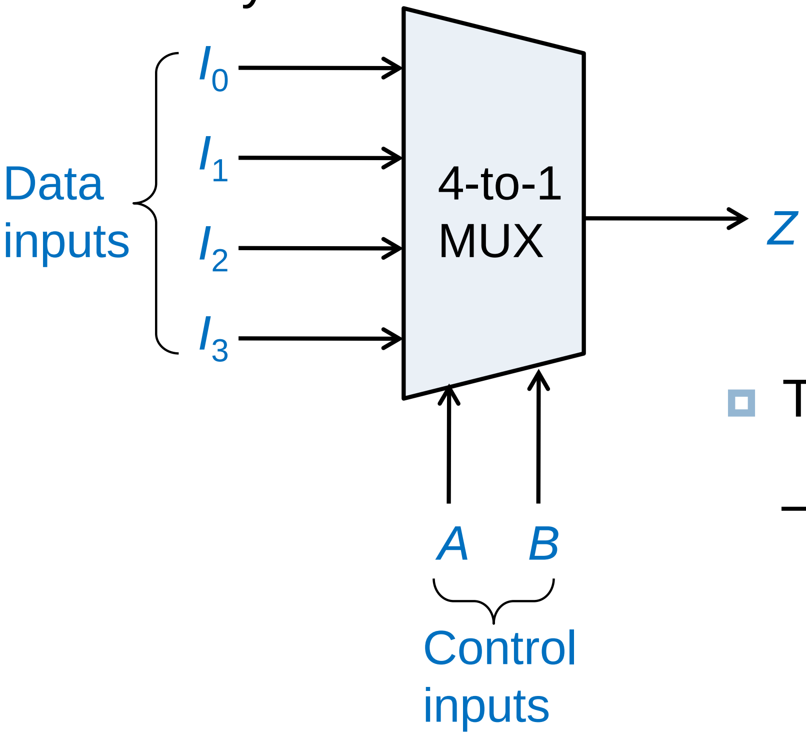
Multiplexers

MUX

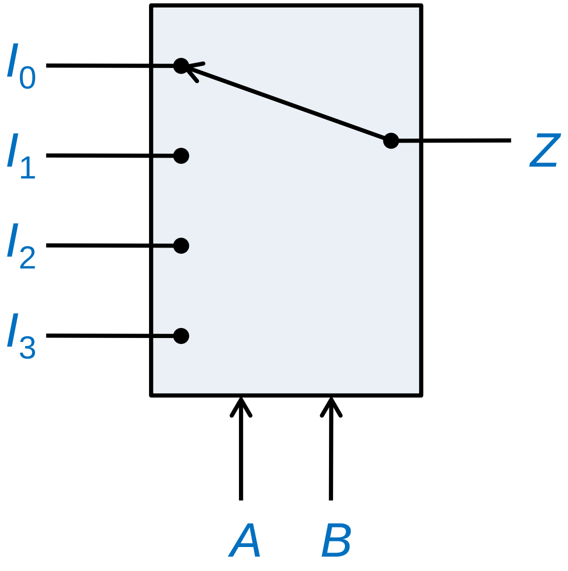


Multiplexers (1/3)

- **A multiplexer (data selector, MUX)**
  - Uses the control inputs to select one of the data inputs and connects it to the output terminal
  - One combination of control inputs corresponds to one data input
  - Symbol



  - Switch



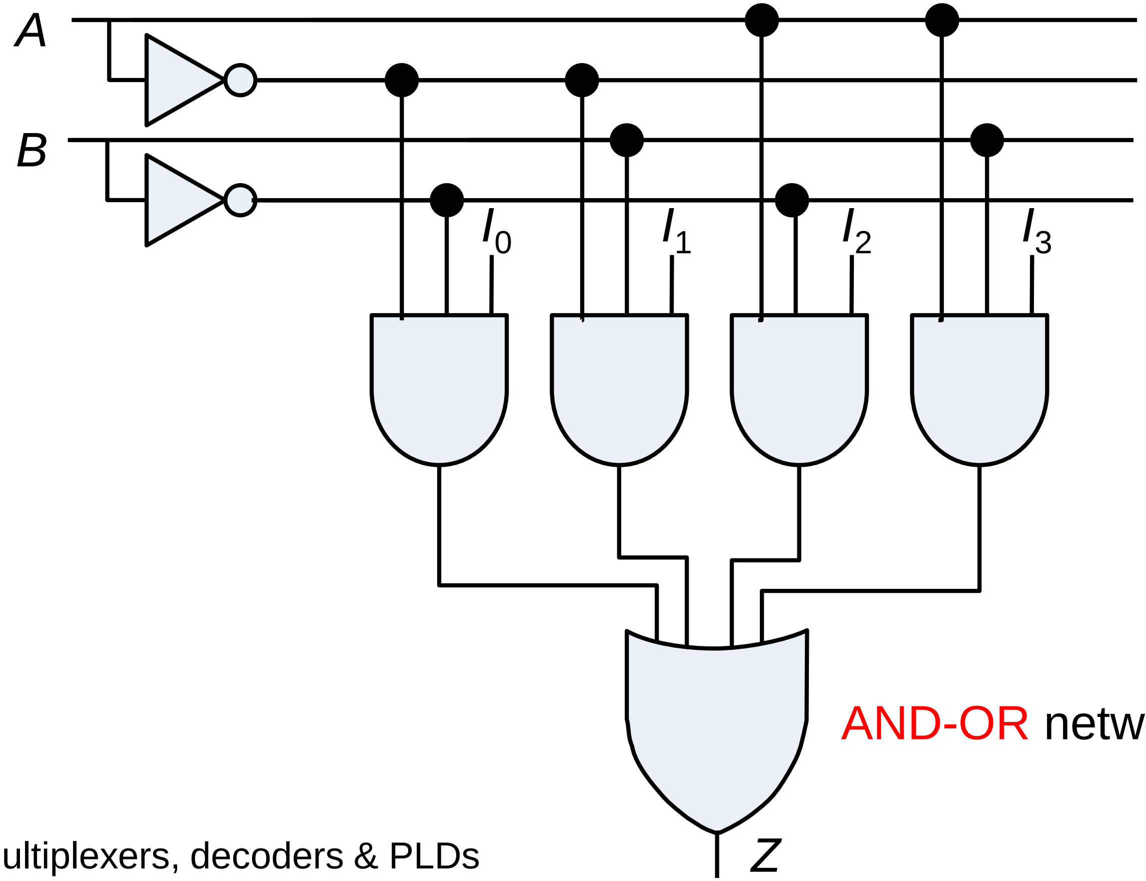
  - Truth table

A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Multiplexers (2/3)

- **Logic equation of 4-to-1 MUX**
  - $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$
    $= m_0I_0 + m_1I_1 + m_2I_2 + m_3I_3$

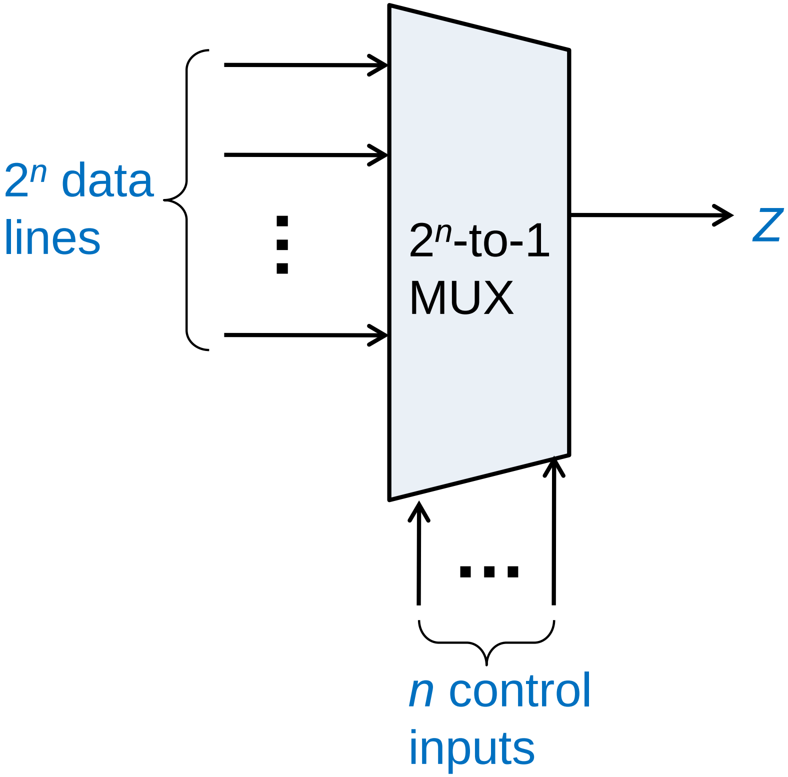
A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



AND-OR network

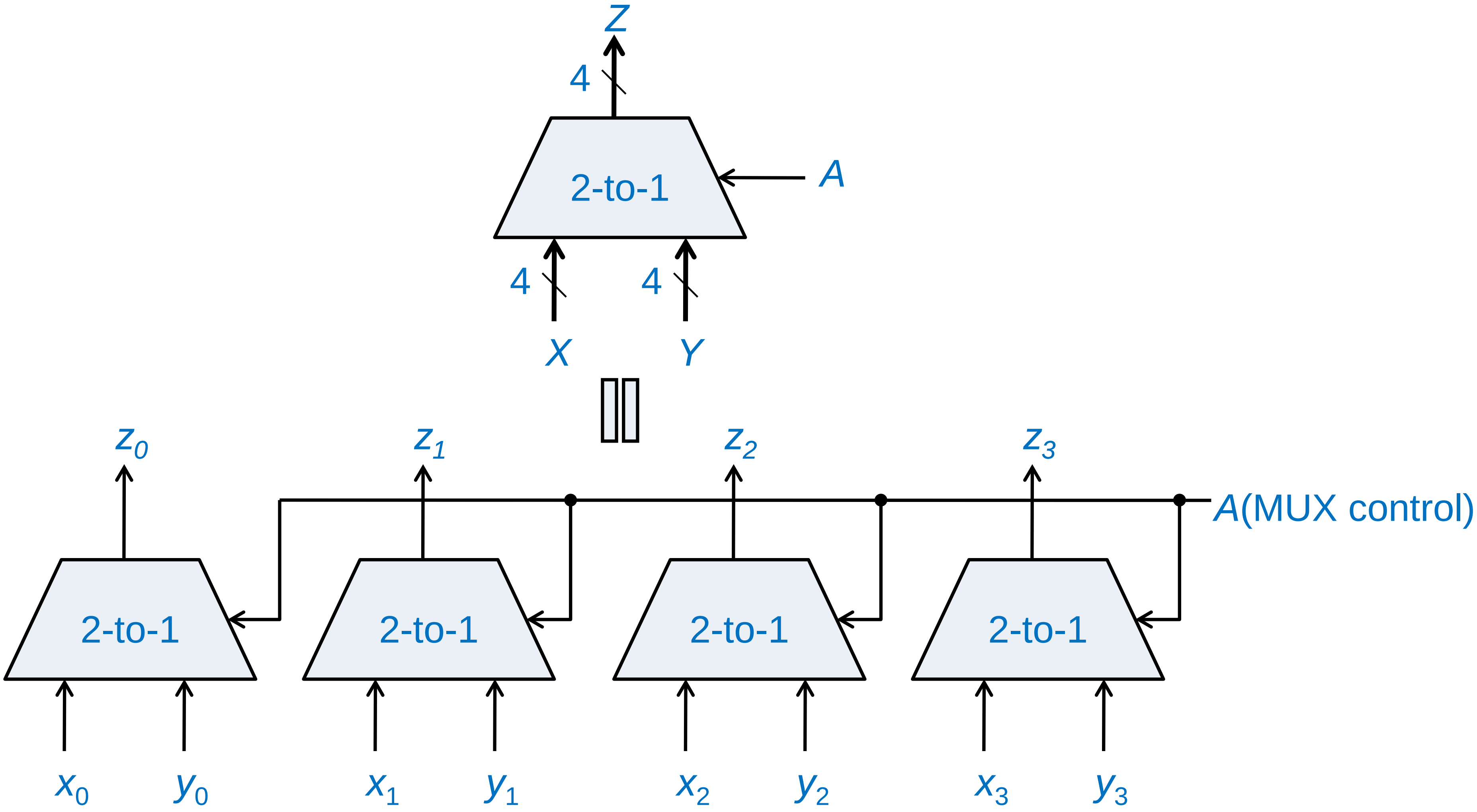
Multiplexers (3/3)

- **The general equation for *n* control inputs**
  - $Z = m_0 I_0 + m_1 I_1 + \ldots + m_i I_i + \ldots + m_{2^n-1} I_{2^n-1} = \sum_{k=0..2^n-1} m_k I_k$



Application: Quad Multiplexer

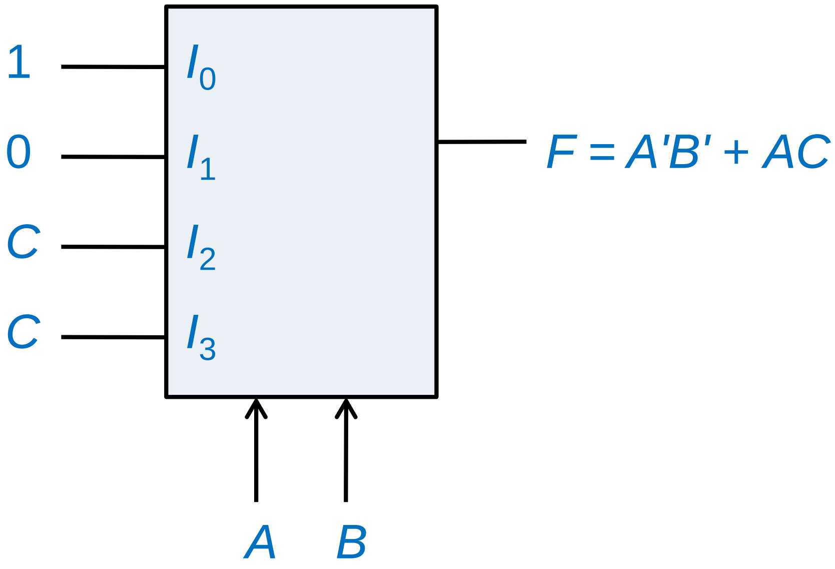
- □ **Select one of two 4-bit data words**



Application: Realizing Combinational Logic

□ **e.g., realize a 3-variable function by a 4-to-1 MUX**

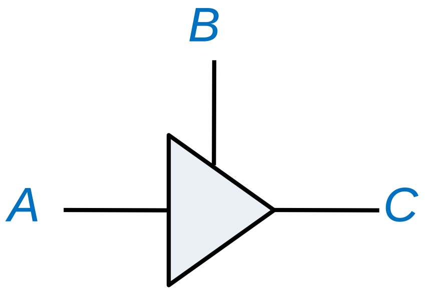
$$\begin{aligned} F(A, B, C) &= A'B' + AC \\ &= A'B' + AC(B + B') \\ &= 1\bullet A'B' + 0\bullet A'B + C\bullet AB' + C\bullet AB \end{aligned}$$



A	*B*	*F*
0	0	1
0	1	0
1	0	*C*
1	1	*C*

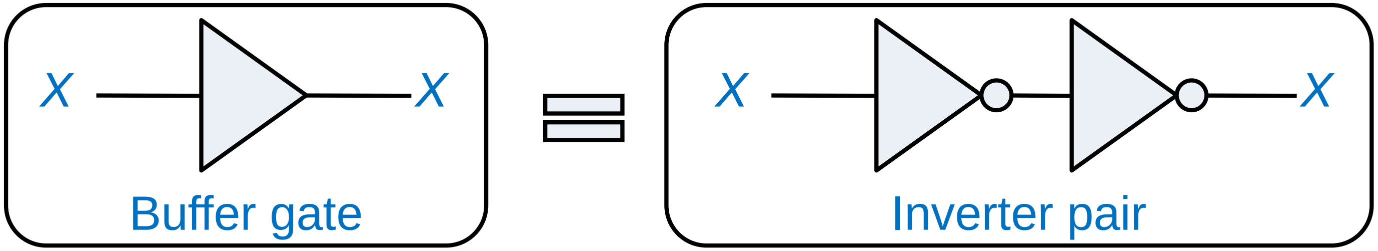
Three-State Buffers

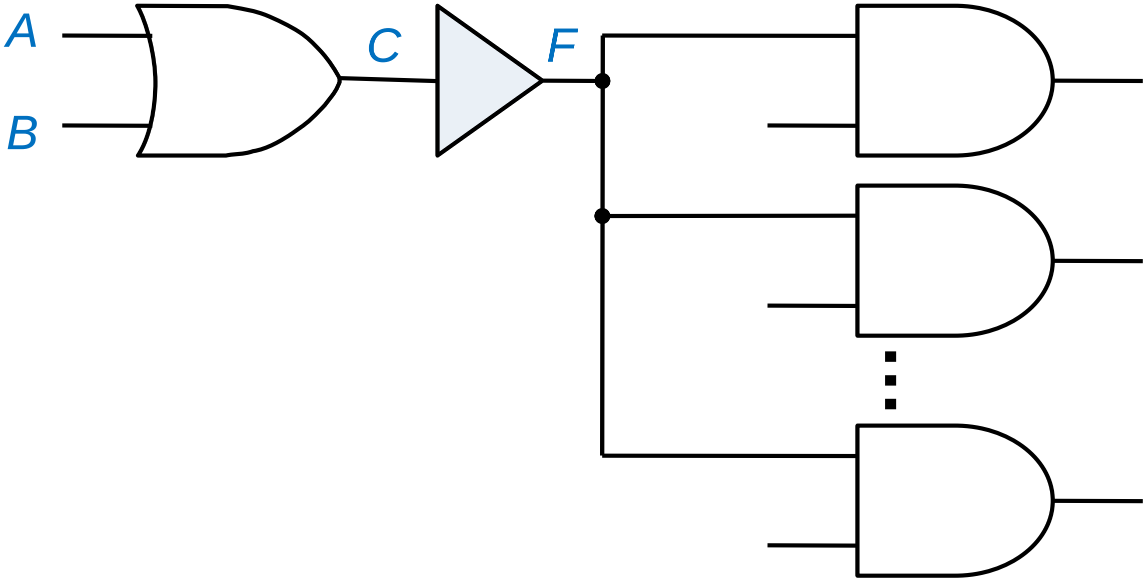
Tri-state buffers



Buffers

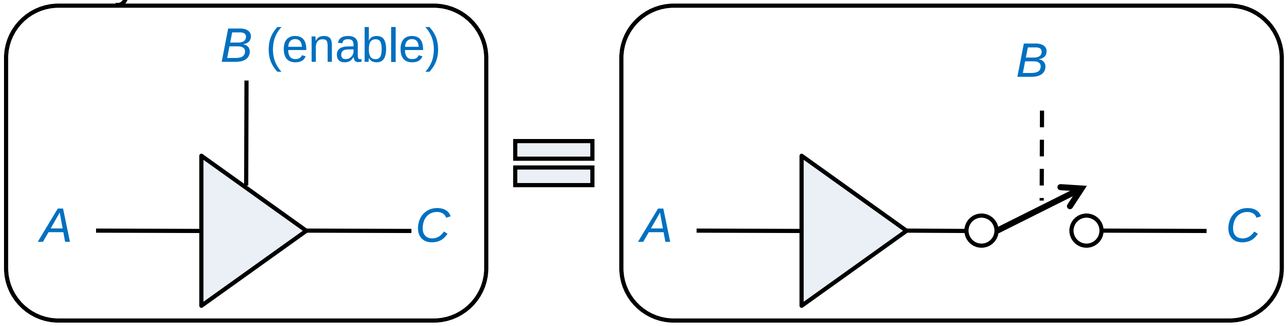
- **Buffers: increase the driving capability of a gate output**
  - Symbol





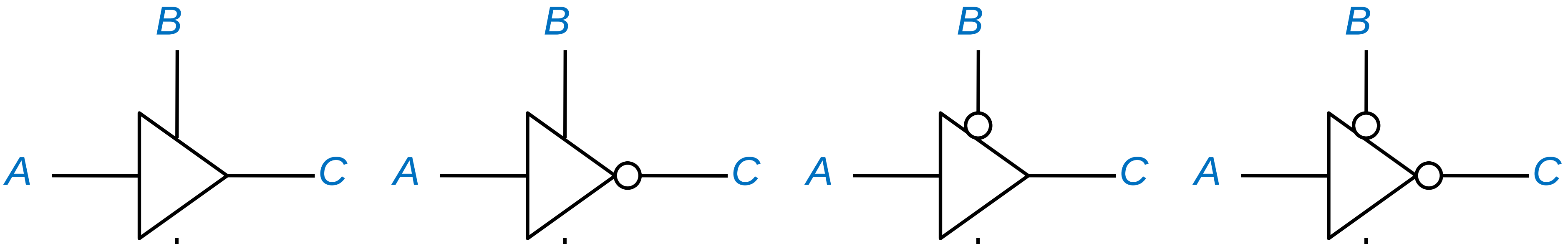
Three-State Buffers

- **Three-state (tri-state) buffers: permits gate outputs to be connected together**
  - Symbol



B	C
0	Z
1	A

  - 4 variants



B	C
0	Z
1	A

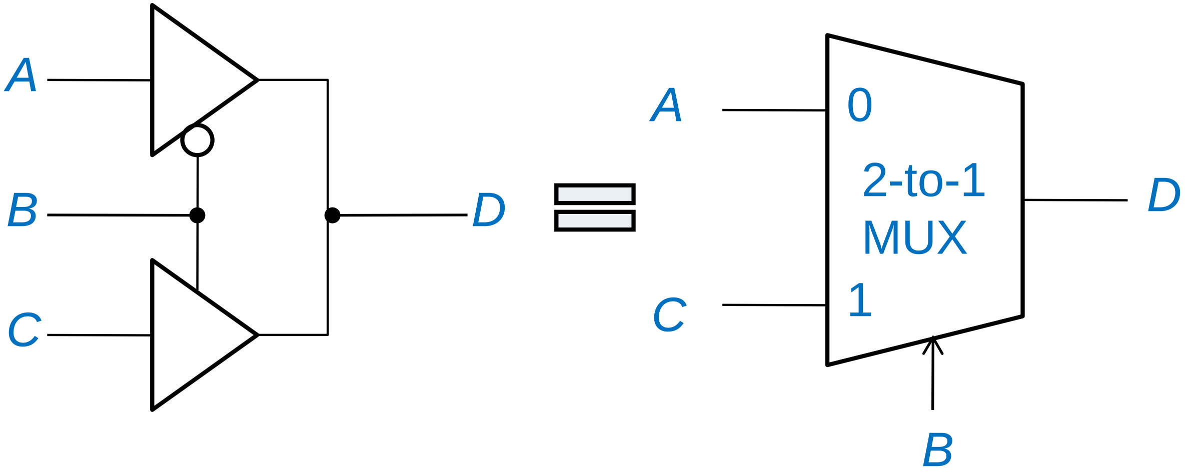
B	C
0	Z
1	A′

B	C
0	A
1	Z

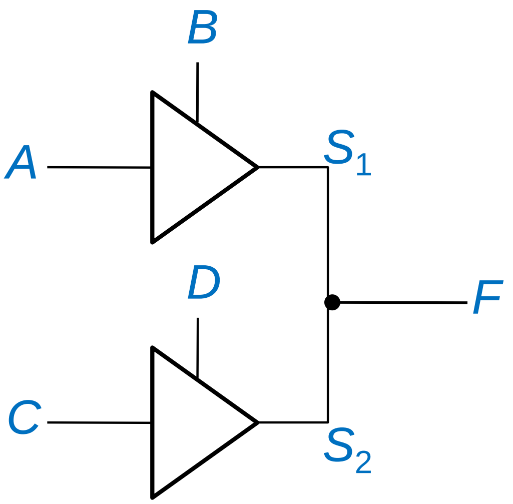
B	C
0	A′
1	Z

Application (1/2)

- Data selection



- Circuit with 2 tri-state buffers

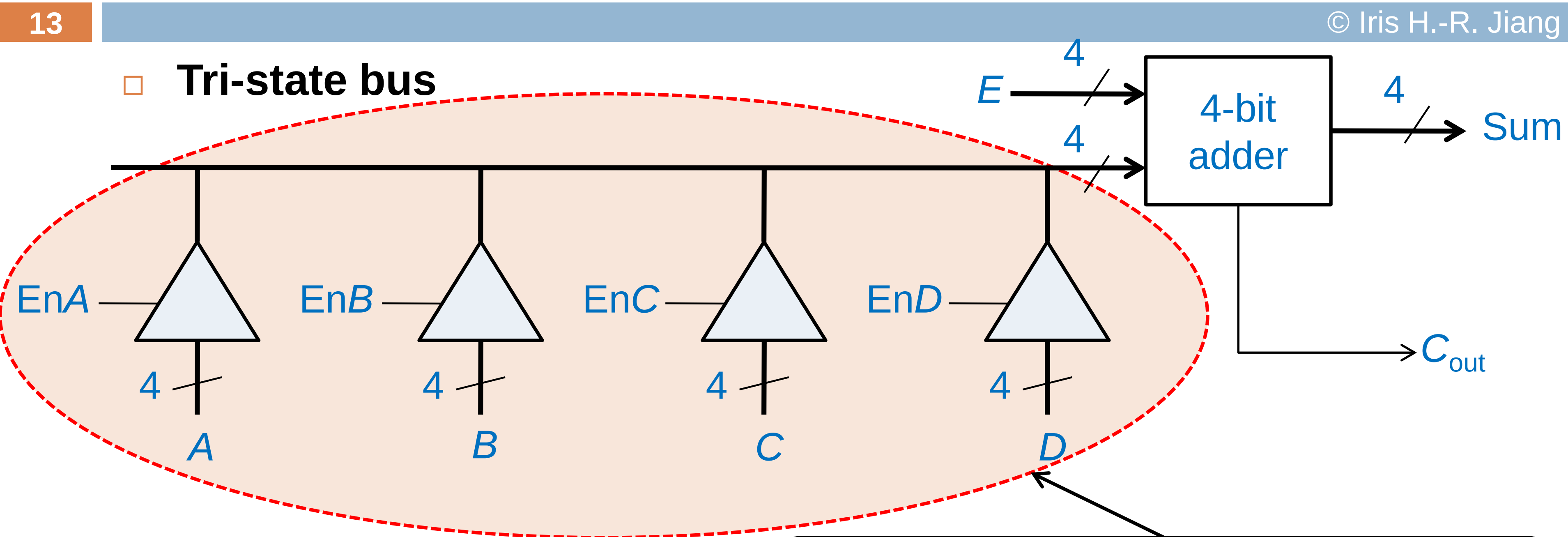


outputs of buffers

S_1 \ S_2	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

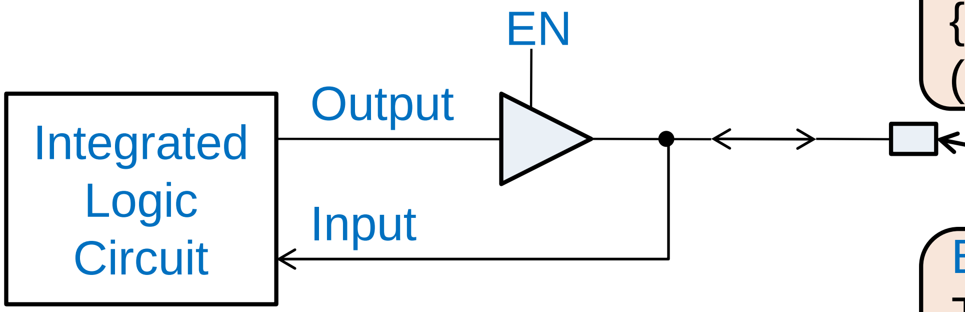
Application (2/2)

- □ **Tri-state bus**



Key:
{En*A*, En*B*, En*C*, En*D*} must be exclusive
(Only 1 active at a time)

- □ **Bi-directional I/O pin**

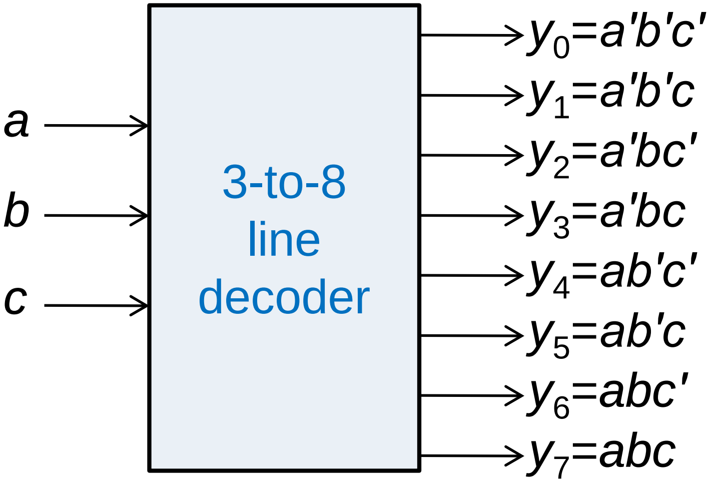


Bi-directional input-output pin:
The same pin can be used as an input pin and as an output pin, but not both at the same time

Decoders and Encoders

Decoders

- **Decoder: generates all of the minterms of input variables**
  - Exactly one output line corresponds to one input combination
  - Symbol



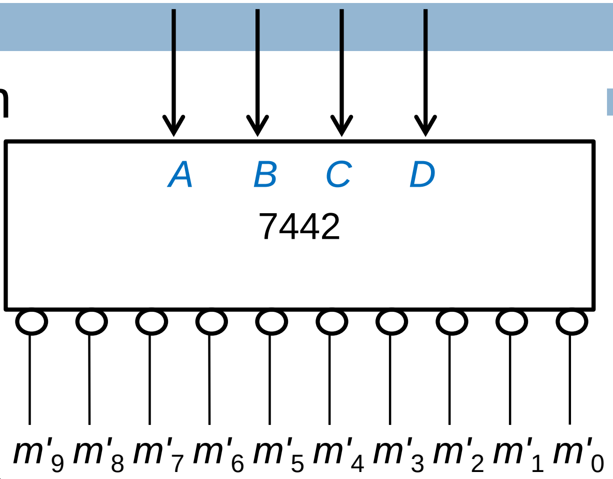
  - Truth table

	a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
m_0	0	0	0	1	0	0	0	0	0	0	0
m_1	0	0	1	0	1	0	0	0	0	0	0
m_2	0	1	0	0	0	1	0	0	0	0	0
m_3	0	1	1	0	0	0	1	0	0	0	0
m_4	1	0	0	0	0	0	0	1	0	0	0
m_5	1	0	1	0	0	0	0	0	1	0	0
m_6	1	1	0	0	0	0	0	0	0	1	0
m_7	1	1	1	0	0	0	0	0	0	0	1

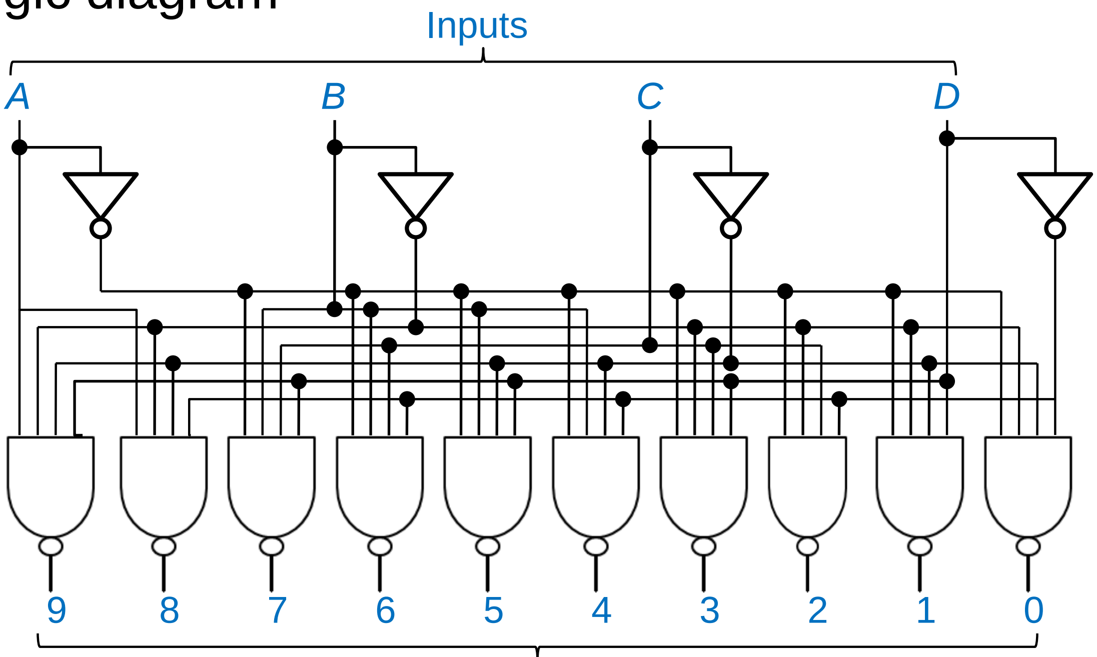
  - General n-to-2^n decoder
    - Noninverted outputs: $y_i = m_i$, i = 0 to $2^n - 1$
    - Inverted outputs: $y_i = m_i' = M_i$, i = 0 to $2^n - 1$

Example: 4-to-10 Line Decoder

- Block diagram



- Logic diagram

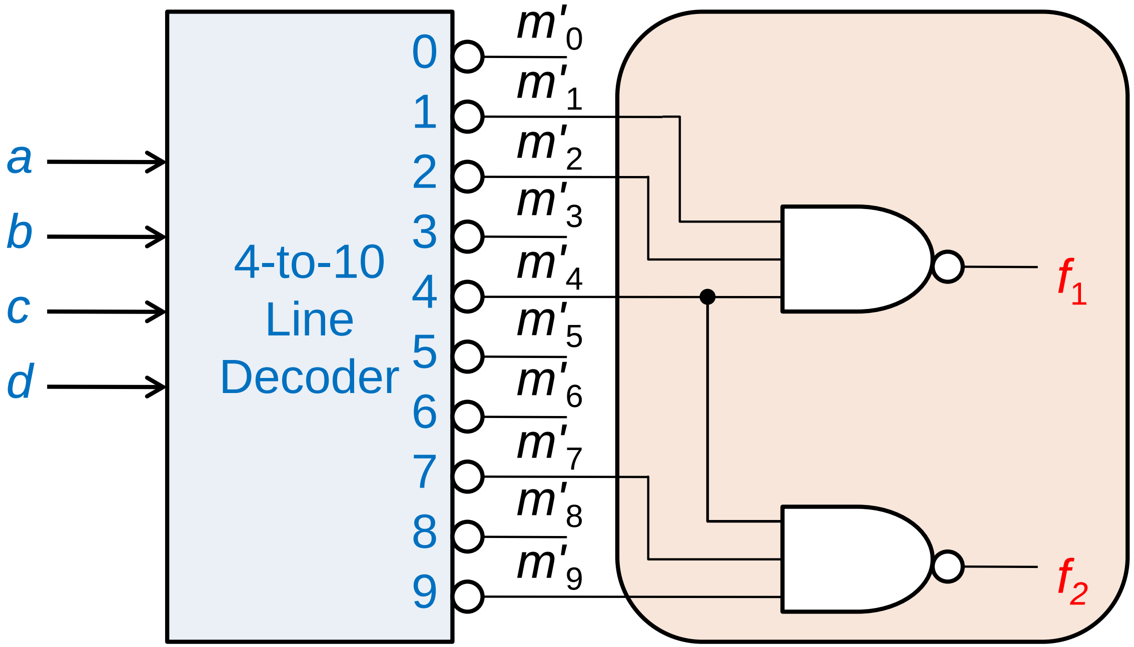


- Truth table

BCD input	Decimal output
ABCD	0123456789
0000	0111111111
0001	1011111111
0010	1101111111
0011	1110111111
0100	1111011111
0101	1111101111
0110	1111110111
0111	1111111011
1000	1111111101
1001	1111111110
1010	1111111111
1011	1111111111
1100	1111111111
1101	1111111111
1110	1111111111
1111	1111111111

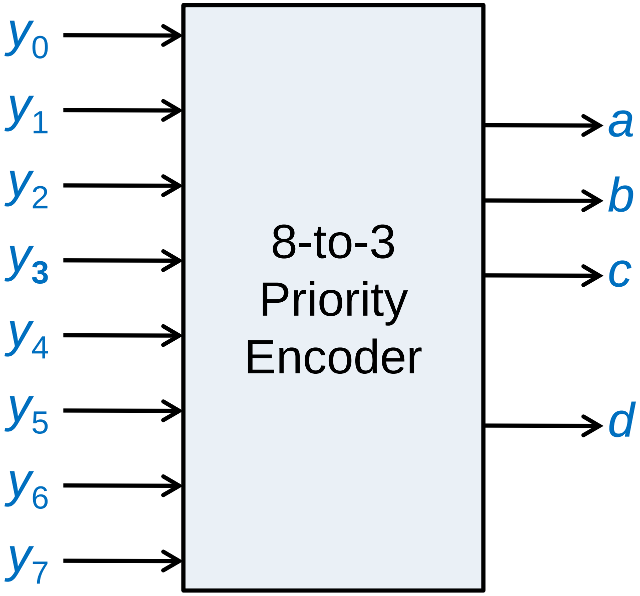
Application: Realizing *n*-Variable Functions

- **Exactly one output line corresponds to one minterm**
  - ⇒ Realize *n*-variable functions by ORing selected minterm outputs from a decoder
  - e.g., $f_1 = \Sigma\, m(1, 2, 4)$, $f_2 = \Sigma\, m(4, 7, 9)$



Encoders

- **Encoder: performs the inverse function of a decoder**
  - If $y_i = 1$, *abc* outputs represent a binary number equal to *i*
  - If more than one input can be 1 at a time, use a priority scheme



	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
m_0	1	0	0	0	0	0	0	0	0	0	0	1
m_1	X	1	0	0	0	0	0	0	0	0	1	1
m_2	X	X	1	0	0	0	0	0	0	1	0	1
m_3	X	X	X	1	0	0	0	0	0	1	1	1
m_4	X	X	X	X	1	0	0	0	1	0	0	1
m_5	X	X	X	X	X	1	0	0	1	0	1	1
m_6	X	X	X	X	X	X	1	0	1	1	0	1
m_7	X	X	X	X	X	X	X	1	1	1	1	1

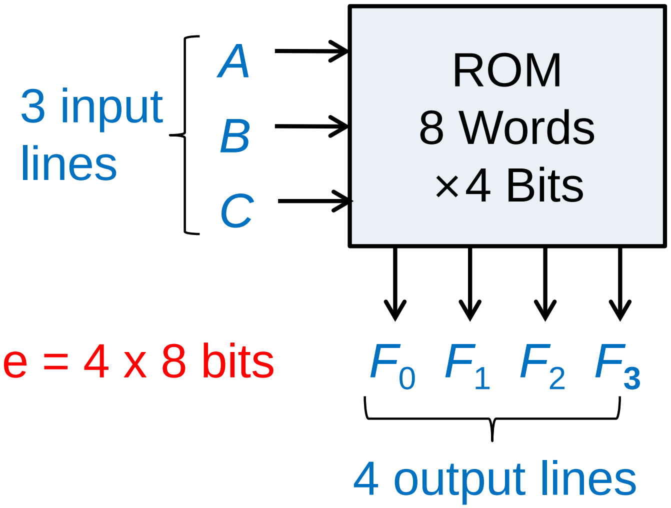
No signal

Read-Only Memories

ROM

Read-Only Memories (1/3)

- **A read-only memory (ROM): stores an array of binary data**
  - Stored data cannot be changed
  - e.g., 8-word × 4-bit ROM: each word is 4-bit, total 8 words
    - Input: *ABC*: 3-bit lines can index 2^3 values (0~7 address)
    - Output: $F_0F_1F_2F_3$: each output pattern is called a word



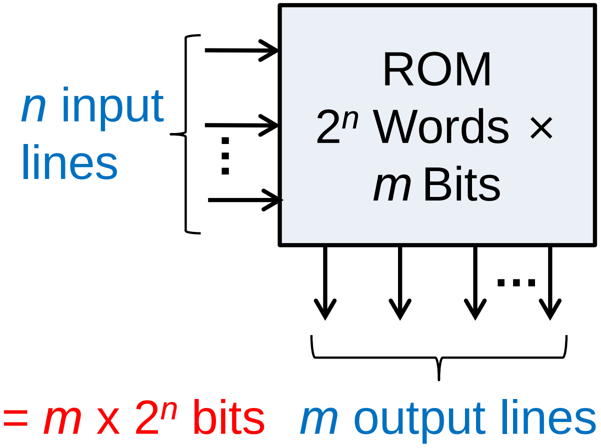
Size = 4 x 8 bits

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	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

typical data stored in ROM (2^3 words of 4 bits each)

ROM (2/3)

- **Generalized form: 2^n Words × *m* Bits ROM (*n*-inputs *m*-outputs)**

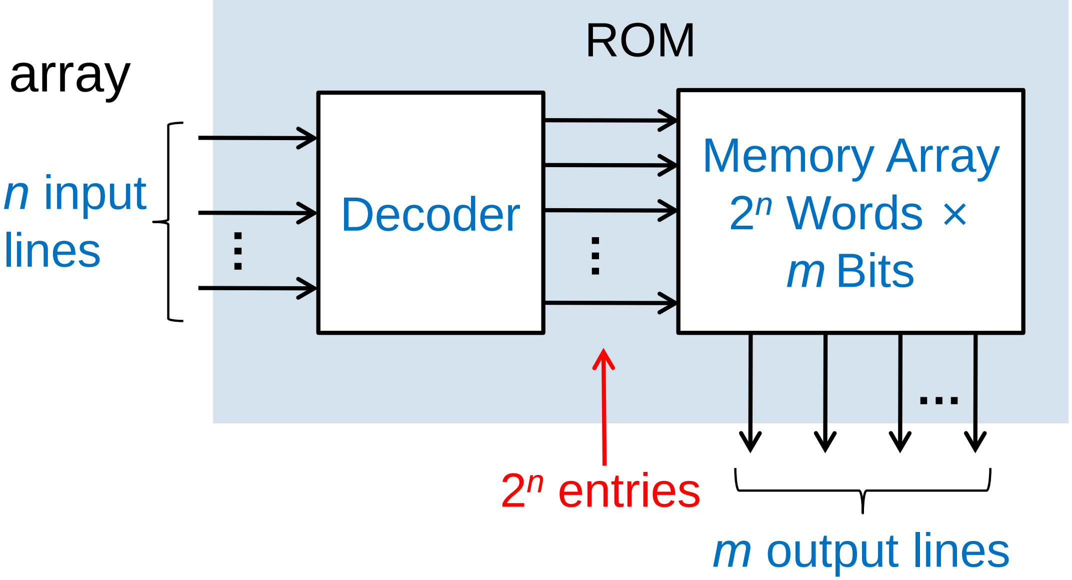


Size = $m \times 2^n$ bits

n-bit input address	*m*-bit output data
…	…

typical data stored in ROM (2^n words of *m* bits each)

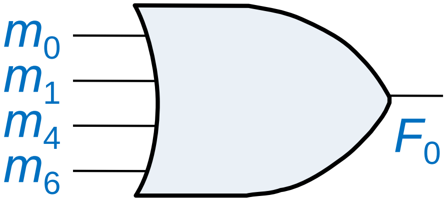
- **Basic ROM structure**
  - A decoder + memory array

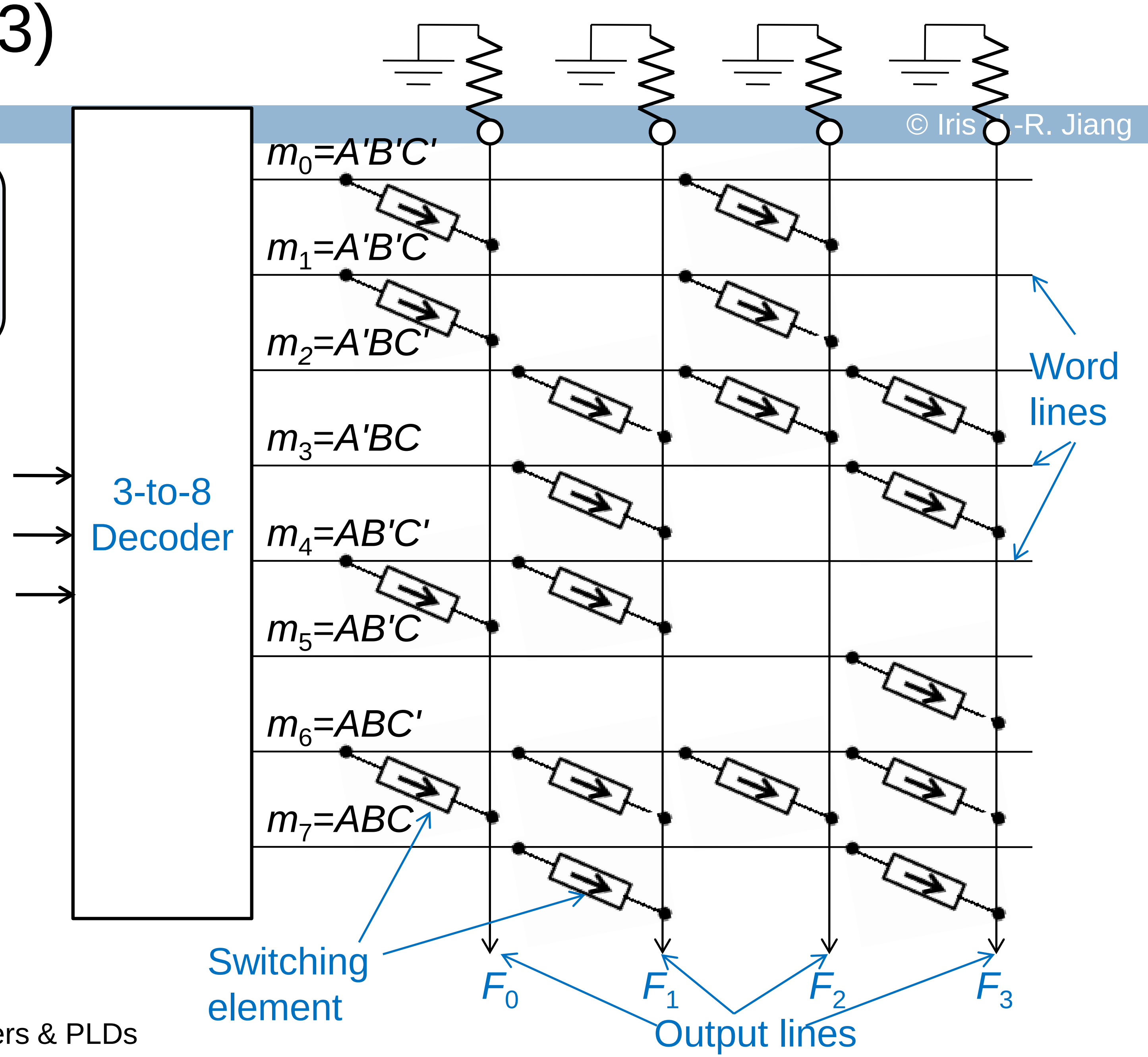


ROM (3/3)

The contents of a ROM are usually specified by a truth table

ABC	$F_0F_1F_2F_3$
000	1 0 1 0
001	1 0 1 0
010	0 1 1 1
011	0 1 0 1
100	1 1 0 0
101	0 0 0 1
110	1 1 1 1
111	0 1 0 1





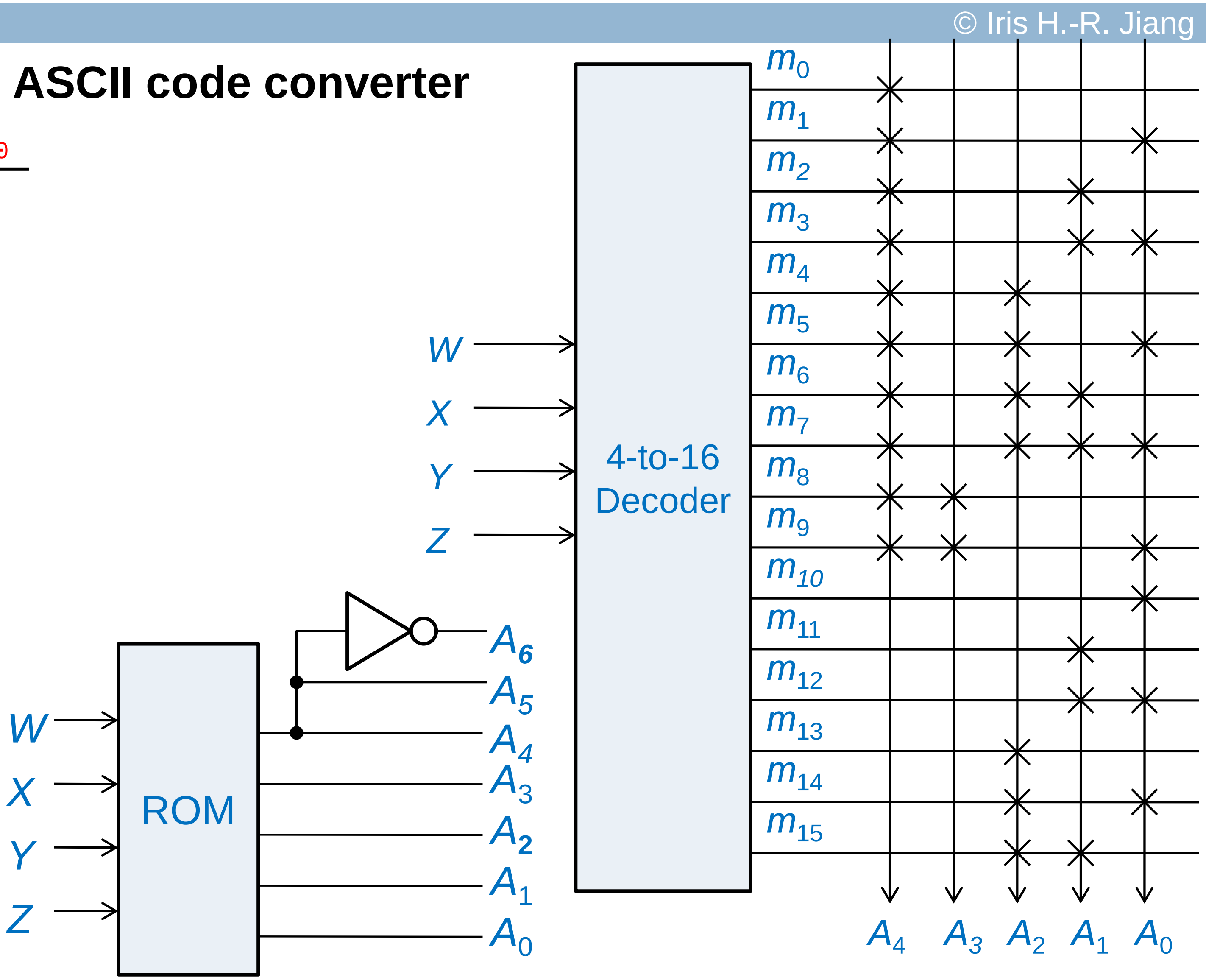
Application: Multi-Output Combinational Ckts

- □ **e.g., hex to ASCII code converter**

Hex	$A_6A_5A_4A_3A_2A_1A_0$
0	011 0000
1	011 0001
2	011 0010
3	011 0011
4	011 0100
5	011 0101
6	011 0110
7	011 0111
8	011 1000
9	011 1001
A	100 0001
B	100 0010
C	100 0011
D	100 0100
E	100 0101
F	100 0110

$A_5 = A_4$

$A_6 = A'_4$



Three Common Types of ROMs

- **Mask-programmable ROMs**
  - Use mask to program
    - Include/omit switching elements
- **Programmable ROMs (PROMs)**
  - Program once
- **Electrically erasable programmable ROMs (EEPROMs)**
  - Can reprogram 100—1000 times

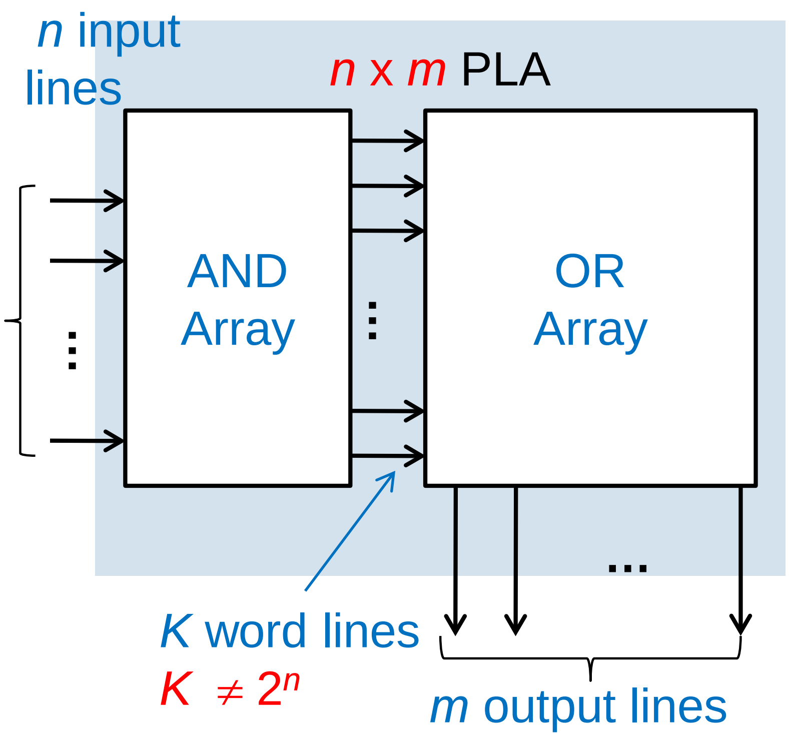
Programmable Logic Devices

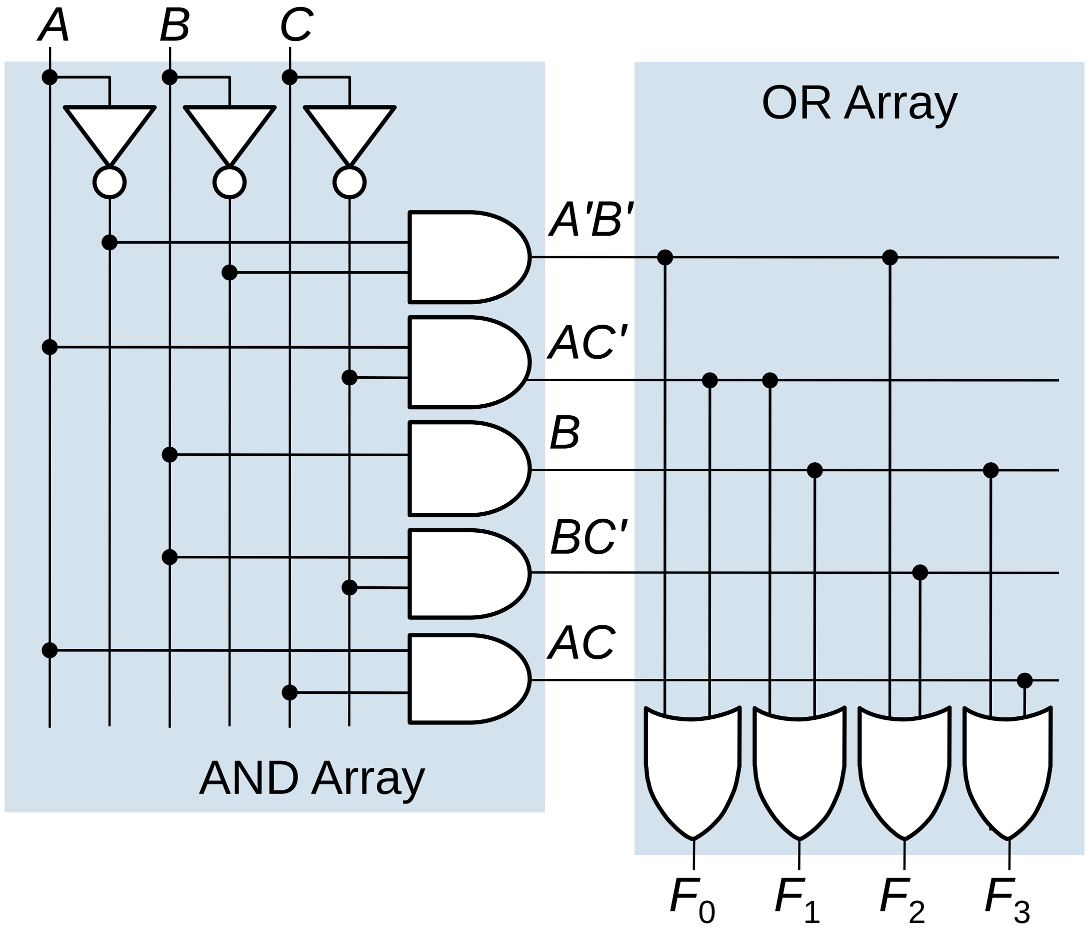
PLD

PLA / PAL

Programmable Logic Arrays (1/3)

- **Programmable logic arrays (PLAs): 2-level SOP implementation**
  - AND plane generates product terms
  - OR plane sums the product terms





PLA (2/3)

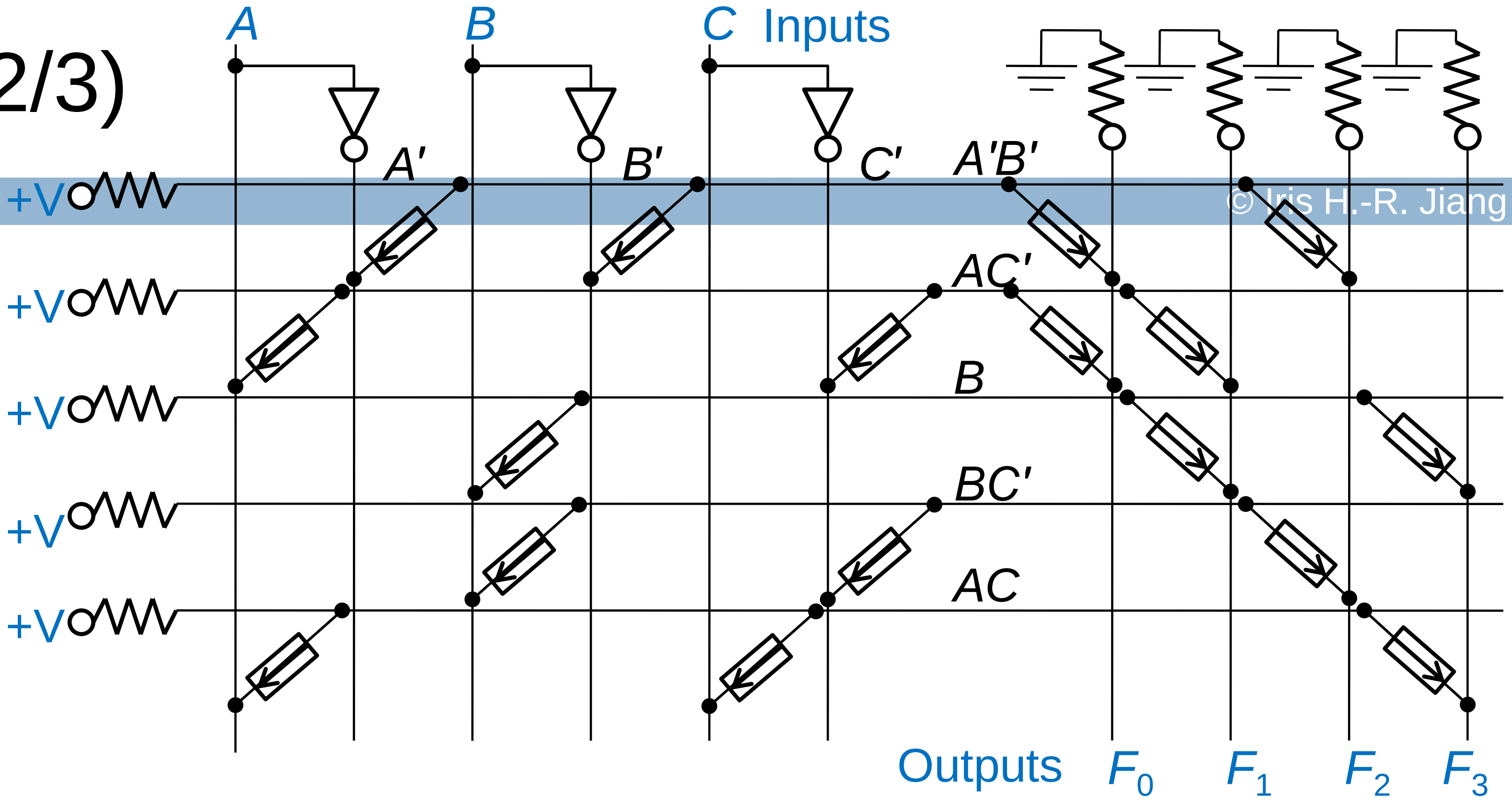
- e.g.,

$F_0 = A'B' + AC'$

$F_1 = AC' + B$

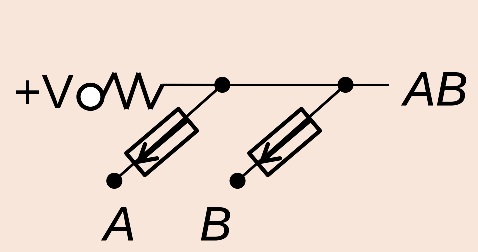
$F_2 = A'B' + BC'$

$F_3 = B + AC$

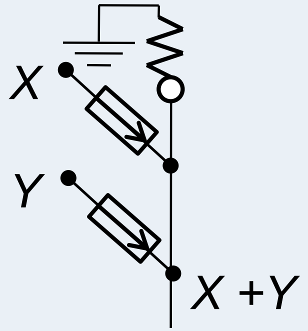


Product term	Inputs			Outputs			
	A	B	C	F_0	F_1	F_2	F_3
$A'B'$	0	0	–	1	0	1	0
AC'	1	–	0	1	1	0	0
B	-	1	–	0	1	0	1
BC'	-	1	0	0	0	1	0
AC	1	-	1	0	0	0	1

AND plane



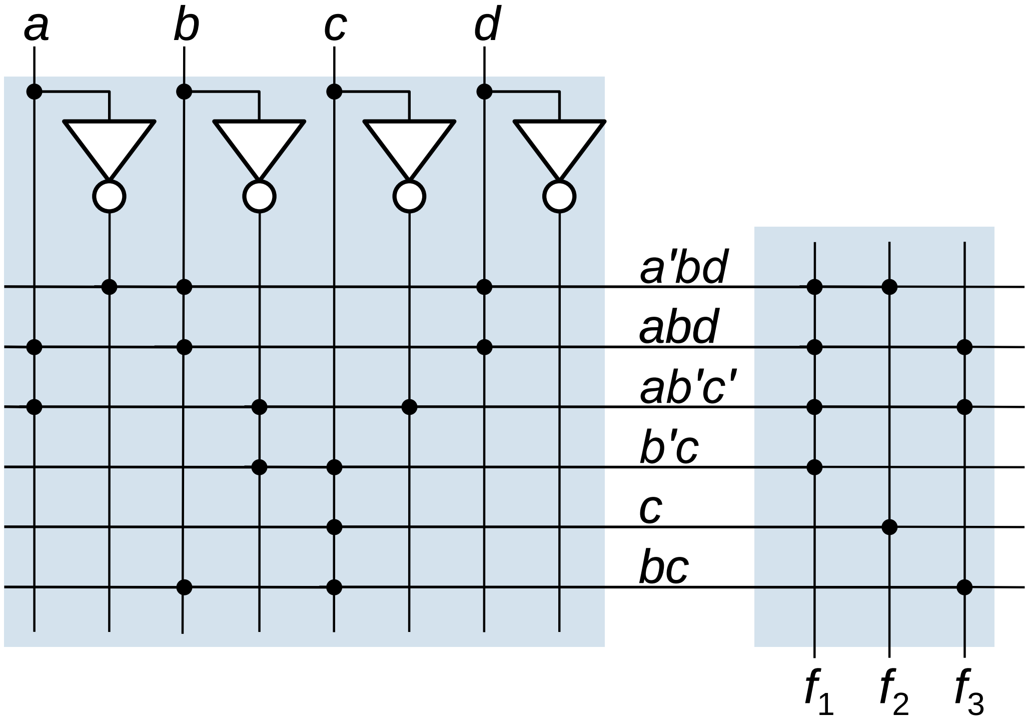
OR plane



PLA (3/3)

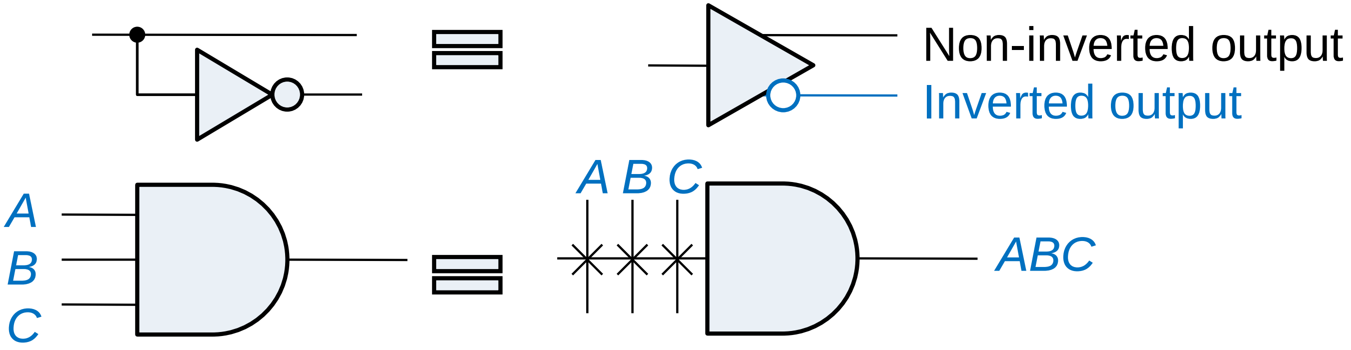
- e.g.,
  - $f_1(a, b, c, d) = \Sigma\, m(2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$
  - $f_2(a, b, c, d) = \Sigma\, m(2, 3, 5, 6, 7, 10, 11, 14, 15)$
  - $f_3(a, b, c, d) = \Sigma\, m(6, 7, 8, 9, 13, 14, 15)$
- Minimize using K-map
  - $f_1 = abd + a'bd + b'c + ab'c'$
  - $f_2 = c + a'bd$
  - $f_3 = bc + ab'c' + abd$

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	0	0
-	-	1	–	0	1	0
-	1	1	-	0	0	1



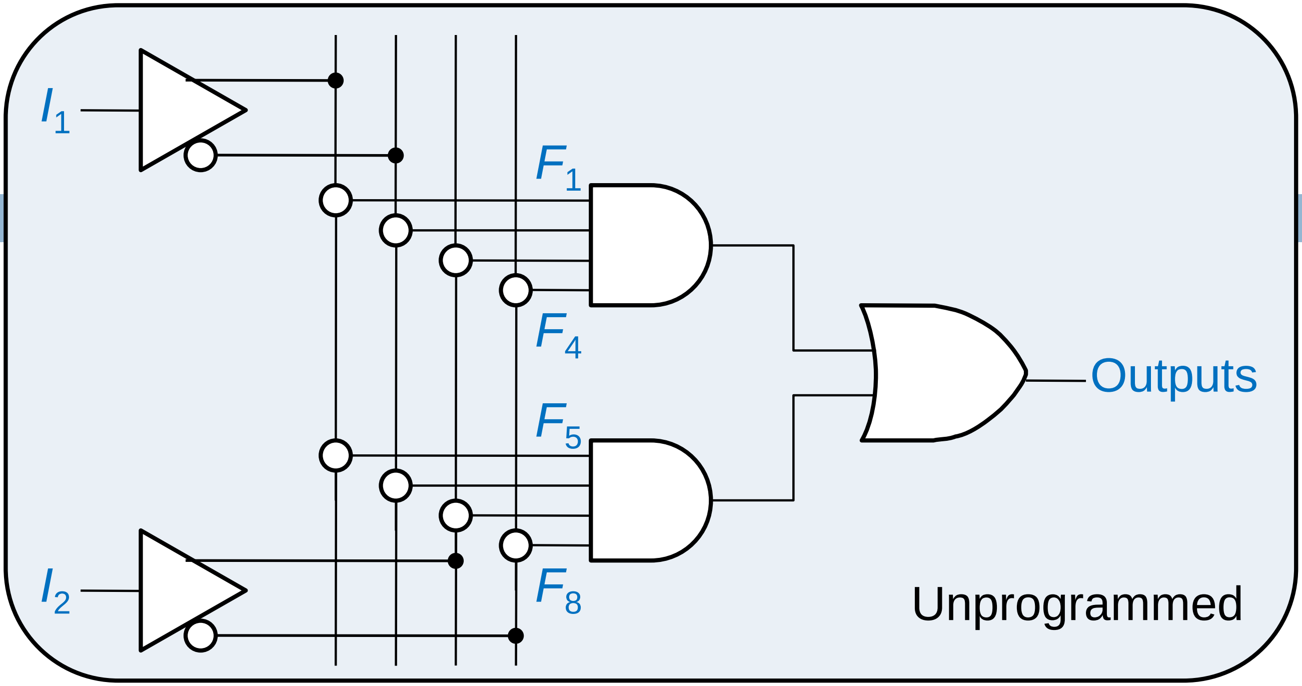
Programmable Array Logic (1/2)

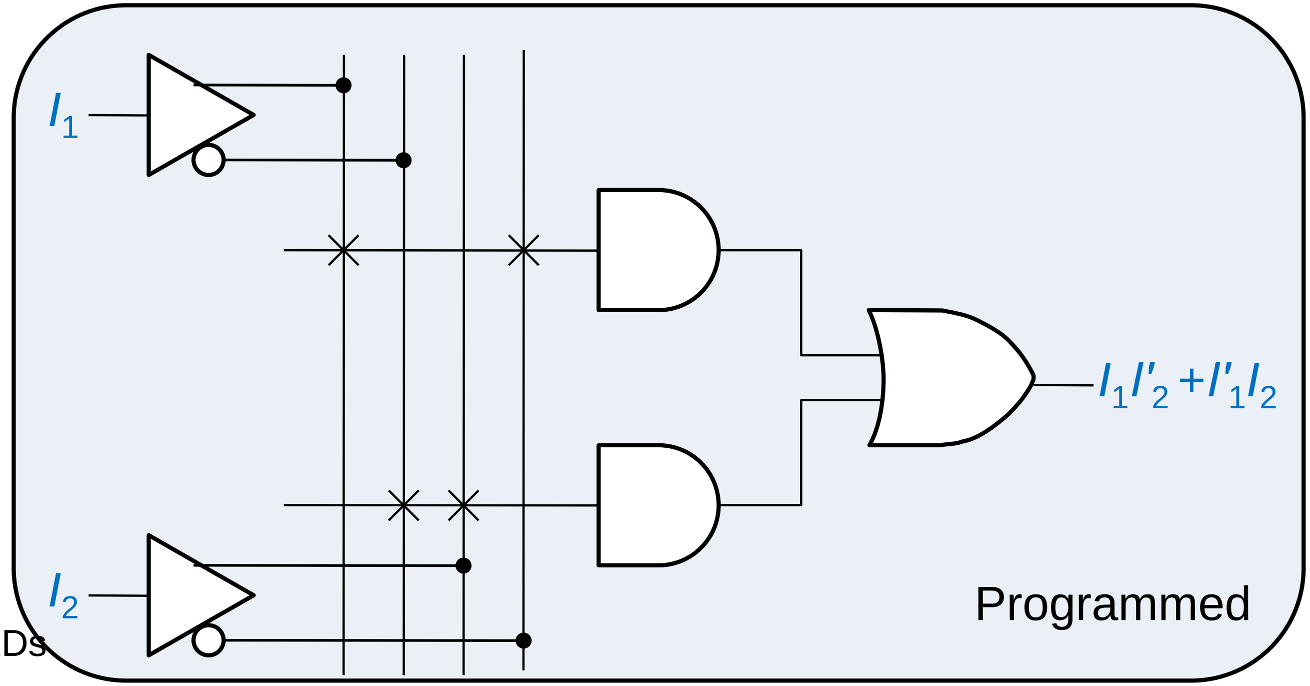
- **Programmable array logic (PAL): A special case of PLA**
  - AND array: programmable
  - OR array: fixed
  - Symbol



PAL (2/2)

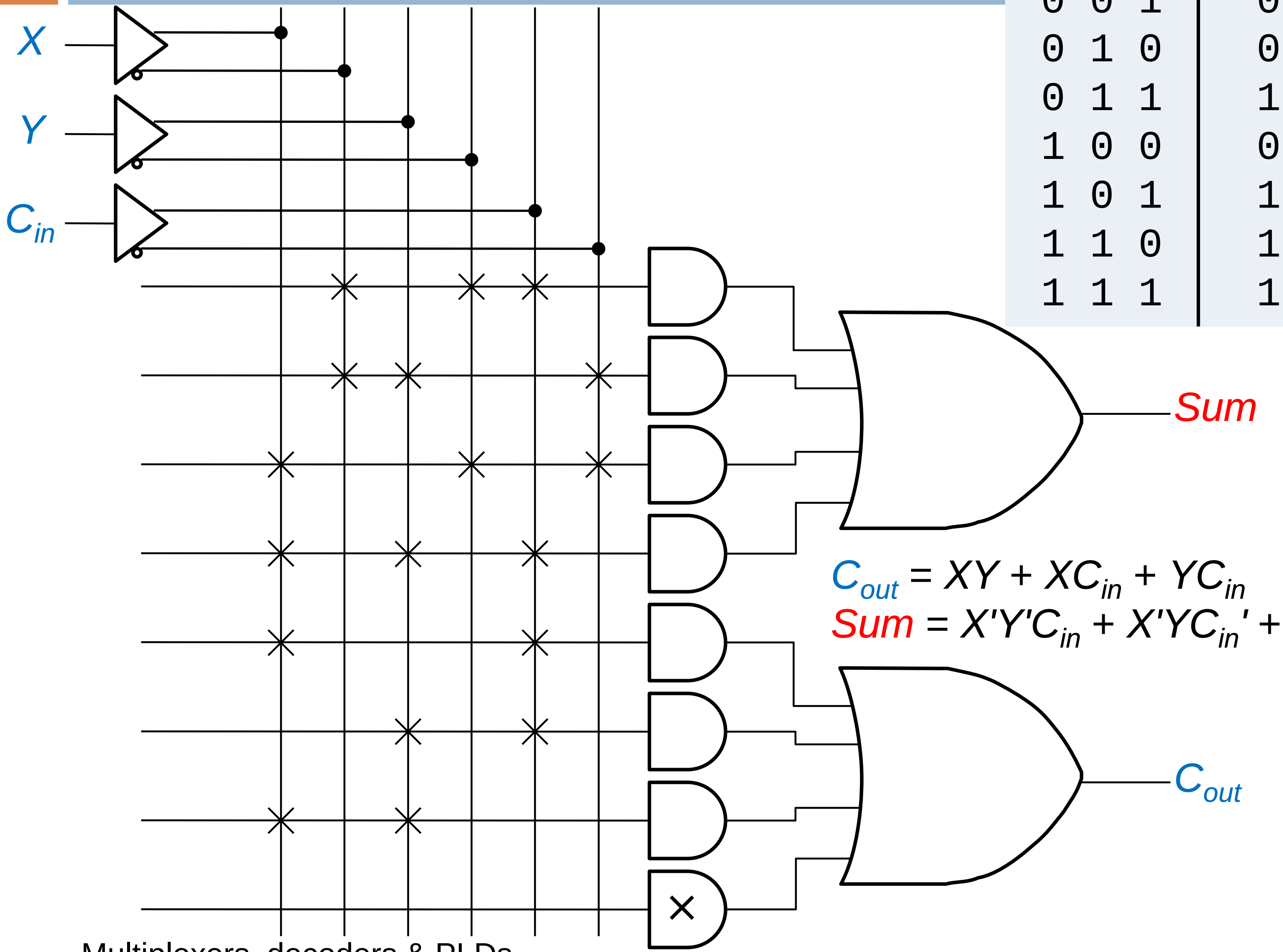
- Fixed OR array





Application: Full Adder

X	Y	C_{in}	C_{out}	Sum
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



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

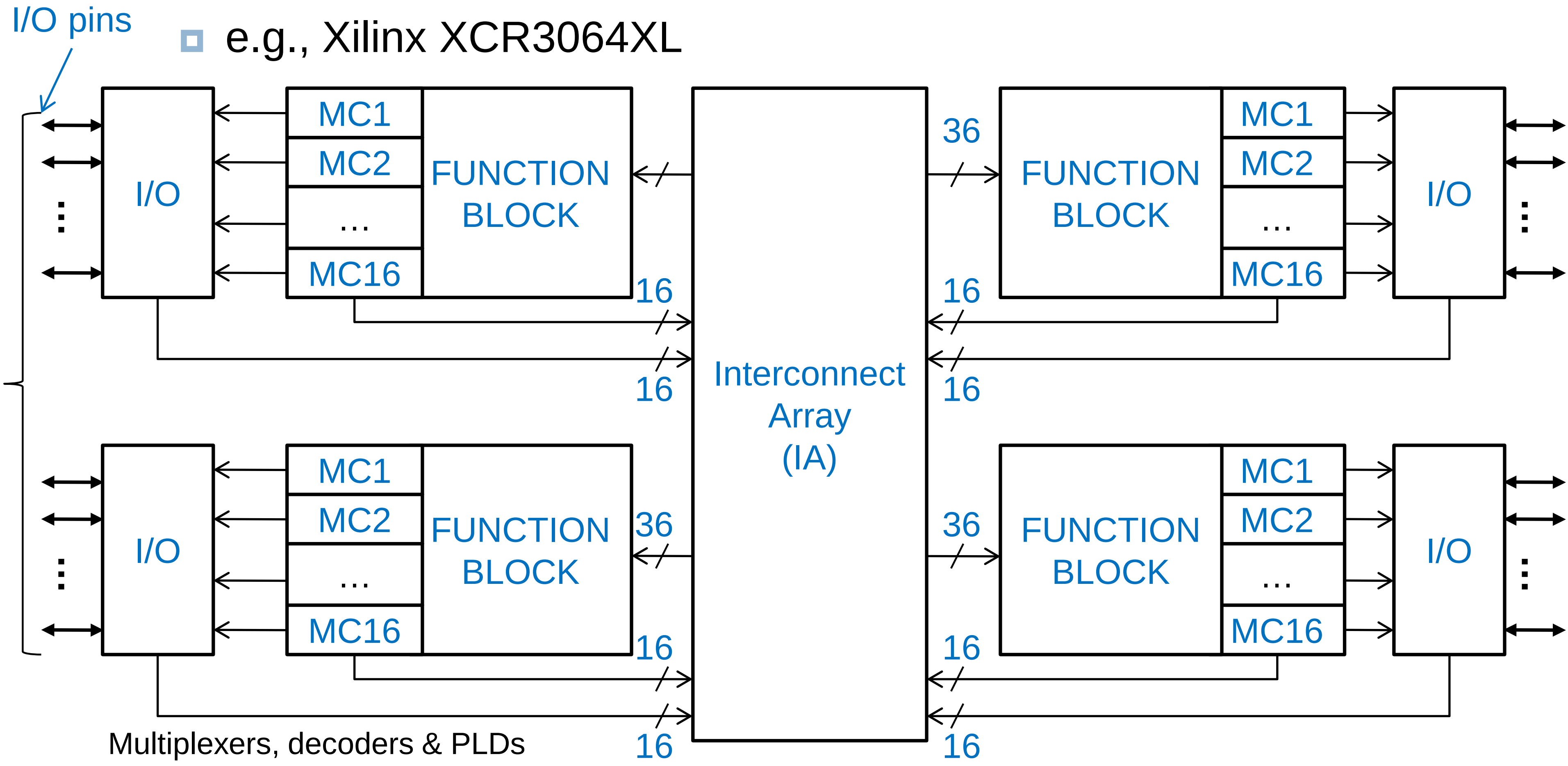
$Sum = X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in}$

Complex Programmable Logic Devices

CPLD

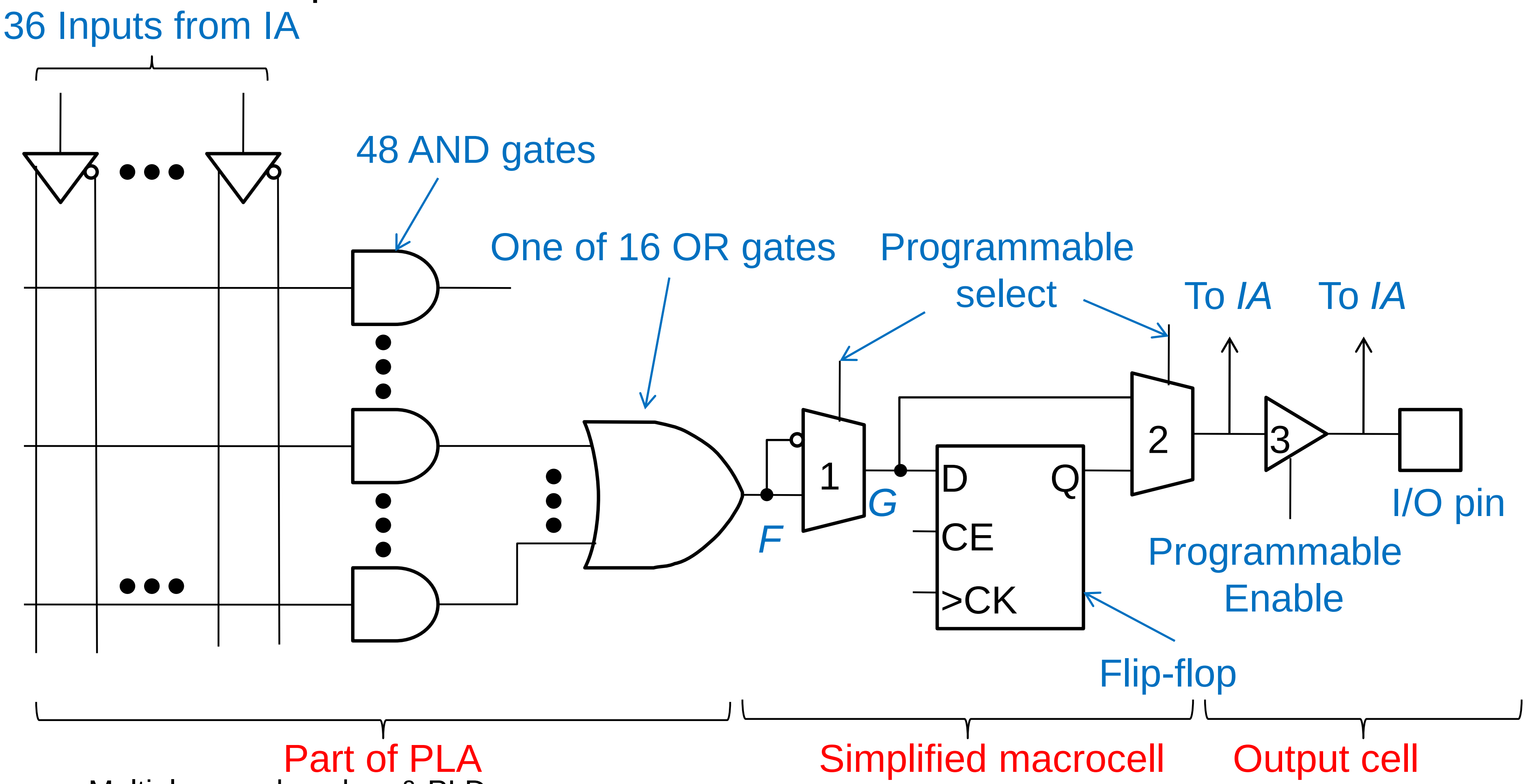
Complex Programmable Logic Devices (1/2)

- **Complex programmable logic device (CPLD): integrate and interconnect many PALs and PLAs on a single chip**
  - Tools will program for you
  - e.g., Xilinx XCR3064XL



CPLD (2/2)

- **CPLD function block and macrocell**
  - A simplified version of XCR3064XL

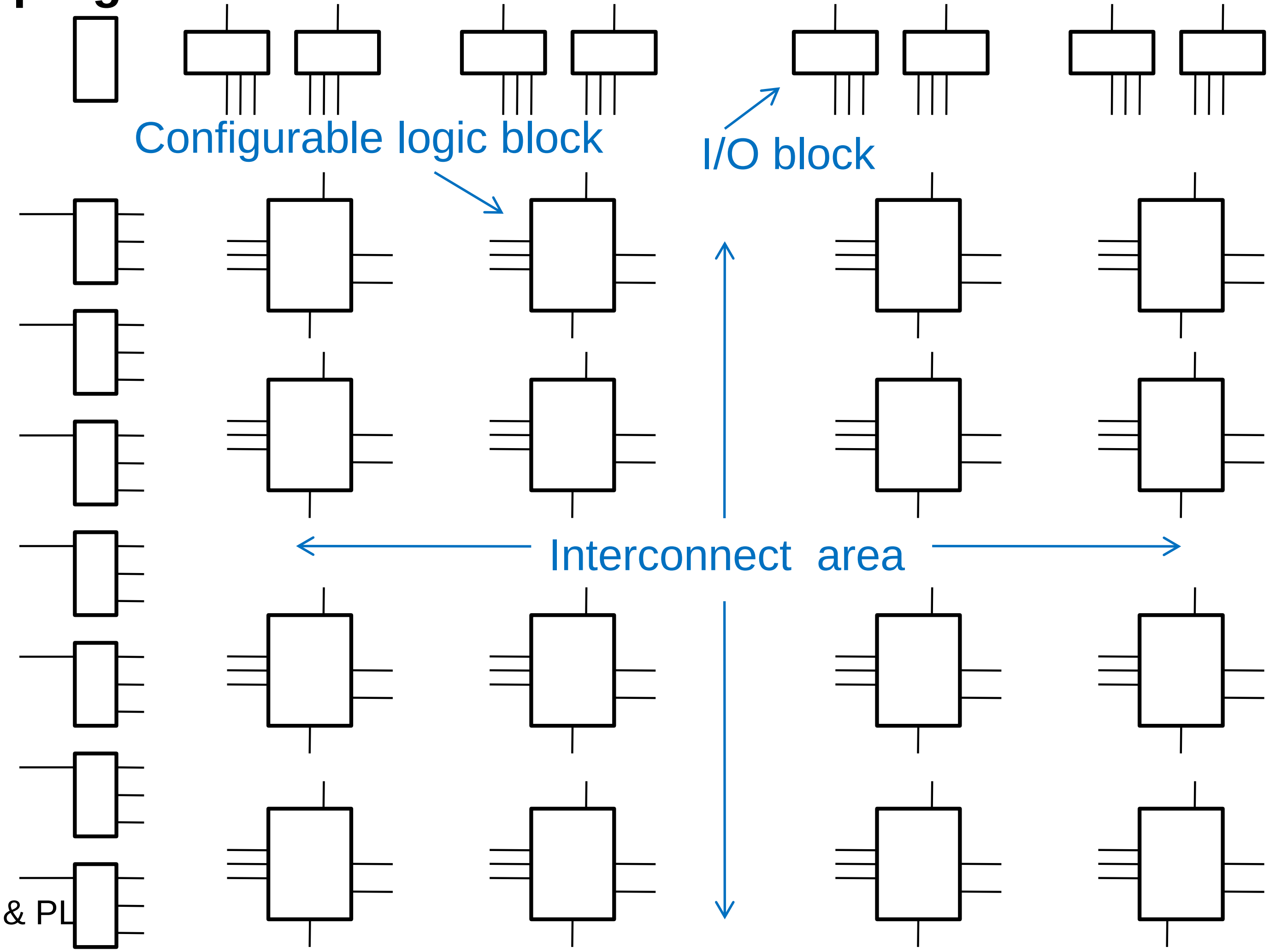


Field Programmable Gate Arrays

FPGA

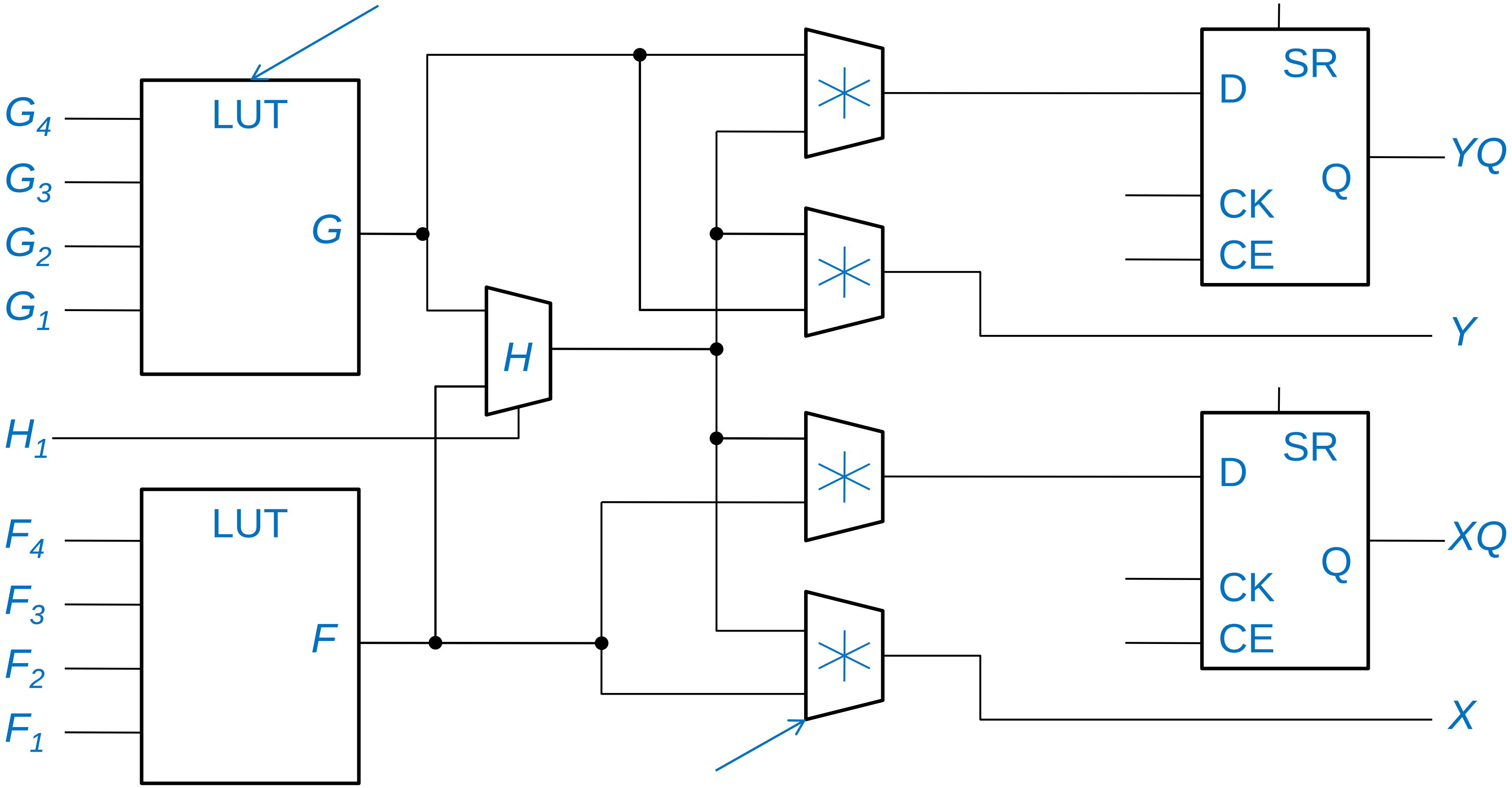
Field Programmable Gate Arrays

- **Field programmable gate arrays (FPGA): an array of identical logic cells + programmable interconnections**



Simplified Configurable Logic Block (CLB)

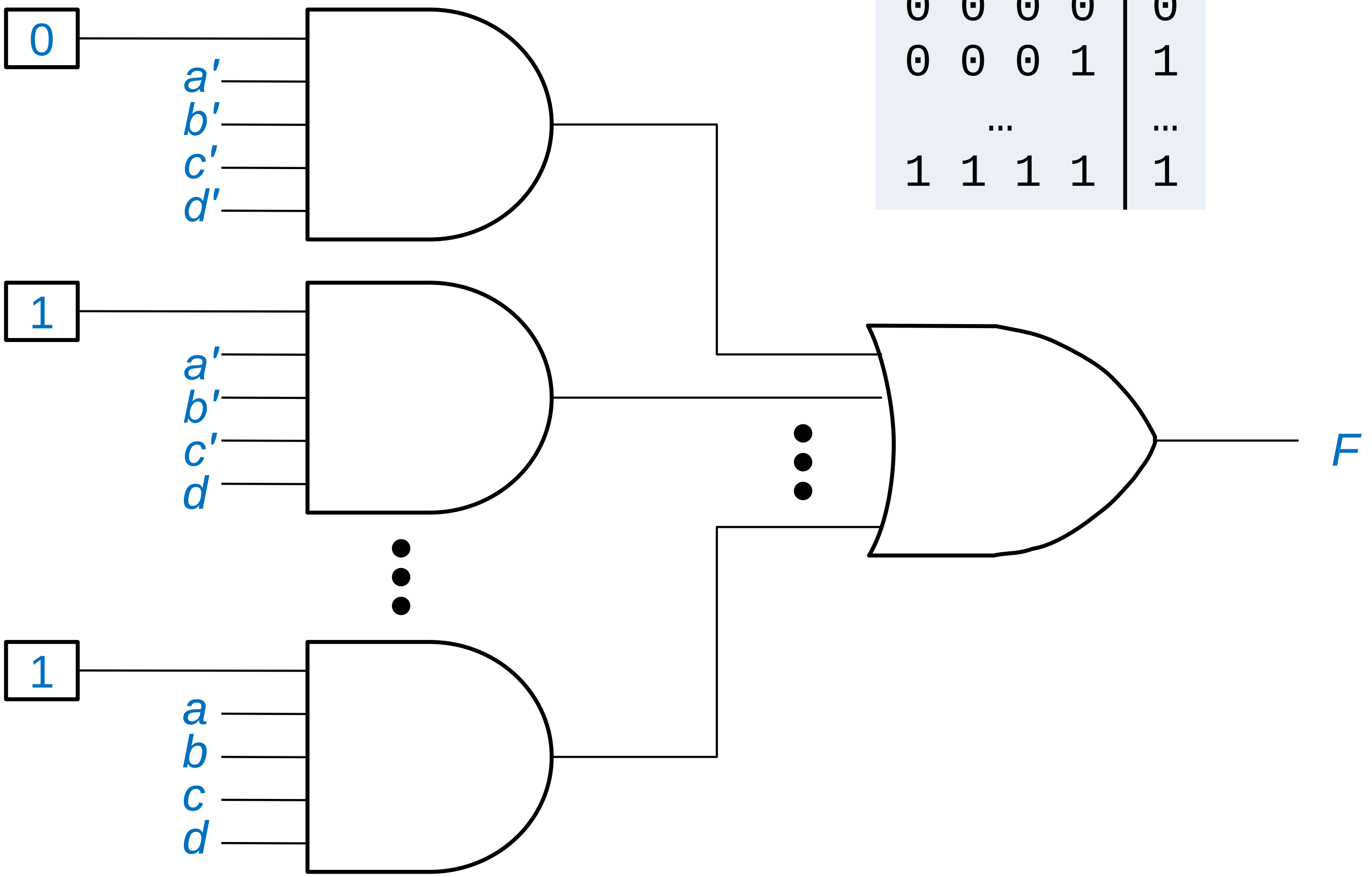
Lookup table



Programmable MUX

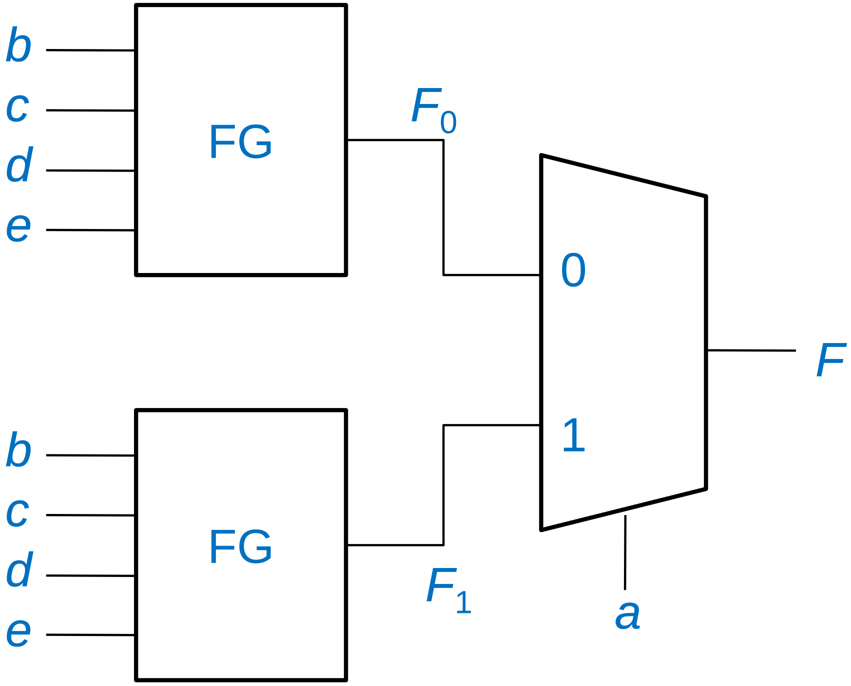
Implementation of a Lookup Table (LUT)

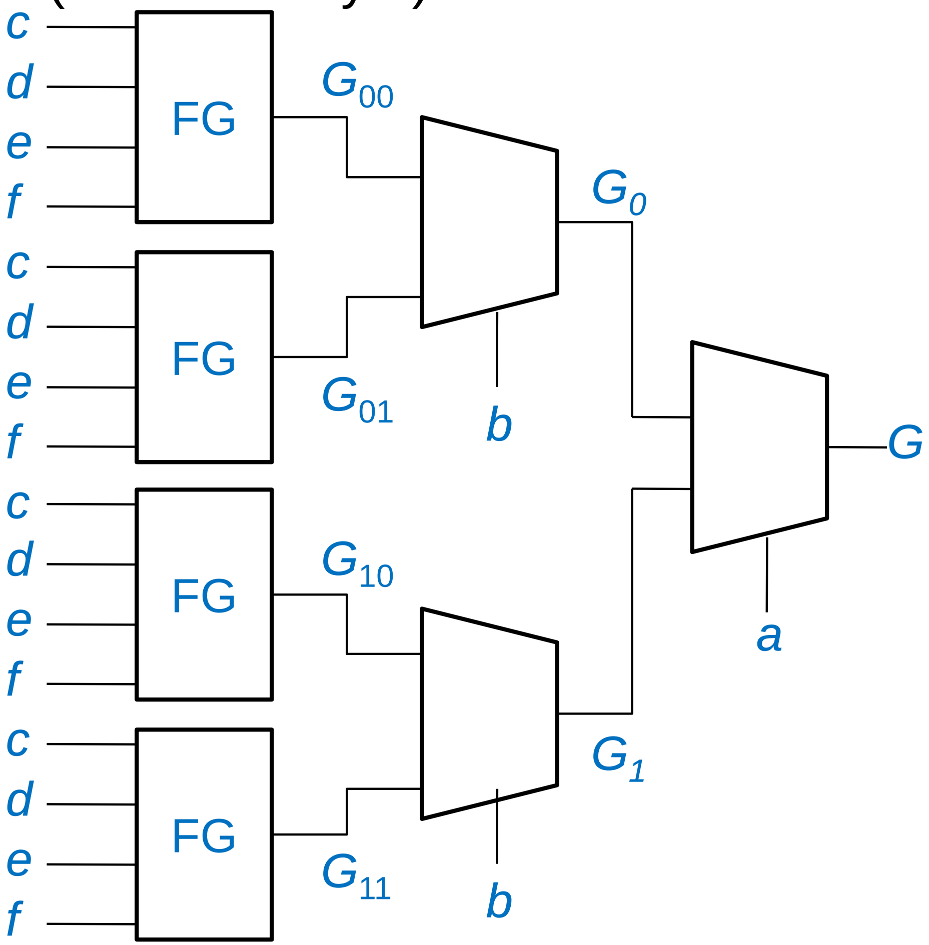
a	*b*	*c*	*d*	*F*
0	0	0	0	0
0	0	0	1	1
…				…
1	1	1	1	1



Realization with Function Generators

- □ **Realize a 5-variable function with 4-variable FGs:**
  - $F(a, b, c, d, e) = a'F(0, b, c, d, e) + aF(1, b, c, d, e) = a'F_0 + aF_1$
  - 2 4-variable FGs + 2-to-1 MUX (controlled by a)
- □ **Q: 6-variable function?**

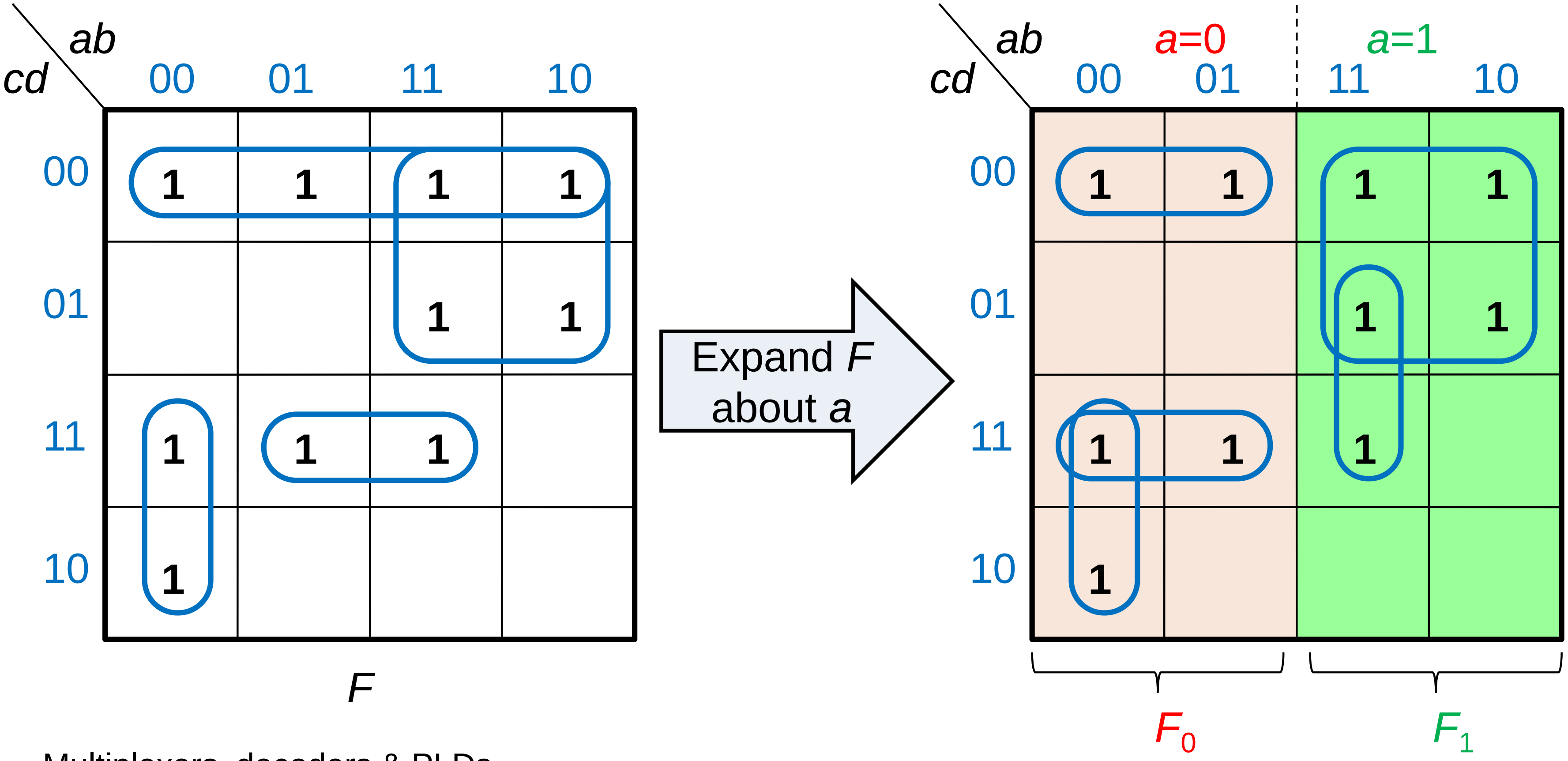




Decomposition of Switching Functions

- **Shannon's expansion theorem: reduces input variables**
  - $f(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = x'_i f_0 + x_i f_1$

    $= x'_i\, f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i\, f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$



F

cd \ ab	00	01	11	10
00	1	1	1	1
01			1	1
11	1	1	1	
10	1			

Expand F about a

cd \ ab	00	01	11	10
00	1	1	1	1
01			1	1
11	1	1	1	
10	1			