Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and Valerio Selis

dmc@liv.ac.uk

v.selis@liv.ac.uk

Digital 3: Tristate gates, ROM, PLA and PAL

UNIVERSITY OF LIVERPOOL

Outline

- Tri-state gates
- Read only memory
- Programmable logic arrays
- Programmable array logic

Previous material

Multiplexers✓

Decoders and encoders ✓

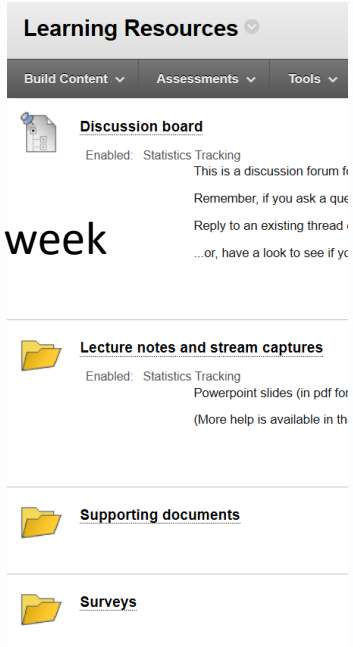
Minterms ✓

Implementing functions ✓

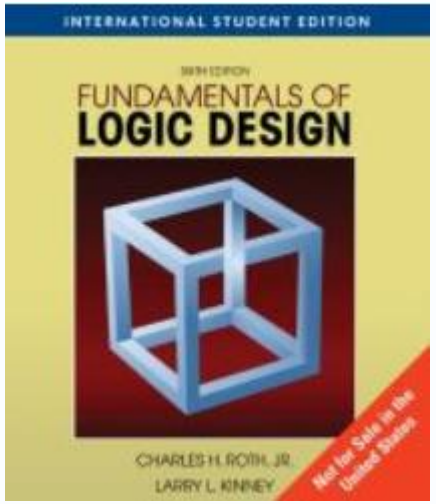
Use VITAL!:

- Stream lectures
- Handouts
- Notes and Q&A each week
- Discussion Board
- Exam resources

www.liv.ac.uk/vital



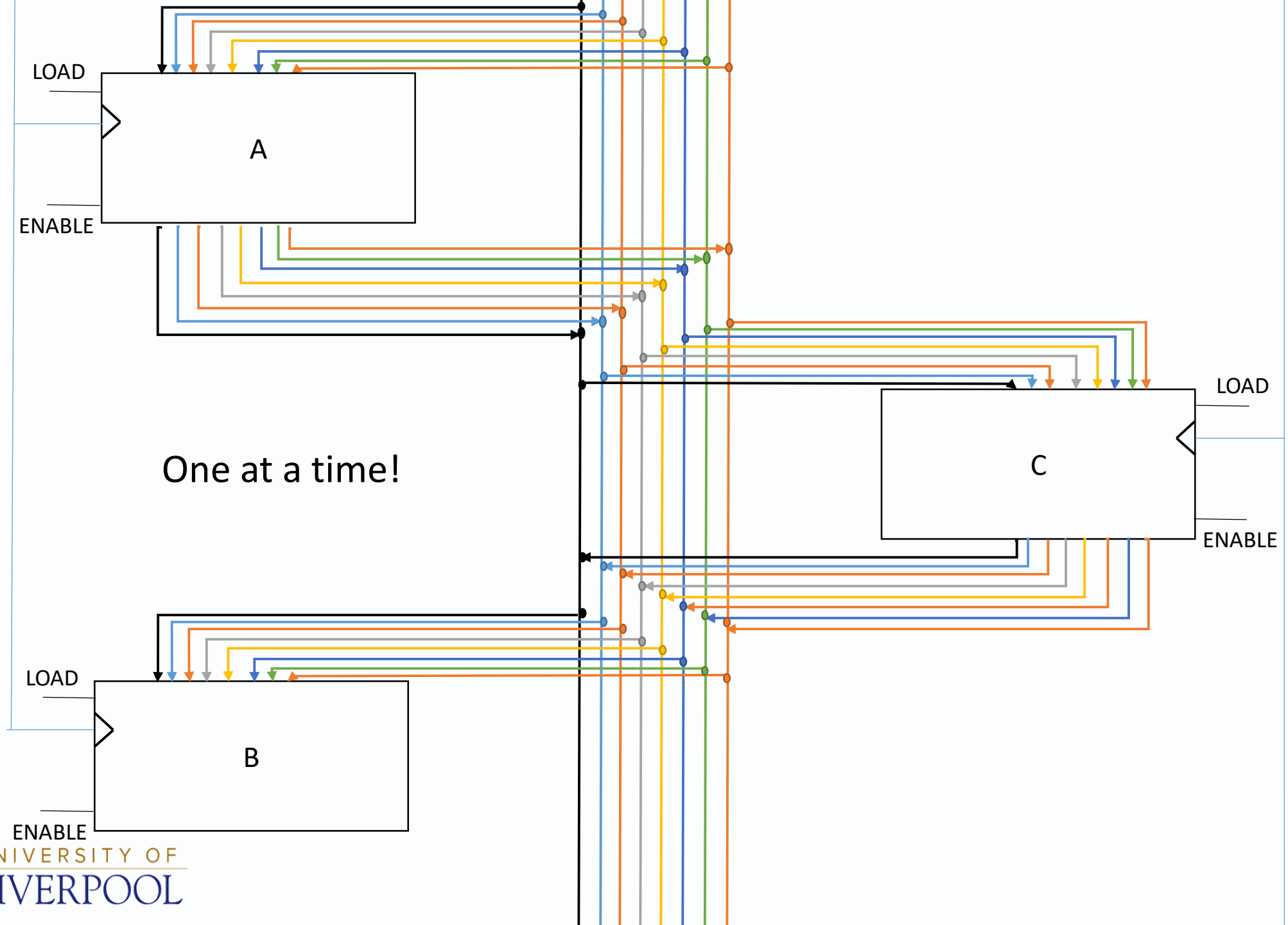
Course textbook – please borrow and use it!



UNIVERSITY OF LIVERPOOL

From previous lecture

- The BCD decoders in the previous lecture took ‘natural’ BCD (NBCD, 8421 code) i.e. equivalent to straight binary input for the 10 decimal digits 0-9
  - (Not 5421, 7321, etc.)



One at a time!

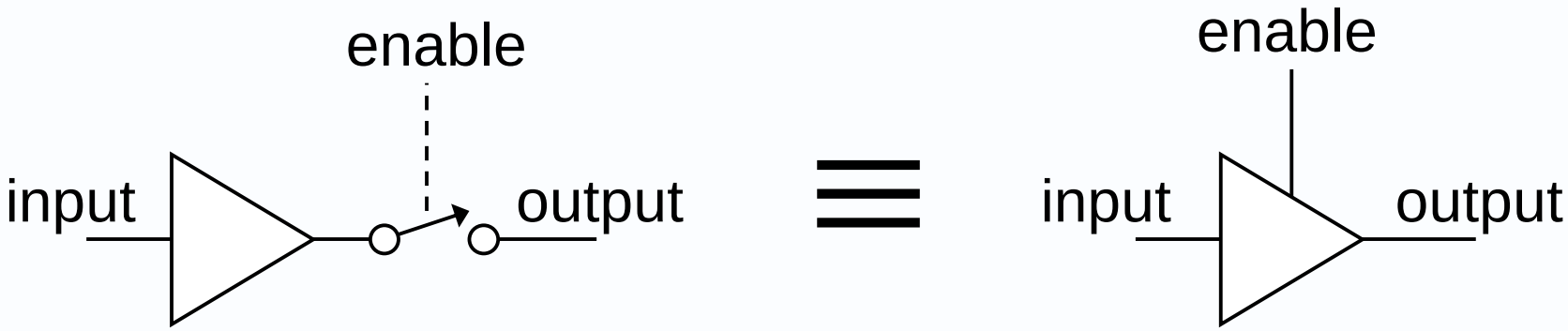
Good discussion:
https://www.youtube.com/watch?v=faAjse109Q8

Tri-state gates

Only one memory location should be connected to the bus at one time – one device can not drive a logic 0 onto a bus line at the same time as another device drives a logic 1 onto the same line.

This is achieved using ‘tri-state gates’ which have an ‘enable’ input – when not enabled the output acts like an ‘open circuit’ which is referred to as ‘high impedance’.

Tri-state gates

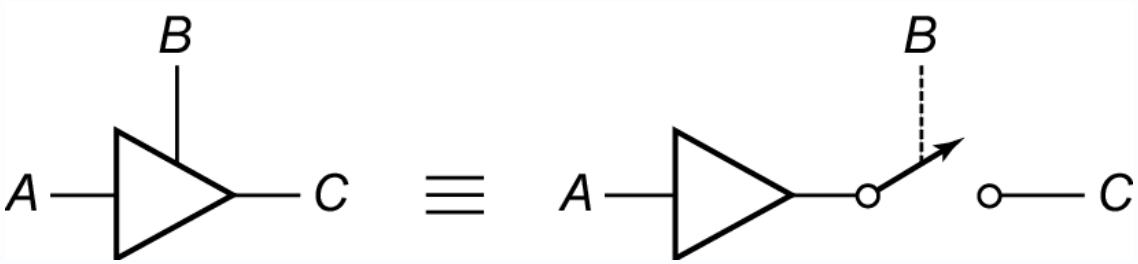


enable	input	output
0	0	high-impedance
0	1	high-impedance
1	0	0
1	1	1

Three-State Buffers

When the enable input B of a three-state buffer is 1 the output C equals A; when enable (B) is 0, the output C acts like an open circuit, commonly known as ‘high impedance’.

The three states of the buffer output are therefore: 1, 0, and Hi-Z



B	C
0	Hi-Z
1	A

A	B	C
0	0	Hi-Z
0	1	0
1	0	Hi-Z
1	1	1

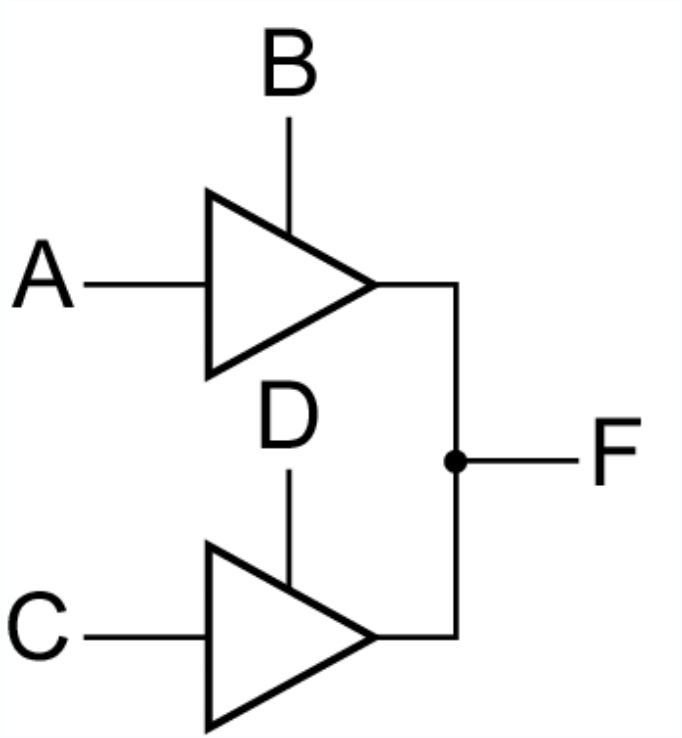
Three-State Buffers

When we connect the outputs of two three-state buffers:

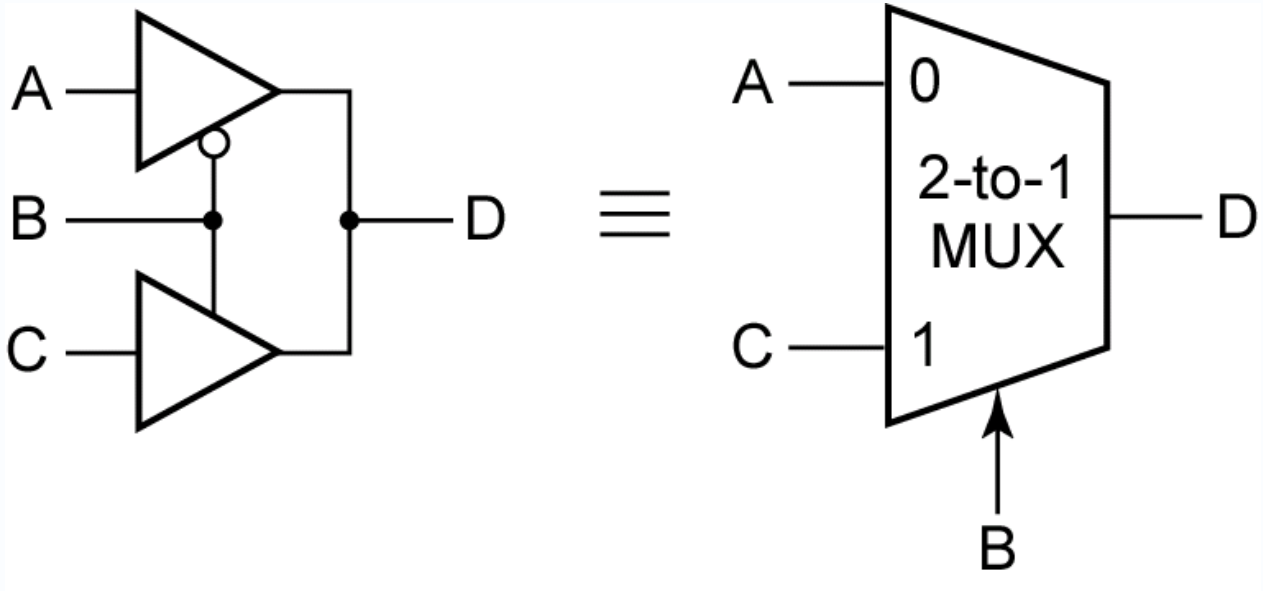
If one of the buffers is disabled the combined output is the same as the other output.

If both buffers are disabled the output is Hi-Z.

If both buffers are enabled, a conflict can occur. The connection of logic 1 and logic 0 is unknown (X)



Three-State Buffers as multiplexers



Data selection using three-state buffers

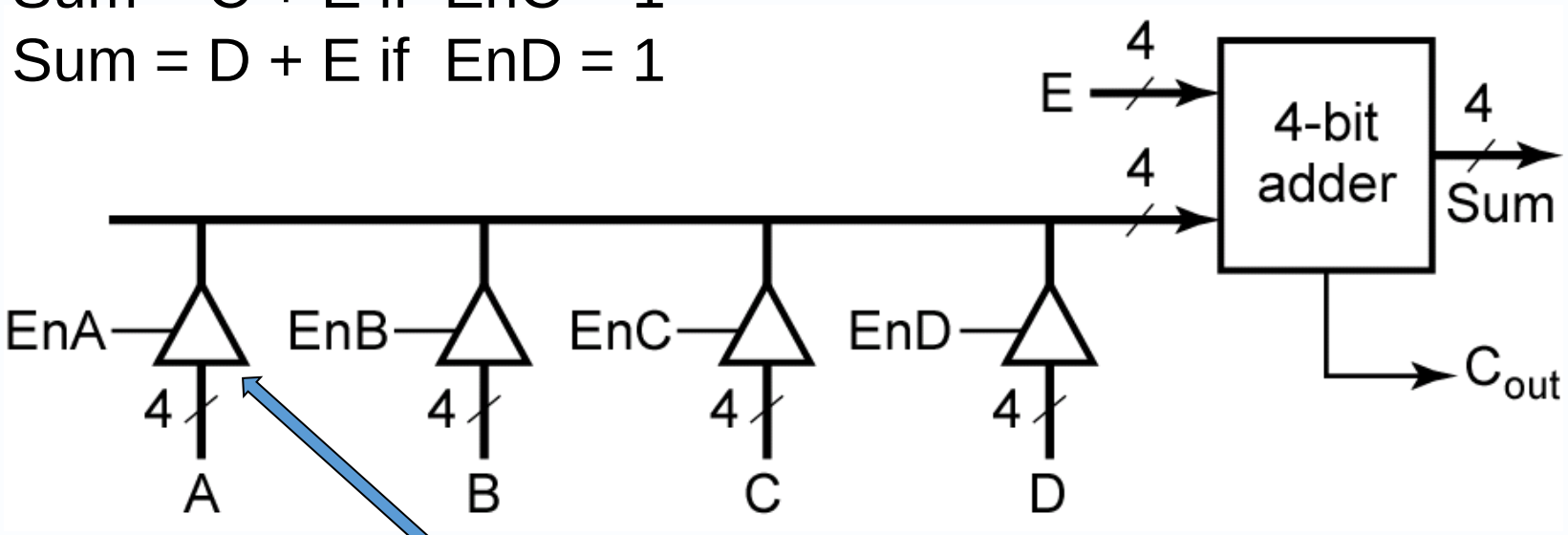
When B=0 the top buffer is enabled D=A

When B=1 the bottom buffer is enabled D=C

Three-State Buffers Applications

4-bit adder with four sources for one operand. Three-state buffers have been used as multiplexers.

Sum = A + E if EnA = 1
Sum = B + E if EnB = 1
Sum = C + E if EnC = 1
Sum = D + E if EnD = 1



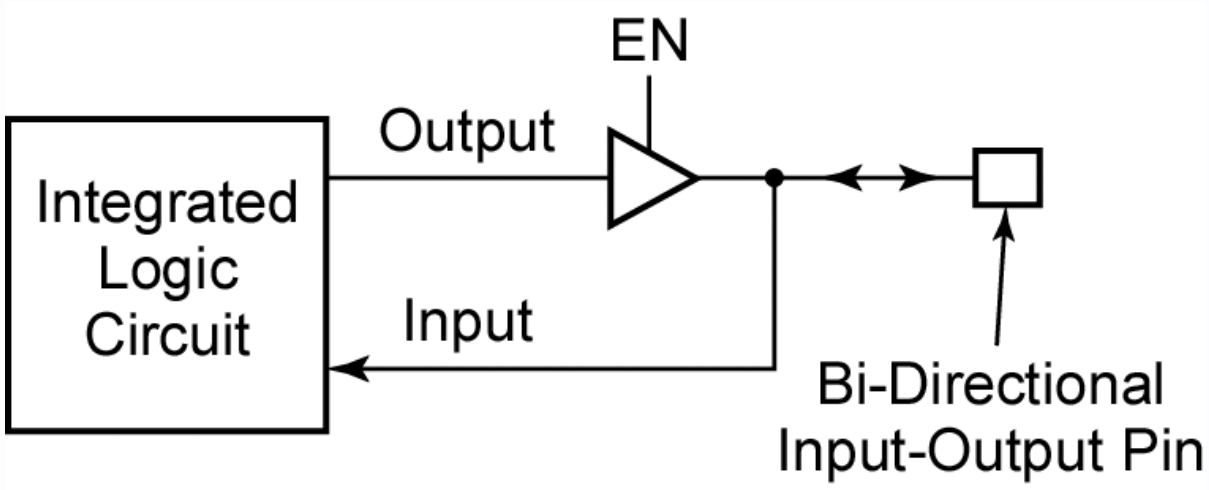
*Here, **4** tri-state buffers are represented by each buffer symbol. They're all enabled by one signal (e.g. EnA).*

Three-State Buffers Applications

Integrated circuit with bi-directional input/output pin.

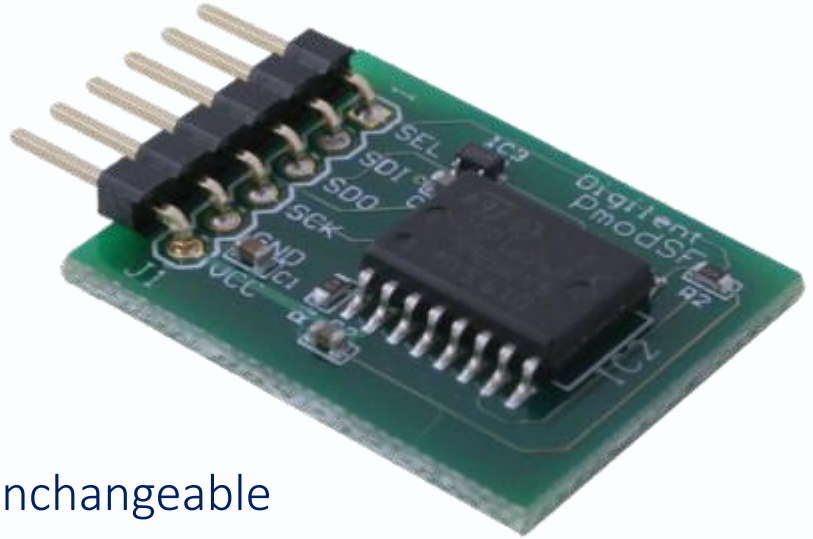
When the buffer is enabled (EN = 1) the pin is driven with the output signal.

When the buffer is disabled (EN = 0) an external source can drive the input pin.



Read-Only Memory (ROM)

- ✓A read only memory chip consists of an array of semiconductor devices that are interconnected to store an array of binary data
- ✓Once binary data is stored in the ROM, it can be read out whenever desired, but it can't be changed under normal operating conditions
  - ✓*Particularly important in systems where programs should not be modified*
- ✓Several variants have been developed over time:
  - ✓Mask ROM (data fixed during manufacture)
  - ✓PROM (programmable once after mfct. – fuses & transistors)
  - ✓EPROM (erasable *en bloc* by UV, reprogrammable - transistors)
  - ✓EEPROM (*individual* portions *electrically** erasable)
  - ✓Flash (faster – writes/erases in larger blocks e.g. 512-byte)
- ✓All have in common:
  - ✓A 'special' operation is needed to change stored data – or it's unchangeable
  - ✓'Non-volatile' – data is still remembered without power on

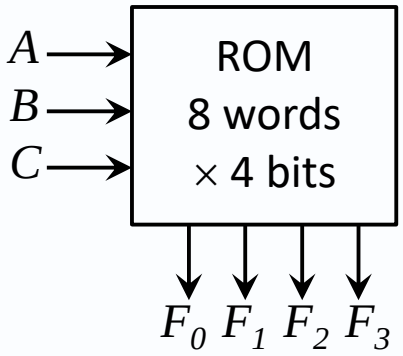


Pmod serial flash rom

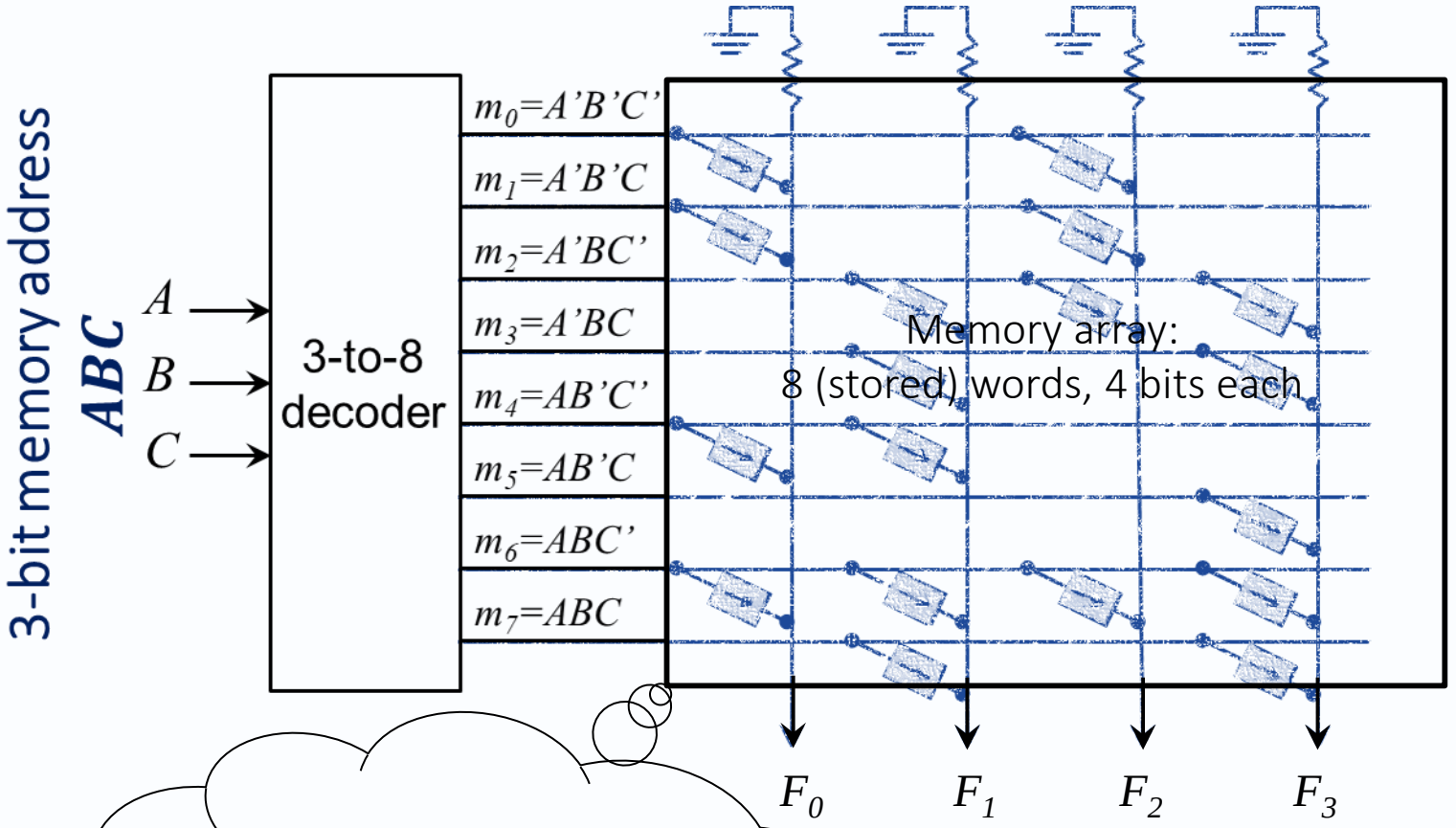
https://commons.wikimedia.org/wiki/File:Pmod-serial-flash-rom-16-mbit.jpg

*i.e. end-user-modifiable

ROM example: 8-word x 4-bit



=



A ROM consists of:

- Decoder *and*
- Memory array

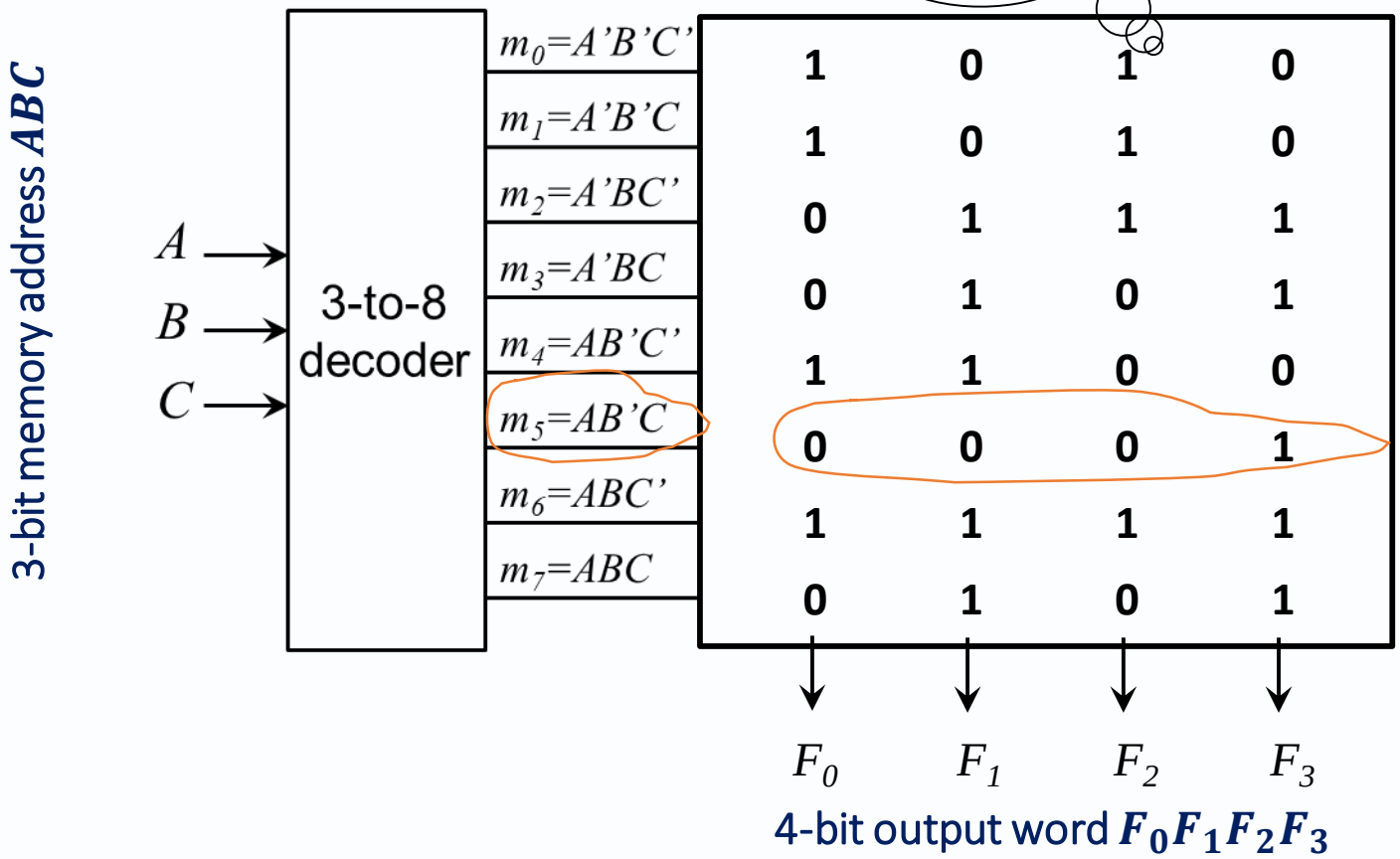
ROM example: 8 x 4

To re-emphasise: this is just a memory array; we have programmed what the eight words might be (there is no logic to this; it is just an example.)

address
- 3 input lines = address
- e.g. $ABC = 101$

decode
- $2^3 = 8$ possible addresses
- Each activates one minterm

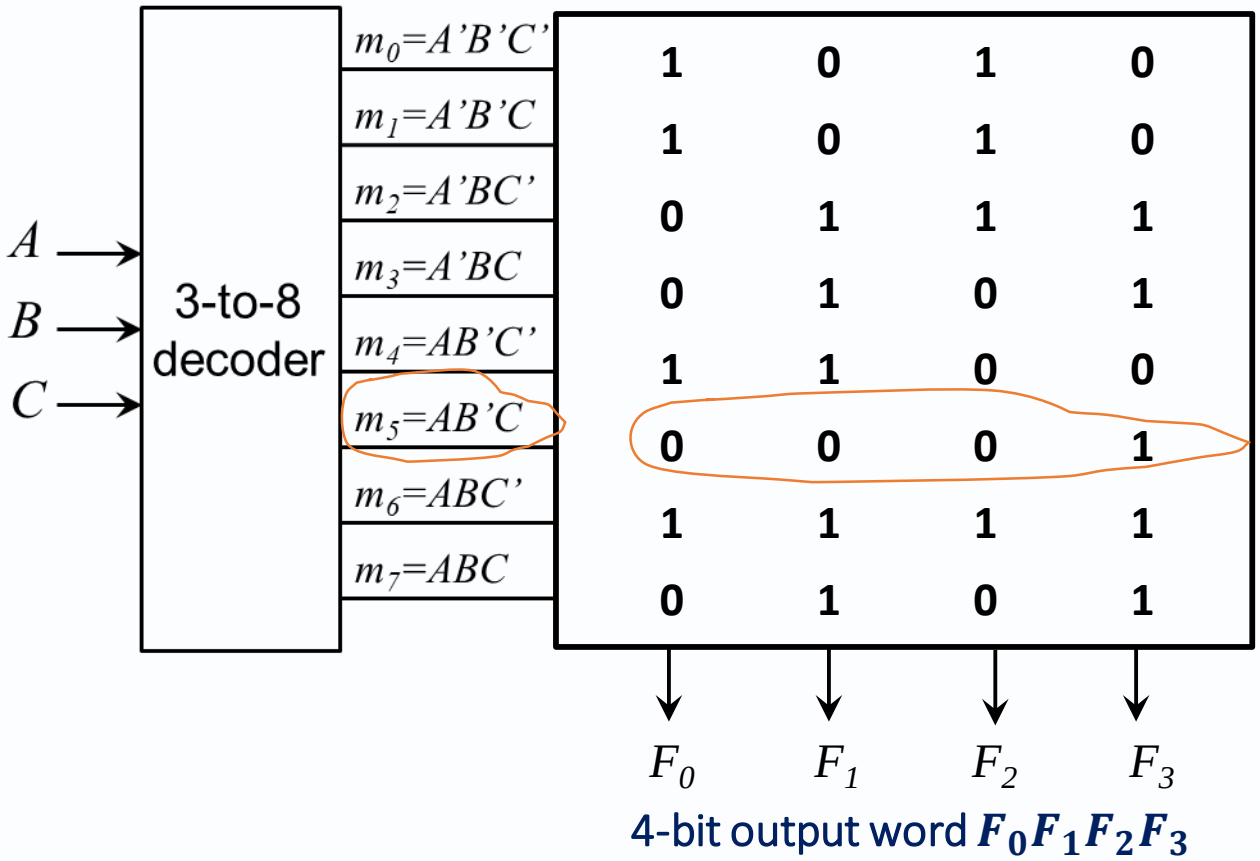
word
- Each minterm selects a stored word
- Each word 4 bits long



8 x 4 ROM: example truth table

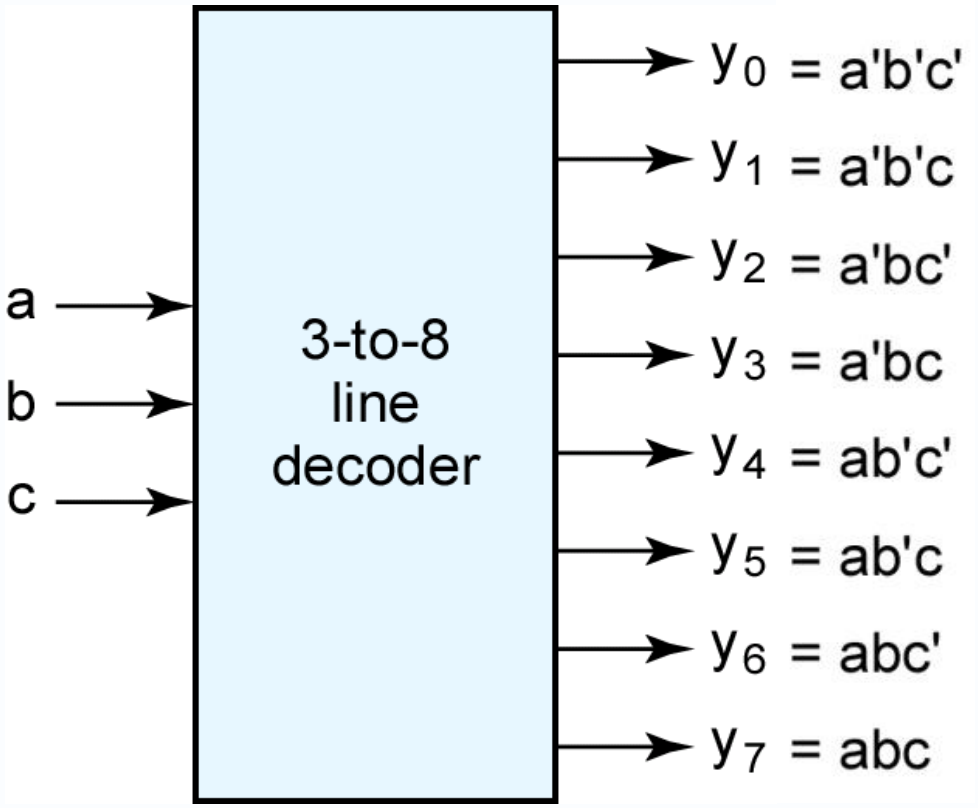
A	B	C	F_0	F_1	F_2	F_3
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

3-bit memory address ABC



(Last lecture: 3-to-8 line decoder truth table)

*(y_i might be assigned to represent the presence/absence of the decimals 0 1 2 3 4 5 6 7 ….)**



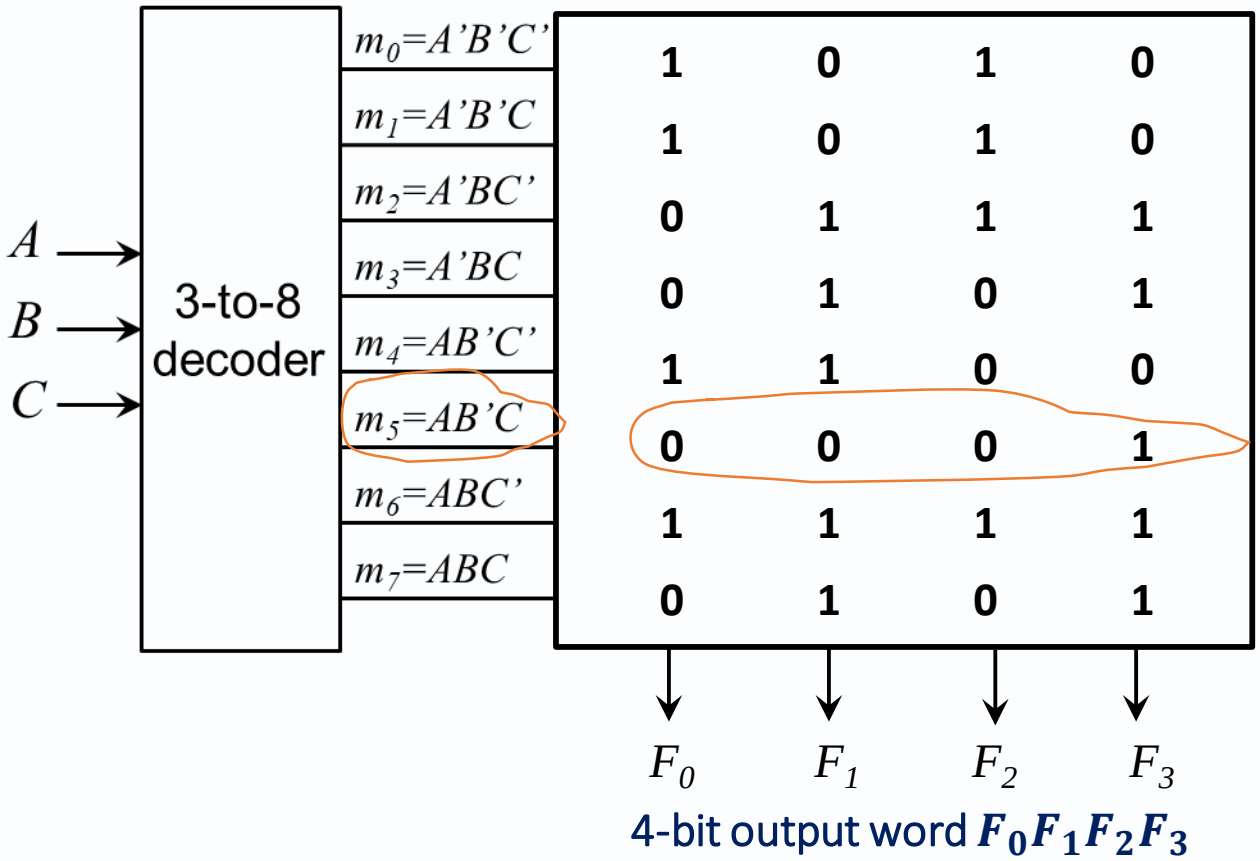
a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

***Note**: if we had two more output lines available, and one extra input, we could assign the state of each output line to represent presence or absence of one of the ten decimal digits…… see top*

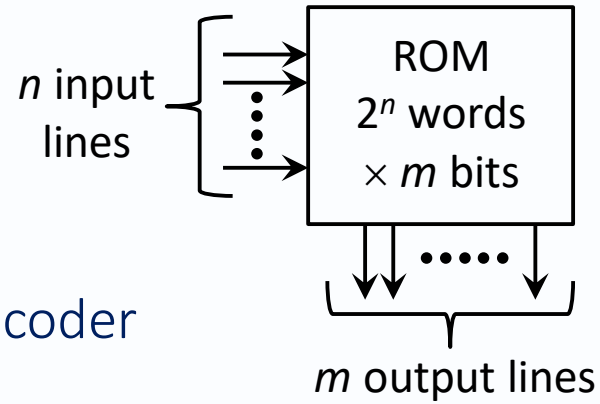
8 x 4 ROM: example truth table

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

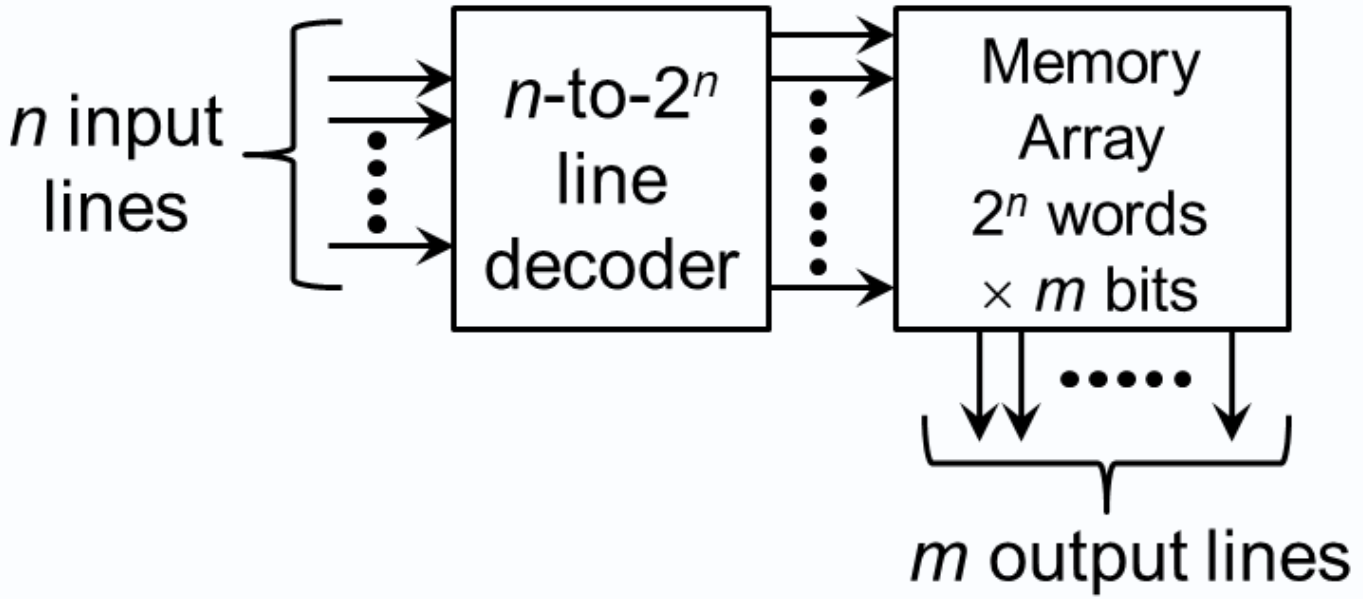
3-bit memory address ABC



ROM simple summary: $n \times m$



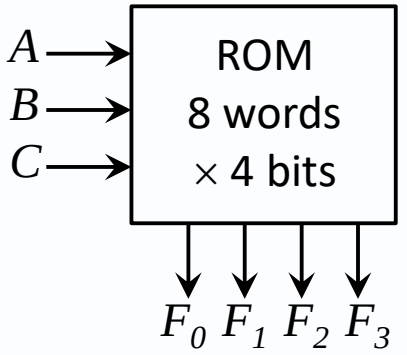
- ✓When an address is applied to the decoder inputs, exactly one of the 2^n decoder outputs (minterms) is 1
- ✓This decoder output line selects one of the 2^n words stored in the memory array
- ✓The bit pattern stored in this word is transferred to the m memory output lines

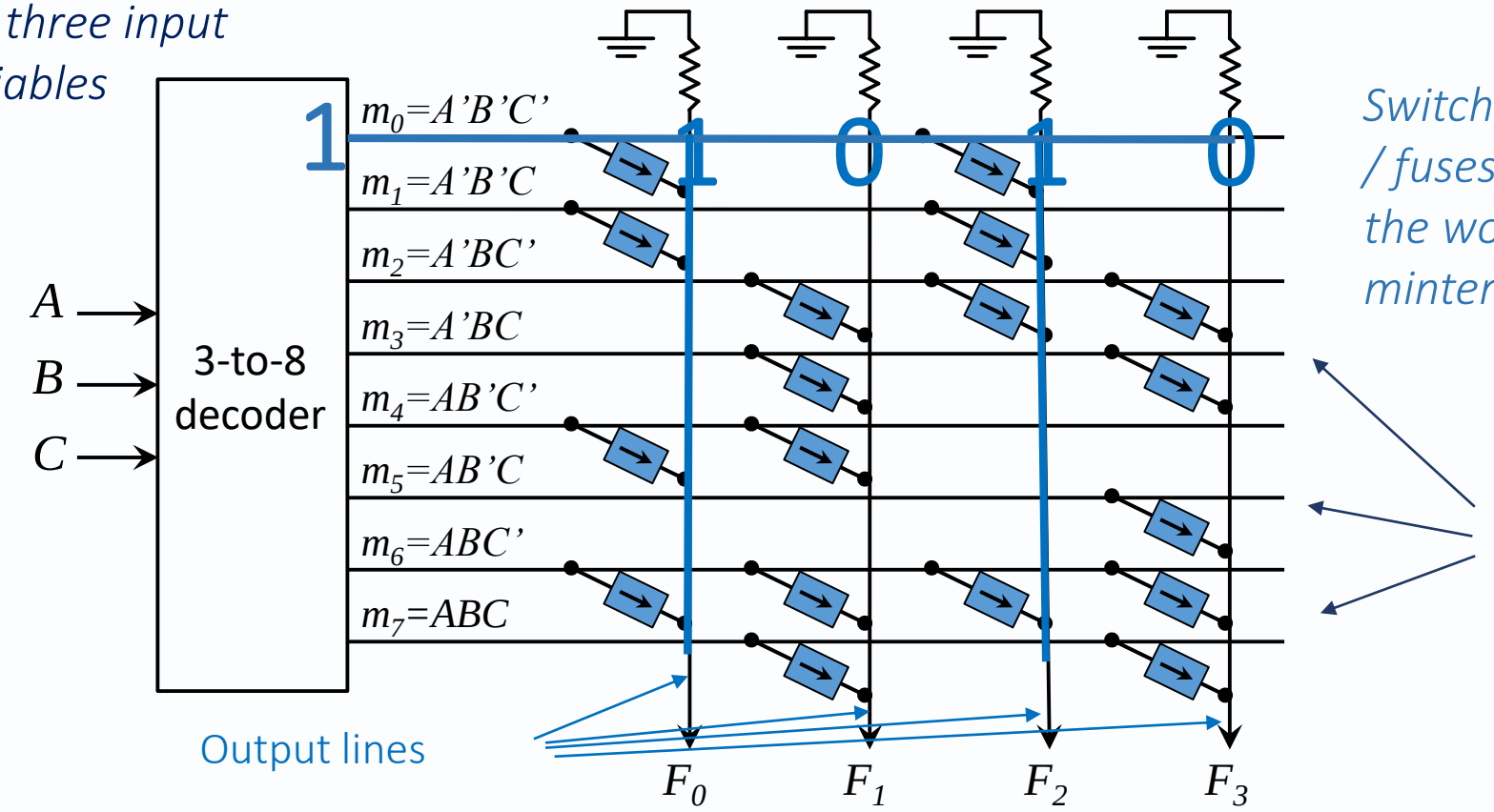


ROM – how?

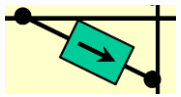
Decoder generates the eight minterms of the three input variables

Pull-down resistors force the output line to be 0 when none of the connected minterms are 1





Switching elements (diodes / fuses / transistors) define the word belonging to each minterm

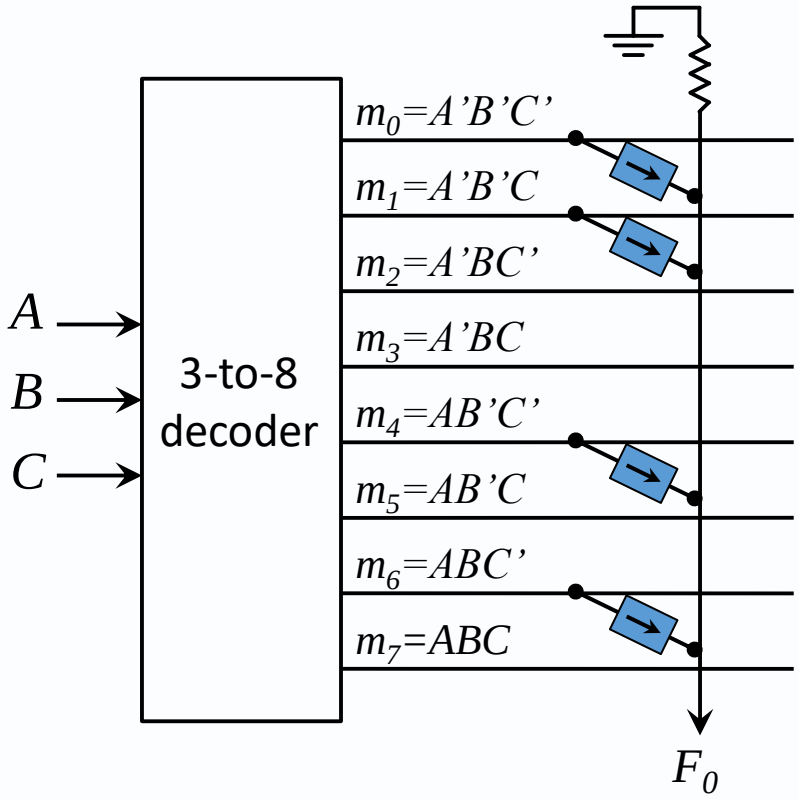


Word lines

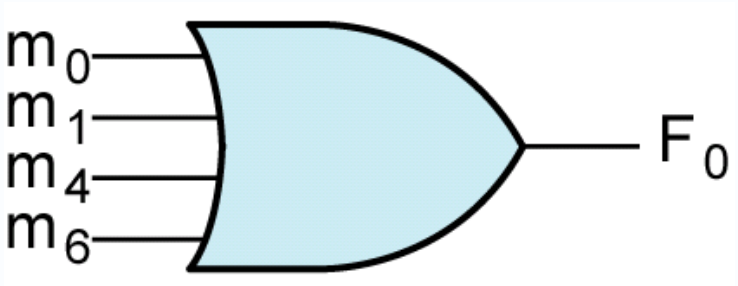
Output lines

Logically, we can think of the memory array as forming each of the four output functions by ORing together relevant minterms…

Combinational logic of ROM



- The pull-down resistor forces the output line to be 0 when none of the connected minterms are 1
- This is a "wired" OR

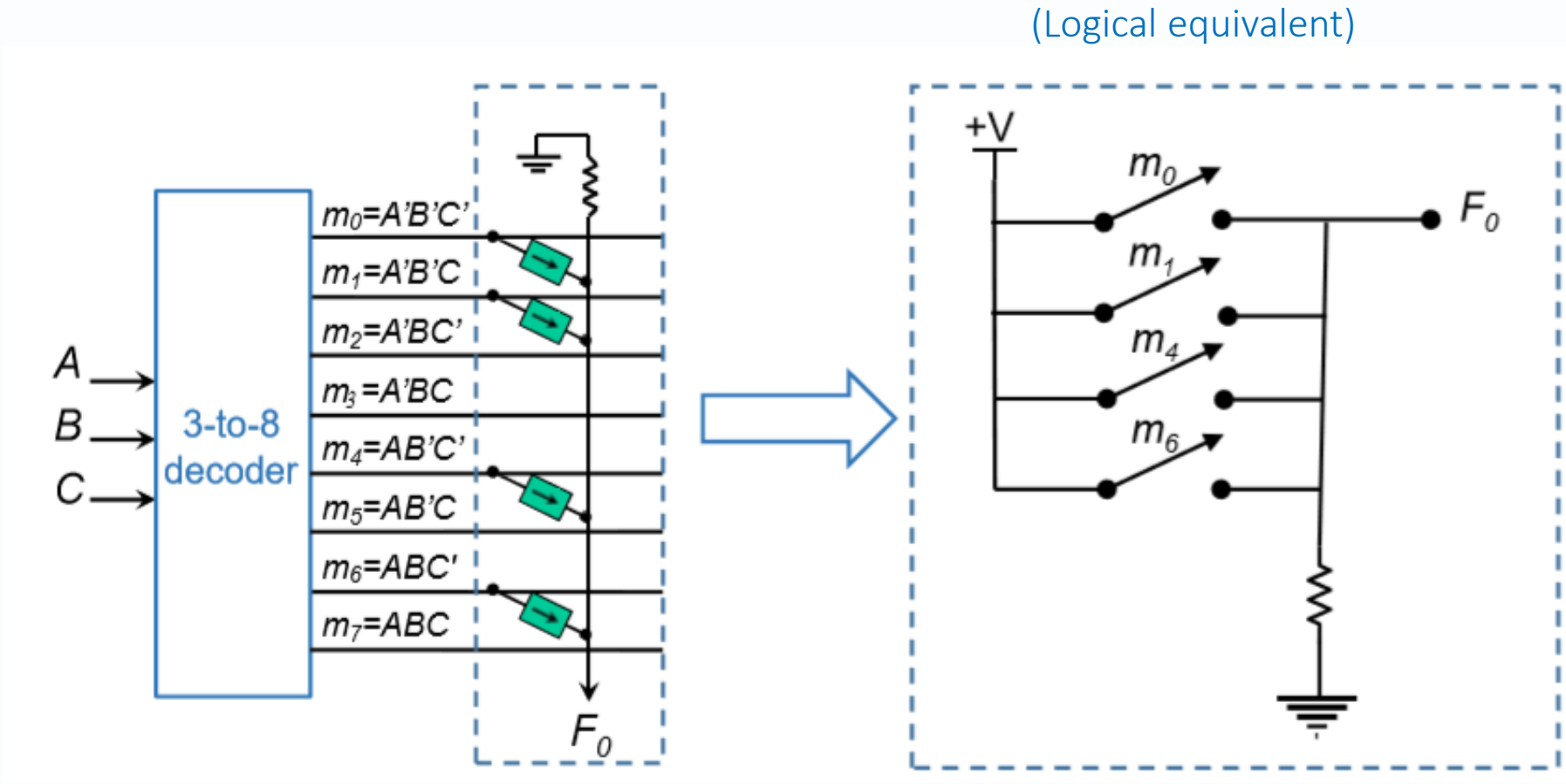


$F_0=\Sigma m(0,1,4,6)=A'B'C'+A'B'C+AB'C'+ABC'$

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

$=A'B'+AC'$

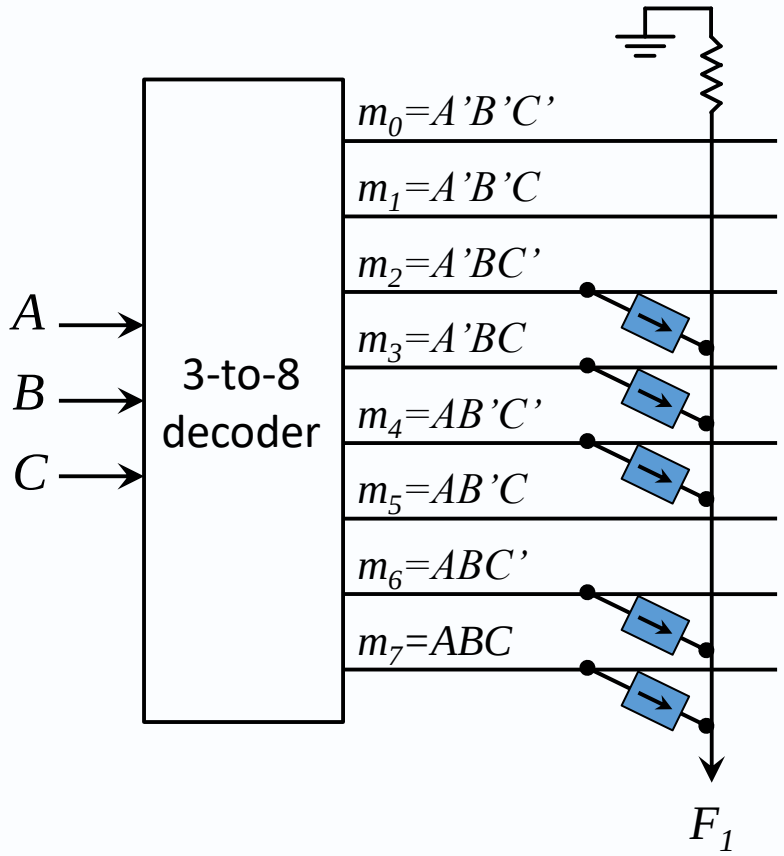
Combinational logic of ROM



If m_0, m_1, m_4 or m_6 are in logic state 0, the switches are open. If they are in logic state 1, the switches are closed. The result is an OR operation (wired OR).

$$F_0=\Sigma m(0,1,4,6)=A'B'C'+A'B'C+AB'C'+ABC'$$
$$=A'B'(C'+C)+AC'(B'+B)$$
$$=A'B'+AC'$$

Combinational logic of ROM



Karnaugh map

F_1

	$A'B'$	$A'B$	AB	AB'
C'	0	1	1	1
C	0	1	1	0

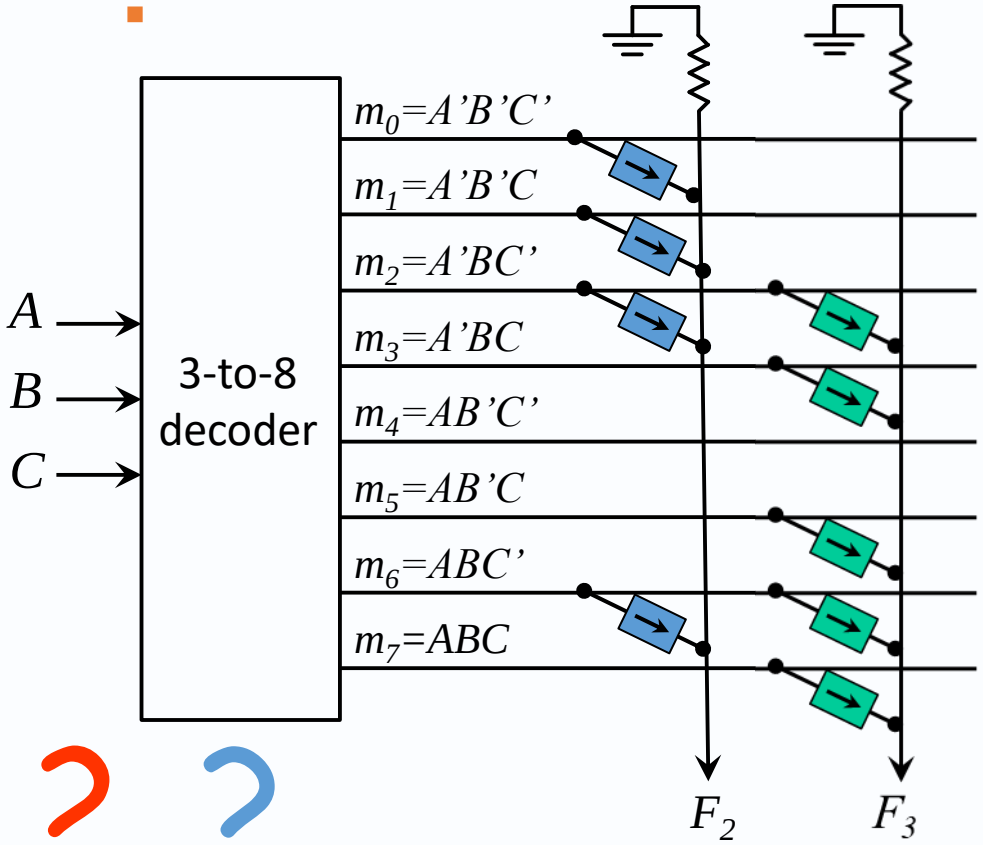
$$F_1 = \sum m(2,3,4,6,7)$$

$$= \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.B.\overline{C} + A.B.C$$

$$= B.(\overline{A}.\overline{C} + \overline{A}.C + A.\overline{C} + A.C) + A.\overline{C}.(\overline{B} + B)$$

$$= B + A.\overline{C}$$

Question

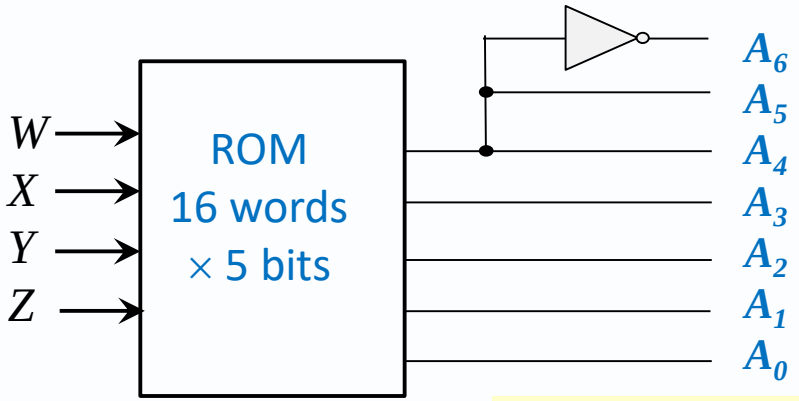


- Find the 'sum of products' expressions for outputs F_2 and F_3 and simplify if possible.

Using ROM for multi-output combinational circuits

Easy to realize - e.g. code converter: 4-bit binary # to hex digit, outputs the 7-bit ASCII code

					ASCII Code for Hex Digit						
W	X	Y	Z	Hex Digit	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	**0**	0	1	1	0	0	0	0
0	0	0	1	**1**	0	1	1	0	0	0	1
0	0	1	0	**2**	0	1	1	0	0	1	0
0	0	1	1	**3**	0	1	1	0	0	1	1
0	1	0	0	**4**	0	1	1	0	1	0	0
0	1	0	1	**5**	0	1	1	0	1	0	1
0	1	1	0	**6**	0	1	1	0	1	1	0
0	1	1	1	**7**	0	1	1	0	1	1	1
1	0	0	0	**8**	0	1	1	1	0	0	0
1	0	0	1	**9**	0	1	1	1	0	0	1
1	0	1	0	**A**	1	0	0	0	0	0	1
1	0	1	1	**B**	1	0	0	0	0	1	0
1	1	0	0	**C**	1	0	0	0	0	1	1
1	1	0	1	**D**	1	0	0	0	1	0	0
1	1	1	0	**E**	1	0	0	0	1	0	1
1	1	1	1	**F**	1	0	0	0	1	1	0

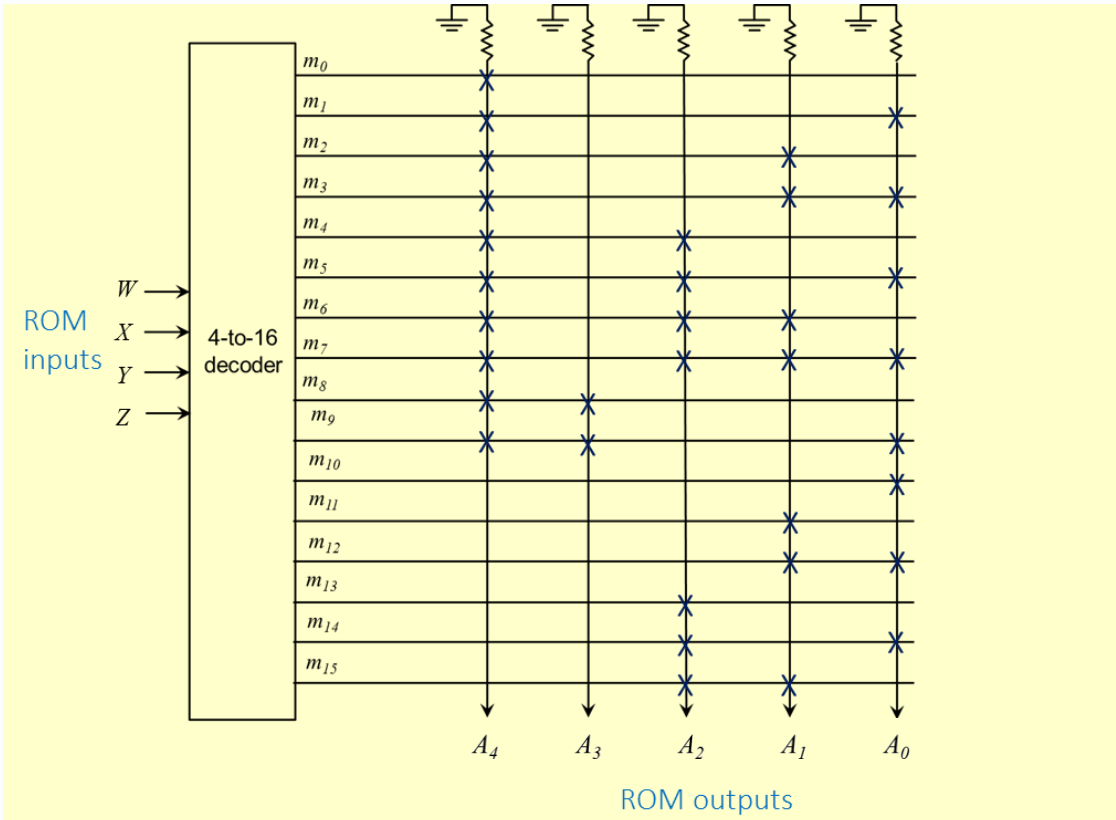


Here, a cross X at the intersection of word line and output line (bit line) indicates the presence of a switching element

$A_5 = A_4$

$A_6 = A_4'$

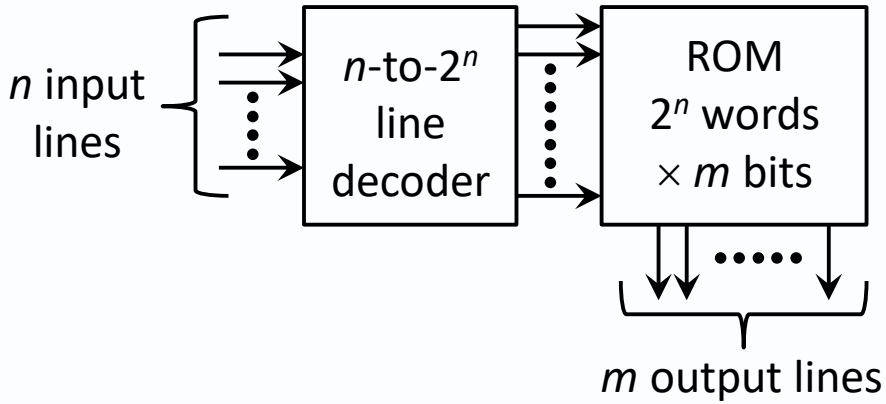
ROM needs only 5 outputs



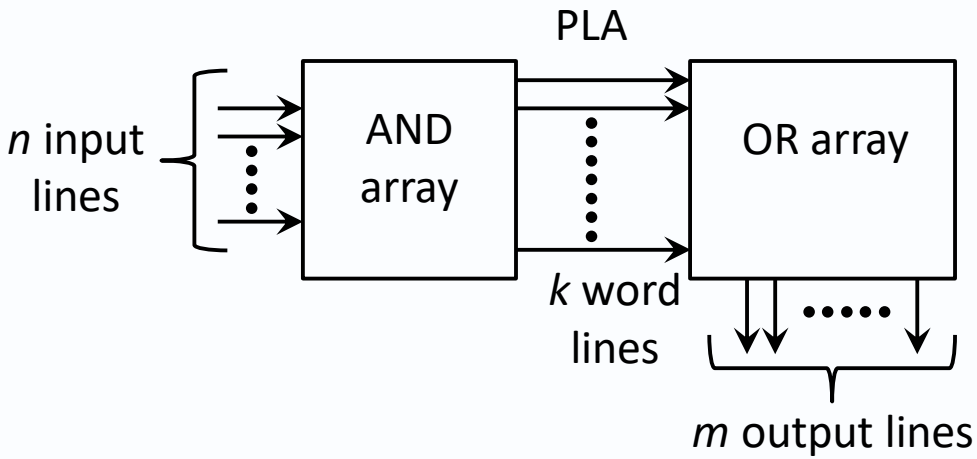
UNIVERSITY OF LIVERPOOL

Programmable Logic Arrays (PLA)

A ROM directly implements a truth table, after using a decoder to produce every minterm.



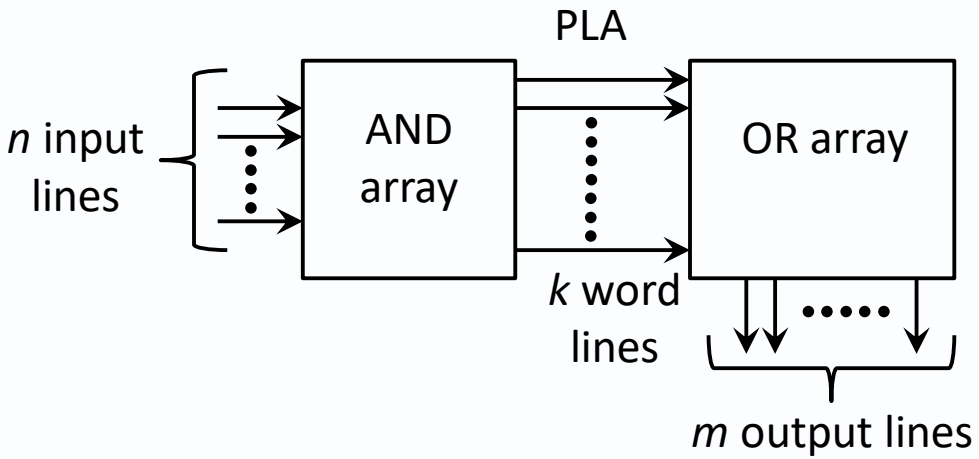
A PLA implements a **sum-of-products** expression, after using an AND array to produce each product term.



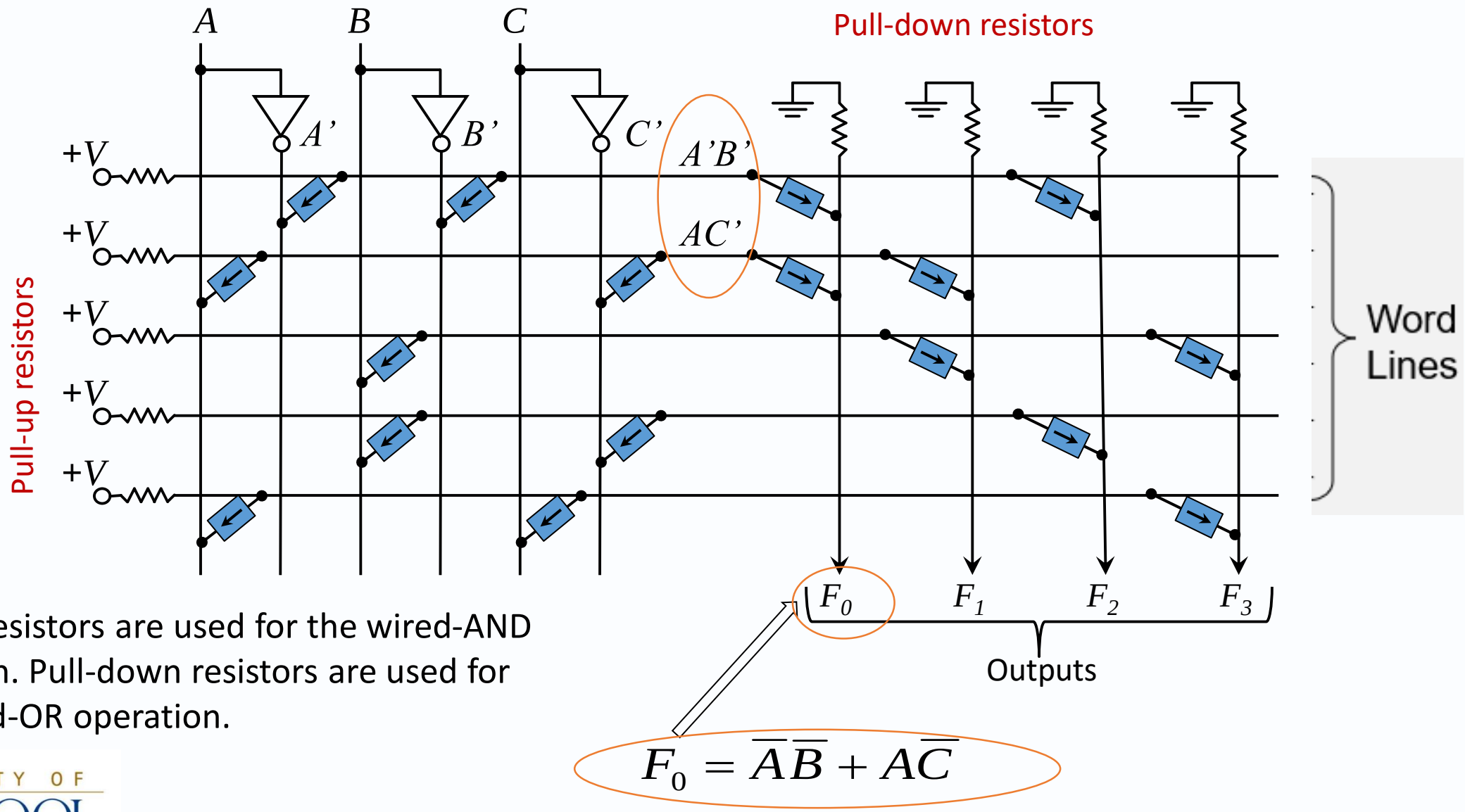
- *Generally, programmable logic devices (PLD) are digital integrated circuits which can be programmed to provide different logic functions*
  - *Can be combinational or sequential, replacing a large number of ICs*
  - *Hence lower cost designs*

Programmable Logic Arrays (PLA)

- PLA performs the same basic function as a ROM.
- With *m* outputs and *n* inputs, can realize *m* functions of *n* variables (e.g. *n* words of *m* bits each)
- The AND array realizes the selected product terms of the input variables.
- The OR array ORs together the product terms needed to form the output functions.



3 inputs, 5 product terms & 4 outputs

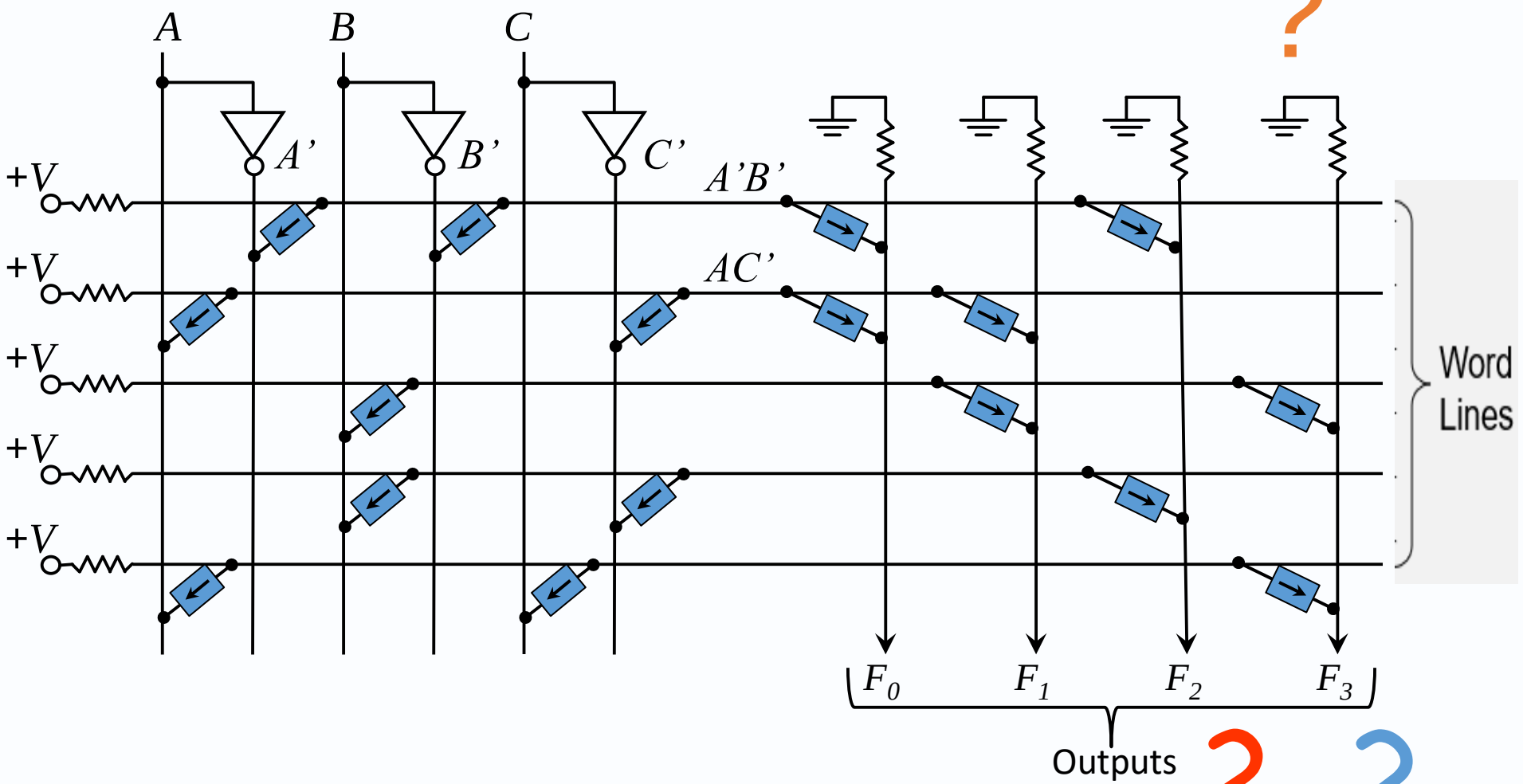


Pull-up resistors are used for the wired-AND operation. Pull-down resistors are used for the wired-OR operation.

Question

Find the ‘sum of products’ expressions for the outputs

F_1, F_2, F_3.

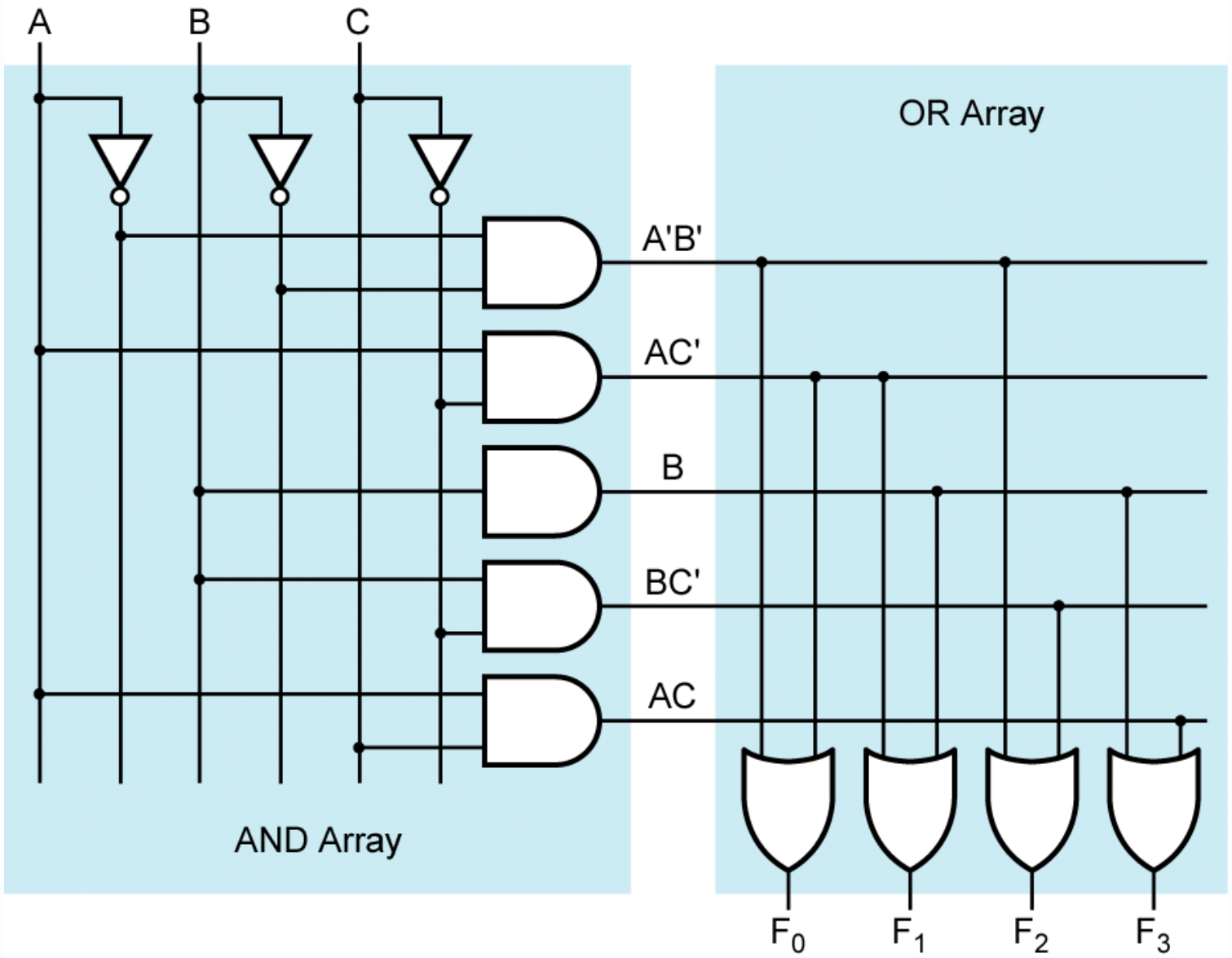


$$F_0 = \overline{A}\,\overline{B} + A\overline{C}$$

UNIVERSITY OF LIVERPOOL

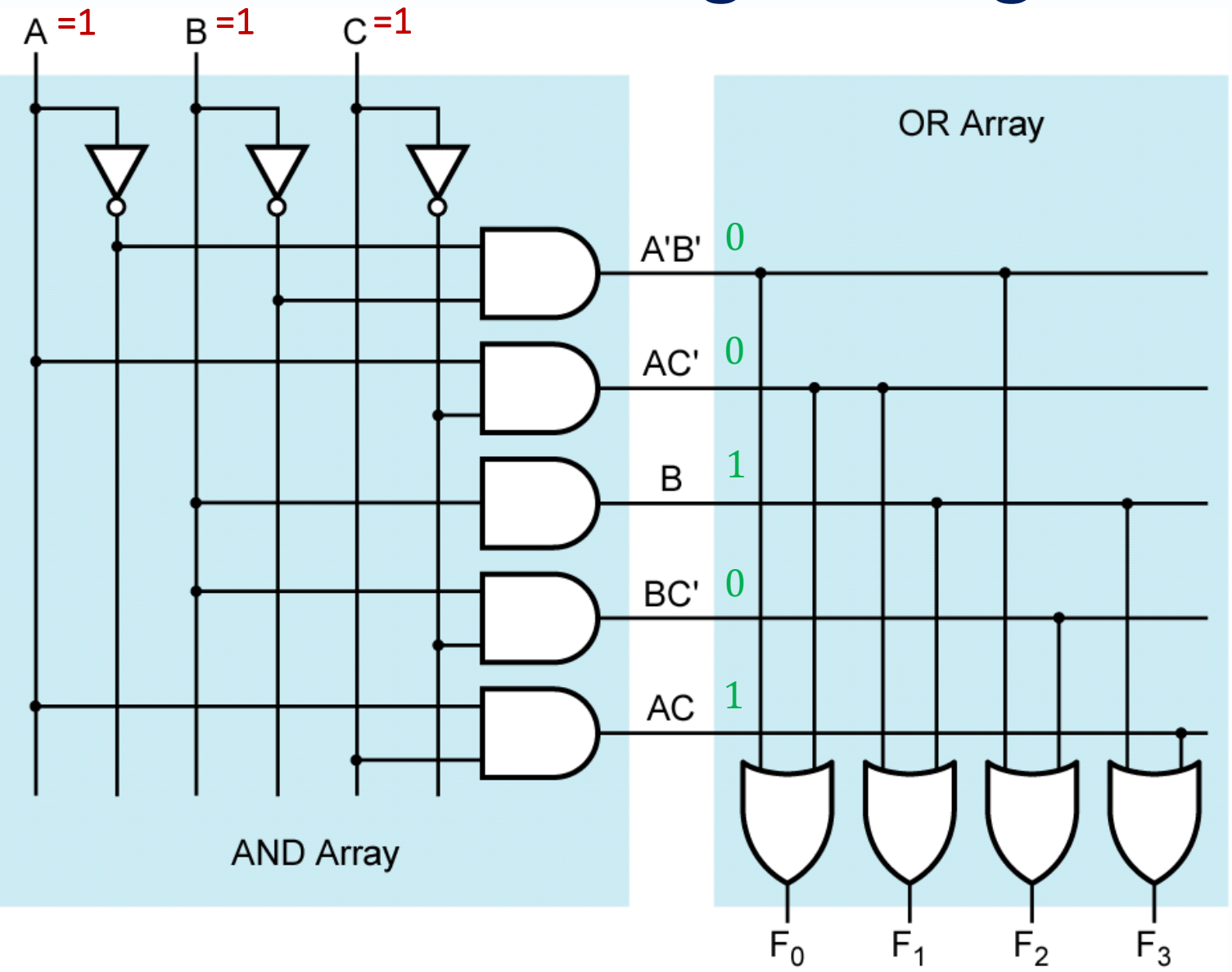
AND-OR Array Equivalent Circuit

$$F_0 = A'B' + AC'$$
$$F_1 = B + AC'$$
$$F_2 = A'B' + BC'$$
$$F_3 = B + AC$$



Following the logic: if *A,B,C = 1,1,1*

$$F_0 = A'B' + AC'$$
$$F_1 = B + AC'$$
$$F_2 = A'B' + BC'$$
$$F_3 = B + AC$$



$$A' \cdot B' = 0 \cdot 0 = 0$$
$$A \cdot C' = 1 \cdot 0 = 0$$
$$B = 1$$
$$B \cdot C' = 1 \cdot 0 = 0$$
$$A \cdot C = 1 \cdot 1 = 1$$

$$F_0 = A'B' + AC' = 0 + 0 = \mathbf{0}$$
$$F_1 = B + AC' = 1 + 0 = \mathbf{1}$$
$$F_2 = A'B' + BC' = 0 + 0 = \mathbf{0}$$
$$F_3 = B + AC = 1 + 1 = \mathbf{1}$$

Output: 0 1 0 1

UNIVERSITY OF LIVERPOOL

PLA Realization of Equations

Realize the following functions with a PLA

$$F_1 = A'BD + ABD + AB'C' + B'C$$

$$F_2 = C + A'BD$$

$$F_3 = BC + AB'C' + ABD$$

Making a PLA Table:

Inputs side: this variable, in this product term, is….

complemented – 0

not complemented – 1

not present – -

Outputs side: is the product term in the function?

does not appear in function – 0

appears in function – 1

PLA Table

PRODUCT TERM	INPUTS				OUTPUTS		
	A	B	C	D	F_1	F_2	F_3
$A'BD$	0	1	-	1	1	1	0
ABD	1	1	-	1	1	0	1
$AB'C'$	1	0	0	-	1	0	1
$B'C$	-	0	1	-	1	0	0
C	-	-	1	-	0	1	0
BC	-	1	1	-	0	0	1

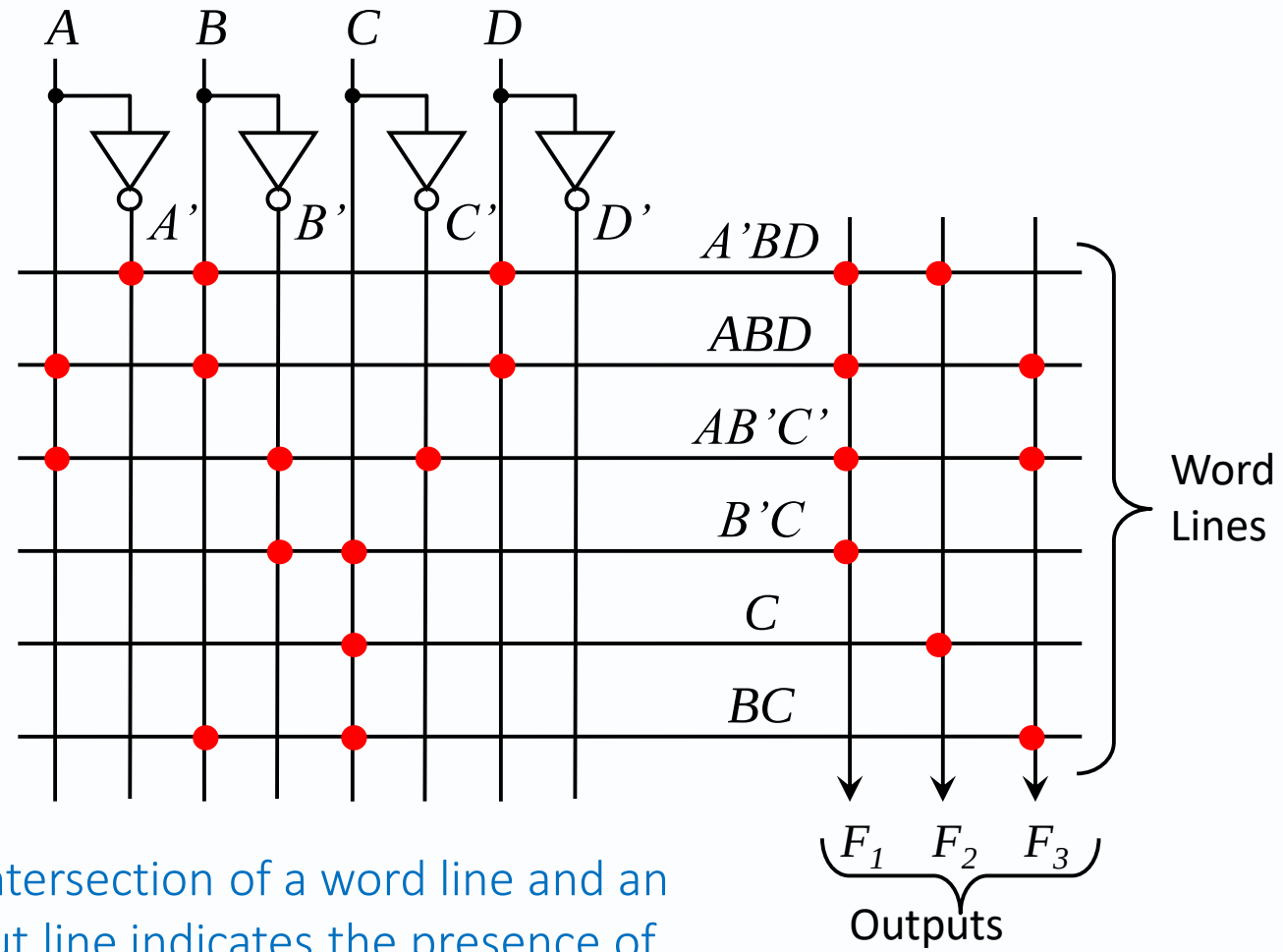
PLA Realization of Equations

$F_1 = A'BD + ABD + AB'C' + B'C$

$F_2 = C + A'BD$

$F_3 = BC + AB'C' + ABD$

PRODUCT TERM	INPUTS				OUTPUTS		
	A	*B*	*C*	*D*	F_1	F_2	F_3
A'BD	0	1	-	1	1	1	0
ABD	1	1	-	1	1	0	1
AB'C'	1	0	0	-	1	0	1
B'C	-	0	1	-	1	0	0
C	-	-	1	-	0	1	0
BC	-	1	1	-	0	0	1



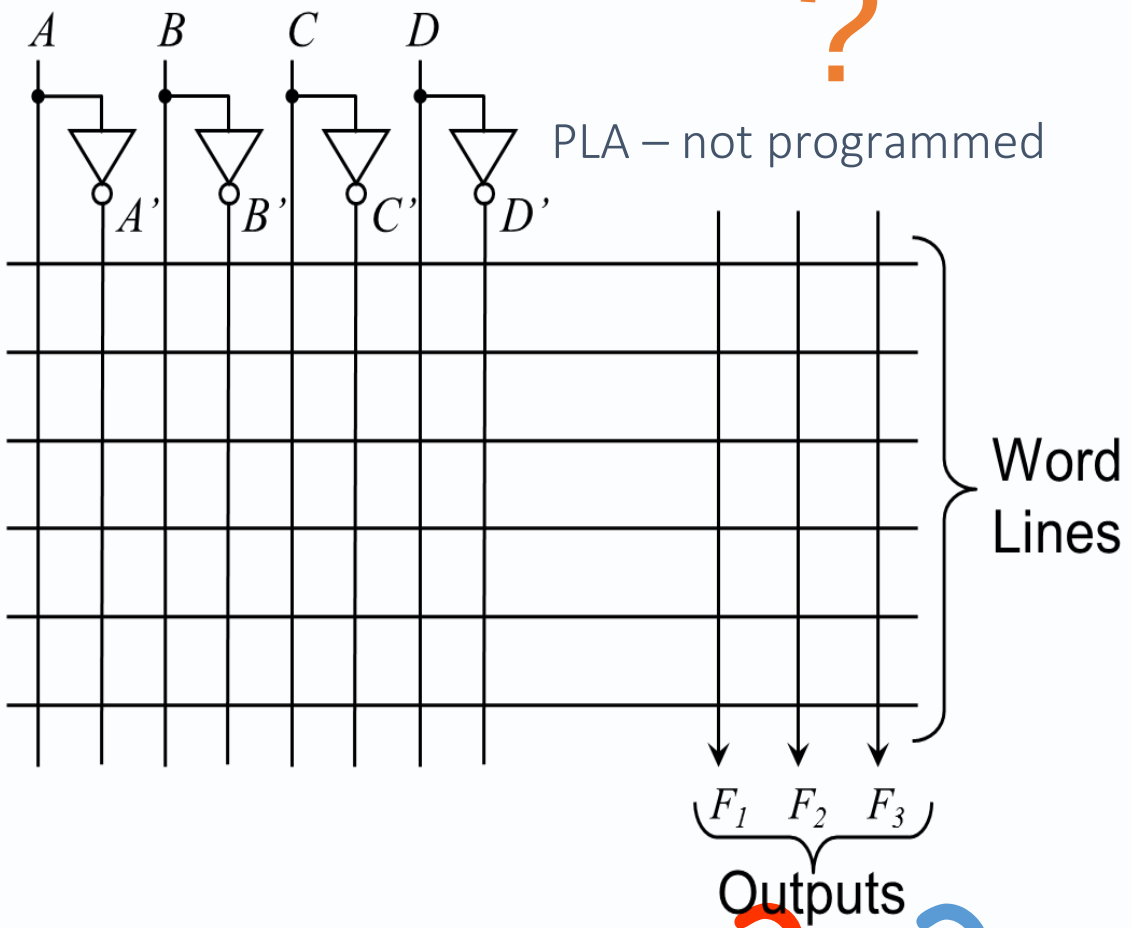
- A dot at the intersection of a word line and an input or output line indicates the presence of a switching element in the array.

Question

- Design a circuit with outputs F_1, F_2 and F_3 using a PLA

$$F_1 = BC' + A'D + B'D$$
$$F_2 = A'C' + B'D + CD$$
$$F_3 = BC' + AC'D$$

INPUTS				OUTPUTS		
A	B	C	D	F_1	F_2	F_3



UNIVERSITY OF LIVERPOOL

PLA relevance?

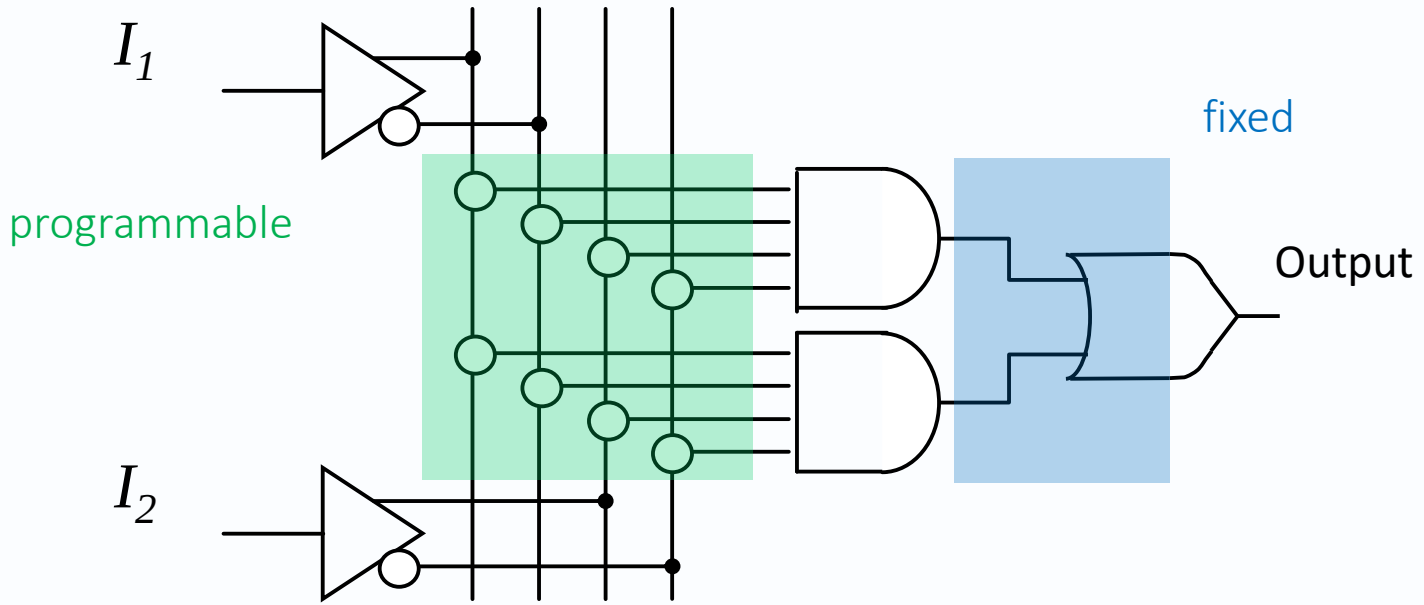
- Similarly to ROM, PLA are available as:
  - Mask-programmable
    - Programmed at manufacture
  - Field-programmable
    - Programmable interconnection points
    - Electronic charges store a pattern in the AND and OR arrays
- Small number of input variables: PROM might be more economical
- Large number of input variables: PLAs often more economical
- E.g. 8 functions of 16 variables → PROM: 16 million 8-bit words

→ decompose the functions to use a number of smaller PROMs

- → Single PLA (potentially)

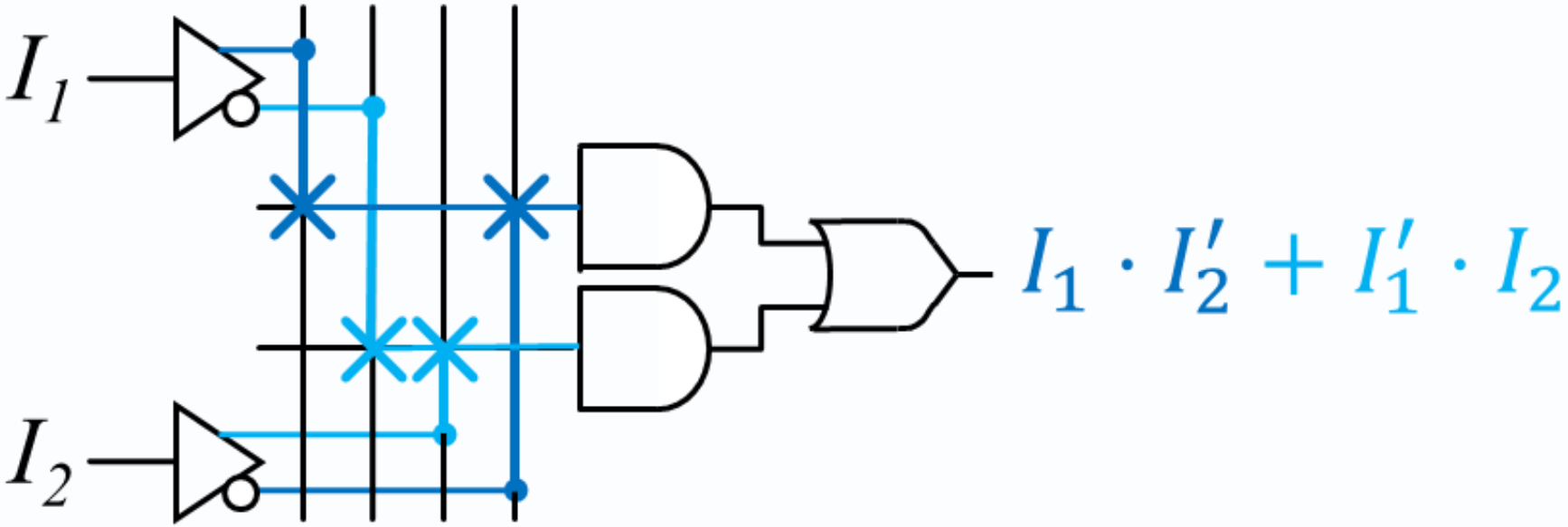
Programmable Array Logic (PAL)

- PAL is a special case of PLA in which:
  - AND array is programmable
  - OR array is fixed
- PAL is less expensive than PLA and easier to program

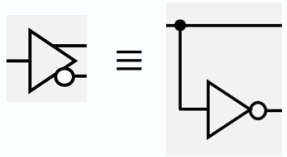


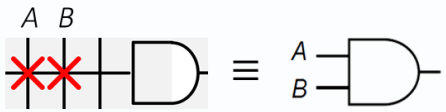
Programmable Array Logic (PAL)

- A buffer is used for the inputs as they drive many AND gates.
- For a given type of PAL the number of AND gates that feed each output OR gate is fixed and limited.



Symbols:





Full 1 bit 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_{in} = Carry input
C_{out} = Carry output

$$C_{out} = X \cdot C_{in} + Y \cdot C_{in} + X \cdot Y$$

$$Sum = X' \cdot Y' \cdot C_{in} + X' \cdot Y \cdot C'_{in} + X \cdot Y' \cdot C'_{in} + X \cdot Y \cdot C_{in}$$

Full 1 bit 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

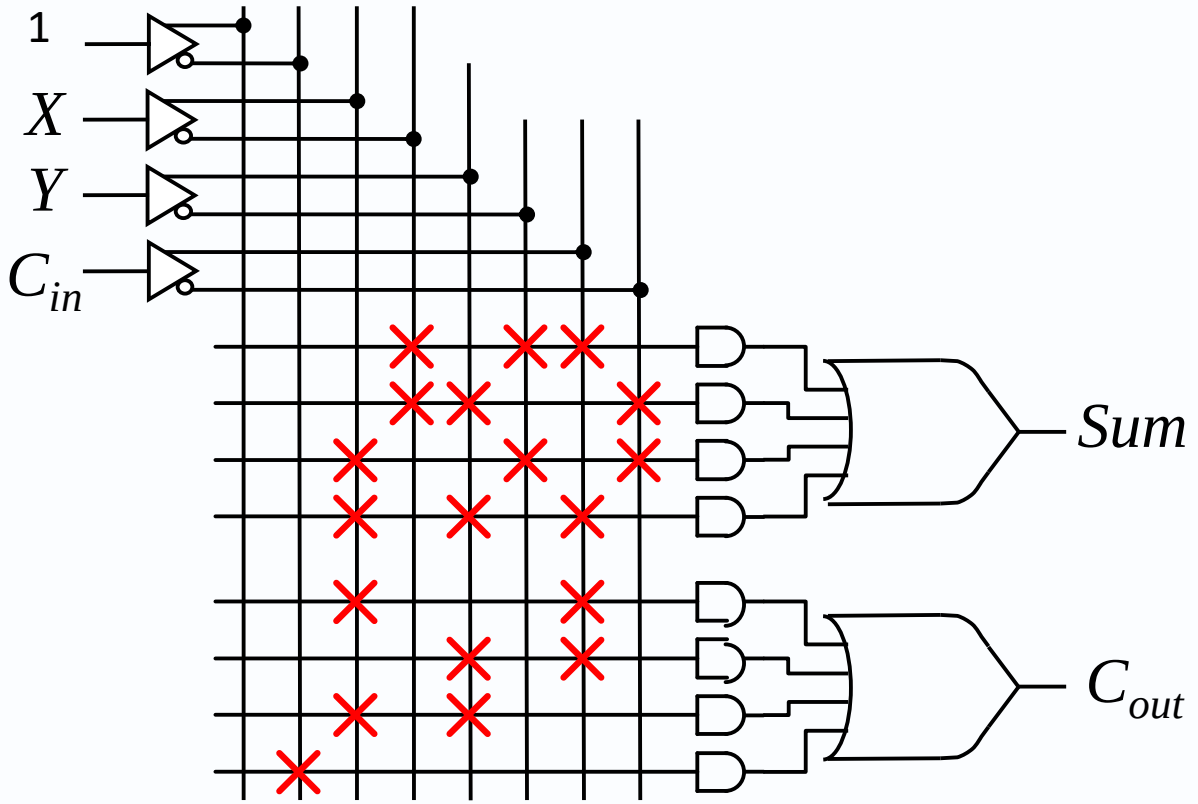
C_{in} = Carry input
C_{out} = Carry output

$$C_{out} = X \cdot C_{in} + Y \cdot C_{in} + X \cdot Y$$

Note: $X \cdot C_{in} + Y \cdot C_{in} + X \cdot Y \cdot C_{in}$ simplifies to $X \cdot C_{in} + Y \cdot C_{in}$ (Try it!)

$$Sum = X' \cdot Y' \cdot C_{in} + X' \cdot Y \cdot C'_{in} + X \cdot Y' \cdot C'_{in} + X \cdot Y \cdot C_{in}$$

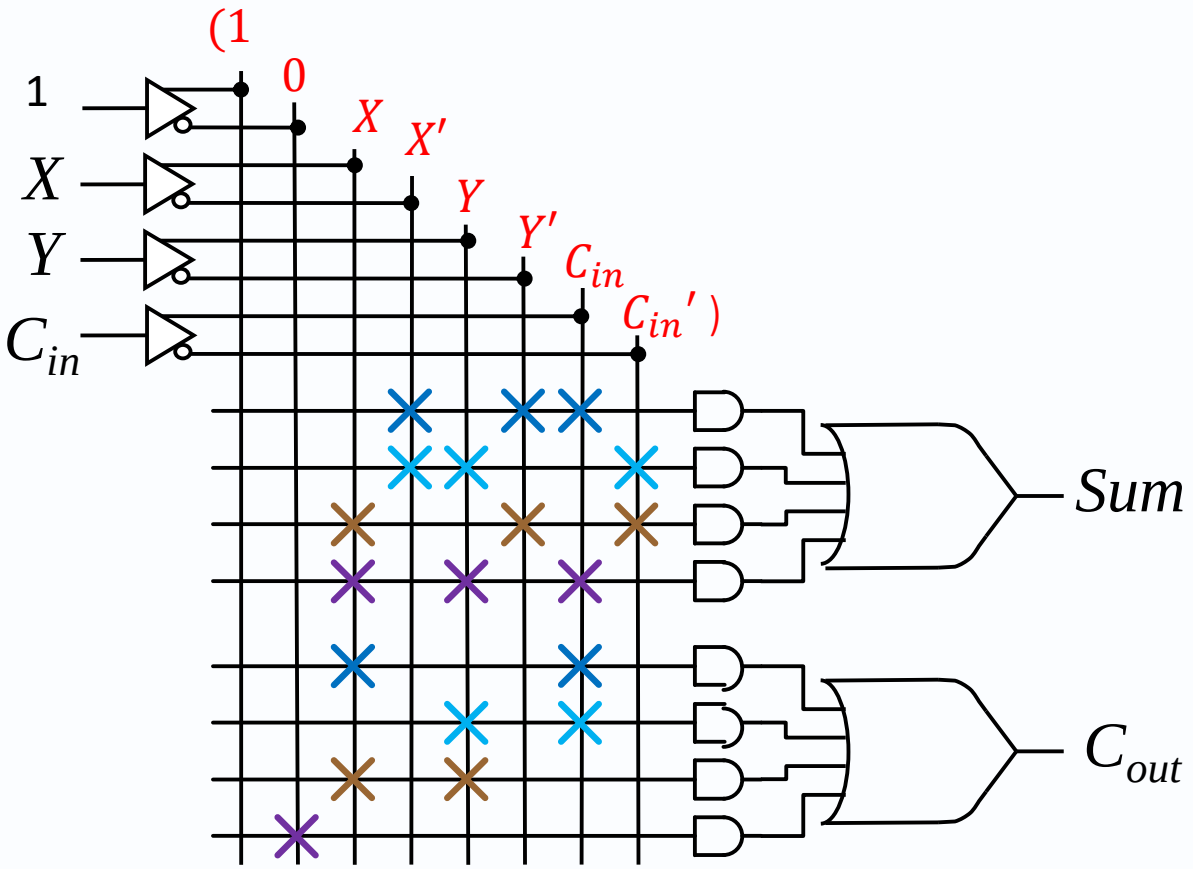
Full 1 bit Adder using a PAL



$$Sum = X' \cdot Y' \cdot C_{in} + X' \cdot Y \cdot C'_{in} + X \cdot Y' \cdot C'_{in} + X \cdot Y \cdot C_{in}$$

$$C_{out} = X \cdot C_{in} + Y \cdot C_{in} + X \cdot Y + 0$$

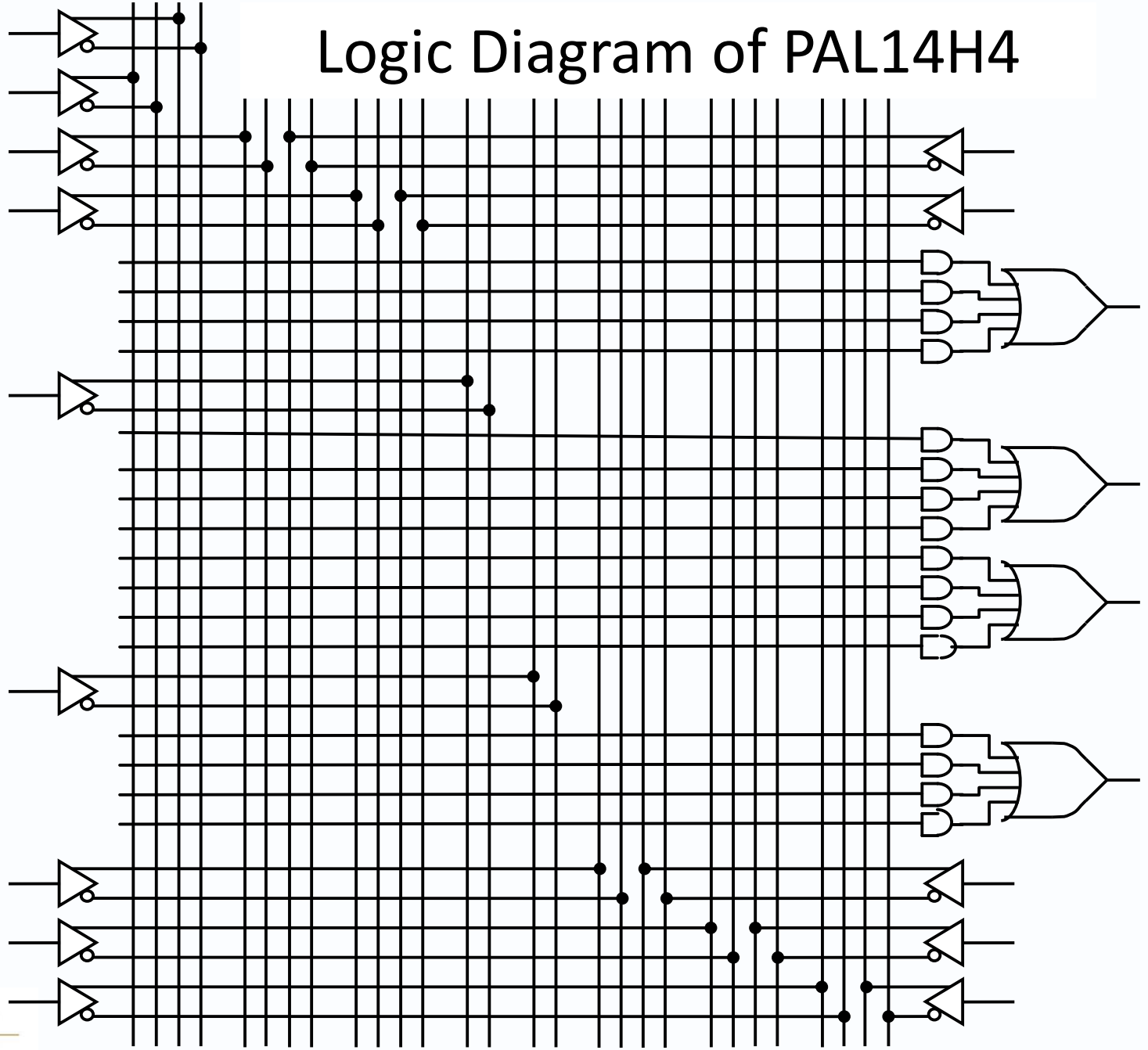
Full 1 bit Adder using a PAL



$$Sum = X' \cdot Y' \cdot C_{in} + X' \cdot Y \cdot C'_{in} + X \cdot Y' \cdot C'_{in} + X \cdot Y \cdot C_{in}$$

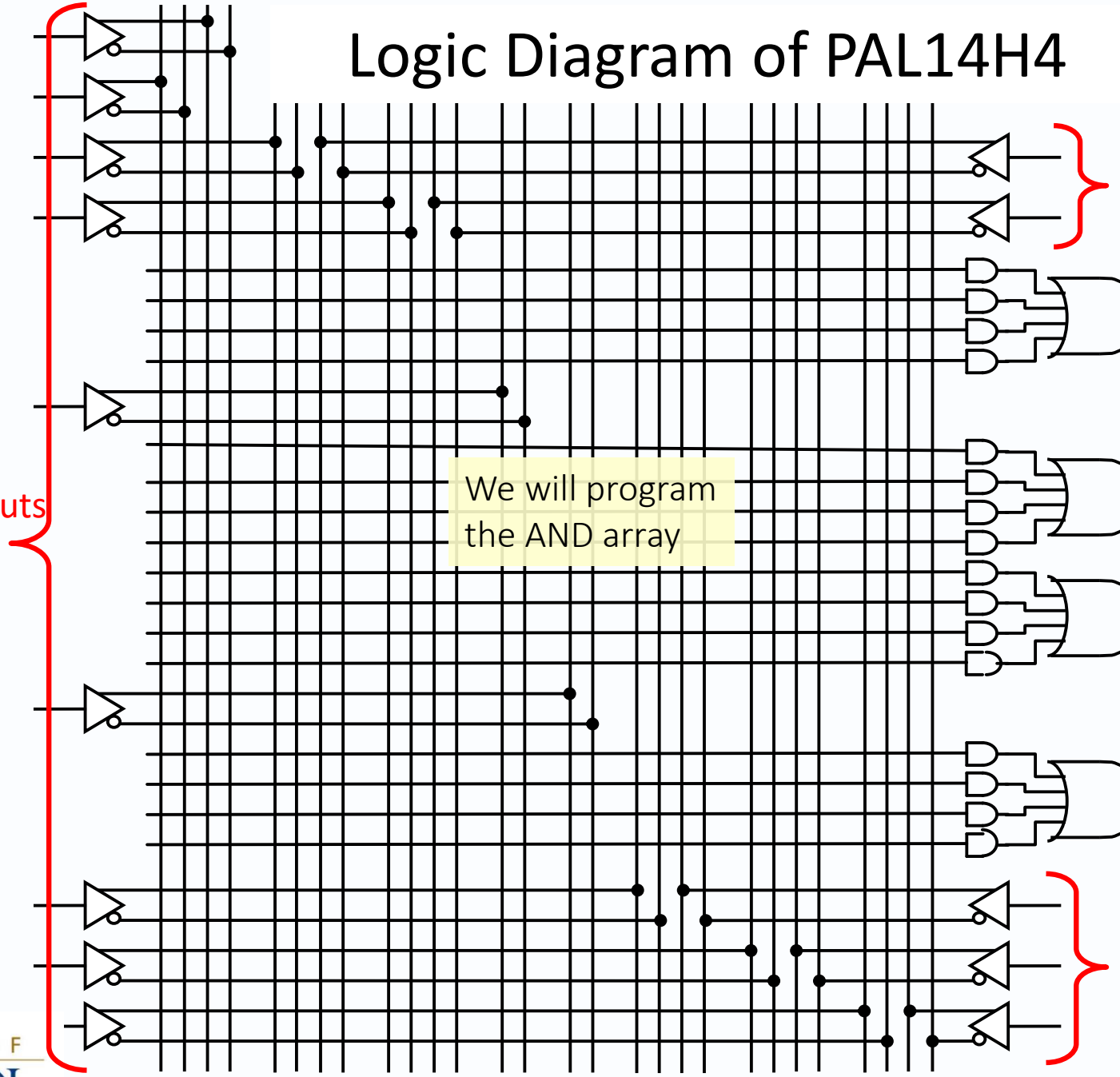
$$C_{out} = X \cdot C_{in} + Y \cdot C_{in} + X \cdot Y + 0$$

Logic Diagram of PAL14H4



(See p.2-10 of MMI Pal handbook on VITAL)

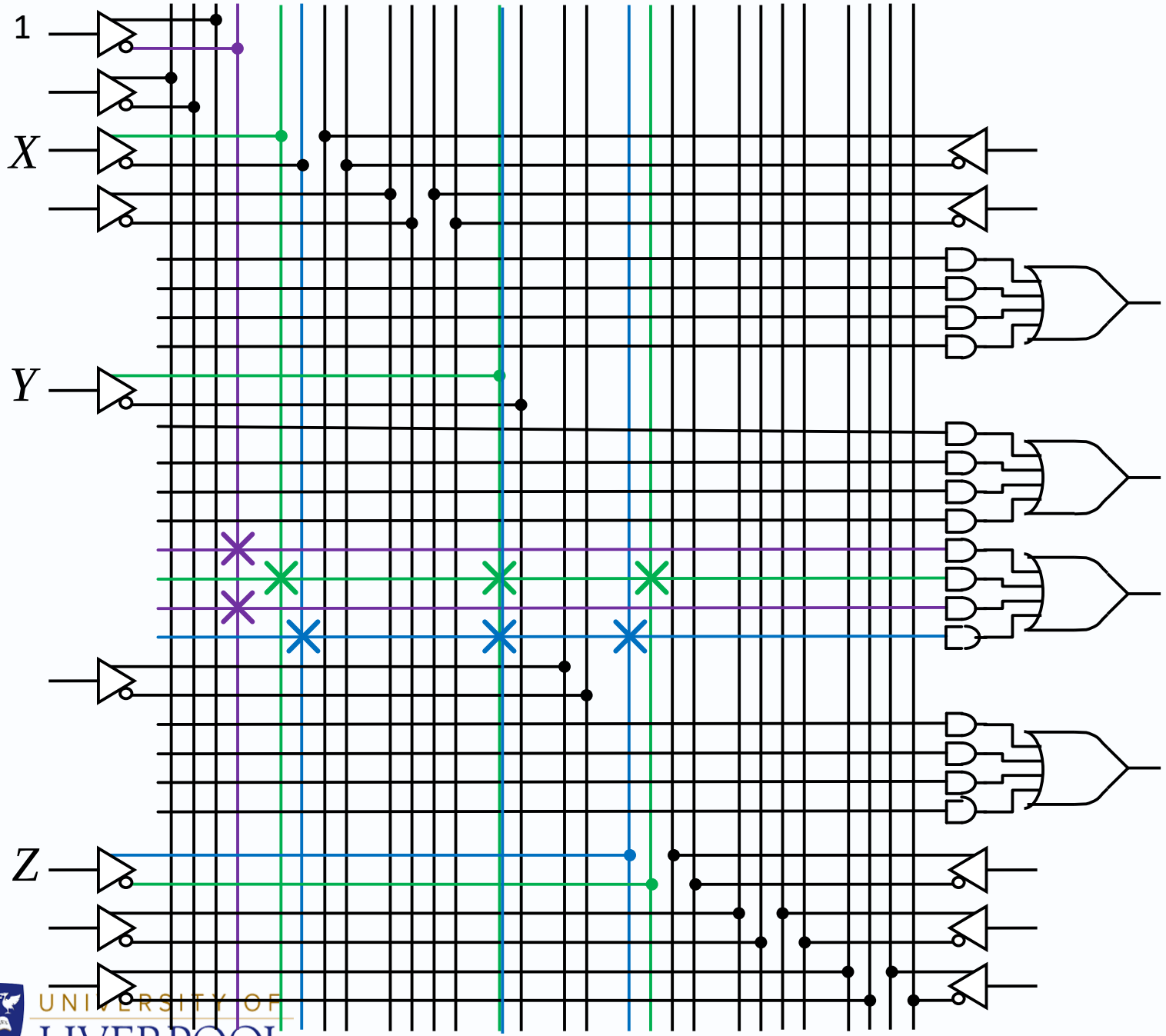
Logic Diagram of PAL14H4



We will assign inputs to variables of our choice

Note some inputs are on the right!

Similarly, we will 'name' the output functions



Example

Design a circuit with output

$A = XYZ' + X'YZ$

using a PAL14H4.

$A = (0 +)\ XYZ' + (0 +)\ X'YZ$

Question

- Design a circuit with outputs B, C and D using a PAL14H4.

$$B = U \cdot V' + V \cdot W' + U' \cdot W \cdot X + X' \cdot Y \cdot Z$$

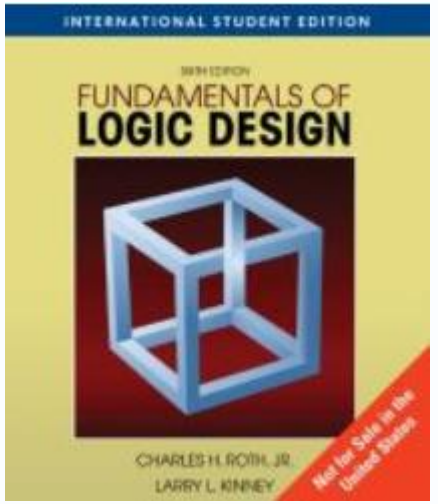
$$C = S' \cdot Y' + T \cdot W' + S \cdot V \cdot W$$

$$D = P \cdot Z + R' \cdot S \cdot W + P' \cdot R \cdot V + W \cdot X$$

Summary and suggested reading

- Three state buffers (Section 9.3)
- ROM
- PLA and PAL

Roth and Kinney *Fundamentals of Logic Design*



UNIVERSITY OF LIVERPOOL