CS2100
Recitation 7

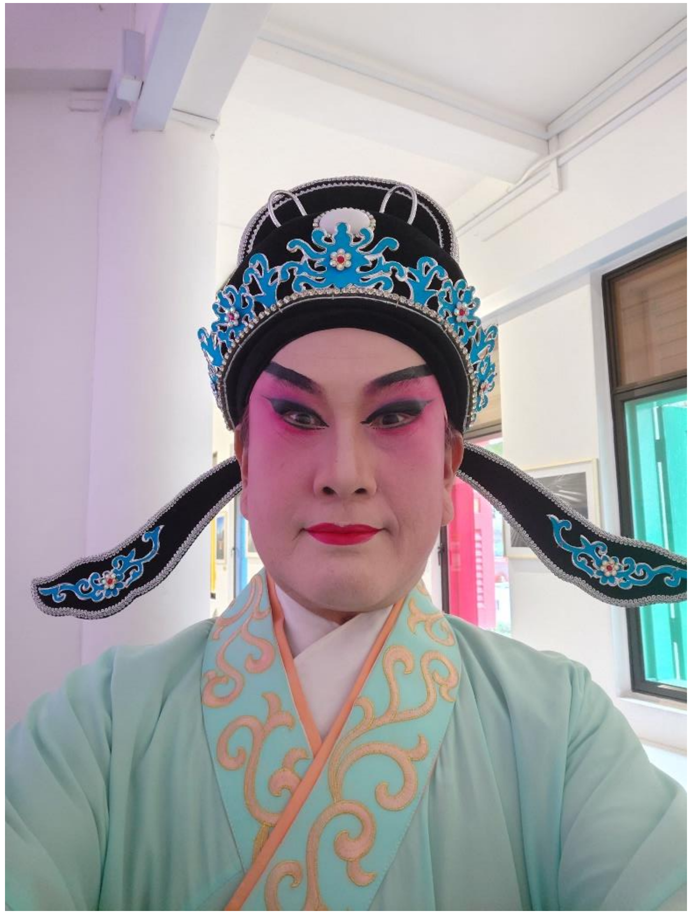
Boolean Algebra, Circuit Design, Simplification

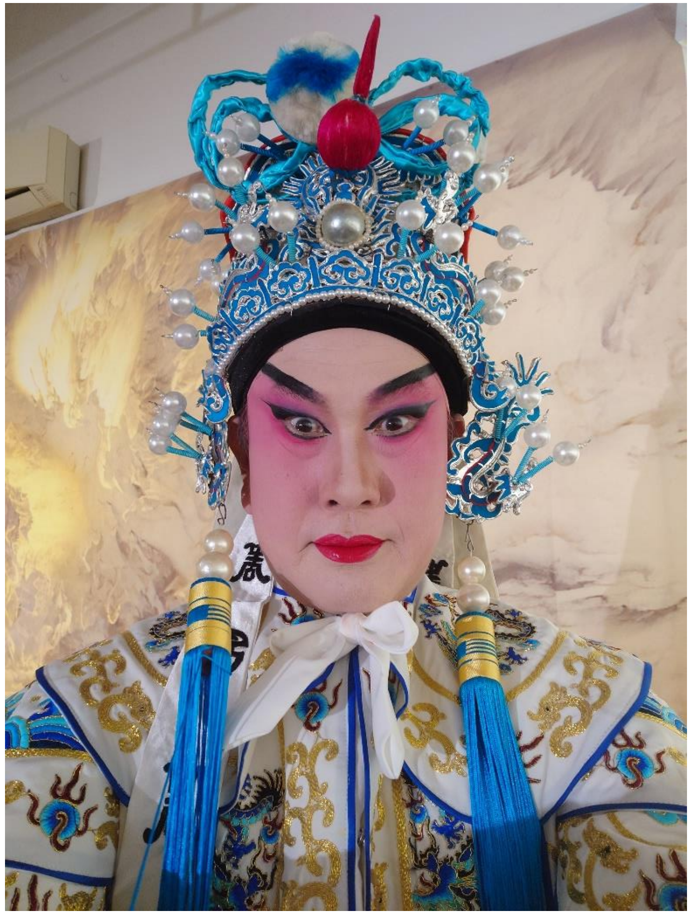
10 March 2025

Aaron Tan

NUS National University of Singapore | School of Computing

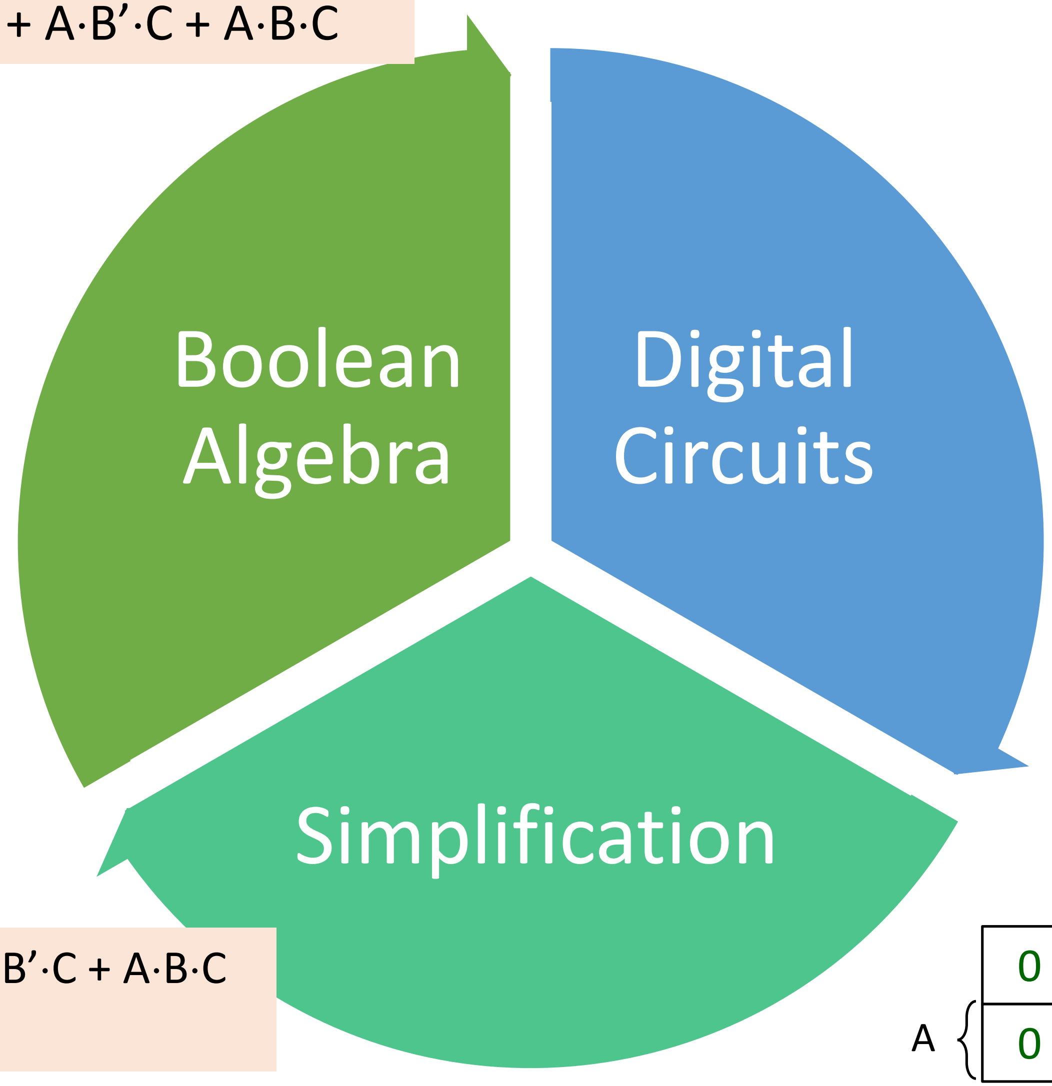
Who is this?

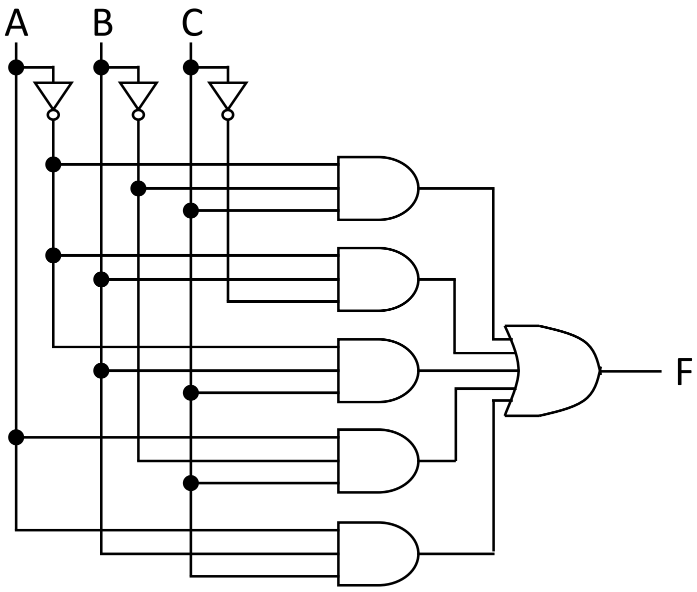


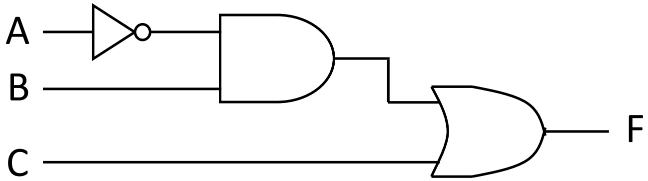


$$F = \Sigma m(1,2,3,5,7)$$
$$= A'\cdot B'\cdot C + A'\cdot B\cdot C' + A'\cdot B\cdot C + A\cdot B'\cdot C + A\cdot B\cdot C$$

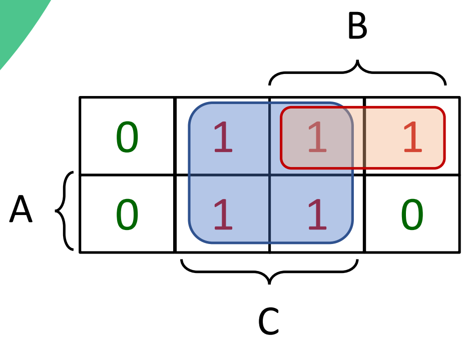
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1







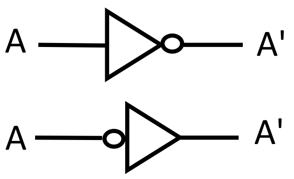
$$F = A'\cdot B'\cdot C + A'\cdot B\cdot C' + A'\cdot B\cdot C + A\cdot B'\cdot C + A\cdot B\cdot C$$
$$= C + A'\cdot B$$



$F = C + A'\cdot B$

NOT (‘)

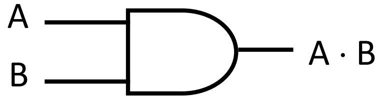
A	A'
0	1
1	0



Inverter = NOT gate

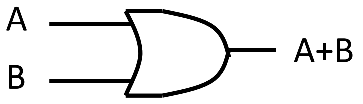
AND (·)

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1



OR (+)

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1



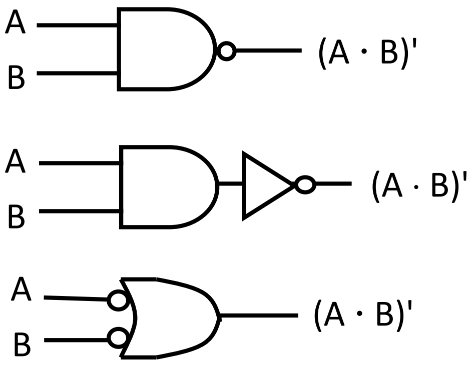
0 = FALSE; 1 = TRUE

Order of precedence:
NOT > AND > OR

- Examples:
  - A · B + C = (A · B) + C
  - X + Y' = X + (Y')
  - P + Q' · R = P + ((Q') · R)
- Parentheses to overwrite precedence. Examples:
  - A · (B + C) [without parenthesis → (A·B)+C]
  - (P + Q)' · R [without parenthesis → P+((Q’)·R)]

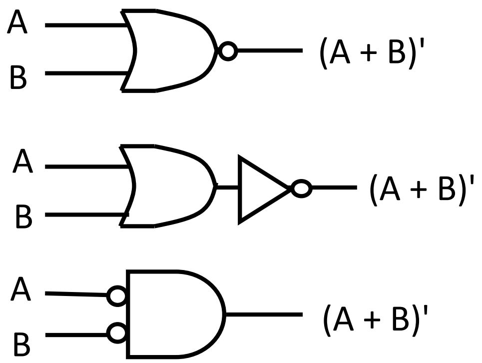
NAND

A	B	(A · B)'
0	0	1
0	1	1
1	0	1
1	1	0



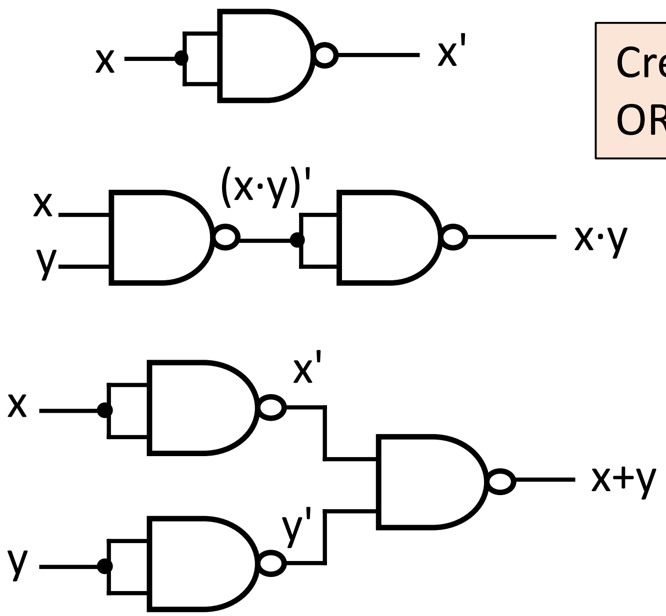
NOR

A	B	(A + B)'
0	0	1
0	1	0
1	0	0
1	1	0



Universal Gates

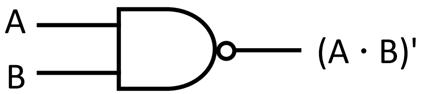
- The set {AND,OR,NOT} is a complete set of logic, ie. any Boolean function can be implemented using these 3 operations.
- {NAND} is also a complete set of logic. So is {NOR}.



Creating NOT, AND and OR from NAND gates.

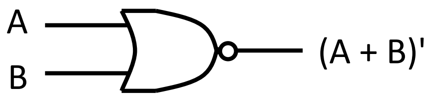
NAND

A	B	(A · B)'
0	0	1
0	1	1
1	0	1
1	1	0



NOR

A	B	(A + B)'
0	0	1
0	1	0
1	0	0
1	1	0

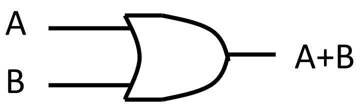


XOR/XNOR Gates

- Two other gates – XOR and XNOR – are useful for applications such as parity bit generation.

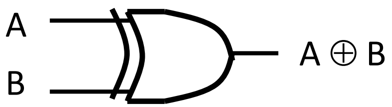
OR (+)

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1



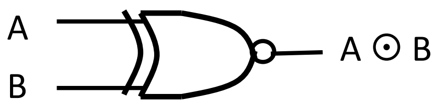
XOR ($\oplus$)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



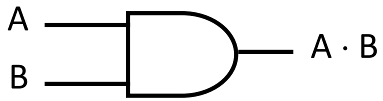
XNOR ($\odot$)

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1



AND (·)

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1



- For AND operations, <u>must include</u> the AND operator · (instead of omitting it)
  - Example: Write ***A·B*** instead of just ***AB***
  - Why? Writing ***AB*** could mean that it is a 2-bit value.

Laws/theorems of Boolean Algebra (CS2100)

Identity laws	
$A+0 = 0+A = A$	$A \cdot 1 = 1 \cdot A = A$
Inverse/complement laws	
$A+A' = A'+A = 1$	$A \cdot A' = A' \cdot A = 0$
Commutative laws	
$A+B = B+A$	$A \cdot B = B \cdot A$
Associative laws	
$A+B+C = A+(B+C) = (A+B)+C$	$A \cdot B \cdot C = A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributive laws	
$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$	$A+(B \cdot C) = (A+B) \cdot (A+C)$

Idempotency	
$X+X = X$	$X \cdot X = X$
One element / Zero element	
$X+1 = 1+X = 1$	$X \cdot 0 = 0 \cdot X = 0$
Involution	
$(X')' = X$	
Absorption 1	
$X+(X \cdot Y) = X$	$X \cdot (X+Y) = X$
Absorption 2	
$X+(X' \cdot Y) = X+Y$	$X \cdot (X'+Y) = X \cdot Y$
DeMorgans' (can be generalised to > 2 variables)	
$(X+Y)' = X' \cdot Y'$	$(X \cdot Y)' = X'+Y'$
Consensus	
$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$	$(X+Y) \cdot (X'+Z) \cdot (Y+Z) = (X+Y) \cdot (X'+Z)$

Duality

Given a Boolean equality that is true, if the AND/OR operators and identity elements 0/1 in the equality are interchanged, the new equality is also true.

Boolean Algebra: Standard Forms

Why should we know the different forms?

Because of their practical value.

- A numerical value can be written in different forms.

  Eg: $12.34 = 1.234 \times 10 = 0.01234 \times 10^3 = 1234 \times 10^{-2} = \frac{617}{50}$

- Likewise, a Boolean expression can be written in different forms.
- Eg: A Boolean function F(A,B,C) is given as A·B + A'·C + A'·B'
- Then F(A,B,C) = A·B + A'·C + A'·B'

  = A·B + B·C + A'·B'

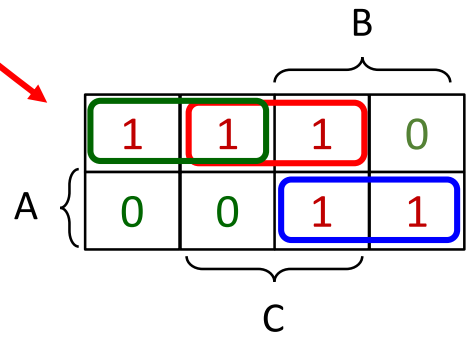
  = A·B + B·(A'+C)

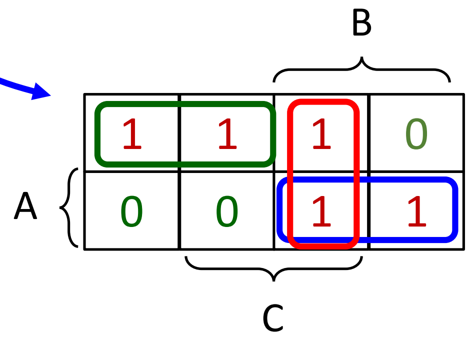
  = A'·B'·C' + A'·B'·C + A'·B·C + A·B·C' + A·B·C

  = m0 + m1 + m3 + m6 + m7

Sum-of-products (SOP) expression

Sum-of-minterms expression





Definitions! (Let’s do it the CS1231S way!)

Literal

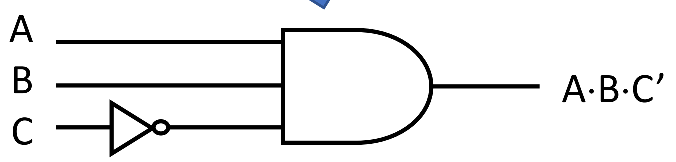
- A Boolean variable on its own or in its complemented form
- Examples:

x	x'	y	A'

Product term

- A single literal or a logical product (AND) of several literals
- Examples:

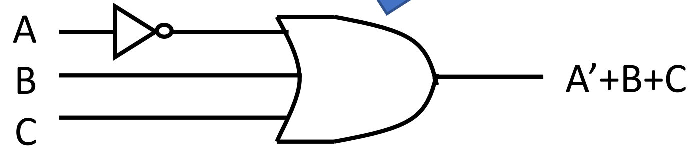
x	x·y·z'	A'·B	A·B·C’	d·g'·v·w



Sum term

- A single literal or a logical sum (OR) of several literals
- Examples:

y'	x+y’+z	A+B	A’+B+C	c+d+h’+j



Definitions! (Let's do it the CS1231S way!)

Literal
- A Boolean variable on its own or in its complemented form

Product term
- A single literal or a logical product (AND) of several literals

Sum term
- A single literal or a logical sum (OR) of several literals

Sum-of-Products (SOP) expression

- A product term or a logical sum (OR) of several product terms
- Examples:

x
$x + y \cdot z'$
$x \cdot y' + x' \cdot y \cdot z$
$A \cdot B + A' \cdot B'$
$A + B' \cdot C + A \cdot C' + C \cdot D$

Product-of-Sums (POS) expression

- A sum term or a logical product (AND) of several sum terms
- Examples:

x'
$x \cdot (y+z')$
$(x+y') \cdot (x'+y+z)$
$(A+B) \cdot (A'+B')$
$(A+B+C) \cdot D' \cdot (B'+D+E')$

Product term
- A single literal or a logical product (AND) of several literals

Sum term
- A single literal or a logical sum (OR) of several literals

SOP expression
- A product term or a logical sum (OR) of several product terms

POS expression
- A sum term or a logical product (AND) of several sum terms

Put a tick ✓ or cross ✗

	Expression	*SOP?*	*POS?*
(1)	A'·B + A·B'	✓	✗
(2)	(X+Y')·(X'+Y)·(X'+Z')	✗	✓
(3)	(B + D·E)·A·C'	✗	✗
(4)	P + Q' + R'	✓	✓
(5)	W·X'·Y'·Z	✓	✓
(6)	W·X'·Y + V·(X·Z + W')	✗	✗

Why SOP and POS?

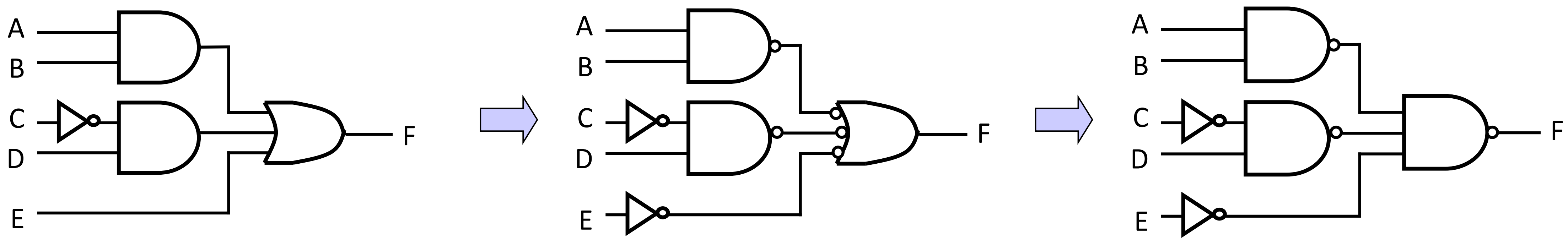
- An SOP expression can be <u>easily</u> implemented using a 2-level AND-OR circuit, or a 2-level NAND circuit.

(Likewise, a POS expression can be easily implemented using a 2-level OR-AND circuit, or a 2-level NOR circuit.)

- Example: $F = A \cdot B + C' \cdot D + E$

2-level AND-OR circuit

2-level NAND circuit



Minterms and Maxterms

- A minterm of n variables is a product term that contains n literals from all the variables.
- A maxterm of n variables is a sum term that contains n literals from all the variables.
- Examples: On 2 variables **x** and **y**

minterms | Function F(x,y) | maxterms

minterms	
x'·y'	**= m0**
x'·y	**= m1**
x·y'	**= m2**
x·y	**= m3**

maxterms	
M0 =	**x+y**
M1 =	**x+y’**
M2 =	**x’+y**
M3 =	**x’+y’**

- In general, with n variables we have up to 2^n minterms and 2^n maxterms.
- Note that mx and Mx are complement of each other (eg: m2 = M2’)

Minterms and Maxterms

- Ability to convert minterms and maxterms from its Boolean expression to its notation (and vice versa) is important.
- Test yourself with the following quiz, assuming that you are given a Boolean function F on 4 variables A, B, C, D, i.e. F(A,B,C,D).

	Minterm	***Number notation***
(1)	A'·B'·C·D	m3
(2)	**A·B'·C·D'**	m10
(3)	**A·B'·C·D**	m11
(4)	A·B·C·D'	**m14**
(5)	A·B'·C'·D	**m9**

	Maxterm	***Numbern otation***
(1)	A+B+C'+D'	M3
(2)	**A'+B'+C+D'**	M13
(3)	**A+B+C+D**	M0
(4)	A+B+C'+D	**M2**
(5)	A'+B+C+D'	**M9**

Sum-of-minterms and Product-of-maxterms

- Canonical/normal form: a unique form of representation.
  - Sum-of-minterms expression = Canonical sum-of-products expression
  - Product-of-maxterms expression = Canonical product-of-sums expression

Sum-of-minterms:
Pick minterms where the function values are 1.

x	y	z	F1	F2	F3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0

$F1 = x \cdot y \cdot z' = m6$

$$F2 = x' \cdot y' \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z + x \cdot y \cdot z' + x \cdot y \cdot z$$
$$= m1 + m4 + m5 + m6 + m7$$
$$= \Sigma m(1,4,5,6,7) \text{ or } \Sigma m(1,4 - 7)$$

$$F3 = x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z$$
$$= m1 + m3 + m4 + m5$$
$$= \Sigma m(1,3,4,5) \text{ or } \Sigma m(1,3 - 5)$$

Sum-of-minterms and Product-of-maxterms

- Canonical/normal form: a unique form of representation.
  - Sum-of-minterms expression = Canonical sum-of-products expression
  - $\boxed{\text{Product-of-maxterms expression}}$ = Canonical product-of-sums expression

Product-of-maxterms:
Pick maxterms where the function values are 0.

x	y	z	F1	F2	F3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0

$$F2 = (x+y+z) \cdot (x+y'+z) \cdot (x+y'+z')$$
$$= M0 \cdot M2 \cdot M3 = \Pi M(0,2,3)$$

$$F3 = (x+y+z) \cdot (x+y'+z) \cdot (x'+y'+z) \cdot (x'+y'+z')$$
$$= M0 \cdot M2 \cdot M6 \cdot M7 = \Pi M(0,2,6,7)$$

Conversion between Sum-of-minterms and Product-of-maxterms

- Very easy!
- Example: Given F2(x,y,z) = Σm(1,4,5,6,7)

x	y	z	F2	F2'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Product-of-maxterms expression

F2(x,y,z) = ΠM(0,2,3)

Why?

- F2' = m0 + m2 + m3
  Therefore,
  F2 = (m0 + m2 + m3)'
  = m0' · m2' · m3' (by DeMorgan's)
  = M0 · M2 · M3 (as m*x*' = M*x*)

Boolean Algebra: Standard Forms

Why should we know the different forms?

Because of their practical value.

- A numerical value can be written in different forms.

  Eg: $12.34 = 1.234 \times 10 = 0.01234 \times 10^3 = 1234 \times 10^{-2} = \frac{617}{50}$

- Likewise, a Boolean expression can be written in different forms.
- Eg: A Boolean function F(A,B,C) is given as A·B + A'·C + A'·B'
- Then F(A,B,C) = A·B + A'·C + A'·B'

  = A·B + B·C + A'·B'

  Sum-of-products (SOP) expression

  = A·B + B·(A'+C)

  = A'·B'·C' + A'·B'·C + A'·B·C + A·B·C' + A·B·C

  = m0 + m1 + m3 + m6 + m7

  Sum-of-minterms expression

  = M2 · M4 · M5

  = (A+B'+C) · (A'+B+C) · (A'+B+C')

  Product-of-maxterms expression

  = (A+B'+C) · (A'+B)

  Product-of-sums (POS) expression

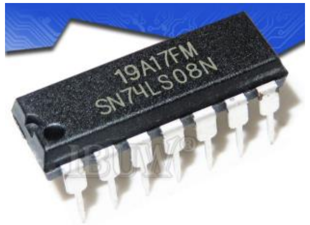
From an earlier slide:

Logic Trainer Labs! Starting next week.

Comments from many past semesters' students:

> Logic trainer labs are the most fun thing about CS2100!

Logic Trainer

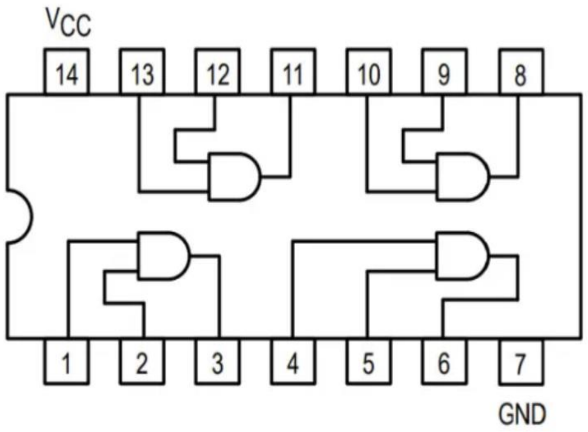


Lab 6: Intro to Logic Trainer

- Please be punctual.
- Your labTA will show you a demo.
- No report submission.
- Attendance 5 marks.

IC chips:

- Inverter: 74LS04
- 2-input AND: 74LS08
- 2-input OR: 74LS32
- 2-input NAND: 74LS00
- 2-input NOR: 74LS02



Why SOP and POS?

- An SOP expression can be <u>easily</u> implemented using a 2-level AND-OR circuit, or a 2-level NAND circuit.

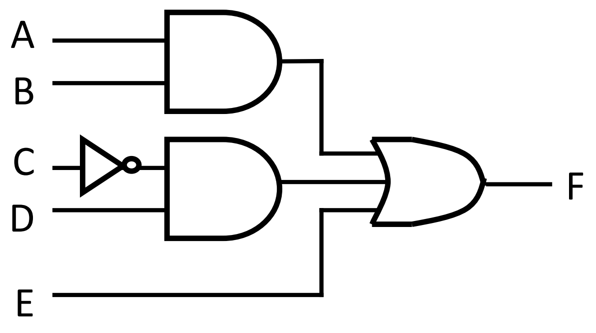
(Likewise, a POS expression can be easily implemented using a 2-level OR-AND circuit, or a 2-level NOR circuit.)

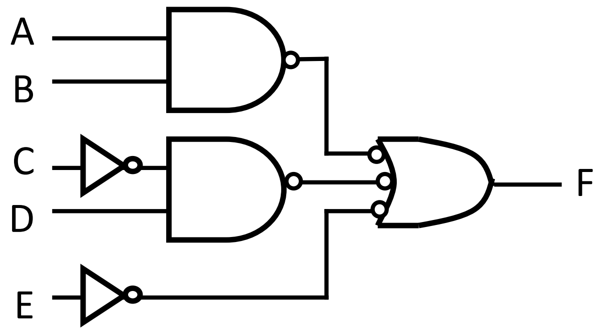
- Example: $F = A \cdot B + C' \cdot D + E$

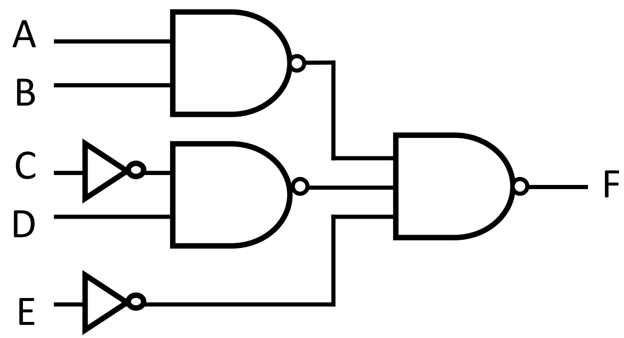
2-level AND-OR circuit

Recall this earlier slide?

2-level NAND circuit

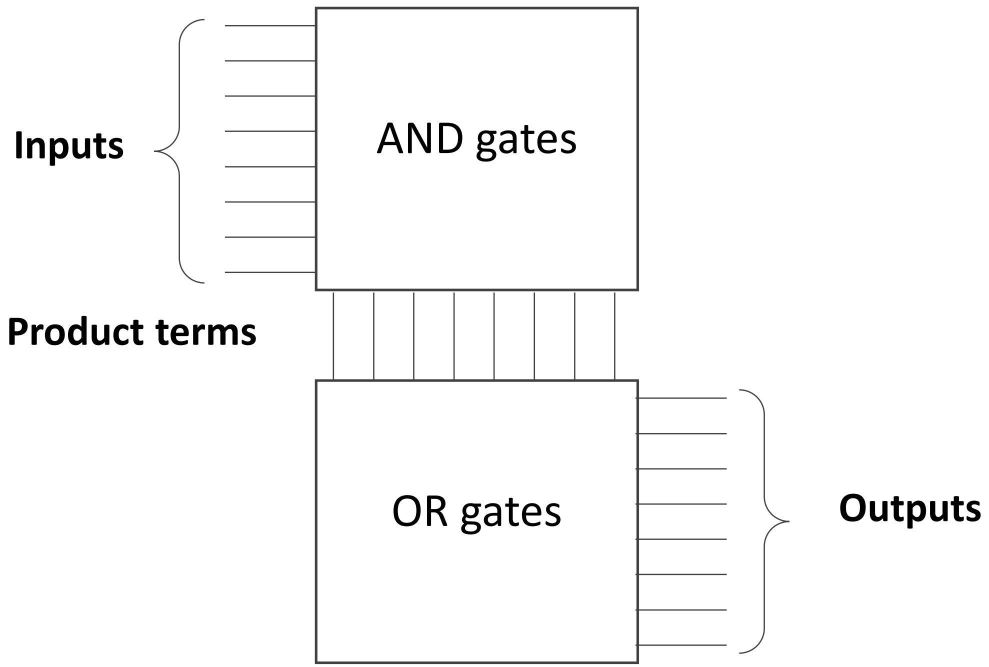






Programmable Array Logic (PLA)

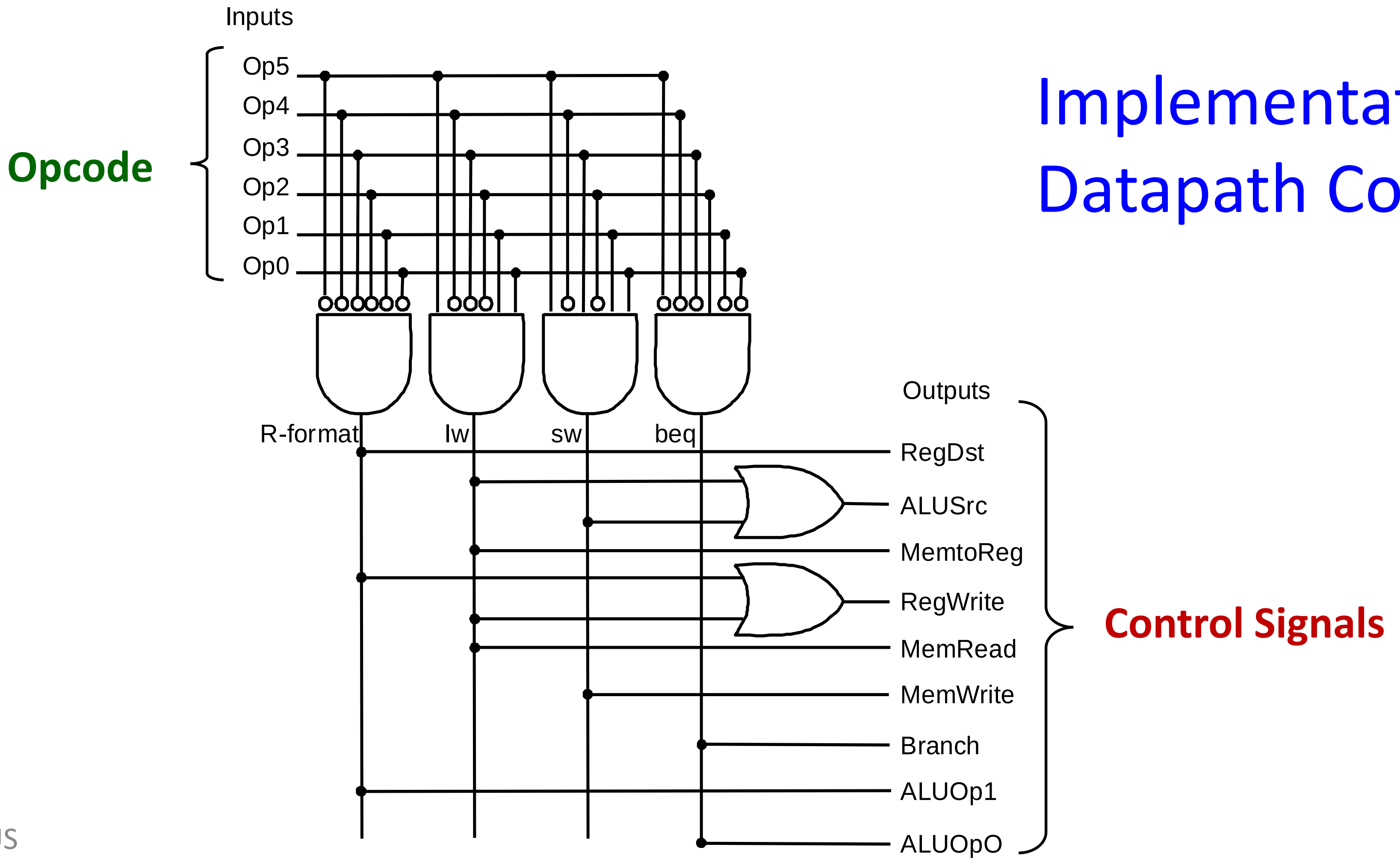
- A programmable integrated circuit – implements circuits for sum-of-products (SOP) expressions.
- 2 stages
  - AND gates = product terms
  - OR gates = outputs
- Connections between inputs and the planes can be 'burned'.



	Opcode					
	Op5	Op4	Op3	Op2	Op1	Op0
R-type	0	0	0	0	0	0
lw	1	0	0	0	1	1
sw	1	0	1	0	1	1
beq	0	0	0	1	0	0

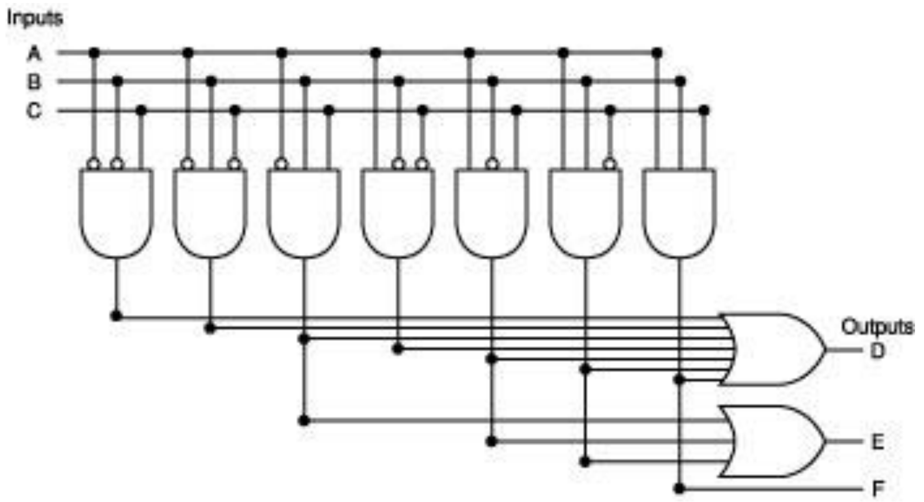
	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Implementation of Datapath Control

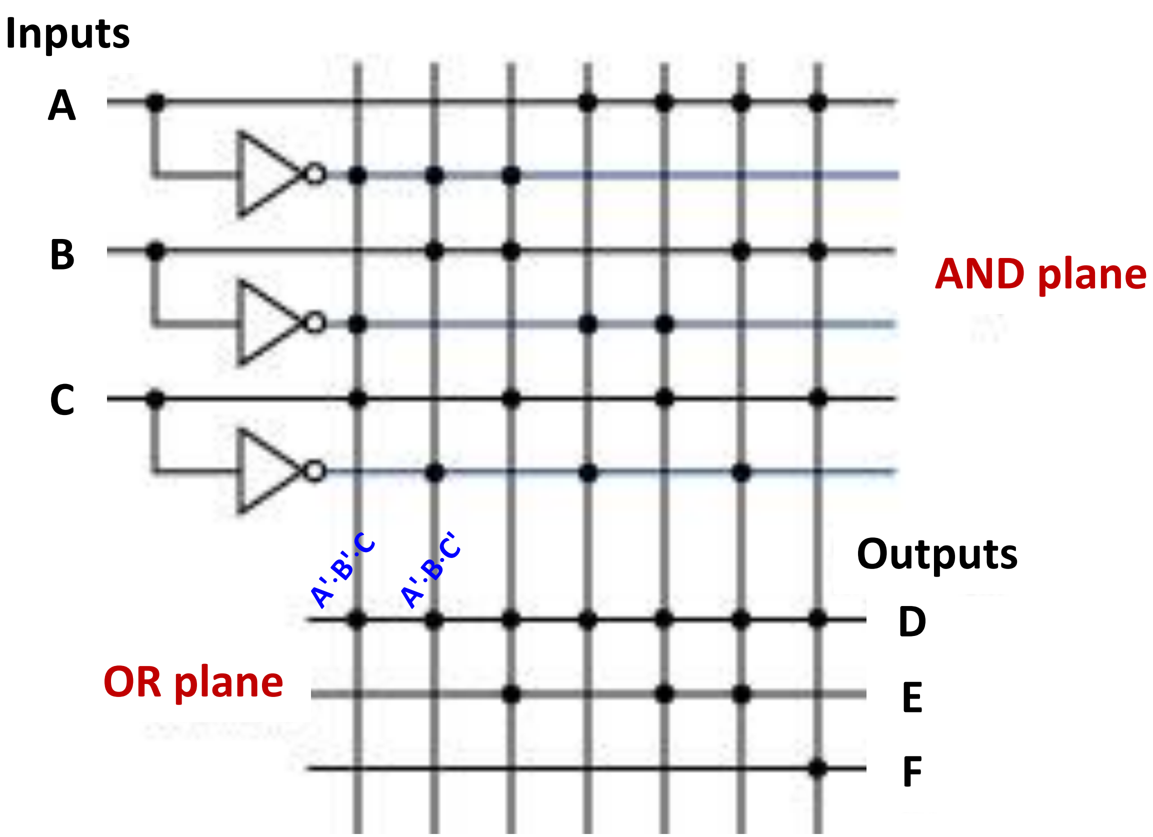


Programmable Array Logic (PLA): Example

Inputs			Outputs		
A	*B*	*C*	*D*	*E*	*F*
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1



- Simplified representation



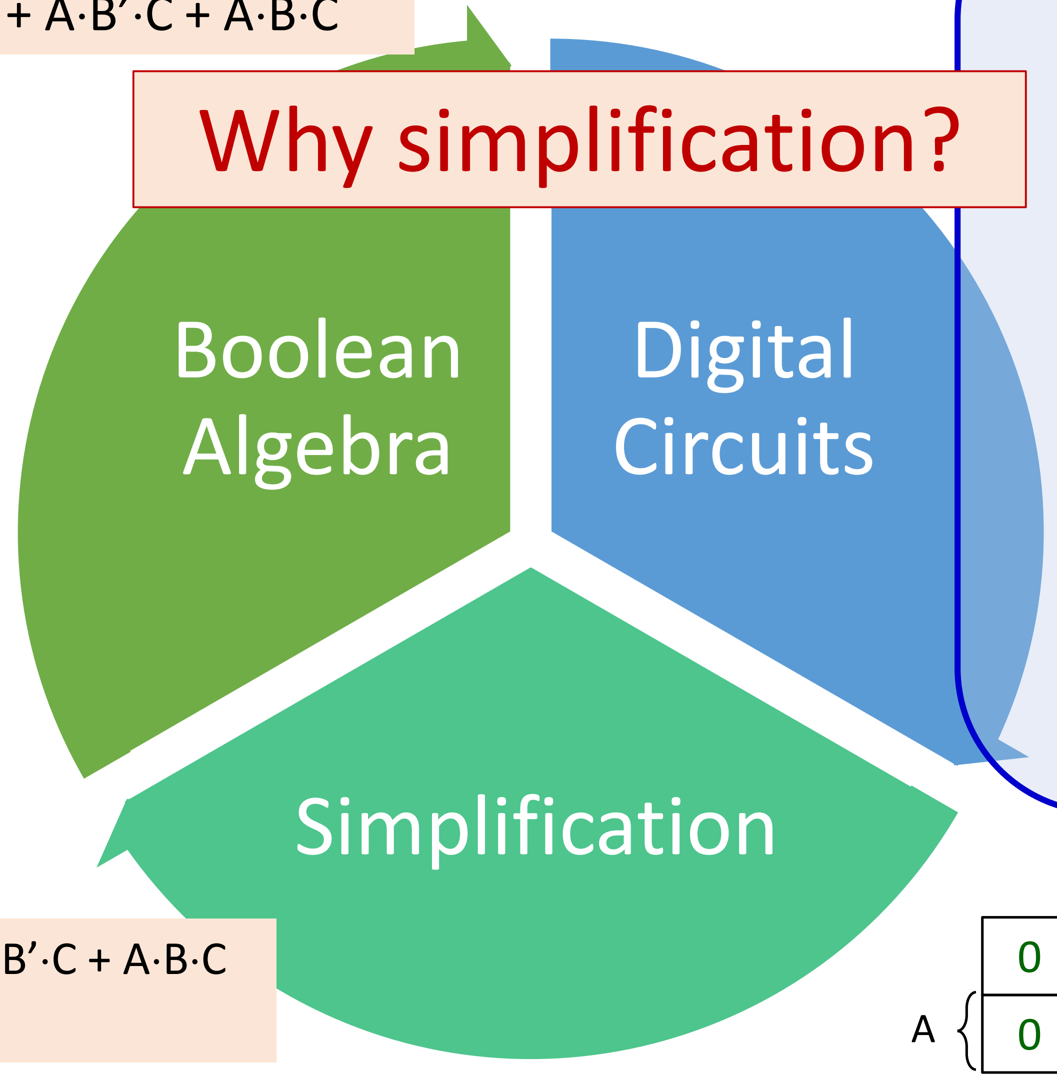
Simplification

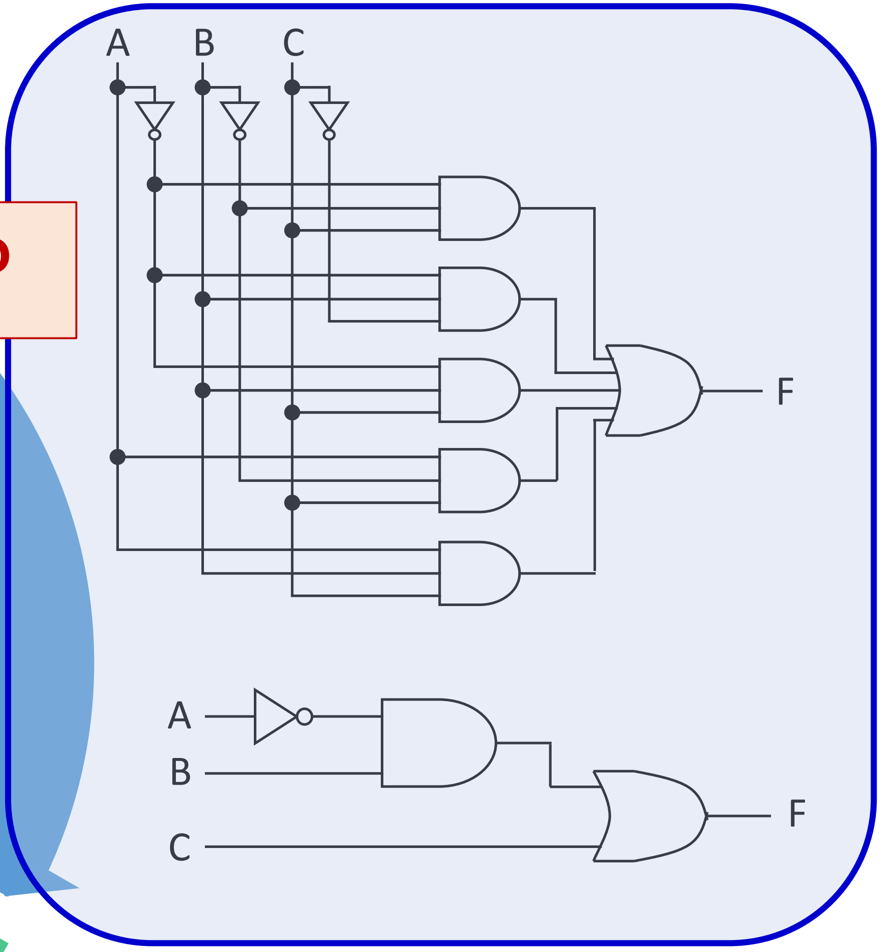
K-Maps

Why simplification?

$F = \Sigma m(1,2,3,5,7)$
$= A' \cdot B' \cdot C + A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C$

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





$F = A' \cdot B' \cdot C + A' \cdot B \cdot C' + A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C$
$= C + A' \cdot B$

			B	B
	0	1	1	1
A	0	1	1	0
		C	C	

$F = C + A' \cdot B$

Laws/theorems of Boolean Algebra (CS2100)

Recall this earlier slide?

Identity laws	
$A+0 = 0+A = A$	$A \cdot 1 = 1 \cdot A = A$
Inverse/complement laws	
$A+A' = A'+A = 1$	$A \cdot A' = A' \cdot A = 0$
Commutative laws	
$A+B = B+A$	$A \cdot B = B \cdot A$
Associative laws	
$A+B+C = A+(B+C) = (A+B)+C$	$A \cdot B \cdot C = A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributive laws	
$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$	$A+(B \cdot C) = (A+B) \cdot (A+C)$

$X+X = X$	$X \cdot X = X$
One element / Zero element	
$X+1 = 1+X = 1$	$X \cdot 0 = 0 \cdot X = 0$
Involution	
$(X')' = X$	
Absorption 1	
$X+(X \cdot Y) = X$	$X \cdot (X+Y) = X$
Absorption 2	
$X+(X' \cdot Y) = X+Y$	$X \cdot (X'+Y) = X \cdot Y$
DeMorgans' (can be generalised to > 2 variables)	
$(X+Y)' = X' \cdot Y'$	$(X \cdot Y)' = X'+Y'$
Consensus	
$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$	$(X+Y) \cdot (X'+Z) \cdot (Y+Z) = (X+Y) \cdot (X'+Z)$

Duality

Given a Boolean equality that is true, if the AND/OR operators and identity elements 0/1 in the equality are interchanged, the new equality is also true.

Algebraic Simplification

- Example: Find the simplified SOP expression of

  F(a,b,c,d) = a·b·c + a·b·d + a'·b·c' + c·d + b·d'

a·b·c + **a·b·d** + a'·b·c' + c·d + **b·d'**
= a·b·c + **a·b + a'·b·c'** + c·d + b·d' (absorption 2)
= **a·b·c + a·b** + b·c' + c·d + b·d' (absorption 2)
= a·b + **b·c'** + c·d + **b·d'** (absorption 1)
= a·b + c·d + b·(**c'+d'**) (distributivity)
= a·b + **c·d + b·(c·d)'** (DeMorgan's)
= **a·b** + c·d + **b** (absorption 2)
= b + c·d (absorption 1)

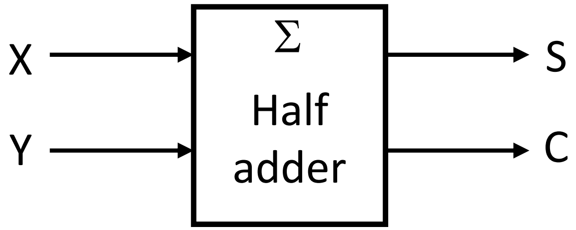
a·b·d + **b·d'**
= **b**· (**a·d** + **d'**)
= **b**· (**a** + **d'**)
= **a·b** + **b·d'**

Number of literals reduced from 13 to 3.

Absorption 1: X+X·Y = X
Absorption 2: X+X'·Y = X+Y

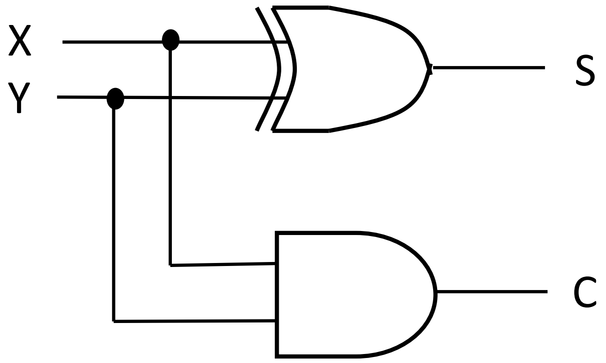
Half Adder

- **Half adder** is a circuit that adds 2 single bits (X, Y) to produce a result of 2 bits (C, S).
- The black-box representation and truth table for half adder are shown below.

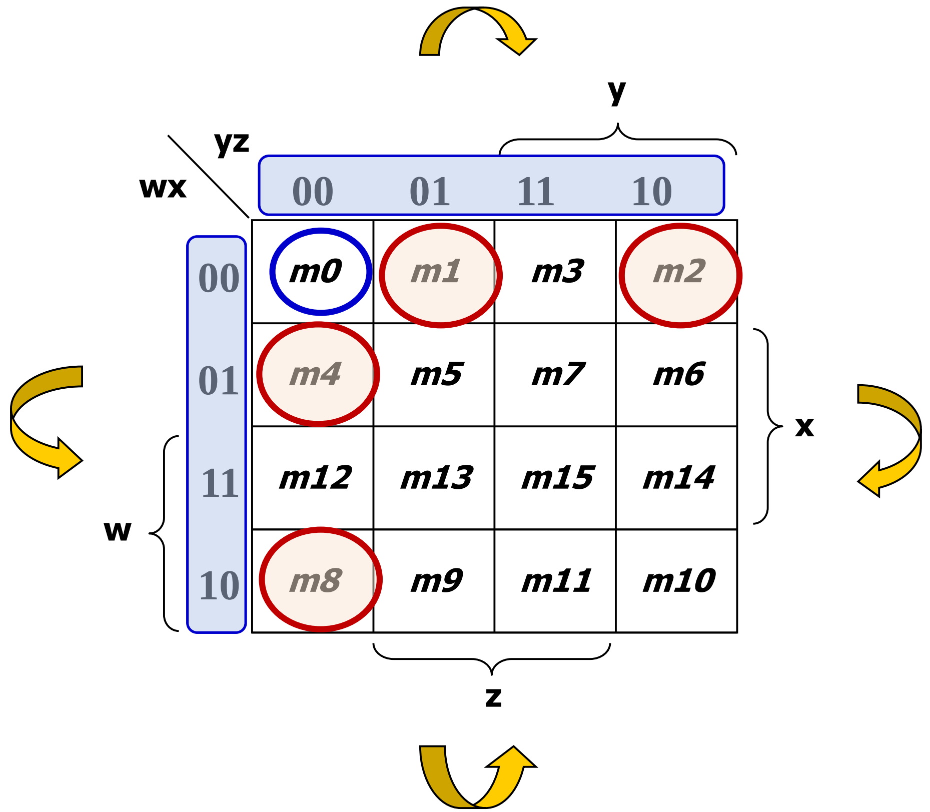


Inputs		*Outputs*	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- In canonical form (sum-of-minterms):
  - $C = X \cdot Y$
  - $S = X' \cdot Y + X \cdot Y'$
- Output S can be simplified further (though no longer in SOP form):
  - $S = X' \cdot Y + X \cdot Y' = X \oplus Y$
- Implementation of a half adder



4-Variable Karnaugh-Map (K-map)



- Rows and columns in Gray code sequence (next value differs from the current value by 1 bit): eg: 00 → 01 → 11 → 10 and back to 00.
- There are 2 wrap-arounds.
- Every cell has 4 neighbours.
  - Neighbours of minterm mx are those that differ by one literals from mx.
  - Examples: The cell corresponding to minterm *m0* has neighbours *m1*, *m2*, *m4* and *m8*. The minterm *m14* has neighbours *m6*, *m15*, *m10* and *m12*.

In general, an *n*-variable K-map has 2^n cells and each minterm has *n* neighbours.

How to use K-maps

Unifying Theorem (complement law) $A' + A = 1$

- Two minterms that differ by one literal can be merged into a product term.
- Eg: 4 variables A,B,C,D
  - $m10 + m14 = A \cdot B' \cdot C \cdot D' + A \cdot B \cdot C \cdot D' = A \cdot (B'+B) \cdot C \cdot D' = A \cdot C \cdot D'$
  - $m2 + m3 + m10 + m11 = A' \cdot B' \cdot C \cdot D' + A' \cdot B' \cdot C \cdot D + A \cdot B' \cdot C \cdot D' + A \cdot B' \cdot C \cdot D$
    $= A' \cdot B' \cdot C \cdot (D'+D) + A \cdot B' \cdot C \cdot (D'+D) = A' \cdot B' \cdot C + A \cdot B' \cdot C = (A'+A) \cdot B' \cdot C = B' \cdot C$

- Minterms that differ by one literal will appear as neighbours on the K-map (due to the Gray code sequence arrangement).

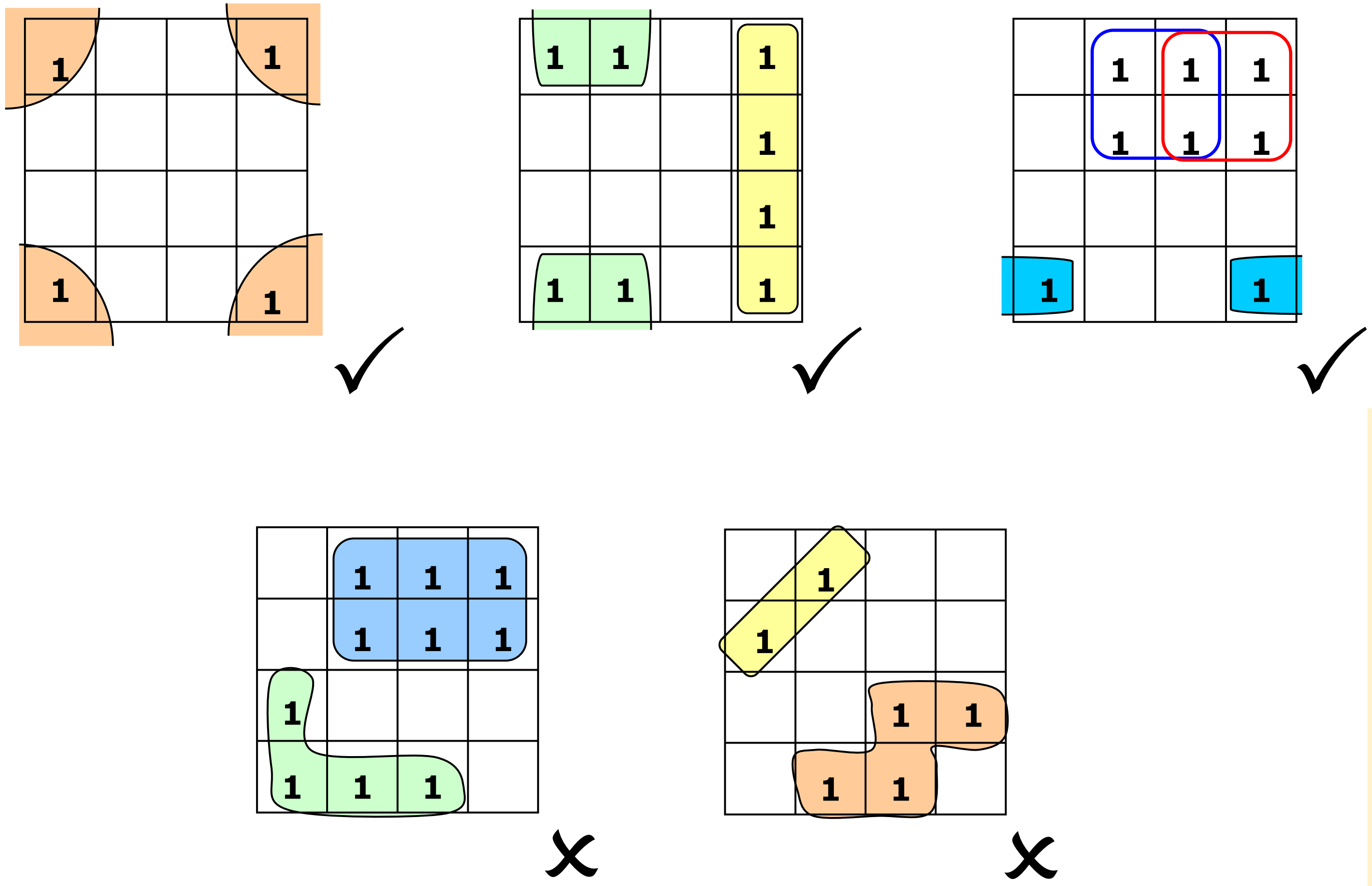
AB \ CD	00	01	11	10
00	*m0*	*m1*	*m3*	*m2*
01	*m4*	*m5*	*m7*	*m6*
11	*m12*	*m13*	*m15*	*m14*
10	*m8*	*m9*	*m11*	*m10*

How to use K-maps

- In a K-map, each cell containing a '1' corresponds to a minterm of the given function *F* where the output is 1.
- Each valid grouping of adjacent cells containing '1' then corresponds to a simpler product term of *F*.
  - A group must have size in powers of two: 1, 2, 4, 8, …
  - Grouping 2 adjacent cells eliminates 1 variable from the product term; grouping 4 cells eliminates 2 variables; grouping 8 cells eliminates 3 variables, and so on. In general, grouping 2^n cells eliminates n variables.
- Group as many cells as possible
  - The larger the group, the shorter is the resulting product term.
- Select as few groups as possible to cover all the cells (minterms) of the function
  - The fewer the groups, the fewer is the number of product terms in the simplified SOP expression.

How to use K-maps

- Examples of valid and invalid groupings.



For illustration purpose here, unfilled cells are 0s. In practice, you need to fill in all cells with either 0 or 1 (or don't care – we will cover that later.)

K-maps: Prime Implicants (PIs) and Essential PIs (EPIs)

$F = A'\cdot B'\cdot C + A'\cdot B\cdot C' + A'\cdot B\cdot C + A\cdot B'\cdot C + A\cdot B\cdot C$
$= C + A'\cdot B$

- To get the simplest SOP expression from a K-map, you need to obtain:
  - Minimum number of literals per product term (i.e. the shortest product terms); and
  - Minimum number of product terms.

By finding all the biggest groupings (PIs) on the K-maps

By removing redundant PIs.

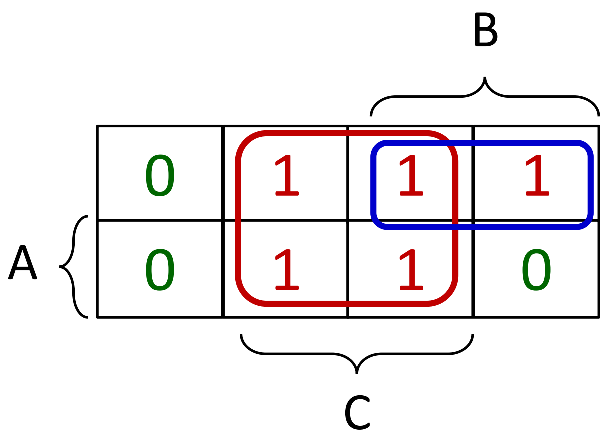
However, it might be hard to find the redundant PIs. Instead we identify the EPIs among the PIs.

K-maps: Prime Implicants (PIs) and Essential PIs (EPIs)

F(A,B,C) = A’·B’·C + A’·B·C’ + A’·B·C + A·B’·C + A·B·C

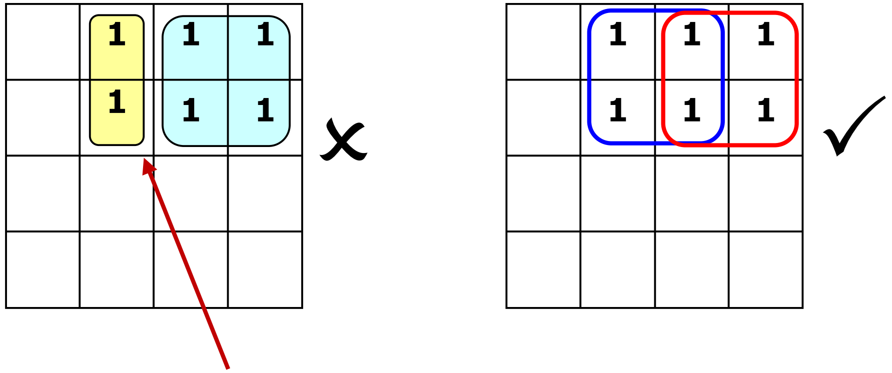
Implicants

- A product term that could be used to cover some minterms of the function.
- Implicants on this K-map: A’·B’·C, A’·B·C’, A’·B·C, A·B’·C, A·B·C, A’·C, A’·B, A·C, B’·C, B·C, and C.



Prime Implicants

- A product term obtained by combining the largest number of neighbouring minterms.
- PIs on this K-map: C and A’·B.
- Always look for PIs on a K-map.

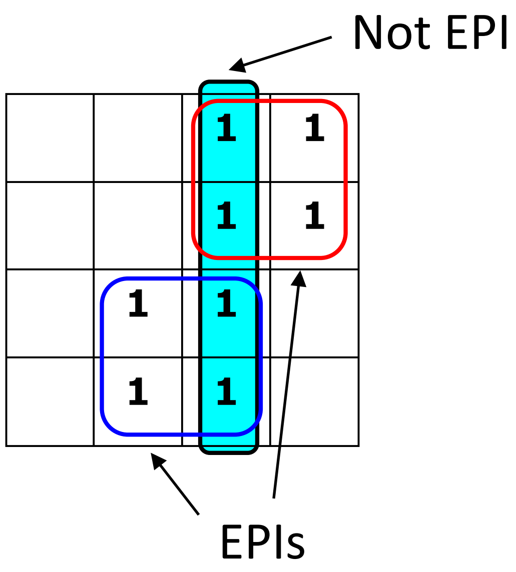


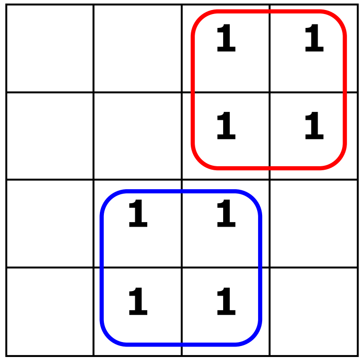
Not a PI.

K-maps: Prime Implicants (PIs) and Essential PIs (EPIs)

Essential Prime Implicants

- A prime implicant that includes at least one minterm in it that is not covered by any other prime implicant.

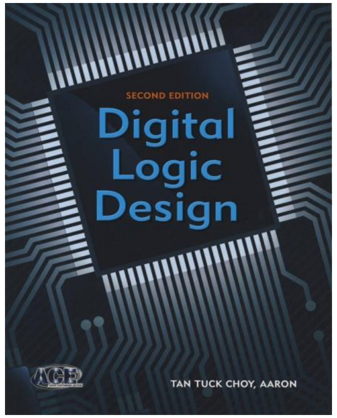




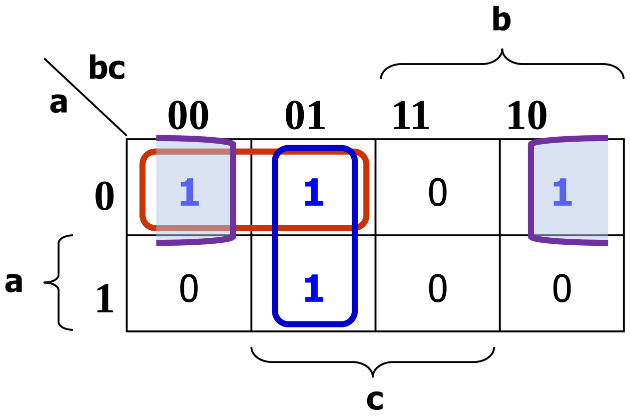
Solution

K-maps: Prime Implicants (PIs) and Essential PIs (EPIs)

DLD page 106, question 5-3.

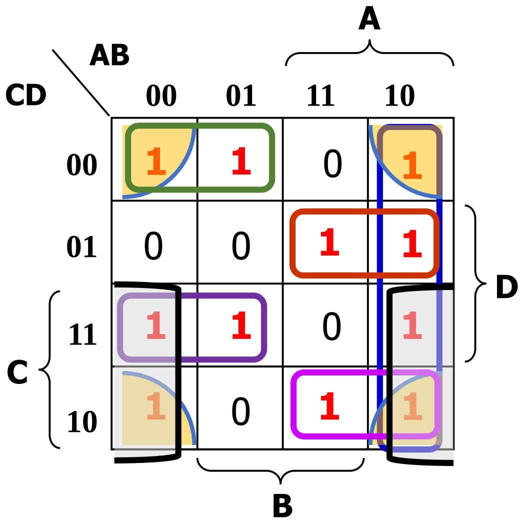


Identify the PIs and EPIs of the K-maps below.



How many PIs? 3 PIs

How many EPIs? 2 EPIs



PIs:	EPIs:
B’·D’	✕
B’·C	✕
A·B’	✕
A’·C’·D’	✓
A·C’·D	✓
A’·C·D	✓
A·C·D’	✓

How many PIs? 7 PIs

How many EPIs? 4 EPIs

Finding simplified SOP expression on K-map

Algorithm

1. Circle all prime implicants on the K-map.
2. Identify and select all essential prime implicants for the cover.
3. Select a minimum subset of the remaining prime implicants to complete the cover, that is, to cover those minterms not covered by the essential prime implicants.

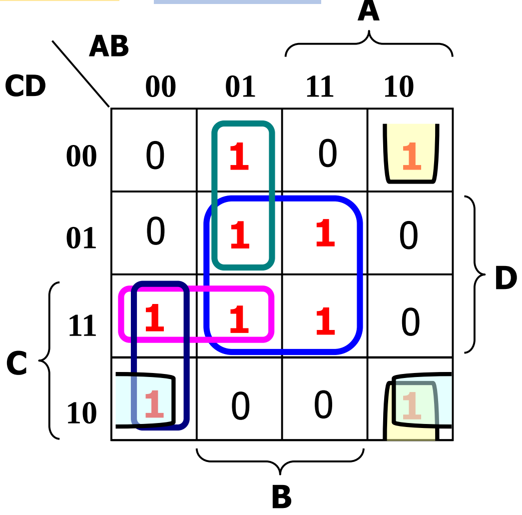
Example #1:

$F(A,B,C,D) = \Sigma\, m(2,3,4,5,7,8,10,13,15)$

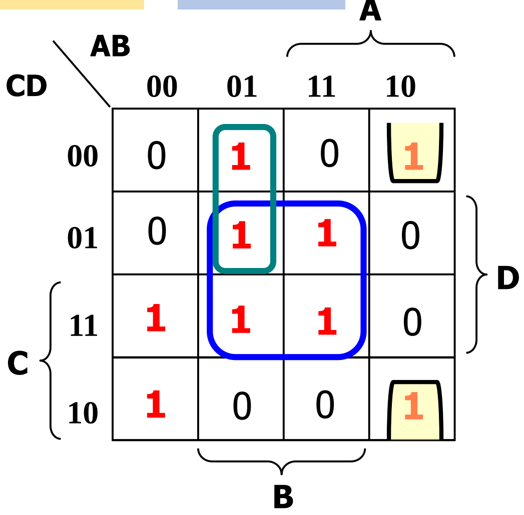
CD \ AB	00	01	11	10
00	0	1	0	1
01	0	1	1	0
11	1	1	1	0
10	1	0	0	1

$F = B\cdot D + A'\cdot B\cdot C' + A\cdot B'\cdot D' + A'\cdot B'\cdot C$

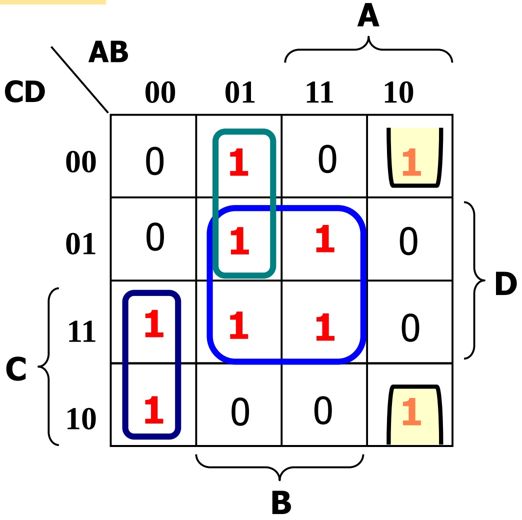
Find PIs | 6 PIs



Find EPIs | 3 EPIs

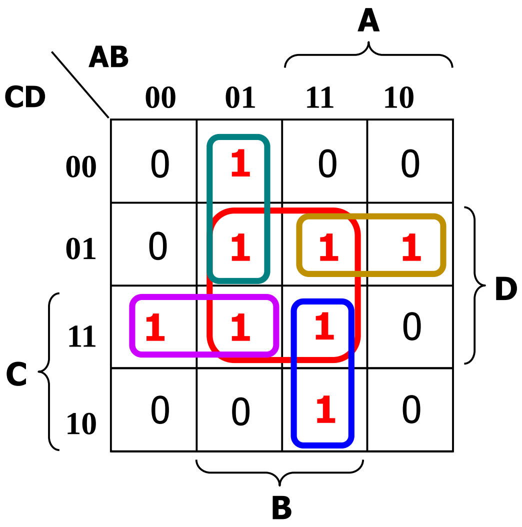


Solution



Finding simplified SOP expression on K-map

Find the simplified SOP expression for G(A,B,C,D).



PIs:	EPIs:
B·D	✗
A'·B·C'	✓
A·C'·D	✓
A'·C·D	✓
A·B·C	✓

Solution

G = A'·B·C' + A·C'·D + A'·C·D + A·B·C

Finding simplified POS expression on K-map

We use K-map to find the simplified SOP expression of a function.
What if we want to find the simplified POS expression?

Algorithm: To find simplified POS expression of a Boolean function F

1. Find the simplified SOP expression of F' on the K-map of F'.
2. Complement the SOP expression of F'.
3. Resulting expression is the simplified POS expression of F.

Finding simplified POS expression on K-map

1. Find the simplified SOP expression of F’ on the K-map of F’.
2. Complement the SOP expression of F’.
3. Resulting expression is the simplified POS expression of F.

Find the simplified POS expression of F:

$F(A,B,C,D) = \Sigma m(0,1,2,3,5,7,8,9,10,11)$

Draw K-map of F’:

K-map of F

CD \ AB	00	01	11	10
00	1	0	0	1
01	1	1	0	1
11	1	1	0	1
10	1	0	0	1

K-map of **F'**

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	0	1	0
11	0	0	1	0
10	0	1	1	0

$F' = A\cdot B + B\cdot D'$

Complement both sides:

$F = (A\cdot B + B\cdot D')'$

$\quad = (A\cdot B)' \cdot (B\cdot D')'$

$\quad = (A'+B') \cdot (B'+D)$

Don’t Care Values

Datapath control

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

- In certain problems, some outputs are not specified or are invalid. Hence, these outputs can be either ‘1’ or ‘0’.
- They are called don’t-care conditions, denoted by X (or d).

K-maps with Don’t Care Values

- Example: A circuit takes in a 3-bit value ABC and outputs 2-bit value FG which is the sum of the input bits. It is also known that inputs 000 and 111 never occur.

Assuming all inputs are valid.

A	B	C	F	G
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

F (B over columns 3–4, C over columns 2–3, A on row 2)

	0	0	1	0
A	0	1	1	1

G (B over columns 3–4, C over columns 2–3, A on row 2)

	0	1	0	1
A	1	0	1	0

F(A,B,C) = Σm(3,5,6,7)
= A·C + A·B + B·C

G(A,B,C) = Σm(1,2,4,7)
= A·B'·C' + A'·B'·C + A·B·C + A'·B·C'

Don’t-cares could be chosen to be either ‘1’ or ‘0’, depending on which choice results in a simpler expression.

Assuming inputs 000 and 111 are invalid.

F (B over columns 3–4, C over columns 2–3, A on row 2)

	X	0	1	0
A	0	1	X	1

G (B over columns 3–4, C over columns 2–3, A on row 2)

	X	1	0	1
A	1	0	X	0

A	B	C	F	G
0	0	0	X	X
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	X	X

F(A,B,C) = Σm(3,5,6) + Σd(0,7)
= A·C + A·B + B·C

G(A,B,C) = Σm(1,2,4) + Σd(0,7)
= B'·C' + A'·B' + A'·C'

K-maps with Don't Care Values

- Example #3 (with don't-cares):

  $F(A,B,C,D) = \Sigma m(2,8,10,15) + \Sigma d(0,1,3,7)$

- Find the simplified SOP expression for F.

CD \ AB	00	01	11	10
00	X	0	0	1
01	X	0	0	0
11	X	X	1	0
10	1	0	0	1

Do we need this A'·B'?

Solution

$F(A,B,C,D) = B' \cdot D' + B \cdot C \cdot D$

Finding simplified POS expression with Don't Care Values

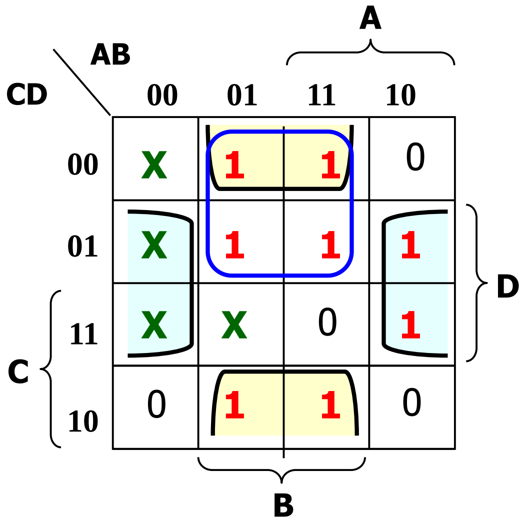
- Example #3 (with don't-cares):

  $F(A,B,C,D) = \Sigma m(2,8,10,15) + \Sigma d(0,1,3,7)$

- Find the simplified POS expression for F.

  $F'(A,B,C,D) = \Sigma m(4,5,6,9,11,12,13,14) + \Sigma d(0,1,3,7)$

K-map of F'



$F' = B{\cdot}C' + B{\cdot}D' + B'{\cdot}D$

$F = (B{\cdot}C' + B{\cdot}D' + B'{\cdot}D)'$

$= (B'+C){\cdot}(B'+D){\cdot}(B+D')$

Quine McCluskey Method

- K-maps are visualization form of Quine McCluskey tabulation method.
- Quine McCluskey method is procedural; it works for any number of variables, and it can be programmed.

- However, Quine McCluskey method is <u>very tedious</u>.
- Lecture #16: Quine McCluskey is for optional reading. Non-examinable. But may help you understand the principle behind K-maps better.

	Column I			Column II			Column III	
group 0	0	0000	✓	0, 1	000–	✓	0, 1, 8, 9	–00–
group 1	1	0001	✓	0, 2	00–0	✓	0, 2, 8, 10	–0–0
	2	0010	✓	0, 8	–000	✓	~~0, 8, 1, 9~~	~~–00–~~
	8	1000	✓	1, 5	0–01		~~0, 8, 2, 10~~	~~–0–0~~
group 2	5	0101	✓	1, 9	–001	✓	2, 6, 10, 14	-- 10
	6	0110	✓	2, 6	0–10	✓	~~2, 10, 6, 14~~	~~-- 10~~
	9	1001	✓	2, 10	–010	✓		
	10	1010	✓	8, 9	100–	✓		
group 3	7	0111	✓	8, 10	10–0	✓		
	14	1110	✓	5, 7	01–1			
				6, 7	011–			
				6, 14	–110	✓		
				10, 14	1–10	✓		

QUIZZES

16: Boolean Algebra Quiz

16: Boolean Algebra Quiz Q1, Q2

Q1. Are the following two Boolean expressions equivalent? Choose YES or NO.

Expression 1: $x'.(y + y'.(z + x.z))'$ $= x'.(y+y'.z)' = x'.(y+z)'$

Expression 2: $x'.(y + z)$

Absorption Theorem 1: $A + A \cdot B = A$

Absorption Theorem 2: $A + A' \cdot B = A + B$

Let $x=0$, $y=0$, $z=0$, then

$x' \cdot (y+z)' = 1 \cdot (0+0)' = 1$

$x' \cdot (y+z) = 1 \cdot (0+0) = 0$

NO

Q2. Are the following two Boolean expressions equivalent? Choose YES or NO

Expression 1: $x.((y + y.z) + z)'$ $= x.(y+z)' = x.(y'.z')$

Expression 2: $x.y'.z'$

Absorption Theorem 1: $A + A \cdot B = A$

De Morgan's Theorem: $(A + B)' = A' \cdot B'$

YES

16: Boolean Algebra Quiz Q3

Question 3 **1 pts**

Canonical sum of products
= sum of minterms.

Complement law: A + A' = A' + A = 1

Identity law: A = A·1 = 1·A = A

We have the following function:

f(a, b, c) = b.c + a.b' = (a'+a).b.c + a.b'.(c'+c) = a'.b.c + a.b.c + a.b'.c' + a.b'.c

Which of the following products are NOT in the canonical sum of products for f?

Distributive law: A·(B+C) = (A·B) + (A·C)

- ✓ a.b.c
- ✗ a'.b'.c'
- ✓ a.b'.c
- ✗ a'.b.c'
- ✓ a.b'.c'

16: Boolean Algebra Quiz Q4

Question 4 **1 pts**

Canonical product of sums
= product of maxterms.

Complement law: A·A' = A'·A = 0

Identity law: A = A+0 = 0+A = A

We have the following function:

g(x, y, z) = (x + y').(y+ z) = (x+y'+z'.z) · (x'.x+y+z)
= (x+y'+z') · (x+y'+z) · (x'+y+z) · (x+y+z)

Which of the following sums are NOT in the canonical product of sums for g?

Distributive law: A+(B·C) = (A+B)·(A+C)

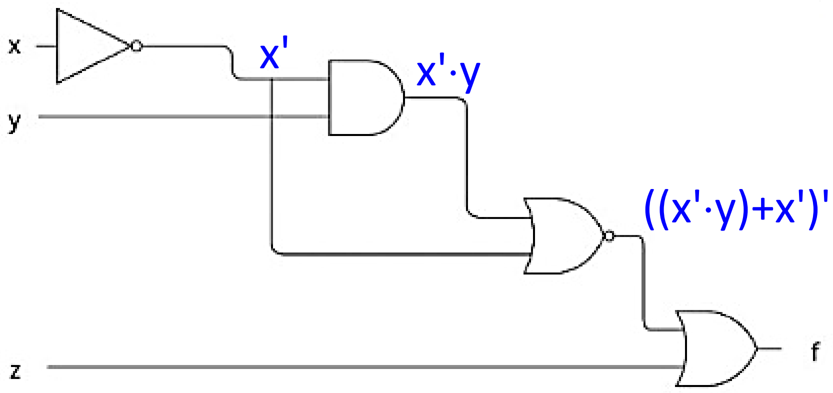
- [x] (x' + y' + z) ✗
- [x] (x + y' + z) ✓
- [x] (x + y + z) ✓
- [x] (x' + y + z) ✓
- [x] (x + y' + z') ✓

QUIZZES

17: Logic Circuit Quiz

17: Logic Circuit Quiz Q1

We are given the circuit below:



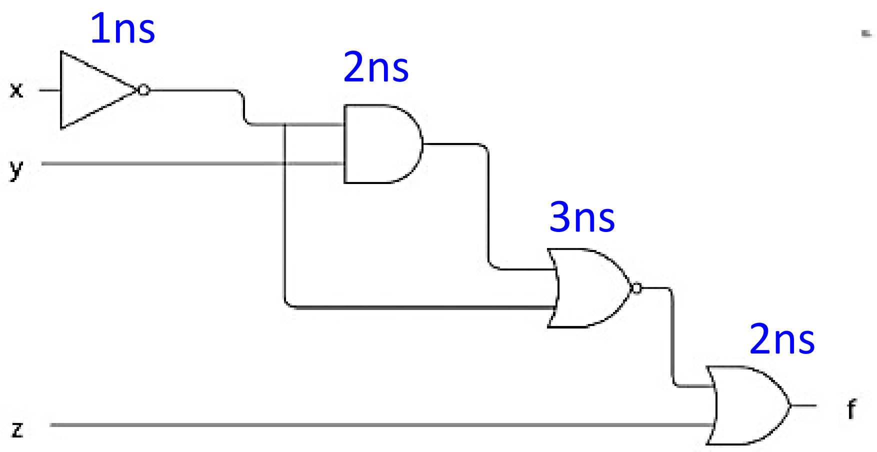
$((x'\cdot y)+x')' = (x'\cdot y)' \cdot x = (x+y')\cdot x = x$

Choose the most simplified expression for f(x, y, z)

- ○ x'.y + x' + z
- ○ x + z
- ○ x'.y + z
- ○ x' + z
- ○ x'.y + z + x'
- ○ x.y + z

17: Logic Circuit Quiz Q2

We are given the circuit below:



If NOT gates have a propagation delay of 1 ns, AND and OR gates have a propagation delay of 2ns and NOR gates have a propagation delay of 3 ns, what is the total propagation delay of the circuit above in ns? 1 ns = 1 nanosecond.

8ns

QUIZZES

18: Simplification Quiz

18: Simplification Quiz Q1

Question 1 **1 pts**

Given the following function:

$f(x, y, z) = \boxed{y'.z'} + \boxed{x'.y'.z} + \boxed{x'.y} + \boxed{x.y.z}$

Use K-maps to simplify this expression as much as possible, and choose all the possible simplest SOP expressions here:

- ☐ x'.y' + y'.z' + x'.y + y.z
- ☐ x'.z' + y'.z' + y.z + x'.z
- ☐ x' + y.z'
- ☑ x' + y'.z' + y.z

f

			y	y
	1	1	1	1
x	1	0	1	0
		z	z	

18: Simplification Quiz Q2

Question 2

Again using the same function:

f(x, y, z) = y'.z' + x'.y'.z + x'.y + x.y.z

Use K-maps to simplify this expression as much as possible, and choose all the possible simplified POS expressions below:

- ✓ (x' + y + z').(x' + y' + z)
- ○ x.(y + z).(y;' + z')
- ○ x.(y' + z)
- ○ (x+ y).(y + z).(x + y').(y' + z')

f

			y	y
	1	1	1	1
x	1	0	1	0
		z	z	

f'

			y	y
	0	0	0	0
x	0	$\boxed{1}$	0	$\boxed{1}$
		z	z	

$$f' = x \cdot y' \cdot z + x \cdot y \cdot z'$$

$$f = (x \cdot y' \cdot z + x \cdot y \cdot z')' = (x'+y+z') \cdot (x'+y'+z)$$

Too few Quiz questions on K-maps.

More K-maps questions in Tutorial 6 Discussion Questions. Try them out yourself.

Reminder:

CS2100 Midterm Test on 12 March 2025, 7pm, at MPSH2B.

End of File