Lecture 3
Boolean Algebra and
Two Level Logic

Outline

- Boolean Logic Operations (included as background information)
- Methods for building logic circuits- Building 2-level Logic Networks using minimum number of product terms
  – Heuristic Method: Karnaugh Map
  – Exact-Optimal Method: 2-Level Minimization with Quine-McCluskey (QM) approach

Boolean Algebra

- **Boolean Algebra**
  - The algebra of propositions
  - Basis for computation in binary computer systems
- Constants/Truth Values
  - False (0) or True (1)
- Variables / Propositions
  - *A*, *B*, *C*, …, upper case Roman letters
  - Each representing either True or False
- Operations
  - Single variable / Unary operations e.g. ***not (')***
  - Two variables/ Binary operations e.g. ***and*** **(·)**, ***or*** **(+)**

Boolean Algebra

- Boolean Constants:
  - True, T, 1
  - False, F, 0
- Boolean Operators
  - NOT A, NOT(A), A’ , A, ~A
  - A AND B, A * B, A·B, AB, A $\wedge$ B
  - A OR B, A + B, A $\vee$ B

Boolean Algebra

- Boolean Expressions
  - Literals
  - A literal is primed (negated) or unprimed variable name
  - E.g. A, a', b, x'

Boolean Representations

- Boolean **expression**
  - A sequence of zeros, ones and literals separated by Boolean operators
  - E.g. *$A \cdot B + C'$ is a Boolean expression*
- Boolean **equation**
  - Used to express relationships.
  - E.g. *$X = A \cdot B + C'$ is a Boolean equation,* representing the relationship between the value of X and the values of A, B and C
- Truth table
  - another way of represent a Boolean expression /equation
- Karnaugh Map
  - For better visualization

Boolean Algebra

- Boolean vs. binary
  - They are different
  - 1+1
    - Boolean: true and true = true
    - Binary: 1+1=10

AND

- ***A* AND *B* ; *A·B*; *AB*; *AΛB***
- ***True*** *if and only if* ***A*** *and* ***B*** *are both* ***true***

AND (Conjunction)

- AND means to satisfy both
  - E.g. (GPA>3.0 and major="engineering")
  - A^A=A
  - A^T=A
  - A^F=F
  - A^~A=F

Negation
NOT

A	B	AΛB
0	1	0
1	0	0
0	0	0
1	1	1

OR

- ***A* OR *B* ; *A*+*B*; *A* V *B***
- ***False*** *if and only if* ***A*** *and* ***B*** *are both* ***false***

OR (Disjunction)

- OR means to satisfy either
  - E.g. (weather="sunny" or temperature>80)
  - A v A = A
  - A v T = T
  - A v F = A
  - A v ~A = T

A	B	AVB
0	1	1
1	0	1
0	0	0
1	1	1

Major Theorems

- $X + 0 = X$
- $X+1 = 1$
- $X+X = X$
- $X+X' = 1$
- $X \cdot 1 = X$
- $X \cdot 0 = 0$
- $X \cdot X = X$
- $X \cdot X' = 0$

Major Theorems

- $(X+Y)+Z = X+(Y+Z)$
- $XY + XZ = X(Y+Z)$
- $(X+Y)(X+Z) = X+YZ$
- Many others …

Major Theorems

- Multiple variables:
- DeMorgan's theorem
  - $(X_1X_2\ldots X_n)' = X'_1+X'_2+\ldots+X'_n$
  - $(X_1+X_2+\ldots+X_n)' = X'_1X'_2\ldots X'_n$
- Shannon's Theorem
  - $f(X_1, X_2, \ldots, X_n) = X_1 f(1, X_2, \ldots X_n)+X'_1 f(0, X_2, \ldots, X_n)$
  - $f(X_1, X_2, \ldots, X_n) = [X_1+f(0, X_2, \ldots X_n)]\cdot[X'_1+f(1, X_2, \ldots, X_n)]$

Karnaugh Map

- In 1953, Maurice Karnaugh was a telecommunications engineer at Bell Labs.
- While exploring the new field of digital logic and its application to the design of telephone circuits, he invented a graphical way of visualizing and then simplifying Boolean expressions.
- This graphical representation, now known as a Karnaugh map, or Kmap, is named in his honor.

Kmap for two variables

Minterms

Minterm	X	Y
$\bar{X}\bar{Y}$	0	0
$\bar{X}Y$	0	1
$X\bar{Y}$	1	0
XY	1	1

F=X’Y+XY’+XY

Truth Table

F(X,Y) = X+Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

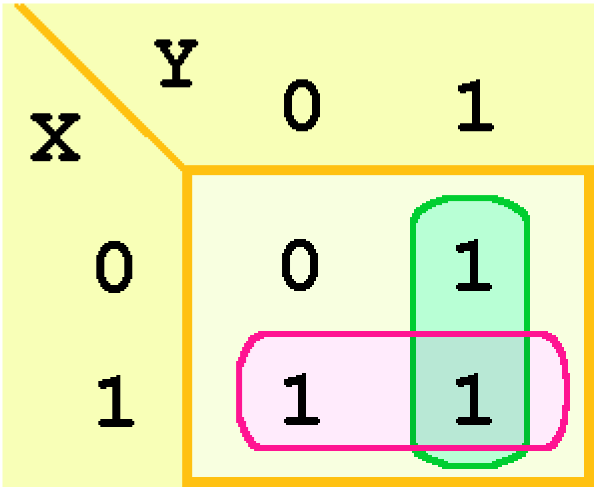
Kmap

X \ Y	0	1
0	0	1
1	1	1

Kmap for two variables

F=X'Y+XY'+XY

F=X + Y



- Find groups of neighboring 1s and simplify the minterms into prime implicants

Kmap for Three Variables

Gray coded (codes of adjacent cells differ by ONLY one bit)

X \ YZ	00	01	11	10
0	$\bar{X}\bar{Y}\bar{Z}$	$\bar{X}\bar{Y}Z$	$\bar{X}YZ$	$\bar{X}Y\bar{Z}$
1	$X\bar{Y}\bar{Z}$	$X\bar{Y}Z$	XYZ	$XY\bar{Z}$

Kmap for Three Variables

F = X'Y'Z+X'YZ+XY'Z+XYZ

X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

Simplified to

F = Z

Rules of Kmap Simplification

The rules of Kmap simplification are:

- Groupings can contain only 1s; no 0s.
- Groups can be formed only at right angles; diagonal groups are not allowed.
- The number of 1s in a group must be a power of 2 – even if it contains a single 1
- The groups must be made as large as possible
- Groups can overlap and wrap around the sides of the Kmap.
- Find minimum set of groups (prime implicants) that cover all 1s

Kmap Example

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

$$F = X'Y'Z' + X'Y'Z + X'YZ + X'YZ' + XY'Z' + XYZ'$$

Kmap Example

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

F = X'Y'Z+X'Y'Z+X'YZ+X'YZ'+XY'Z'+XYZ'

F = X'+Z'

Kmap for Four Variables

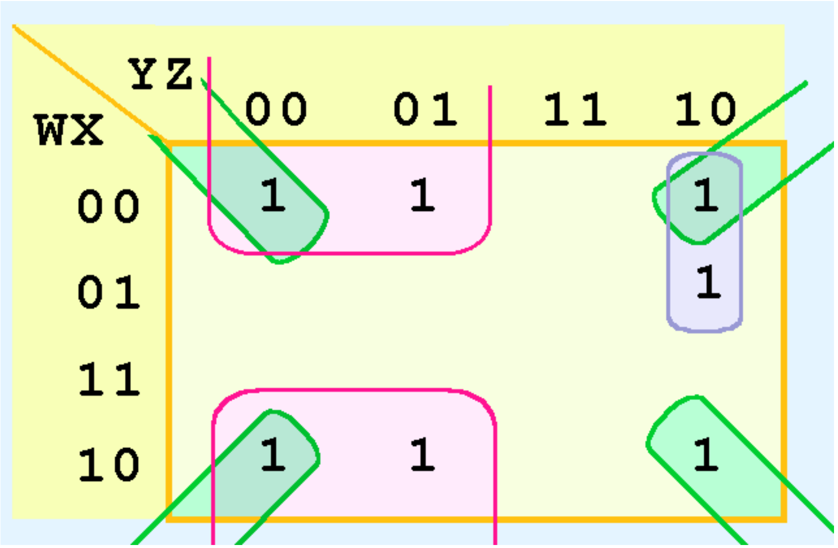
WX \ YZ	00	01	11	10
00	$\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$	$\overline{W}\,\overline{X}\,\overline{Y}Z$	$\overline{W}\,\overline{X}YZ$	$\overline{W}\,\overline{X}Y\overline{Z}$
01	$\overline{W}X\overline{Y}\,\overline{Z}$	$\overline{W}X\overline{Y}Z$	$\overline{W}XYZ$	$\overline{W}XY\overline{Z}$
11	$WX\overline{Y}\,\overline{Z}$	$WX\overline{Y}Z$	$WXYZ$	$WXY\overline{Z}$
10	$W\overline{X}\,\overline{Y}\,\overline{Z}$	$W\overline{X}\,\overline{Y}Z$	$W\overline{X}YZ$	$W\overline{X}Y\overline{Z}$

Kmap for Four Variables

WX \ YZ	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

F =
W'X'Y'Z'+W'X'Y'Z+W'X'YZ'+W'XYZ'+WX'Y'Z'
+WX'Y'Z+WX'YZ'

Kmap for Four Variables



$F = W'X'Y'Z'+W'X'Y'Z+W'X'YZ'+W'XYZ'+WX'Y'Z'+WX'Y'Z+WX'YZ'$

$F = X'Y'+X'Z'+W'YZ'$

Don’t Care Conditions

- In a Kmap, a don’t care condition is identified by an *X* in the cell of the minterm(s) for the don’t care inputs, as shown below.
- In performing the simplification, we are free to include or ignore the *X*’s when creating our groups.

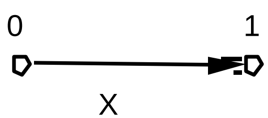
F=YZ+W’X’

WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

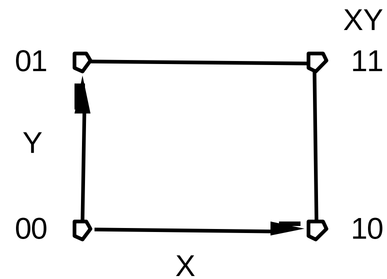
Kmap Summary

- Kmaps provide an easy graphical method of simplifying Boolean expressions.
- A Kmap is a matrix consisting of the outputs of the minterms of a Boolean function.
- In this section, we have discussed 2- 3- and 4-input Kmaps.
- Hard to capture and reason about more complex logic (more variables) beyond 4-inputs

Visualizing Boolean Cubes



1-cube

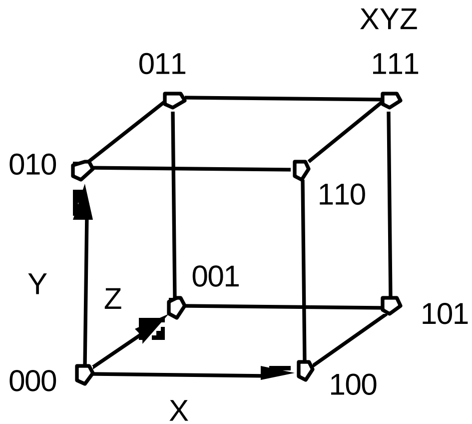


2-cube

Just another way to
represent the truth table

n input variables =
n dimensional "cube"

Visualizing Boolean Cubes



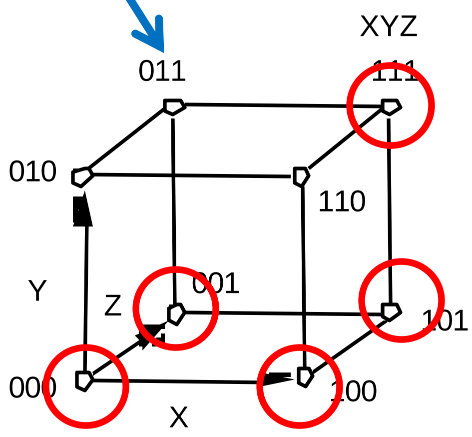
3-cube

YZ

X		00	01	11	10
	0				
	1				

Visualizing Boolean Cubes

Each vertex represents a *minterm* (complete product where each variable appears once)



3-cube

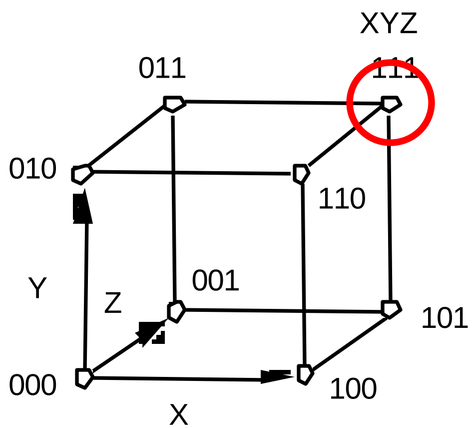
YZ

X	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Karnaugh Map for Logic Function F(X, Y, Z)

- Sum of Products F=X’Y’Z’ + XY’Z’ +X’Y’Z+XY’Z+XYZ

Implicants



3-cube

YZ

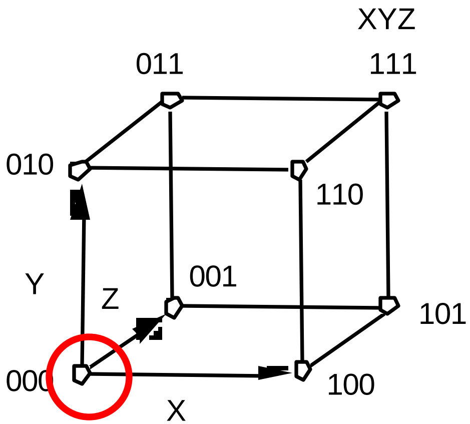
X

	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Implicants: Product Terms covering one or more minterms (power of 2)

Implicant XYZ

Implicants



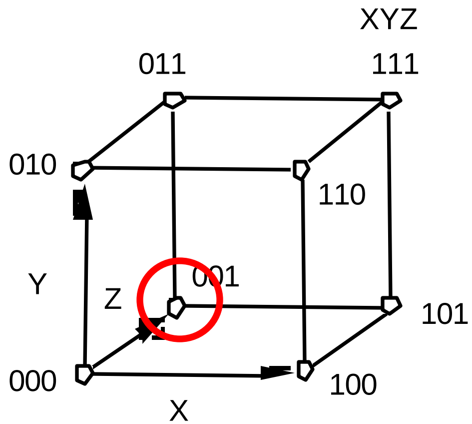
3-cube

YZ

X	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Implicant X’ Y’ Z’

Implicants



3-cube

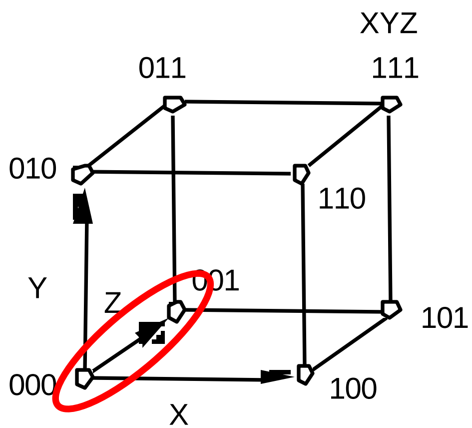
YZ

X

	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Implicant X’ Y’ Z

Implicants



3-cube

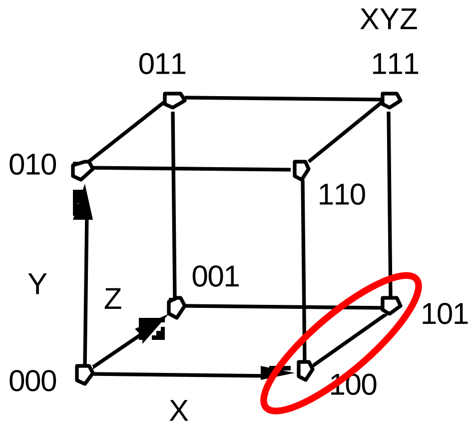
	00	01	11	10
0	1	1	0	0
1	1	1	1	0

YZ

X

Implicant covering (containing) 2 minterms X' Y'

Implicants



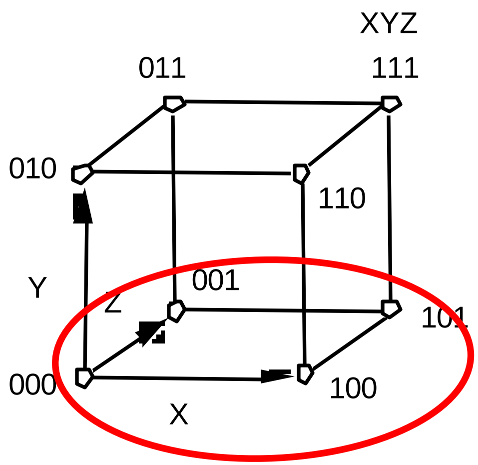
3-cube

YZ

X	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Implicant XY’

Prime Implicants



3-cube

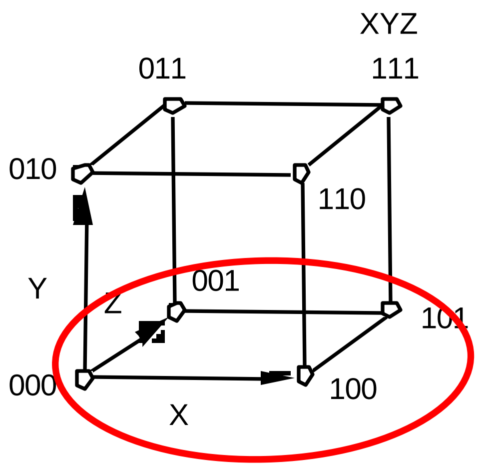
X \ YZ	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Prime Implicant: Y'

Prime Implicant: XZ

Prime Implicants, which cannot be completely covered by any other implicant

Essential Prime Implicants



3-cube

YZ

X

	00	01	11	10
0	1	1	0	0
1	1	1	1	0

Essential Prime Imp.   Essential Prime Imp.

Essential Prime Implicant, uniquely covers one or more minterms, which are NOT covered by any other implicants (If I remove it, some minterms are not covered)

Are all Prime Implicants Essential? NO!

cd

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

Imp_1 = a’c’	Essential Prime
Imp_2 = a’bd	Prime
Imp_3 = bcd	Prime
Imp_4 = abd’	Essential Prime

Summary of Definitions

- Implicant: any product term that covers one or more more minterms
- Prime Implicant: a product term created by merging the maximum possible adjacent minterms (that map to TRUE outputs) on a K-map
- Essential Prime Implicant: A Prime implicant that covers at least one minterm that is not covered by any other Prime Implicant

Two Level Simplification

Three variable function example: Full Adder With Carry Out

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

B Cin

A	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Two Level Simplification

Three variable example: Full Adder Carry Out

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

B Cin

A	00	01	11	10
0	0	0	1	0
1	0	1	1	1

A

Cout=A'BCin + ABCin + ABCin' + AB'Cin

But it is not the MINIMAL representation

Metric for Quality

- Metric for a logic expression being minimal = number of product terms contained
- Goal: phrase the logic function with the minimum number of Prime Implicants

Two Level Simplification

B Cin

A	00	01	11	10
0	0	0	1	0
1 A	0	1	1	1

Cout = A Cin + B Cin + A B

The ON (TRUE) space of this function is covered by the Sum (OR) of three product terms

Logic Functions: Expressions to Gates

More than one way to map an expression to gates

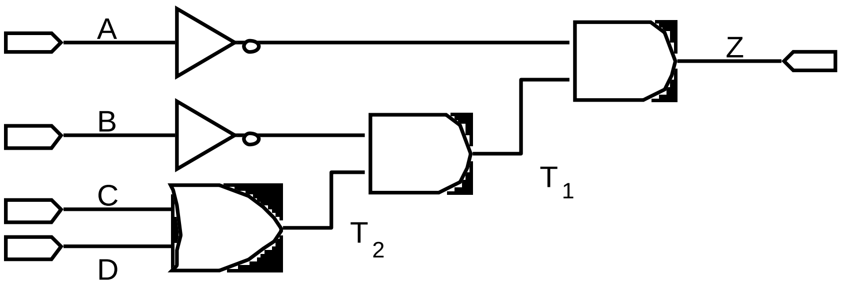
E.g., $Z = (A' \bullet (B' \bullet (C + D)))$

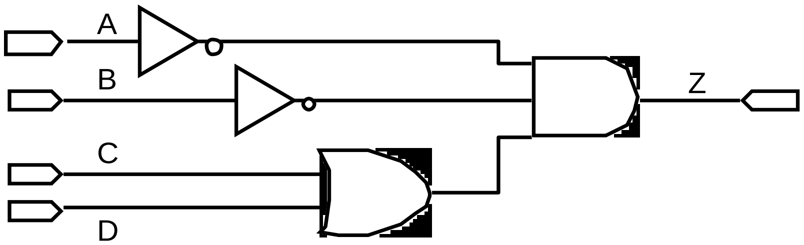
$Z = A' \bullet B' \bullet (C + D)$

Logic Functions: Expressions to Gates

(A' • (B' • (C + D)))

A' • B' • (C + D)

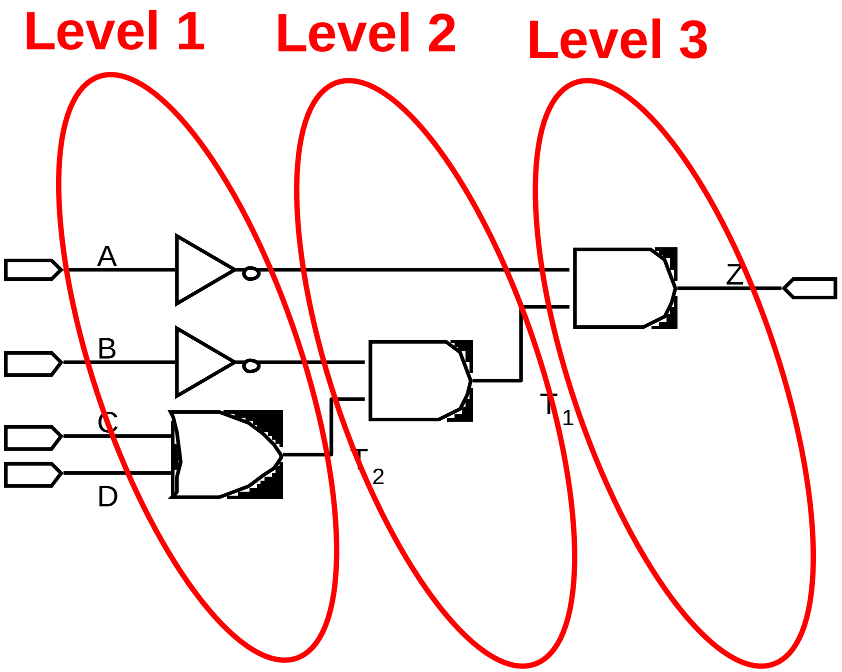


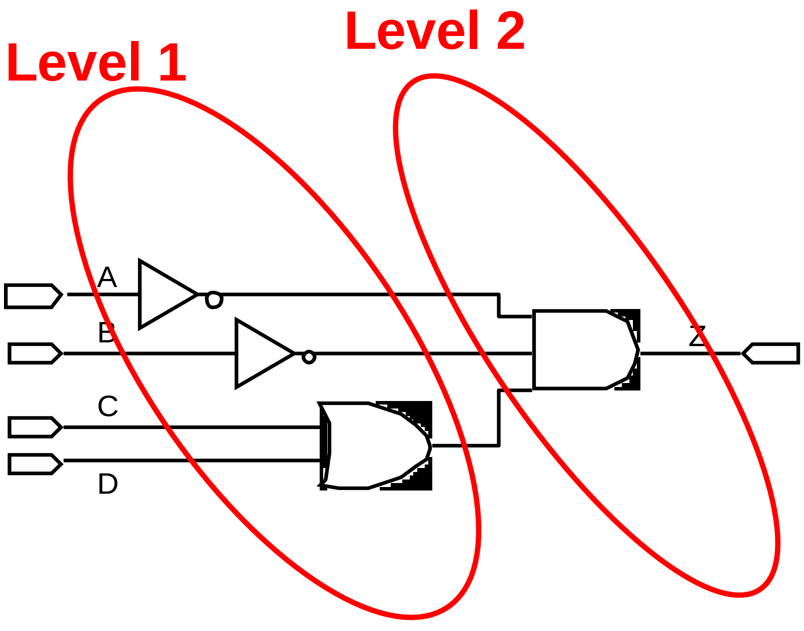


Logic Functions: Expressions to Gates

(A' • (B' • (C + D)))

A' • B' • (C + D)





Simplifying Logic Functions

- Speed: Fewer levels of gates (generally) imply reduced signal propagation delay
- Area: Number of gates influences chip area (costs)
  - Types of gates and number of inputs per gate may matter as well

2-Level Minimization

- Want to reduce area, power consumption, delay of circuits
- Hard to exactly predict circuit area or power just from Boolean equations
  - Can approximate with the number of terms in the Sum of Products in the expression
- Minimize total number of product terms
  - Side effect (minimizing the number of literals in the equation)

2-Level Minimization

- Hunting for all Implicants and considering all combinations of Implicants to arrive at the exhaustive list of Prime Implicants by visually inspecting K-Maps are not practical for large functions
  - Heuristic Methods: Applied in the most general practical settings
  - Perform a limited search to generate a "reasonably comprehensive" list of candidate Prime Implicants
  - Heuristic Methods CANNOT claim OPTIMALITY
- Optimal Method: Quine-McCluskey
  - Useful for small problems but impractical for large ones

Optimal 2-level Minimization

- Quine-McCluskey

QM Method

- TWO MAIN STEPS:
- 1. Compute ALL prime implicants with a well-defined algorithm
  - Start from individual minterms
  - Merge adjacent implicants systematically until further merging is impossible
- 2. Select minimal cover of logic function from prime implicants
  - *Unate covering problem*

Example

$$F = \bar{x}\bar{y}\bar{z}\bar{w} + \bar{x}y\bar{z}w + x\bar{y}\bar{z}w + \bar{x}yzw$$

Don't care

$$D = \bar{y}z + xyw + \bar{x}\bar{y}\bar{z}w + x\bar{y}\bar{w} + \bar{x}y\bar{z}\bar{w}$$

$\bar{x}\,\bar{z}$

	$\bar{x}\,\bar{y}$	$\bar{x}y$	xy	$x\,\bar{y}$
$\bar{z}\,\bar{w}$	1	d	0	d
$\bar{z}w$	d	1	d	1
zw	d	1	d	d
$z\,\bar{w}$	d	0	0	d

$\bar{y}$

w

Primes: $\bar{y}$ + w +$\bar{x}$ $\bar{z}$

Solution: $\bar{y}$ + w is minimum prime cover

(also w+ $\bar{x}\bar{z}$)

Example

- Use prime implicant table:

(1) Find all prime imp

(2) List all minterms

(3) Build Prime Implicant Table

(4) Find subset of primes that cover all minterms

Prime implicant table

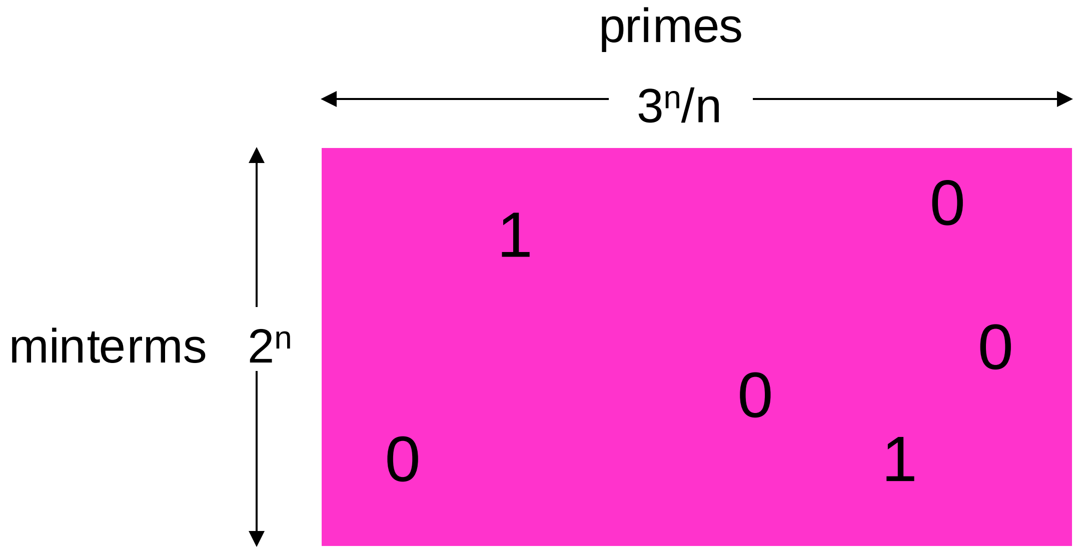
	$\bar{y}$	w	$\bar{x}\bar{z}$
$\bar{x}\bar{y}\bar{z}\bar{w}$	1	0	1
$\bar{x}y\bar{z}w$	0	1	1
$x\bar{y}\bar{z}w$	1	1	0
$\bar{x}yzw$	0	1	0

These two primes can cover all

Kmap Difficulty

Note:

- ~ 2^n minterms
- ~ $3^n/n$ primes



- Optimal 2-level logic synthesis is NP-Complete (means really really hard!!)
- The number of prime implicants grows rapidly with number of inputs, n (variables)
  - Upper bound on number of prime implicants grows as $3^n/n$ where n is the number of inputs
- We need a systematic way of finding primes and optimizing logic

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Gather minterms of the function into groups according to the number of variables that are TRUE in them

Implication Table	
Column I	
0 **0000**	
1 **0100**	
1000	
2 **0101**	
0110	
1001	
1010	
3 **0111**	
1101	
4 **1111**	

- First Goal: find prime implicants

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Compare two minterms from two **consecutive** groups

Identify the location where there is a switch of bit value

Combine the two minterms by placing that location with a Don't Care

Implication Table	
Column I	
0000	0-00
0100	
1000	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + DC(0,7,15)$

Label the two original minterms as “reduced” √

Place the resulting expression into the next column

Implication Table

Column I	Column II
0000 √	0-00
0100 √	
1000	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Repeat for all pairwise matchings

Implication Table	
Column I	**Column II**
0000 √ **0100** √ **1000** √ **0101** **0110** **1001** **1010** **0111** **1101** **1111**	0-00 -000

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Some comparisons will not yield a merge, if more than one bit location is different between the two minterms

Implication Table

Column I	Column II
0000 √	0-00
0100 √	-000
1000 √	
0101 √	010-
0110 √	01-0
1001	
1010	
0111	
1101	
1111	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Implication Table

Column I	Column II
0000 √	0-00
0100 √	-000
1000 √	
0101 √	010-
0110 √	01-0
1001 √	100-
1010 √	10-0
0111 √	01-1
1101 √	-101
	011-
	1-01
1111 √	-111
	11-1

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Repeat the same systematic merge operation among pairs of expressions in the newly created column

Implication Table

Column I	Column II	Column III
0000 √		
	0-00	
0100 √	-000	
1000 √		
	010-	
0101 √	01-0	
	100-	
0110 √	10-0	
1001 √		
1010 √		
	01-1	
	-101	
0111 √	011-	
	1-01	
1101 √		
	-111	
1111 √	11-1	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

Repeat the same systematic merge operation among pairs of expressions in the newly created column

Implication Table

Column I	Column II	Column III
0000 √	0-00 -000	
0100 √ **1000** √	010- √ 01-0 100- 10-0	01--
0101 √ **0110** √ **1001** √ **1010** √	01-1 -101 011- √ 1-01	
0111 √ **1101** √	-111 11-1	
1111 √		

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + DC(0,7,15)$

More than one pair can lead to the same merged expression, then we place the merged expression into the new column only once, but check out all pairs involved

Implication Table

Column I	Column II	Column III
0000 √	0-00	
	-000	
0100 √		
1000 √		
	010- √	
	01-0 √	01--
0101 √	100-	
0110 √	10-0	
1001 √		
1010 √	01-1 √	
	-101	
0111 √	011- √	
1101 √	1-01	
	-111	
1111 √	11-1	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

More than one pair can lead to the same merged expression, then we place the merged expression into the new column only once, but check out all pairs involved

Implication Table

Column I	Column II	Column III
0000 √	0-00	
0100 √	-000	
1000 √	010- √	01--
0101 √	01-0 √	
0110 √	100-	
1001 √	10-0	
1010 √	01-1 √	-1-1
0111 √	-101 √	
1101 √	011- √	
	1-01	
1111 √	-111 √	
	11-1 √	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13)$
$+ DC(0,7,15)$

The process will stop when there is no pair that can be merged

At this point some expressions will remain "unchecked"

Implication Table

Column I	**Column II**	**Column III**
0000 √ **0100** √	0-00 -000	
1000 √ **0101** √ **0110** √ **1001** √	010- √ 01-0 √ 100- 10-0	01--
1010 √ **0111** √ **1101** √	01-1 √ -101 √ 011- √ 1-01	-1-1
1111 √	-111 √ 11-1 √	

Quine-McCluskey Method

$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + DC(0,7,15)$

"Unchecked" expressions correspond to the prime implicants

Implication Table

Column I	Column II	Column III
0000 √		
	0-00	
	-000	
0100 √		
1000 √		
	010- √	
	01-0 √	01--
0101 √	100-	
0110 √	10-0	
1001 √		
1010 √		
	01-1 √	
	-101 √	
0111 √	011- √	
1101 √	1-01	-1-1
	-111 √	
1111 √	11-1 √	

Quine McCluskey Method (Contd)

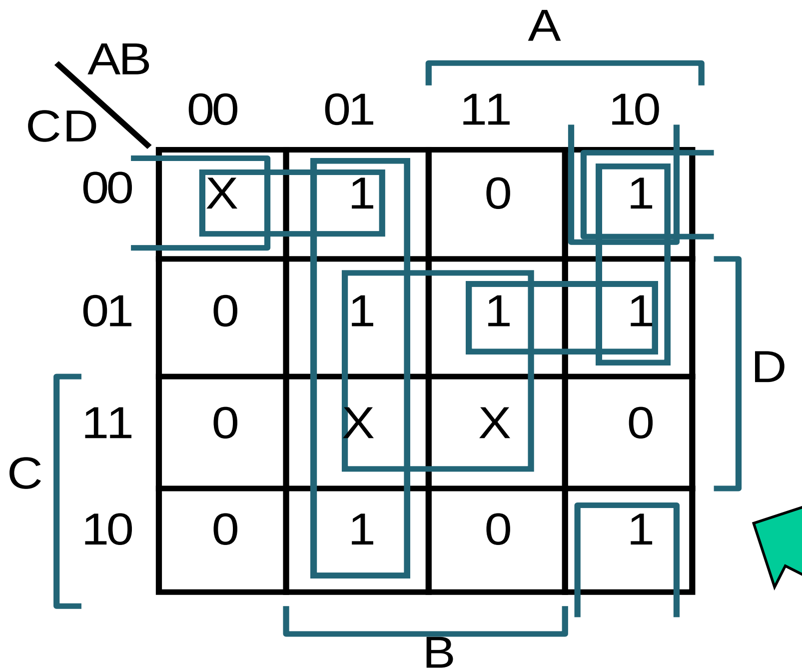
CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

$$f(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + DC(0,7,15)$$

Prime Implicants:

- 0-00 = A' C' D'
- -000 = B' C' D'
- 100- = A B' C'
- 10-0 = A B' D'
- 1-01 = A C' D
- 01-- = A' B
- -1-1 = B D

Quine-McCluskey Method (Contd)



Prime Implicants:

0-00 = A' C' D'

100- = A B' C'

1-01 = A C' D

-1-1 = B D

-000 = B' C' D'

10-0 = A B' D'

01-- = A' B

Kmap leads to the same result

What should be our minimum sets of primes to cover all logic 1s?

Finding the Minimum Cover

- We have so far found all the prime implicants
- The second step of the Q-M procedure is to find the smallest set of prime implicants to cover the complete so called "*on-set*" of the function
  - This problem is an instance of the general Unate Covering Problem

Unate Covering

- DEFINITION:Given a matrix for which all entries are 0 or 1
  - find the minimum cardinality subset of columns such that, for every row, at least one column in the subset contains a 1

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

Minimum two columns can cover all rows

Assume following Prime implicants found in the first step of QM in a hypothetical problem instance

Prime implicants

minterms

	01X	0X0	X00	X11
000		1	1	
010	1	1		
011	1			1
111				1
100			1	

Selecting Prime Implicants

	01X	0X0	X00	X11
~~000~~		1	1	
010	1	1		
011	1			1
111				1
~~100~~			1	

Selecting Prime Implicants

	01X	0X0	X00	X11
~~000~~		1	1	
010	1	1		
~~011~~	1			1
~~111~~				1
~~100~~			1	

Selecting Prime Implicants

	01X	0X0	X00	X11
~~000~~		1	1	
~~010~~	1	1		
~~011~~	1			1
~~111~~				1
~~100~~			1	

Essential Prime Implicants

- If there is a minterm that is covered by only one specific implicant
  - That implicant is essential
  - It must exist in the minimal cover

X00 is an essential prime implicant

minterms

	01X	0X0	X00	X11
000		1	1	
010	1	1		
011	1			1
111				1
100			1	

X11 is an essential prime implicant

minterms

	01X	0X0	X00	X11
000		1	1	
010	1	1		
011	1			1
111				1
100			1	

Dealing with remaining implicants

- Need to reduce the implicant table as much as we can first

1. Eliminate rows covered by essential columns
2. Eliminate rows that dominate other rows
   – The row that can be covered by other rows
3. Eliminate columns dominated by other columns
   – The columns can be covered by other columns

Dealing with remaining implicants

- Eliminate rows that dominate other rows
- Eliminate columns dominated by other columns
- If a row(column) A has a "1" entry in each location that row(column) B has (A may have 1's in some other further entries as well), then A dominates B

Eliminate rows covered by essential columns

	A	B	C
H		1	
I	1		1
J	1	1	
K		1	1

Eliminate rows covered by essential columns

	A	B	C
H		1	
I	1		1
J	1	1	
K		1	1

H, J, K can be eliminated by essential prime B

Eliminate rows that dominate other rows

	A	B	C
H	1		
I	1	1	
J	1		1

Eliminate rows that dominate other rows

	A	B	C
H	1		
I	1	1	
J	1		1

Eliminate rows that dominate other rows

	A	B	C
H	1		
I	1	1	
J	1		1

Eliminate rows that dominate other rows

	A	B	C
H	1		
I	1	1	
J	1		1

Row I, J dominates row H

So, I, J can be eliminated

Eliminate columns dominated by other columns

	A	B	C
H	1		
I	1	1	
J	1		1
K		1	

Eliminate columns dominated by other columns

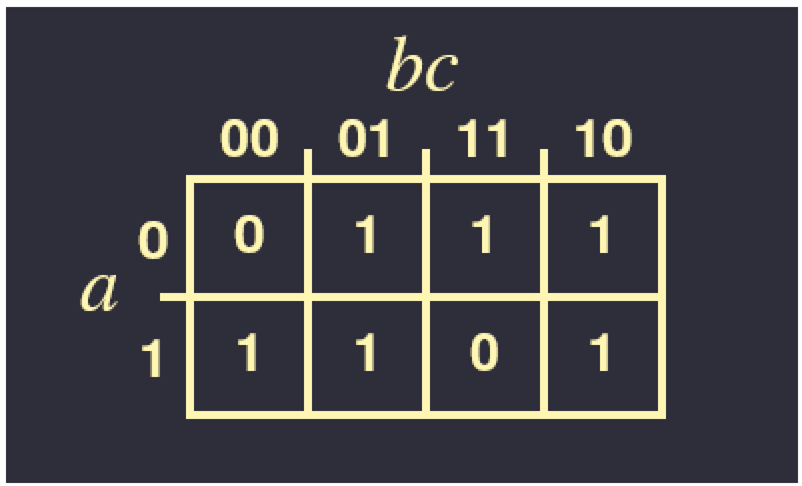
	A	B	C
H	1		
I	1	1	
J	1		1
K		1	

Column A dominates column C

So C can be eliminated

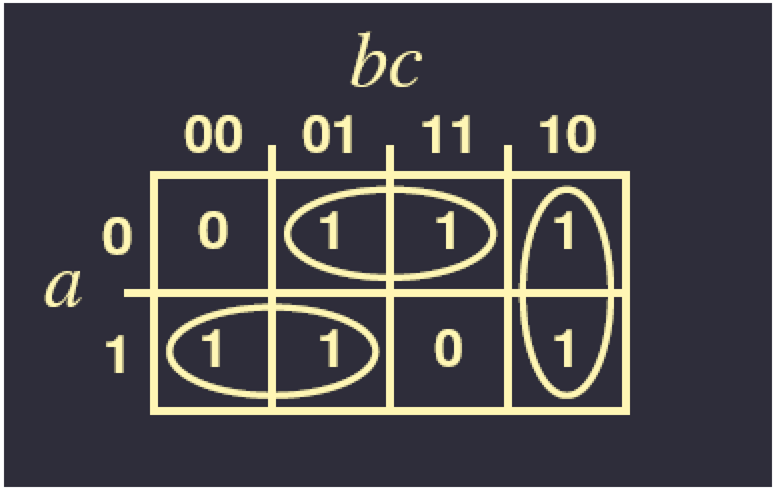
Cyclic Core

After eliminating dominant rows and dominated columns we may end up with a reduced table where there are no more dominance relationships
An implicant table in this form, it is called a cyclic core

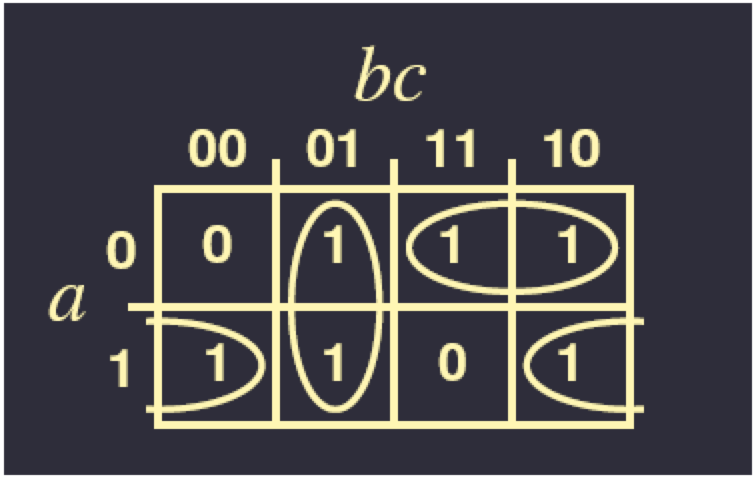


Assume the function shown in the K-Map above is given

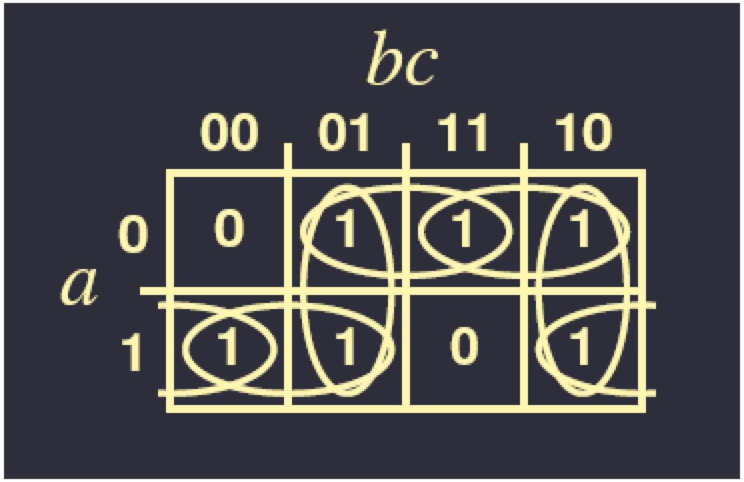
Cyclic Core



Cyclic Core



Cyclic Core



Circled groups of “1’s” are all prime implicants we can identify

Cyclic Core

a \ bc	00	01	11	10
0	0	1	1	1
1	1	1	0	1

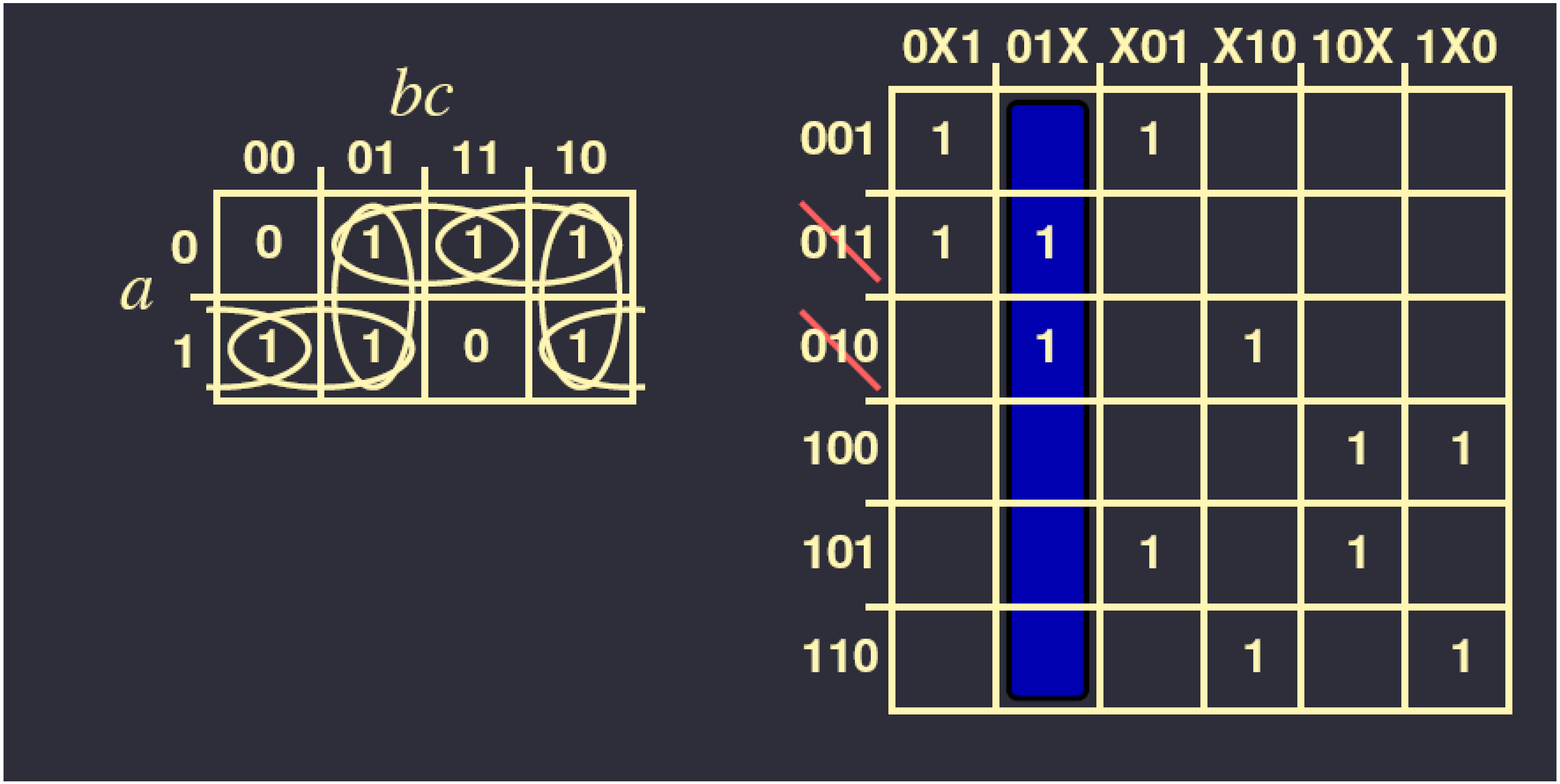
	0X1	01X	X01	X10	10X	1X0
001	1		1			
011	1	1				
010		1		1		
100					1	1
101			1		1	
110				1		1

Cyclic Core

a \ bc	00	01	11	10
0	0	1	1	1
1	1	1	0	1

	0X1	01X	X01	X10	10X	1X0
~~001~~	1		1			
~~011~~	1	1				
010		1		1		
100					1	1
101			1		1	
110				1		1

Cyclic Core


bc
00 01 11 10
a
0 0 1 1 1
1 1 1 0 1
0X1 01X X01 X10 10X 1X0
001 1 1
011 1 1
010 1 1
100 1 1
101 1 1
110 1 1

Solving the Cyclic Core with Branch-and-Bound

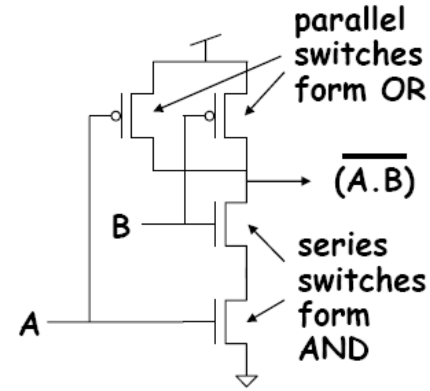
- Will proceed to completed solution if the implicant table reduces to “empty” after eliminating all dominant rows and dominated columns
- If a cyclic core remains we need to apply some exhaustive search method to find which subset of implicants from the cyclic core will yield a covering with minimum cardinality
  - The Branch-and-Bound technique is used for this purpose

Reading on Two-level Minimization

- Implicant Selection for the Quine-McCluskey Algorithm for two-level minimization (QM_ImplicantSelection.pdf)
- Sum of Product (POS) and Product of Sum (POS) for K-Map

Appendix

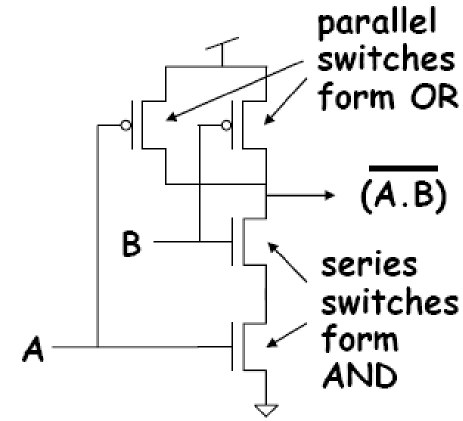
Building CMOS Logic Gate



- Pulldown: realize "0" function
  - $\overline{F}=0$
  - Take inverted function $\overline{F}$
  - Make network using NMOS with inputs
    - AND - in series, OR - in parallel
  - Connect to ground ($\overline{F}=0$)
- Example: NAND: $F=\overline{AB}$
  - $\overline{F} = AB$; series connected NMOS to ground

Building CMOS Logic Gate

- Pullup: realize “1” function
  - F=1
  - But use inverted input
    - Use De Morgan Law: $F=\overline{AB}=\overline{A}+\overline{B}$
  - Make network using PMOS with inputs
    - AND - in series, OR - in parallel
  - Connect to vdd (F=1)
- Example: NAND: $F=\overline{AB}$
  - $F = \overline{A}\overline{B}$; parallel connected PMOS to vdd



Example

- $F = \overline{A \cdot B + C \cdot D}$

Pull-up: $F = (\overline{A}+\overline{B}) \cdot (\overline{C}+\overline{D})$

Pull-down: $\overline{F} = A \cdot B + C \cdot D$