Karnaugh Maps
for Simplification of Digital Logic Functions

CS 64: Computer Organization and Design Logic
Lecture #12
Winter 2019

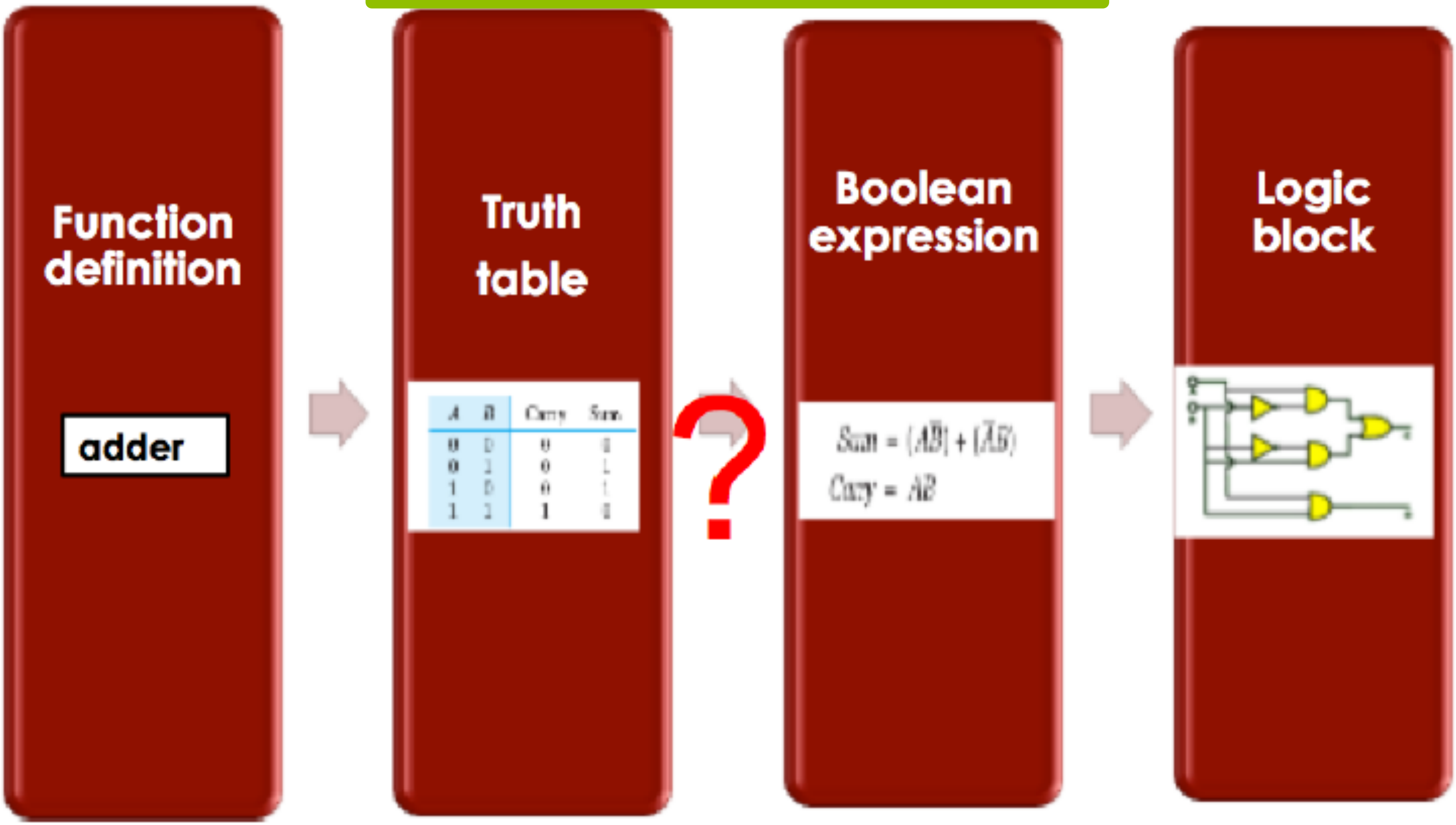
Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- Lab #6 on Thursday
  - You don't have to go to lab
    (so this time, we won't take attendance)
  - But the TAs will be there if you need help!
  - Due by ***<u>Monday</u>***

Digital Circuit Design Process

CAN THIS PROCESS BE REVERSED?



More Simplification Examples

Simplify the Boolean expression:

- **(A+B+C).(D+E)' + (A+B+C).(D+E)**

Simplify the Boolean expression and write it out on a truth table as proof

- **X.Z + Z.(X'+ X.Y)**

Use DeMorgan's Theorm to re-write the expression below using at least one OR operation

- **NOT(X + Y.Z)**

Scaling Up Simplification

- When we get to *more* than 3 variables, it becomes challenging to use truth tables
- We can instead use ***Karnaugh Maps*** to make it immediately apparent as to what can be simplified

Example of a K-Map

	A	B	f(A,B)
0	0	0	a
1	0	1	b
2	1	0	c
3	1	1	d

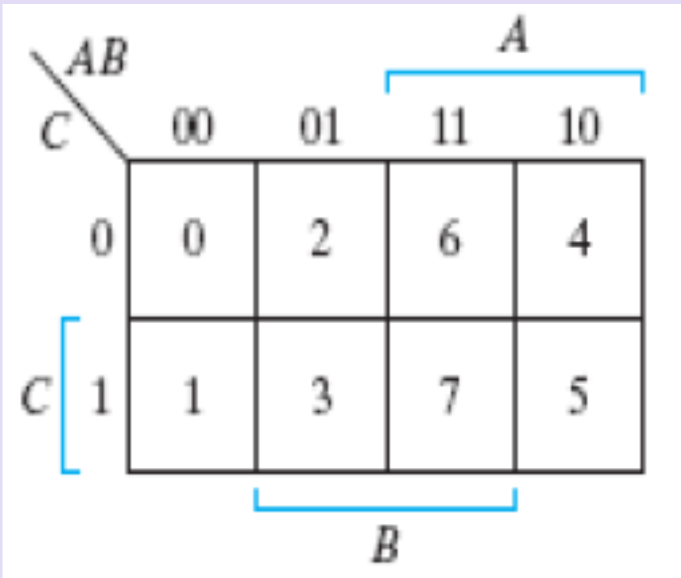
B \ A	0	1
0	a	c
1	b	d

B \ A	0	1
0	0	2
1	1	3

A	B	f(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

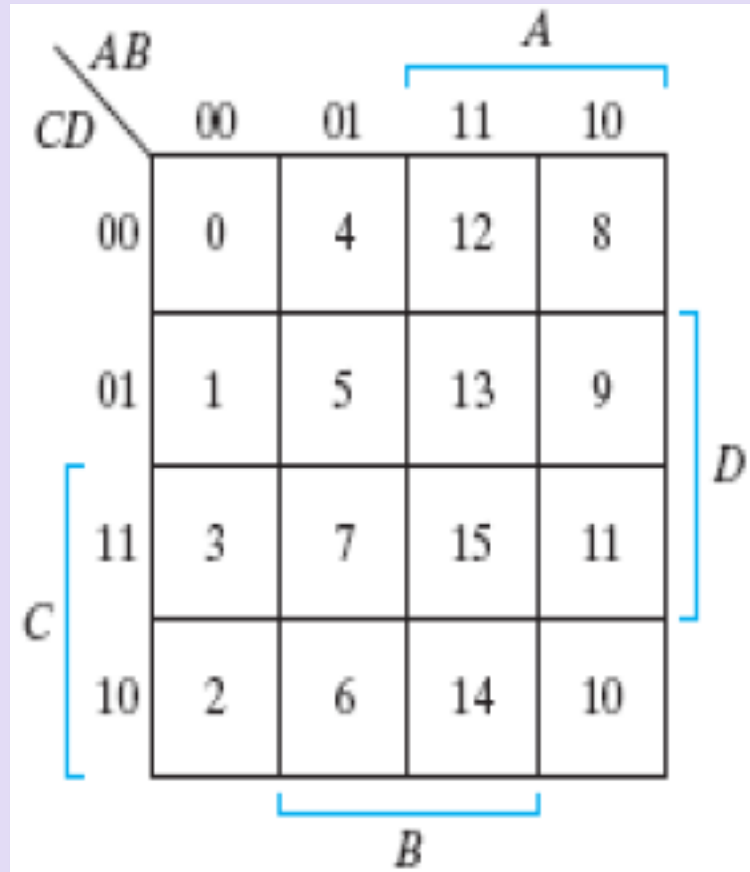
B \ A	0	1
0	0	1
1	1	1

K-Maps with 3 or 4 Variables



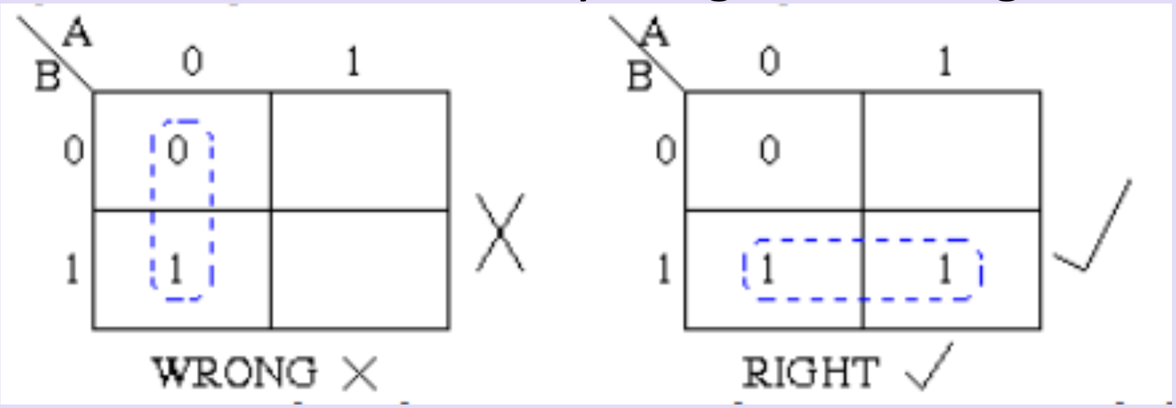
Note the adjacent placement of:
<u>00 01 11 10</u>

It's NOT:
<u>00 01 10 11</u>

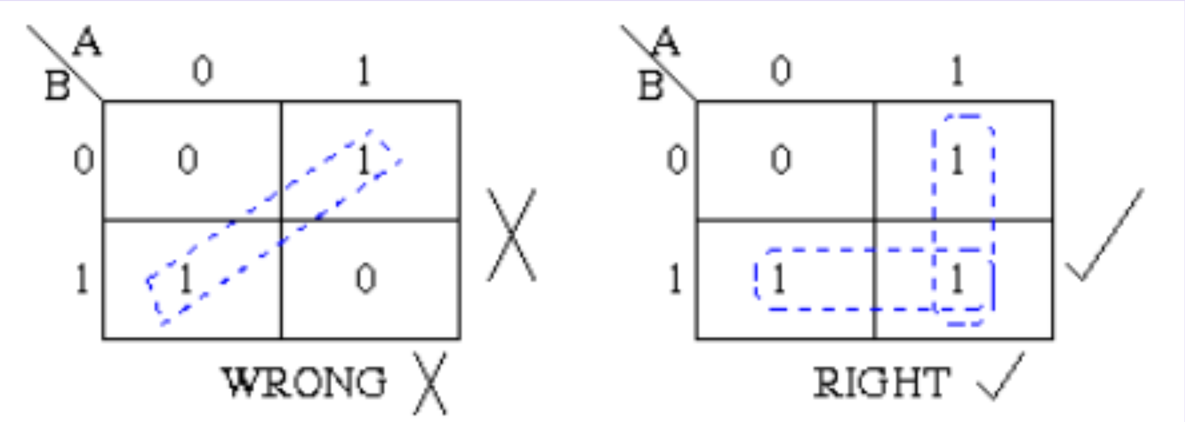


Rules for Using K-Maps for Simplification

1. Group together **adjacent cells** containing "1"
2. Groups should **not include** anything containing "0"

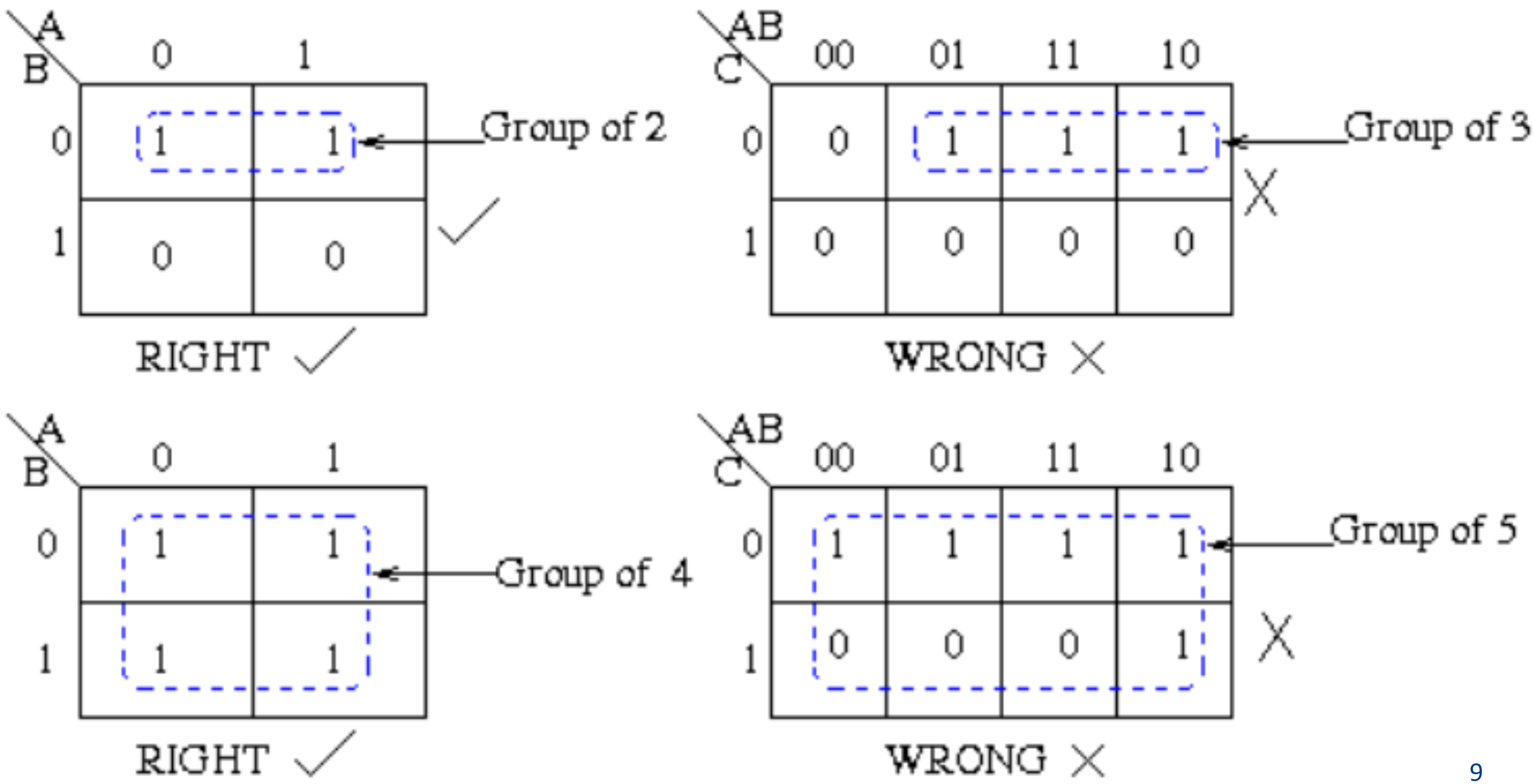


3. Groups may be horizontal or vertical, but **not diagonal**



Rules for Using K-Maps for Simplification

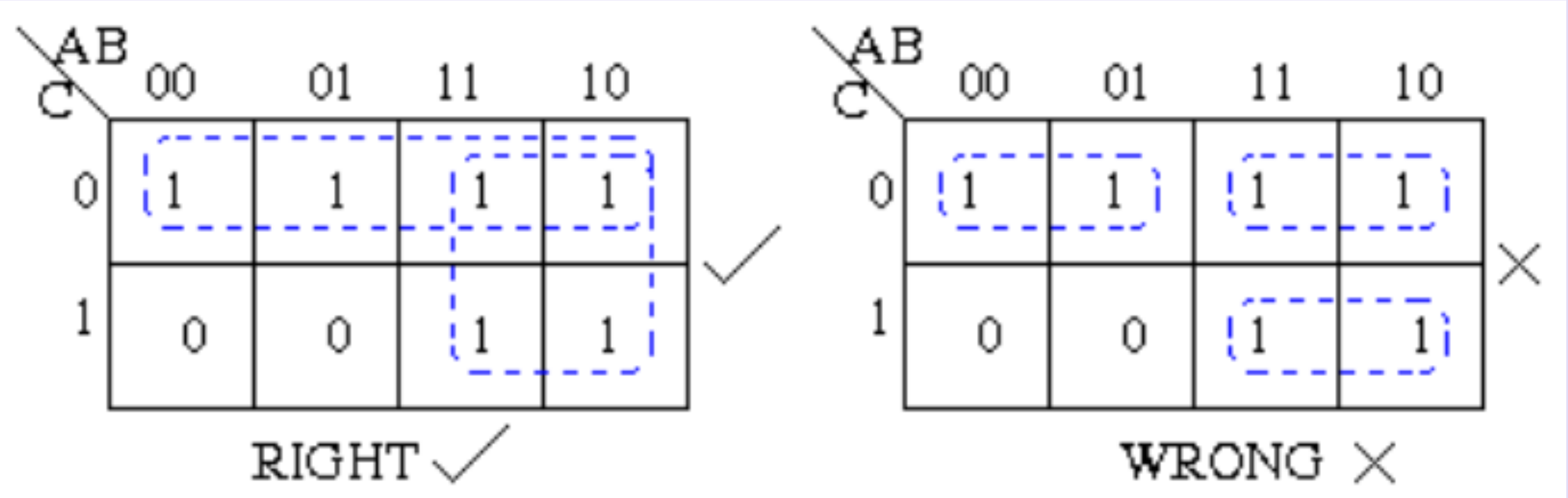
4. Groups must contain 1, 2, 4, 8, or in general 2^n cells.



Rules for Using K-Maps for Simplification

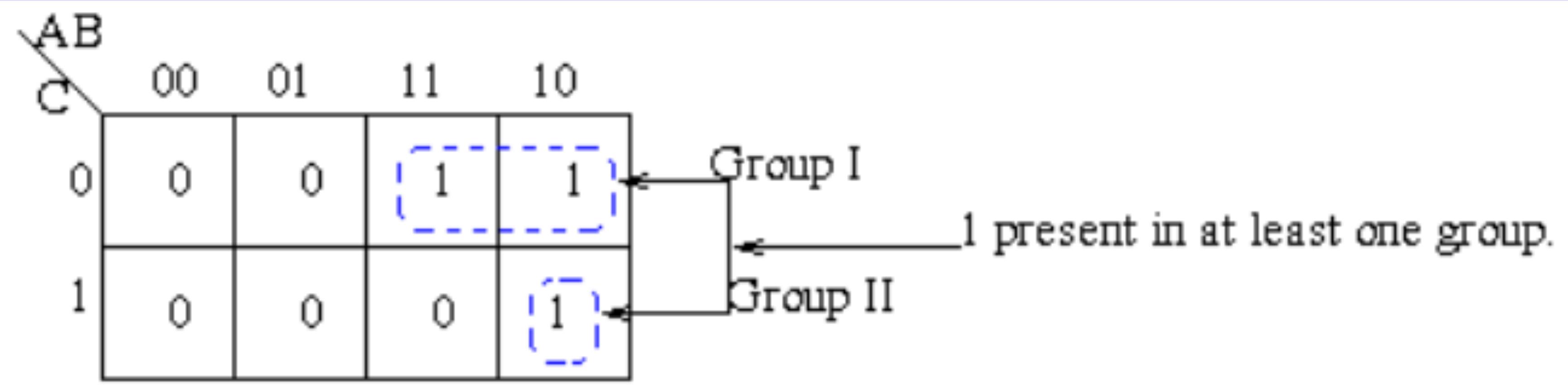
5. **Each group must be as large as possible**

   (Otherwise we're not being as minimal as we can be, even though we're not breaking any Boolean rules)



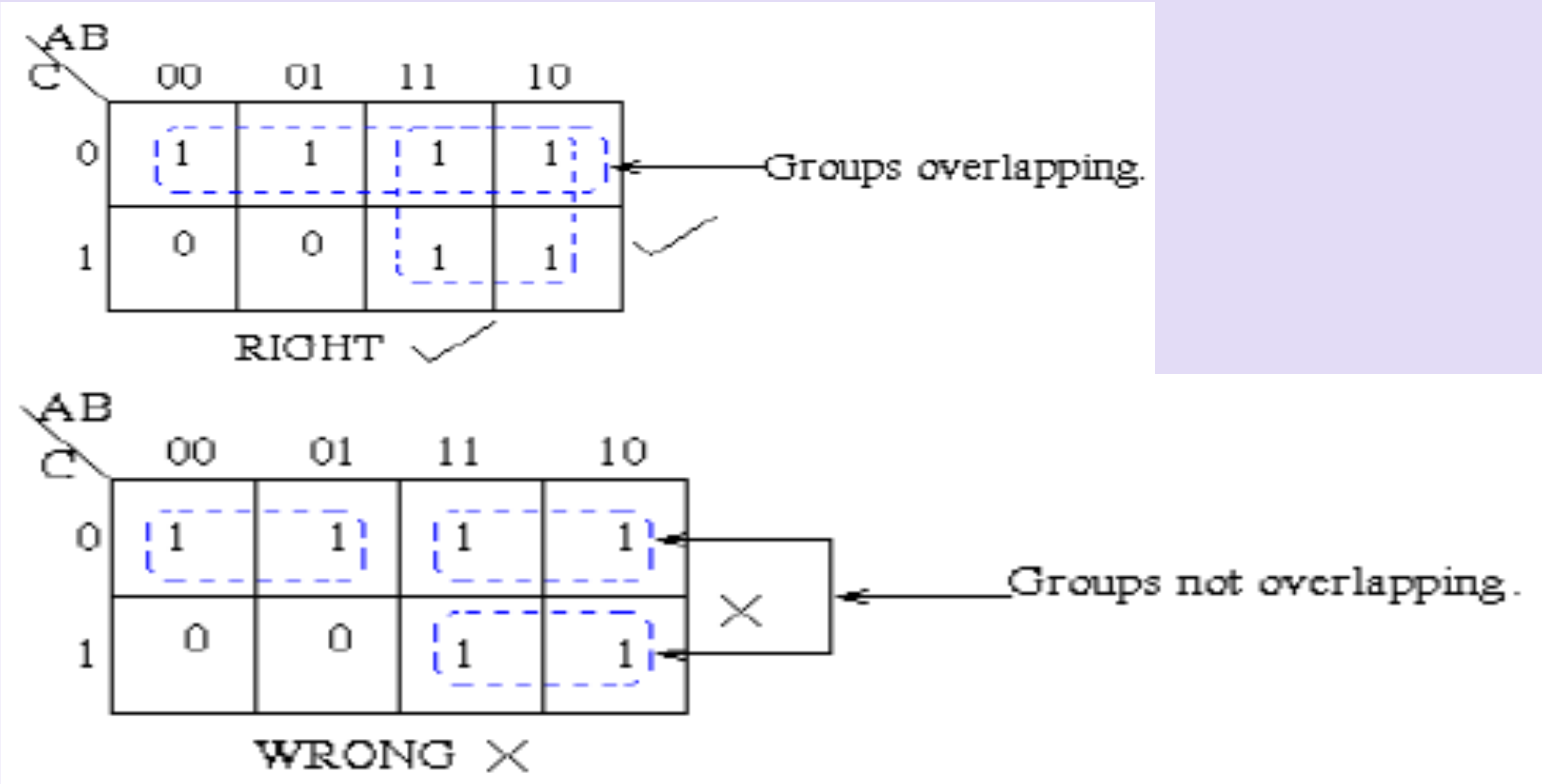
Rules for Using K-Maps for Simplification

6. Each cell containing a "1" must be at least in one group



Rules for Using K-Maps for Simplification

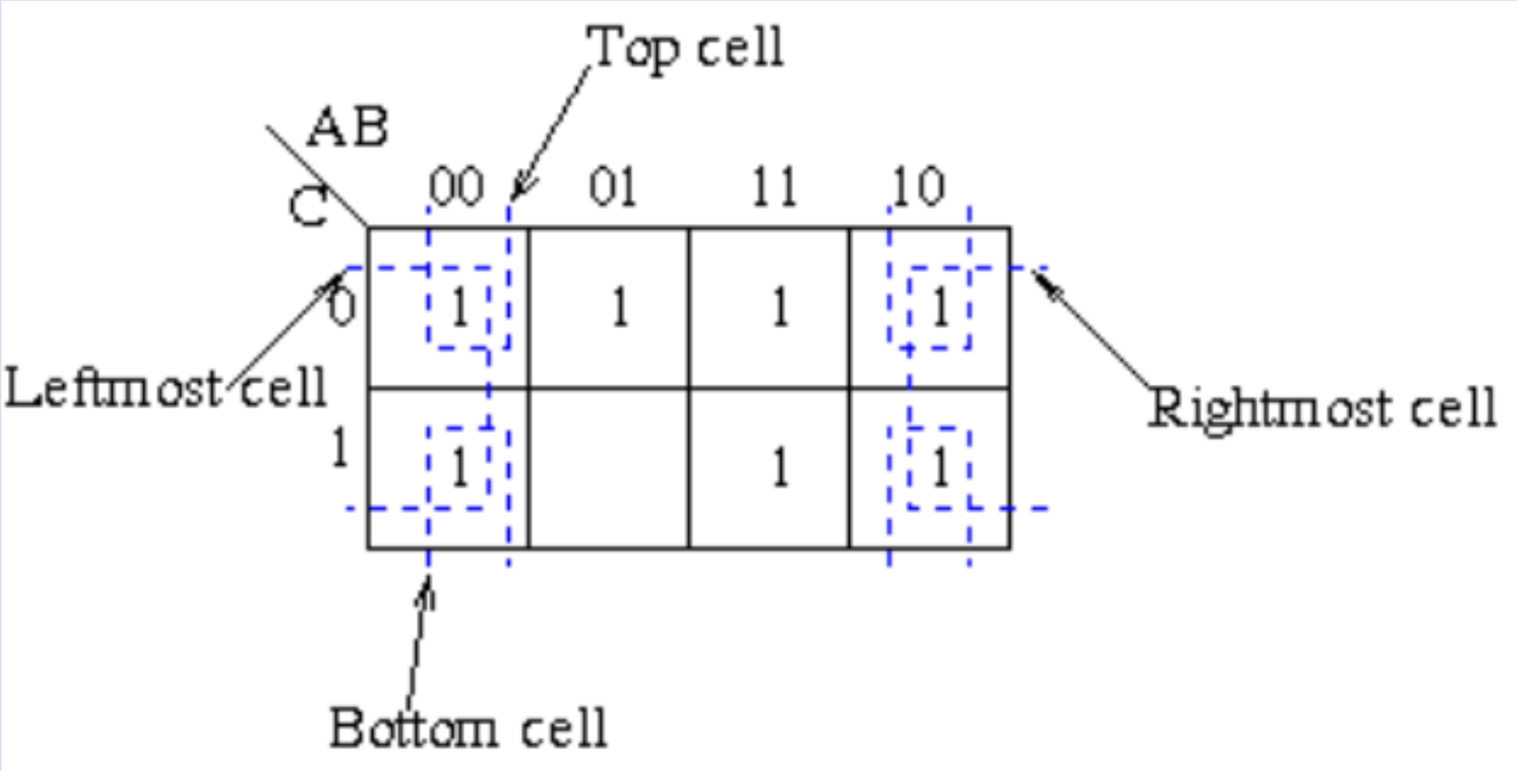
7. Groups may overlap esp. to maximize group size



Rules for Using K-Maps for Simplification

8. Groups may wrap around the table.

The leftmost cell in a row may be grouped with the rightmost cell **and** the top cell in a column may be grouped with the bottom cell.



Example 1
2 vars

F(X,Y)

= XY + Y

= Y (X + 1)

= Y

Verifying results!

F(X,Y) = Y

Y = 1 column

X \ Y	0	1
0		1
1		1

Example 2
3 vars

F(X,Y,Z)

= XZ + Z(X'+ XY)

= XZ + ZX’ + ZXY

= Z (X + X’ + XY)

= Z (1 + XY)

= Z

Y = 1 *X = 1*

Z \ XY	00	01	11	10
0				
1	1	1	1	1

Verifying results!

F(X,Y,Z) = Z

Example 3
3 vars

Class Ex.

!A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

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

F(X,Y,Z) = !C + !A

Example 4
4 vars

F(A,X,Y,Z)

= AX + Z(X+A'+Y)

= AX + ZX+ ZA'+ ZY

Y = 1 *X = 1*

AZ \ XY	00	01	11	10
00				
01	1	1	1	1
11		1	1	1
10			1	1

Z = 1

A = 1

F(A,X,Y,Z) = ZA' + AX + ZY

Example 4
4 vars

Class Ex.

F(A,B,C,D)

$= ABCD' + ABC'D + CD + A'B' + C'D$

B = 1, *A = 1* (AB columns); *D = 1*, *C = 1* (CD rows)

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

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

K-Map Rules Summary

1. Groups can contain only 1s
2. Only 1s in adjacent groups are allowed
3. Groups may ONLY be horizontal or vertical (no diagonals)
4. The number of 1s in a group must be a power of two (1, 2, 4, 8...)
5. Groups must be as large AND as few in no.s as “legally” possible
6. All 1s must belong to a group, even if it’s a group of one element
7. Overlapping groups are permitted
8. Wrapping around the map is permitted

Exploiting "Don't Cares"

- An *output variable* that's designated "don't care" (symbol = X) means that it could be a **0** or a **1** (i.e. we "don't care" which)
  - That is, it is **unspecified**, usually because of invalid inputs

Example of a Don’t Care Situation

- Consider coding all decimal digits (say, for a digital clock app):
  - 0 thru 9 --- requires how many bits?
    - 4 bits
  - But! 4 bits convey more numbers than that!
    - Don’t forget A thru F!
- Not all binary values map to decimal

Example Continued…

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binary	Decimal
1000	8
1001	9
1010	X
1011	X
1100	X
1101	X
1110	X
1111	X

Don’t Care: So What?

- Recall that in a K-map, we can only group 1s
- Because the value of a don’t care is irrelevant, we can treat it as a 1 ***if it is convenient to do so*** (or a 0 if that would be more convenient)

Example

- A circuit that calculates if the 4-bit binary coded single digit decimal **input % 2 == 0**

- So, although 4-bits will give me numbers from 0 to 15, I don't care about the ones that yield 10 to 15.

I3	I2	I1	I0	R
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**1**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**1**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**X**
1	0	1	1	**X**
1	1	0	0	**X**
1	1	0	1	**X**
1	1	1	0	**X**
1	1	1	1	**X**

Example as a K-Map

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

If We **Don't Exploit** "Don't Cares"

$$R = \overline{I_1}\,\overline{I_0}\,\overline{I_3} + I_1\overline{I_0}\,\overline{I_3} + \overline{I_0}\,\overline{I_1}\,\overline{I_2}\,I_3$$

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

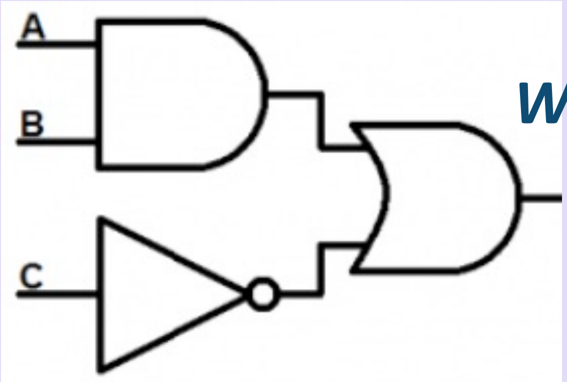
If We **DO Exploit** "Don't Cares"

$R = \overline{I_0}$

I_3I_2 \ I_1I_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

Combinatorial Logic Designs

- When you *combine* multiple logic blocks together to form a more complex logic function/circuit



What is the output?

$A.B + \overline{C}$

What is its truth table?

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

What is its K-Map?

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

Exercise 1

- Given the following truth table, draw the resulting logic circuit
  - **STEP 1**: Draw the K-Map and simplify the function
  - **STEP 2**: Construct the circuit from the now simplified function

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

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

Exercise 1 – Step 1

Get the simplified function

B = 1 *A = 1*

D = 1 *C = 1*

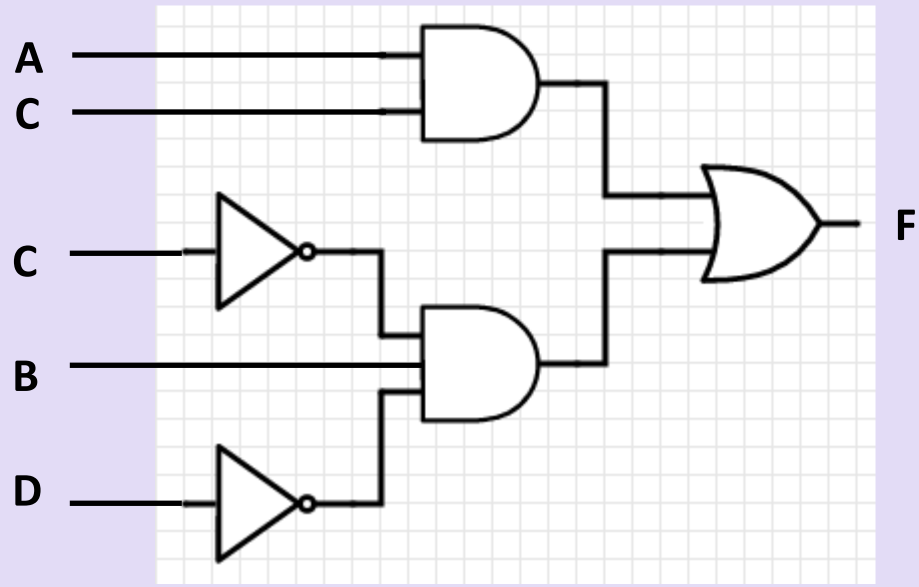
CD \ AB	00	01	11	10
00		1	1	
01				
11			1	1
10			1	1

F(A,B,C) = B.C'.D' + A.C

Exercise 1 – Step 2

Draw the logic circuit diagram

F(A,B,C) = B.C’.D’ + A.C



Exercise 2

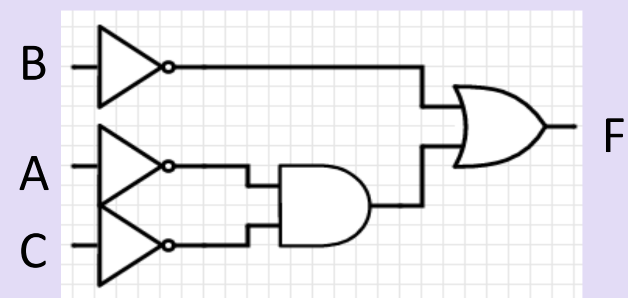
Class Ex.

- Given the following truth table, draw the resulting logic circuit

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

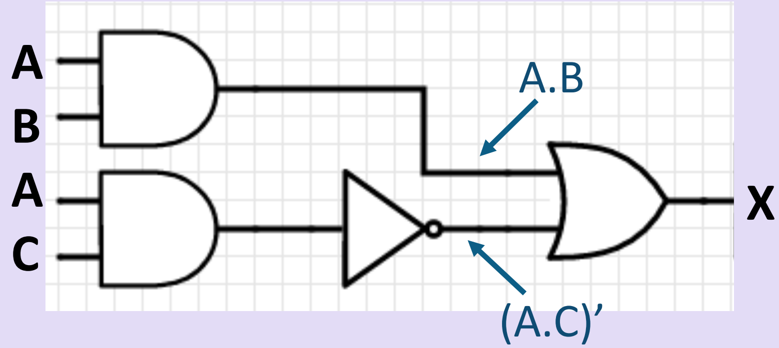
C \ AB	00	01	11	10
0	1	1		1
1	1			1

F(A,B,C) = B' + A'.C'



Exercise 3

- Given the following schematic of a circuit, (a) write the function and (b) fill out the truth table:



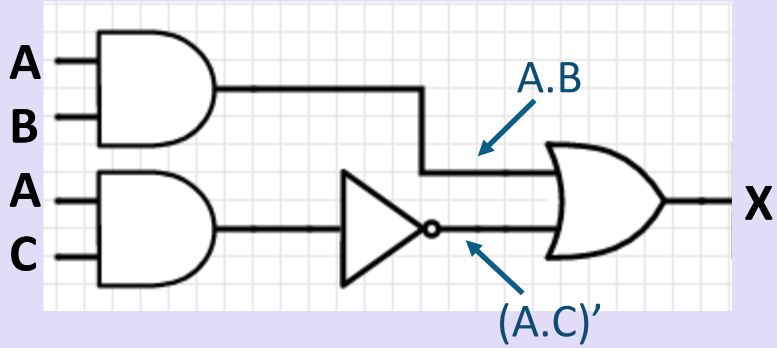
A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

X = A.B + (A.C)'

(note that also means: **X = A.B + A' + C'**)

Exercise 3

- Given the following schematic of a circuit, (a) write the function and (b) fill out the truth table:



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

X = A.B + (A.C)’

(note that also means: **X = A.B + A’ + C’**)

YOUR TO-DOs

- Lab 6!

</LECTURE>