

# Optimizing Our Design!

Prof. Usagi

[Home](#)
[Questions](#)
[Tags](#)

# How do I ask and answer homework questions?

Asked 11 years, 5 months ago   Active 4 months ago   Viewed 50k times

The guidelines outlined below are rooted in two principles:

- **It is okay to ask about homework.** For one, it would be unreasonable to expect a student to have wanted to. Stack Overflow exists to help programmers with programming problems, both simple and complex, and homework is a common source of such problems.
- **Providing an answer that doesn't help a student learn is not the goal.** Therefore you might choose to treat homework questions differently from other questions.

## Asking about homework

- **Make a good faith attempt to solve the problem yourself.** If you haven't tried to solve the problem on your part your question will likely be booted off the site.
- **Ask about *specific* problems with your *existing* implementation.** If you haven't tried to solve the problem on your part or searching for a solution is likely to be a better resource at this stage than Stack Overflow.
- **Be aware of school policy.** If your school has a policy against asking for help with homework, make sure you are aware of it before you ask for / receive help. You should mention any such specific restrictions (for example, you can receive help with understanding the problem, but not with the code) in the question so that those providing assistance are aware of them.
- **Never use code you don't understand.** It definitely won't help you learn (and it could be, at best, worse) if you don't understand the code you turned in.

- **Understand the difference between "asking a question about your homework" and "asking a specific question about the code in your homework".** You should never ask a question *about* your homework because more often than not it will not meet the recommendations in the rest of this question. Instead, ask the question about the code you wrote to solve your homework problem, and be specific with the inputs, desired outputs, and error messages. It is ideal if you take your code and create an [MCVE](#) instead of pasting your entire code, especially if it is a long code block.



stack  
overflow

## Answering homework questions

Help the asker in the correct direction. Genuine attempts to help are appreciated, but trying to provide that is usually appreciated for

complete code sample if you believe it would not help the asker. You can use pseudo-code first, and, in the spirit of creating a learning environment, come back after a suitable amount of time and edit your answer. This way, the student still has to write their own code, but they have a reference available after the assignment has ended.

Do not include artificial constraints, and honor those constraints. This may affect whether or not a question should be closed as

homework questions in good faith, even if they break the rules. Questions that could merit downvotes even if the question weren't obvious at first glance that a question is homework, should be marked as such. See it here. It is a good idea to suggest editing the

question if you haven't yet learned something obvious or developed the question as a seasoned programmer. Do add a *respectful* comment or suggestion for better practices and better style.

- *Don't* downvote a homework question that follows the guidelines and was asked in good faith.
- It's okay to ask if a question is homework, but be **polite**.
- As with homework questions, other questions along the lines of "[plz send teh codez](#)" may be closed as "too broad". Use your best judgment. Remember: students are *new* programmers and often do not yet understand what is expected of them on the site. We should politely and patiently help them gain that understanding.

# Recap: Canonical form — Sum of "Minterms"

Input		Output
X	Y	
0	0	0
0	1	0
1	0	1
1	1	1

A minterm

$$f(X,Y) = \underline{XY'} + \underline{XY}$$

Sum (OR) of "minterms"

**XNOR**

Input		Output
A	B	
0	0	1
0	1	0
1	0	0
1	1	1

$$f(A,B) = \underline{A'B'} + \underline{AB}$$

# Recap: Canonical form — Product of "Maxterms"

A "maxterm"

$$f(X,Y) = \underline{(X+Y)} \underline{(X+Y')}$$

Product of maxterms

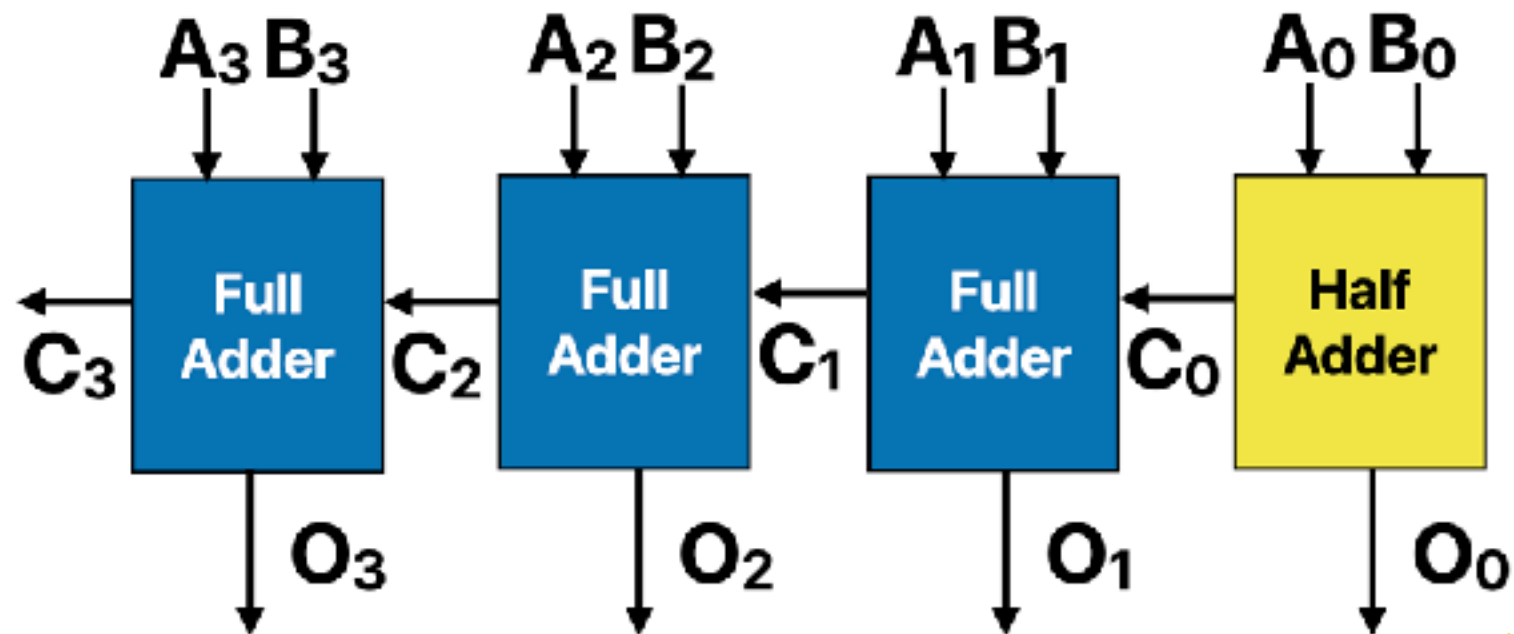
Input		Output
X	Y	
0	0	0
0	1	0
1	0	1
1	1	1

**XNOR**

Input		Output
A	B	
0	0	1
0	1	0
1	0	0
1	1	1

$$f(A,B) = \underline{(A+B')} \underline{(A'+B)}$$

# Recap: The Adder



```
module FA( input a,
           input b,
           input cin,
           output cout,
           output out );
    assign out = (~a&b&~cin)|(a&~b&~cin)|(~a&~b&cin)|(a&b&cin);
    assign cout = (a&b&~cin)|(~a&b&cin)|(a&~b&cin)|(a&b&cin);;
endmodule
```

```
module HA( input a,
           input b,
           output cout,
           output out );
    assign out = (~a & b)|(a & ~b);
    assign cout = a&b;
endmodule
```

```
module adder( input[3:0] A,
              input[3:0] B,
              output[3:0] O,
              output cout);
```

```
    wire [2:0] carries;
```

```
    HA ha0(.a(A[0]), .b(B[0]), .out(O[0]), .cout(carries[0]));
```

```
    FA fa1(.a(A[1]), .b(B[1]), .cin(carries[0]), .out(O[1]), .cout(carries[1]));
```

```
    FA fa2(.a(A[2]), .b(B[2]), .cin(carries[1]), .out(O[2]), .cout(carries[2]));
```

```
    FA fa3(.a(A[3]), .b(B[3]), .cin(carries[2]), .out(O[3]), .cout(cout));
```

```
endmodule
```

**Connecting ports by name yields clearer and less buggy code.**

# Recap: Testing the adder!

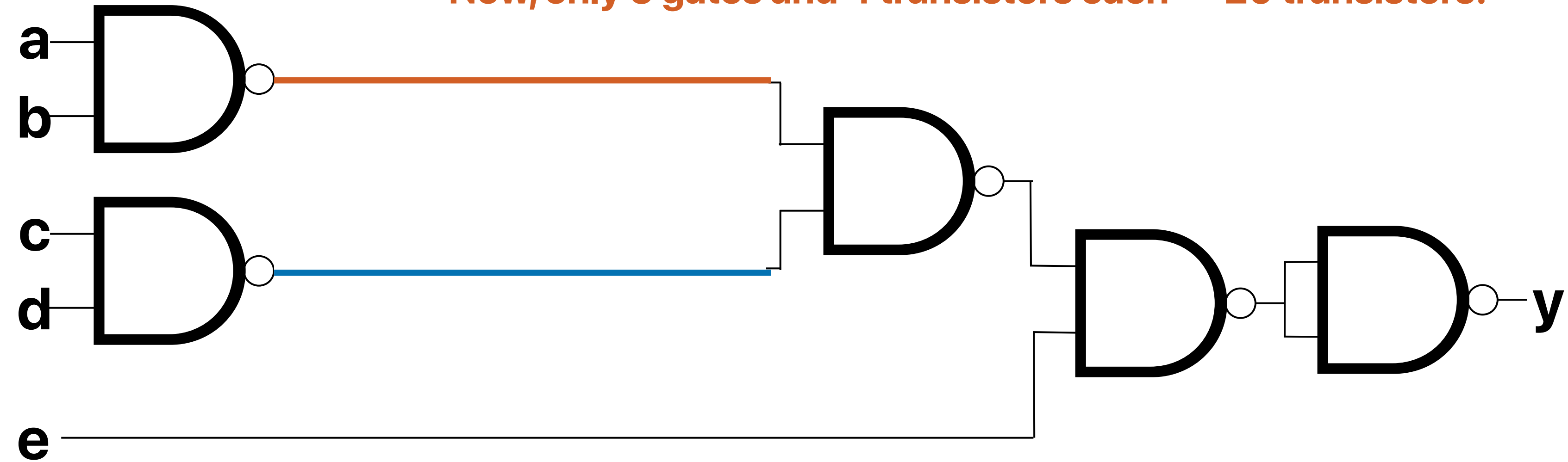
```
`timescale 1ns/1ns // Add this to the top of your file to set time scale
module testbench();
reg [3:0] A, B;
reg C0;
wire [3:0] S;
wire C4;
adder uut (.B(B), .A(A), .sum(S), .cout(C4)); // instantiate adder

initial
begin
A = 4'd0; B = 4'd0; C0 = 1'b0;
#50 A = 4'd3; B = 4'd4; // wait 50 ns before next assignment
#50 A = 4'b0001; B = 4'b0010; // don't use #n outside of testbenches
end

endmodule
```

# Recap: You can also use only NANDs

Now, only 5 gates and 4 transistors each — 20 transistors!



# Outline

- Simplifying design using “theorems”
- Karnaugh maps



**Can we simplify these functions?**

# Laws in Boolean Algebra

	OR	AND
Associative laws	$(a+b)+c=a+(b+c)$	$(a \cdot b) \cdot c=a \cdot (b \cdot c)$
Commutative laws	$a+b=b+a$	$a \cdot b=b \cdot a$
Distributive laws	$a+(b \cdot c)=(a+b) \cdot (a+c)$	$a \cdot (b+c)=a \cdot b+a \cdot c$
Identity laws	$a+0=a$	$a \cdot 1=a$
Complement laws	$a+a'=1$	$a \cdot a'=0$

Duality: We swap all operators between (+,.) and interchange all elements between (0,1). For a theorem if the statement can be proven with the laws of Boolean algebra, then the duality of the statement is also true.

# Some more tools

	OR	AND
DeMorgan's Theorem	$(a + b)' = a'b'$	$a'b' = (a + b)'$
Covering Theorem	$a(a+b) = a+ab = a$	$ab + ab' = (a+b)(a+b') = a$
Consensus Theorem	$ab+ac+b'c = ab+b'c$	$(a+b)(a+c)(b'+c) = (a+b)(b'+c)$
Uniting Theorem	$a(b + b') = a$	$(a+b) \cdot (a+b') = a$
Shannon's Expansion	$f(a,b,c) = a'b' + bc + ab'c$ $f(a,b,c) = a f(1, b, c) + a' f(0, b, c)$	

# Applying Theorems

- Which of the following represents  $CB+BA+C'A$ ?
  - A.  $AB+AC'$
  - B.  $BC+AC'$
  - C.  $AB+BC$
  - D.  $AB+AC$
  - E. None of the above



	OR	AND
Associative laws	$(a+b)+c=a+(b+c)$	$(a \cdot b) \cdot c=a \cdot (b \cdot c)$
Commutative laws	$a+b=b+a$	$a \cdot b=b \cdot a$
Distributive laws	$a+(b \cdot c)=(a+b) \cdot (a+c)$	$a \cdot (b+c)=a \cdot b+a \cdot c$
Identity laws	$a+0=a$	$a \cdot 1=a$
Complement laws	$a+a'=1$	$a \cdot a'=0$
DeMorgan's Theorem	$(a+b)'=a'b'$	$a'b'=(a+b)'$
Covering Theorem	$a(a+b)=a+ab=a$	$ab+ab'=(a+b)(a+b')=a$
Consensus Theorem	$ab+ac+b'c=ab+b'c$	$(a+b)(a+c)(b'+c)=(a+b)(b'+c)$
Uniting Theorem	$a(b+b')=a$	$(a+b) \cdot (a+b')=a$
Shannon's Expansion	$f(a,b,c)=a'b'+bc+ab'c$ $f(a,b,c)=a f(1,b,c)+a' f(0,b,c)$	

# Applying Theorems

- Which of the following represents  $CB+BA+C'A$ ?

A.  $AB+AC'$

B.  $BC+AC'$

C.  $AB+BC$

D.  $AB+AC$

E. None of the above

Consensus Theorem

$$ab+ac+b'c = ab+b'c$$

$$\begin{aligned} &CB + BA1 + C'A \\ &= BC + BA(C'+C) + C'A \\ &= BC + BAC' + BAC + C'A \\ &= (1+A)BC + (B+1)AC' \\ &= BC + AC' \end{aligned}$$



# How many "OR"s?

• For the truth table shown on the right, what's the minimum number of "OR" gates we need?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5

Input			Output
A	B	C	
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	0	0	1
1	1	1	0



# How many "OR"s?

- For the truth table shown on the right, what's the minimum number of "OR" gates we need?



A. 1  $F(A, B, C) =$  Uniting Theorem

B. 2  $A'B'C' + A'B'C + A'BC' + A'BC + AB'C' + ABC'$

C. 3  $= A'B'(C' + C) + A'B(C' + C) + AC'(B' + B)$

D. 4  $= A'B' + A'B + AC'$

E. 5  $= A' + AC' = A'(1 + C') + AC'$  Distributive Laws  
 $= A' + A'C' + AC'$   
 $= A' + (A' + A)C'$   
 $= A' + C'$

Input			Output
A	B	C	
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	0

# Karnaugh maps



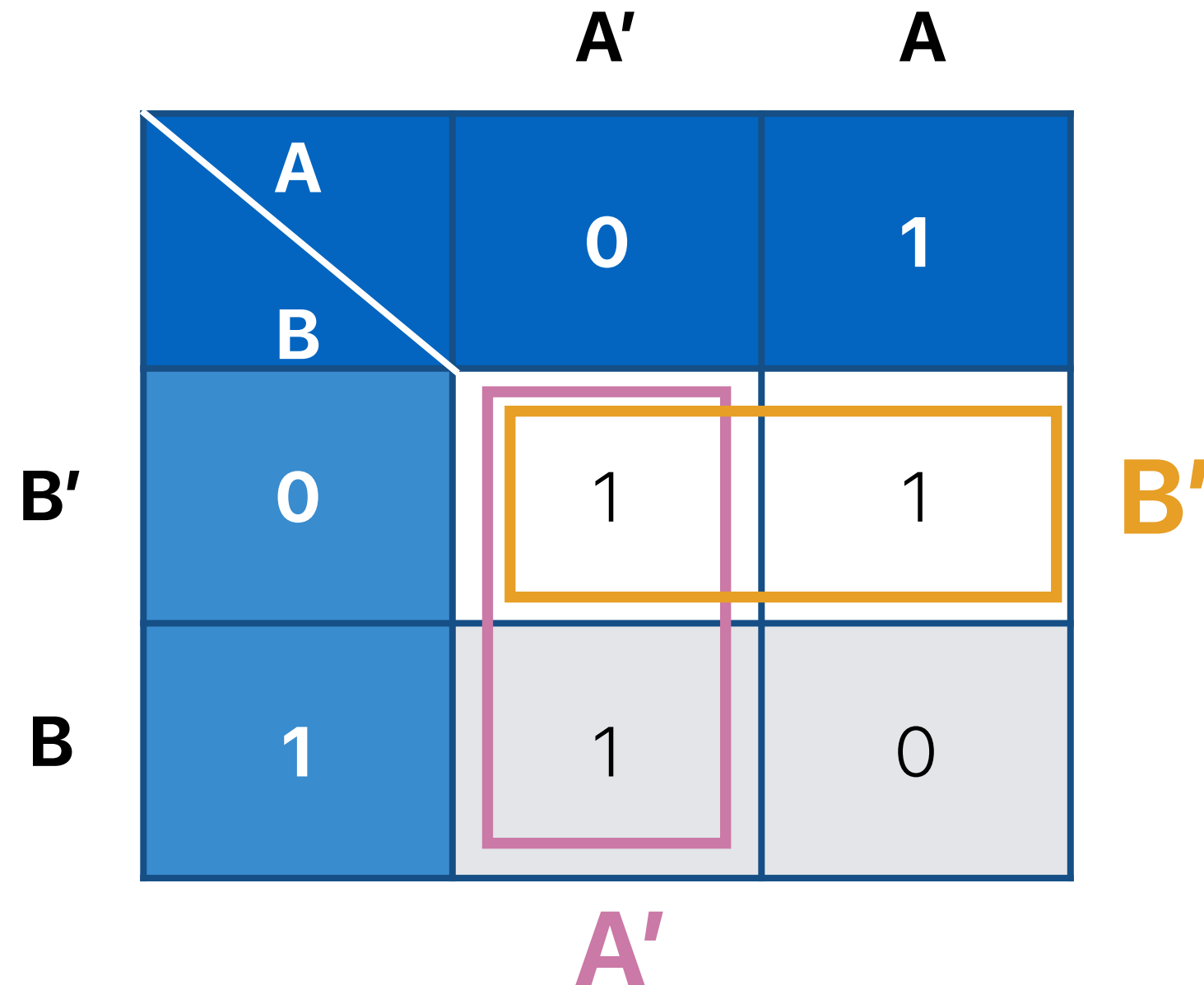
# Karnaugh maps

- Alternative to truth-tables to help visualize adjacencies
- Guide to applying the uniting theorem
- Steps
  - Create a 2-D truth table with input variables on each dimension, and adjacent column(j)/row(i) only change one bit in the variable.
  - Fill each (i,j) with the corresponding result in the truth table
  - Identify ON-set (all 1s) with size of power of 2 (i.e., 1, 2, 4, 8, ...) and "unite" them terms together (i.e. finding the "common literals" in their minterms)
  - Find the "minimum cover" that covers all 1s in the graph
  - Sum with the united product terms of all minimum cover ON-sets



# 2-variable K-map example

Input		Output
A	B	
0	0	1
0	1	1
1	0	1
1	1	0



$$F(A, B) = A' + B'$$

# Practicing 2-variable K-map

- What's the simplified function of the following K-map?
  - A.  $A'$
  - B.  $A'B$
  - C.  $AB'$
  - D.  $B$
  - E.  $A$

<div>A B</div>	0	1
0	0	0
1	1	1

# Practicing 2-variable K-map

- What's the simplified function of the following K-map?

A.  $A'$

B.  $A'B$

C.  $AB'$

D.  $B$

E.  $A$

		$A'$	$A$
	$B$	0	1
	$B'$	0	0
	$B$	1	1

$B$

# 3-variable K-map?

- Reduce to 2-variable K-map — 1 dimension will represent two variables
- Adjacent points should differ by only 1 bit
  - So we only change one variable in the neighboring column
  - 00, 01, 11, 10 — such numbering scheme is so-called **Gray-code**

Input			Output
A	B	C	
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	0

		A'B'	A'B	AB	AB'
(A, B)		0,0	0,1	1,1	1,0
C	C'	0	1	1	1
C	C'	1	1	0	0

**A'**

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

# Minimum number of SOP terms

- Minimum number of SOP terms to cover the following function?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5

Input			Output
A	B	C	
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

# Minimum number of SOP terms

- Minimum number of SOP terms to cover the following function?

A. 1

**B. 2**

C. 3

D. 4

E. 5

$$F(A, B, C) = A'C' + BC'$$

		A'B'		A'B		AB		AB'	
C (A, B)		0,0		0,1		1,1		1,0	
C'	0	1		1		0		0	
	1	0		1		1		0	

*Note: In the original image, a pink box highlights the '1's in the C'=0 row (A'B' and A'B), and a green box highlights the '1's in the C=1 row (A'B and AB). The term A'C' is associated with the pink box, and BC is associated with the green box.*

Input			Output
A	B	C	
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

We don't need A'B to cover all 1s

# Minimum number of SOP terms

- Minimum number of SOP terms to cover the following function?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5

Input			Output
A	B	C	
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



# Minimum number of SOP terms

- Minimum number of SOP terms to cover the following function?

A. 1

B. 2

C. 3

D. 4

E. 5

Input			Output
A	B	C	
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

		A'B'	A'B	AB	AB'	A'B'
C	(A, B)	0,0	0,1	1,1	1,0	0,0
C'	0	1	0	0	1	1
C	1	1	0	0	1	1

# Minimum SOP for a full adder

- Minimum number of SOP terms to cover the "Out" function for a one-bit full adder?

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5

Input			Output	
A	B	Cin	Out	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

# Minimum SOP for a full adder

- Minimum number of SOP terms to cover the "Out" function for a one-bit full adder?

A. 1

B. 2

C. 3

D. 4

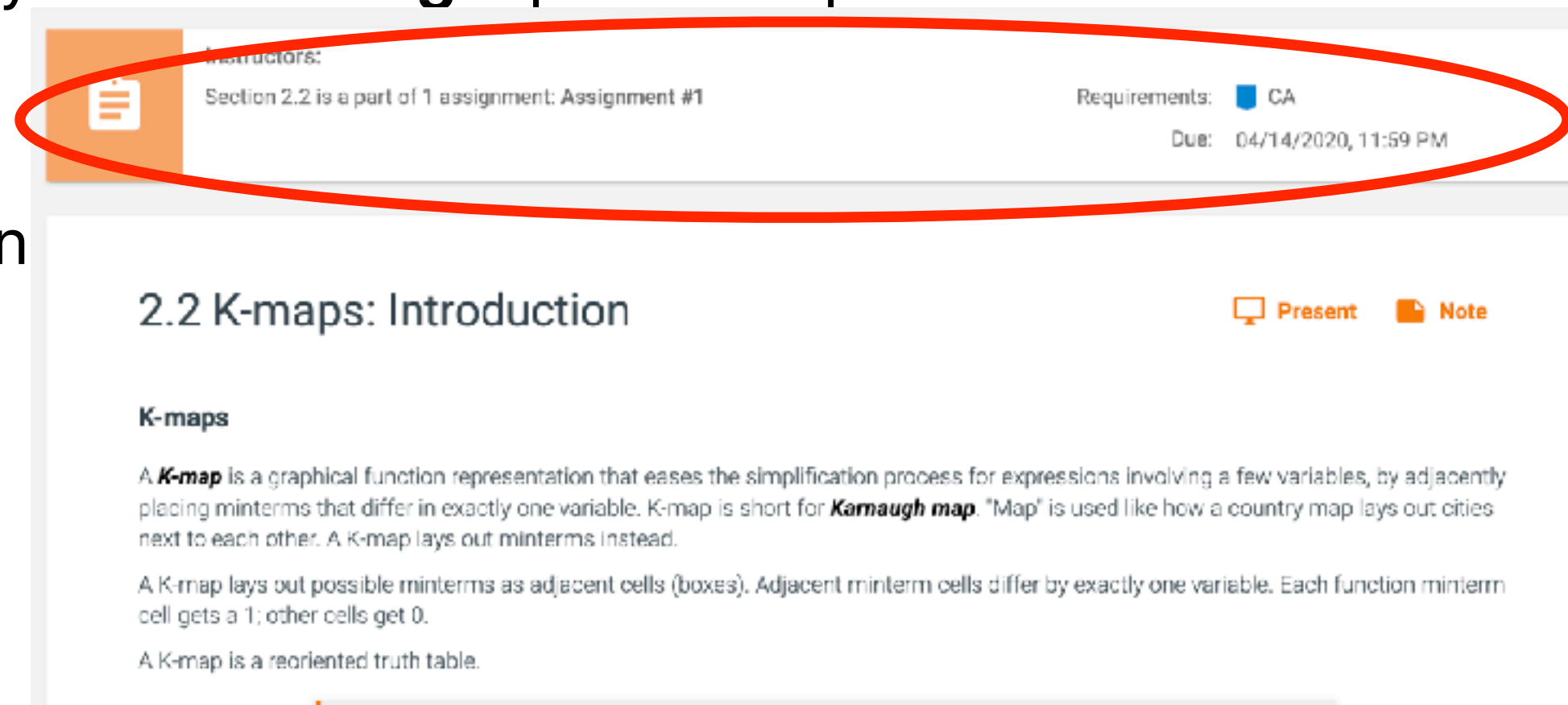
E. 5

	Out(A, B)	A'B'	A'B	AB	AB'
		0,0	0,1	1,1	1,0
Cin'	0	0	1	0	1
Cin	1	1	0	1	0

Input			Output	
A	B	Cin	Out	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

# Announcement

- Lab 1 due tonight
  - Submit through iLearn > Labs
- Reading quiz 3 due 4/14 **BEFORE** the lecture
  - Under iLearn > reading quizzes
- Assignment 1 also due 4/14
  - Submit on **zyBooks.com** directly — all **challenge** questions up to **2.2**
- Lab 2 due 4/16
  - Watch the video and read the instruction **BEFORE** your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs



The screenshot shows a course management system interface. A red oval highlights the top section, which includes a document icon, the text "Instructors:", "Section 2.2 is a part of 1 assignment: Assignment #1", "Requirements: CA", and "Due: 04/14/2020, 11:59 PM". Below this, the title "2.2 K-maps: Introduction" is displayed with "Present" and "Note" icons. The "K-maps" section contains a definition: "A **K-map** is a graphical function representation that eases the simplification process for expressions involving a few variables, by adjacently placing minterms that differ in exactly one variable. K-map is short for **Karnaugh map**. 'Map' is used like how a country map lays out cities next to each other. A K-map lays out minterms instead." It also states: "A K-map lays out possible minterms as adjacent cells (boxes). Adjacent minterm cells differ by exactly one variable. Each function minterm cell gets a 1; other cells get 0." and "A K-map is a reoriented truth table."

# Electrical Computer Science Engineering 120A

くづつ