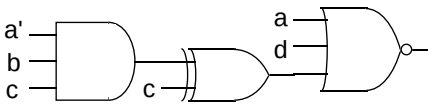
Constructing a Truth Table

- Given a circuit diagram, a truth table can easily be constructed:

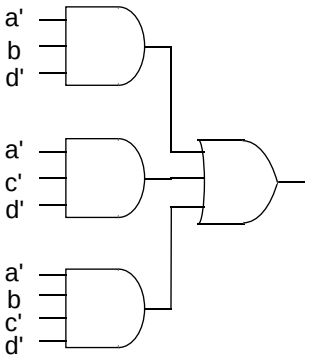


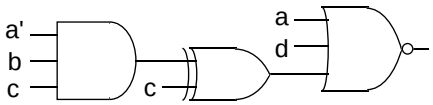
a	b	c	d	a'bc	(a'bc) ⊕c	((a'bc) ⊕c) +a +d	(((a'bc) ⊕c) +a +d)'
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	1	0
1	1	0	1	0	0	1	0
1	1	1	0	0	1	1	0
1	1	1	1	0	1	1	0

Implementing a Truth Table

- For a given truth table, there may be more than one valid implementation

a	b	c	d	x
0	0	0	0	1
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	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0





Implementing a Truth Table (cont.)

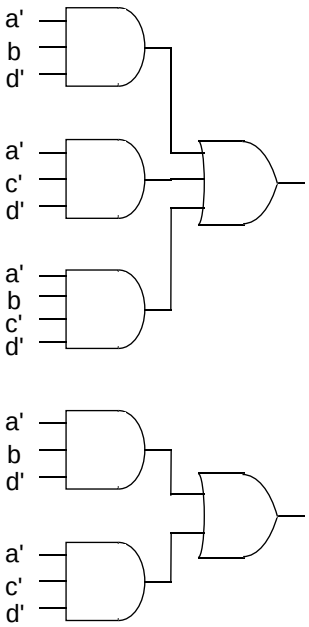
- Unfortunately, there also may be more than one 2-level implementation:

z = a'bd' + a'c'd' + a'bc'd'

= a'bd' + (a'c'd') + (a'c'd')b

= a'bd' + a'c'd'

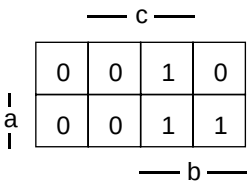
a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



3-Variable Karnaugh Maps

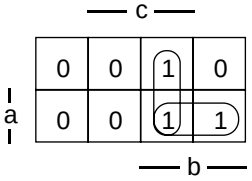
- Given a Boolean expression, we can construct a *Karnaugh map* by drawing a box as shown below and writing 1's in all the appropriate boxes whenever that term is true

z = a'bc + abc + ab



- Then we find the *smallest* set of ovals of size 1, 2, 4, or 8, that cover all the 1s (but none of the 0s), and write out the *minimized expression* from those ovals

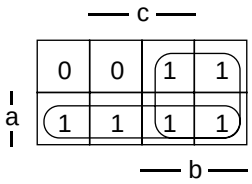
z = bc + ab



3- Variable Karnaugh Maps (cont.)

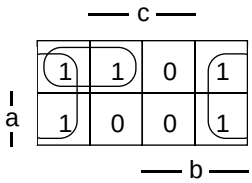
- The ovals should be as big as possible, and two or more ovals can overlap

z = a + b



- Ovals can even wrap around the ends (or top and bottom) of the table

z = c' + a'b'

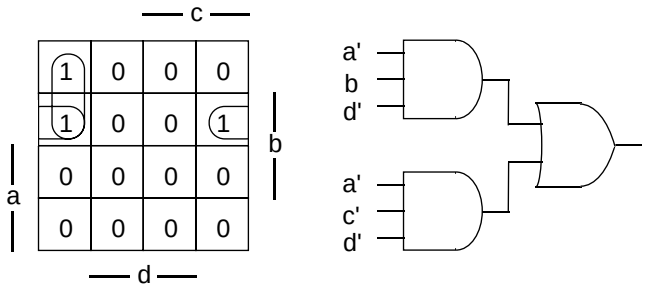


- But ovals can't go diagonally

4-Variable Karnaugh Maps

- 4-variable Karnaugh maps are similar, but now ovals can be of size 16 as well
- Now we can finally make sure that we have the minimum 2-level *and-or* (SOP) implementation of our example:

z = ((a'bc ⊕ c) + a + d)'



= a'bd' + a'c'd'

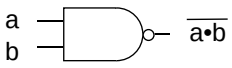
- Could also build Karnaugh map directly from truth table, entering a "1" in each box that corresponds to a combination of inputs that produce an output of 1

Negation / Universal Gate

- *Nand* and *nor* gates are very easy to construct

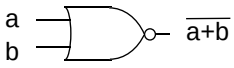
nand

a	b	$\overline{a \bullet b}$
0	0	1
0	1	1
1	0	1
1	1	0

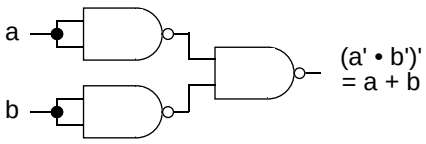


nor

a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

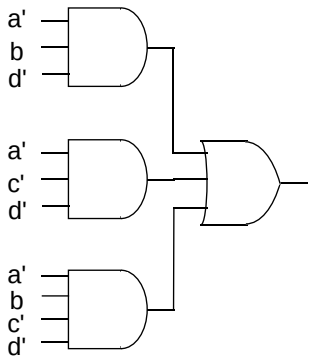


- Any digital circuit can be implemented using only *nand* or *nor* gates

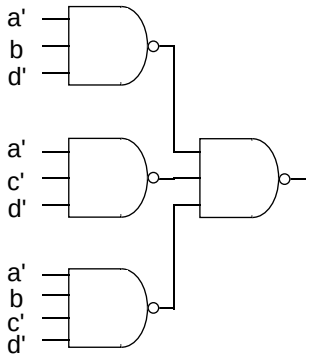


Example from Earlier in the Lecture

- In 2-level *and-or* form:

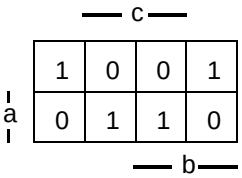


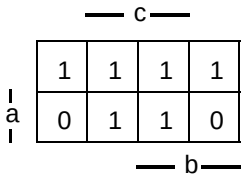
- In 2-level *nand* form:



Homework #2 — Due 9/28/98 (Part 1)

1. Find the minimized expression that corresponds to each of the following Karnaugh maps:





2. Use a Karnaugh map to minimize the 4-variable Boolean expression
   z = a'b + b'c + abc + abcd