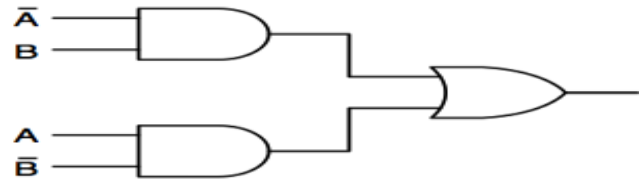


Circuits (II)

Get a Combinational Circuits from a Truth Table

- Taking a look at the basic gate XOR, it actually can be represented by AND and OR gates.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

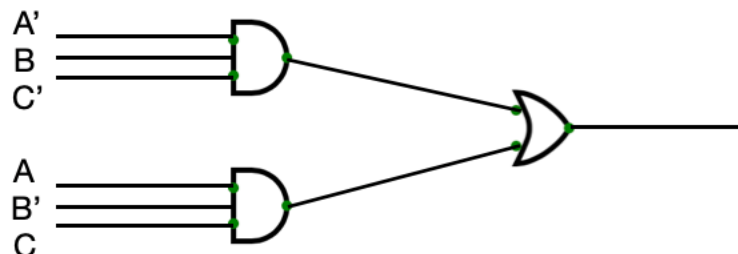


- Do you see the pattern here?
 - General rule (apply when draw a circuit from a truth table): one OR gate for final output. The OR gate is fed with AND gates, one for each 1 in the output column of the table.

Sum of Products

- You can also implement a combinational circuit with 3 inputs.
- **Exercise:** Draw a circuits using the following truth table.

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

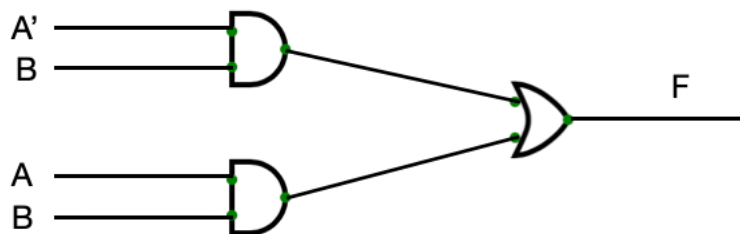


- Please note that you can also use NOT gates in the circuit. For example, the second input of the bottom AND gate is B' . You can have an OR gate and let the input of the OR gate is B . So, the output of the OR gate is B' , which is the second input of the bottom AND gate. In the above exercises, we don't use NOT gates in the circuit to draw these circuits easier.
- We can use boolean expression (**Sum of Products**) to represent a circuit.
 - Exercise: $F = A'BC' + AB'C$

Create a Circuit based on a Specification

- **Example: Odd number finder**
 - Create a circuit that can tell whether a 2-bit number is odd or not. We only need to consider unsigned numbers. The circuit will have 2 inputs which correspond to the 2 bits, and 1 output. If the input is an odd number, the output is 1; Otherwise, the output is 0.

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



Sum of products: $F = A'B + AB$

Karnaugh Map

- Now we know how to build a circuit for a specification from a truth table. However, sometimes, the circuits we got from truth tables are not in a simplest format. There are many unnecessary gates on those circuits. From the perspective of manufacturing, these unnecessary gates cost more while wasting space on the board.

- So, after we create a truth table from a specification, we need to simplify the sum of products so that we can build a circuit with minimal number of gates.
- **Karnaugh map (K-map)** is a convenient way to **simply boolean expressions of a small number of variables**. Usually it's used for up to 4 variables. It can also be used for 5 variable. Complicated used for 6 variables.
- When there are many variables in boolean expressions, simplification is not an easy task. People will then use the basic identities of boolean algebra to simplify the expressions, which is out of the scope of this course.
- So, our focus is on K-map in this course. If you are interested in using the basic identities of boolean algebra to simplify the expressions, please let me know. I can direct you to some resources and get you started.
- The map is an array of 2^n squares, representing all possible combinations of values of n variables.
- Here are the K-maps for 2, 3, and 4 variables.

AB	00	01	11	10

	BC	00	01	11	10
A					
0					
1					

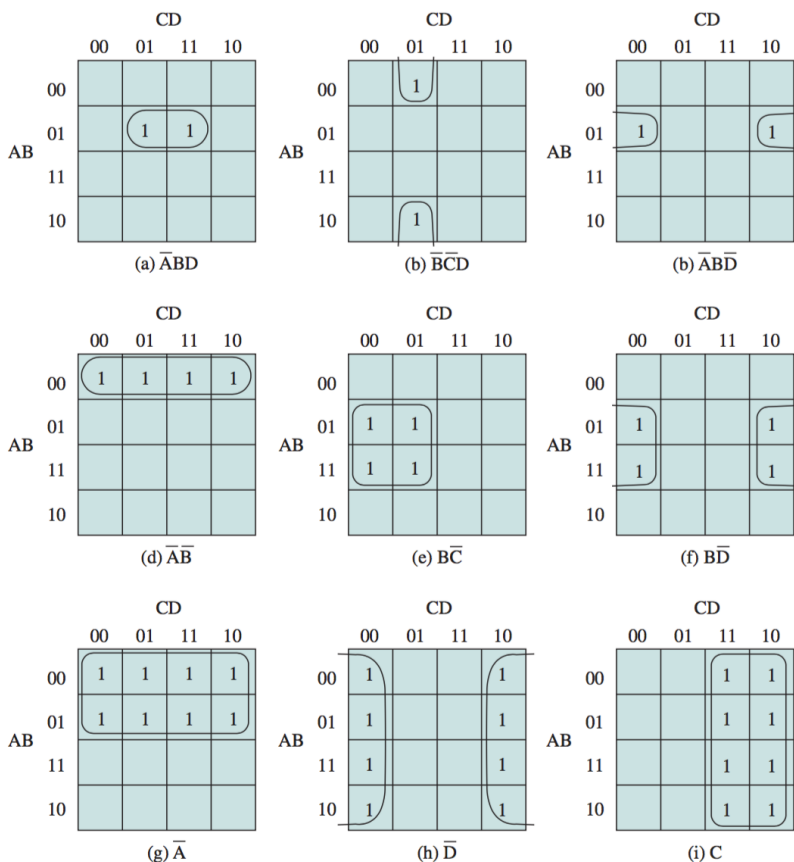
	CD	00	01	11	10
AB					
00					
01					
11					
10					

- Note that the **sequence order** is **00 01 11 10**. *Only one variable can change value at a time.*
- The first step to use K-map is to pick an appropriate map for a truth table depending on the number of variables.
- Then, we need to map the values in the truth table to the K-map by filling the corresponding squares with 1's.
- Let's draw a K-map for the Odd Number Finder example together.
 - Pick the right K-map. Since there are two variables A & B in the example, we pick the K-map for 2 variables.
 - In the truth table of the example, find a row where the F column's value is 1.
 - Find the variable values on that row.
 - Then, find the square in the K-map whose variable values are the same as those in the truth table, and mark that square with 1.

- Repeat the process till each 1 in the F column of the truth table have a corresponding mark on the K-map. The number of 1's on the K-map should be the same as the number of 1's in the F column of the truth table.
- Following the above steps, we get a marked K-map below.

AB	00	01	11	10
		1	1	

- Now, we can use the marked K-map to simplify the expressions.
- The way to simply is: **group 1's by connected blocks of powers of 2**. The concept can be extended to **include wrapping around the edge of the map**. The groups should be as large as possible and as few in number as possible, but **include every square with value 1 at least once**.
- The possible cases with block size 2, 4, and 8 in a K-map for 4 variables.



- One group is a term in the sum of products, which applies an AND operation to the variables whose value remains the same in all squares in the group.
- Why is that?
 - Each square is a row whose F column value is 1. So, that square will be an AND gate with all variables as inputs. Since all such AND gates' outputs will be the inputs for an OR gate.

groups of squares with 1s can be written as a sum of products. E.g., the expression for the group in the K-map (a) in the above figure can be written as

$$F = A'BC'D + A'BCD = A'BD(C' + C) = A'BD$$

- Now, let's revisit the odd number finder example. Apply K-map to the truth table and redesign the circuit. Will we use less gates after simplification?



- After simplification, we can connect the output to input B directly. There is no gate needed.
- To wrap up, the **design process** is:
 - Create the truth table from specification
 - Create a K-map
 - Write the boolean expression
 - Draw the logic diagram
 - Build the circuit