Digital Electronics Basics: Combinational Logic

for
Basic Electronics

http://cktse.eie.polyu.edu.hk/eie209

by
Prof. Michael Tse

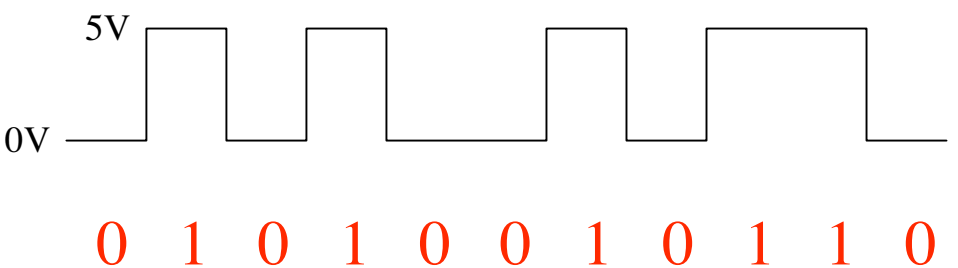
January 2005

Digital versus analog

So far, our discussion about electronics has been predominantly “analog”, which is concerned with continuously changing signals—signals whose values at different times are useful information.

There is another form of signals which is nowadays very important and is being used in an overwhelming number of applications—**DIGITAL**.

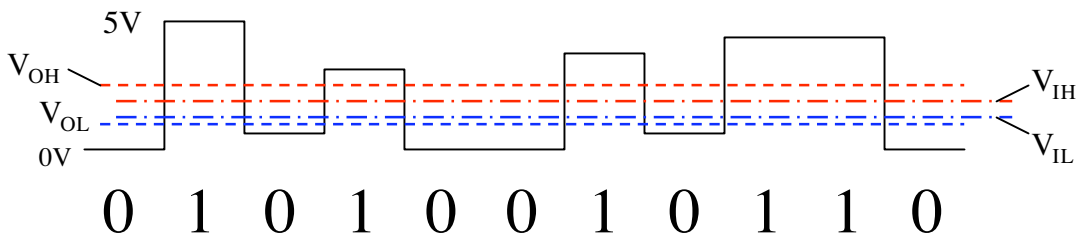
A digital signal assumes just a few analog values. For example, binary signals have two values, e.g., 0V and 5V. They transmit information in the form of a sequence of 0V and 5V segments.



Digital signals

Of course, in practical generation and detection of digital signals, exact values of the assumed voltage levels are not possible.

Noise Margins:



Generation: Any voltage higher than V_{OH} is HIGH. (V_{OH} = 2.4V*)
Any voltage lower than V_{OL} is LOW. (V_{OL} = 0.4V)

Detection: Any voltage higher than V_{IH} is HIGH. (V_{IH} = 2.0V)
Any voltage lower than V_{IL} is LOW. (V_{IL} = 0.8V)

* for TTL logic which is a particular kind of logic circuit family.

Logic

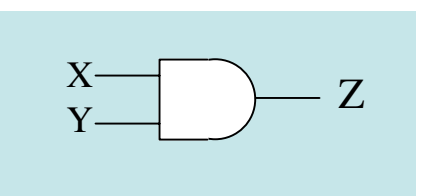
What can a digital circuit do?

The simplest task we can think of is a **combinational** type of logic decision.

For example, we can design a digital electronic circuit to make an *instant* decision based on some information. Here we emphasize "instant" in the decision making process. That means, the process has no time delay.

X = today's weather is good
Y = today is a holiday
decision Z = go to picnic

Suppose our rule is Z = X and Y
The circuit is a simple AND gate.



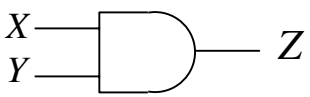
This kind of logic, involving no time delay, is **combinational logic**.

Truth tables

A easy way to represent a combinational logic result is to tabulate all possible inputs.

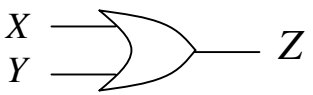
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of **AND**



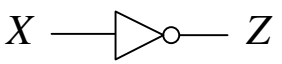
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Truth table of **OR**



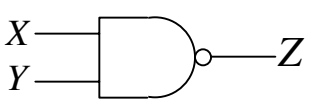
X	Z
0	1
1	0

Truth table of **NOT**



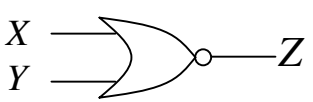
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of **NAND**



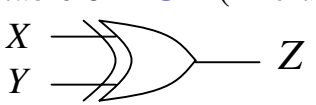
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Truth table of **NOR**



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Truth table of **XOR** (Exclusive OR)



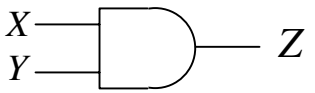
Boolean algebra

Logic can also be expressed in algebraic form.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of AND

$$Z = X.Y$$

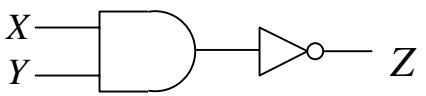


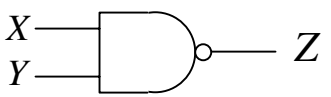
AND gate

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of NAND

$$Z = \overline{X.Y}$$





NAND gate

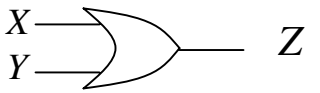
Boolean algebra

Logic can also be expressed in algebraic form.

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Truth table of OR

$$Z = X + Y$$

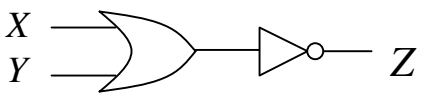


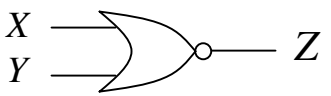
OR gate

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Truth table of NOR

$$Z = \overline{X + Y}$$





NOR gate

Finding expression from truth table

Once we have the truth table, we can find the output expression by adding up all min-terms.

Min-term corresponds to the product term that give a 1 in the output.
For example, here the min-terms are $\overline{X}.Y$, $X.\overline{Y}$, and $X.Y$

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

The expression for Z is

$$Z = \overline{X}.Y + X.\overline{Y} + X.Y$$

Question: This is an OR gate! Can it be simplified down to $Z = X + Y$? How can we do it?

More examples

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

The expression for Z is

$$Z = \overline{X}.\overline{Y} + \overline{X}.Y + X.\overline{Y}$$

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Truth table of NOR

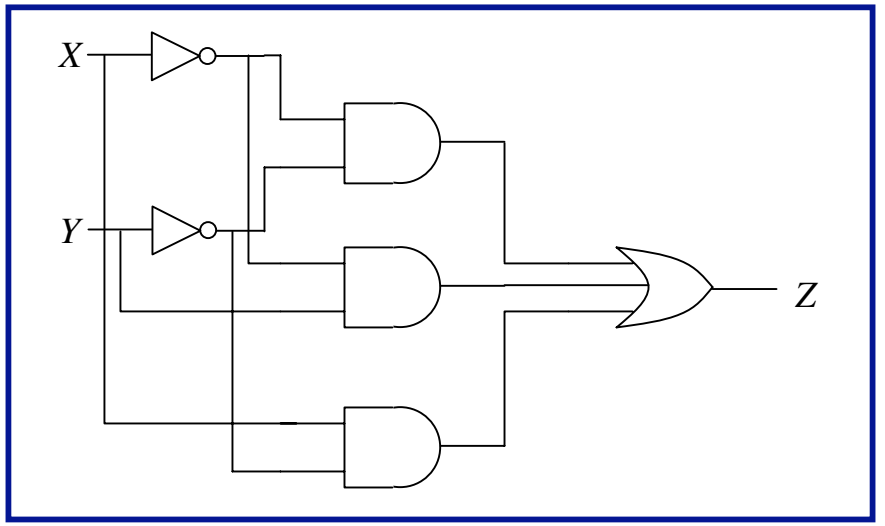
The expression for Z is

$$Z = \overline{X}.\overline{Y}$$

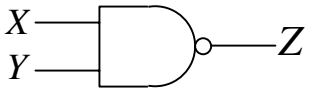
Question: These are NAND and NOR gates! Can they be simplified or converted back to the originally derived forms? How can we do it?

Circuit realization

$$Z = \overline{X}.\overline{Y} + \overline{X}.Y + X.\overline{Y}$$



Note: This is same as a NAND gate (see the truth table), and hence should be the same as



The question is HOW TO SIMPLIFY A MIN-TERM EXPRESSION!

Example: binary code to Gray code conversion

Binary			Gray code		
a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

What are the expressions for x, y and z? Then, can we design a circuit to converter binary code to Gray code?

$$x = a\bar{b}\,\bar{c} + a\bar{b}\,c + ab\bar{c} + abc$$

$$y = \bar{a}b\bar{c} + \bar{a}bc + a\bar{b}\,\bar{c} + a\bar{b}\,c$$

$$z = \bar{a}\bar{b}\,c + \bar{a}b\bar{c} + a\bar{b}\,c + ab\bar{c}$$

So, for each of x, y and z, we need a number of inverters, plus 4 AND gates and a multi-input OR gates.

Boolean algebra simplification

Basic Laws:

Commutative: $X+Y = Y+X$

$X.Y = Y.X$

Associative: $X+Y+Z = (X+Y)+Z = X+(Y+Z)$

$X.Y.Z = (X.Y).Z = X.(Y.Z)$

Distributive: $X.(Y+Z) = X.Y + X.Z$

De Morgan's: $\overline{X+Y} = \overline{X} \,.\, \overline{Y}$

$\overline{X.Y} = \overline{X} + \overline{Y}$

Example:

$$
\begin{aligned}
Z &= \overline{X}.\overline{Y} + \overline{X}.Y + X.\overline{Y} \\
&= \overline{X}.(\overline{Y} + Y) + X.\overline{Y} \\
&= \overline{X}.1 + X.\overline{Y} && \leftarrow\ Y + \overline{Y} = 1 \\
&= \overline{X} + X.\overline{Y} && \leftarrow\ \overline{X}.1 = \overline{X} \\
&= \overline{(X.(\overline{X} + Y))} && \leftarrow\ \text{De Morgan' law} \\
&= \overline{(X.\overline{X} + X.Y)} && \leftarrow\ \text{Distributive} \\
&= \overline{X.Y} && \leftarrow\ \overline{X}.X = 0
\end{aligned}
$$

That's right! It's a NAND gate!

Example:

$$
\begin{aligned}
Z &= \bar{X}.Y + X.\bar{Y} + X.Y \\
&= (\bar{X} + X).Y + X.\bar{Y} \\
&= Y + X.\bar{Y} \\
&= \overline{\bar{Y}.\overline{(X.\bar{Y})}} && \leftarrow \text{De Morgan' law} \\
&= \overline{\bar{Y}.(\bar{X} + Y)} && \leftarrow \text{De Morgan' law} \\
&= \overline{\bar{X}.\bar{Y} + 0} && \leftarrow \bar{Y}.Y = 0 \\
&= \overline{\bar{X}.\bar{Y}} \\
&= X + Y && \leftarrow \text{De Morgan' law}
\end{aligned}
$$

That's right! It's a OR gate!

Boolean algebra can be tedious.

Is there any easier method to simplify the min-term expression?

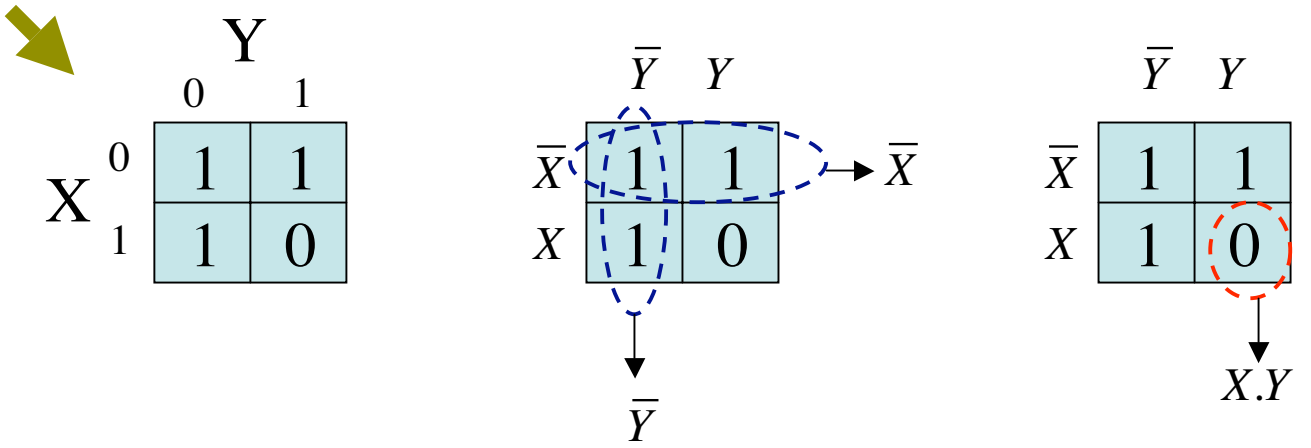
Karnaugh Maps

A very useful design tool for simplifying combinational logic.

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

The expression for Z is

$$Z = \overline{X}.\overline{Y} + \overline{X}.Y + X.\overline{Y}$$



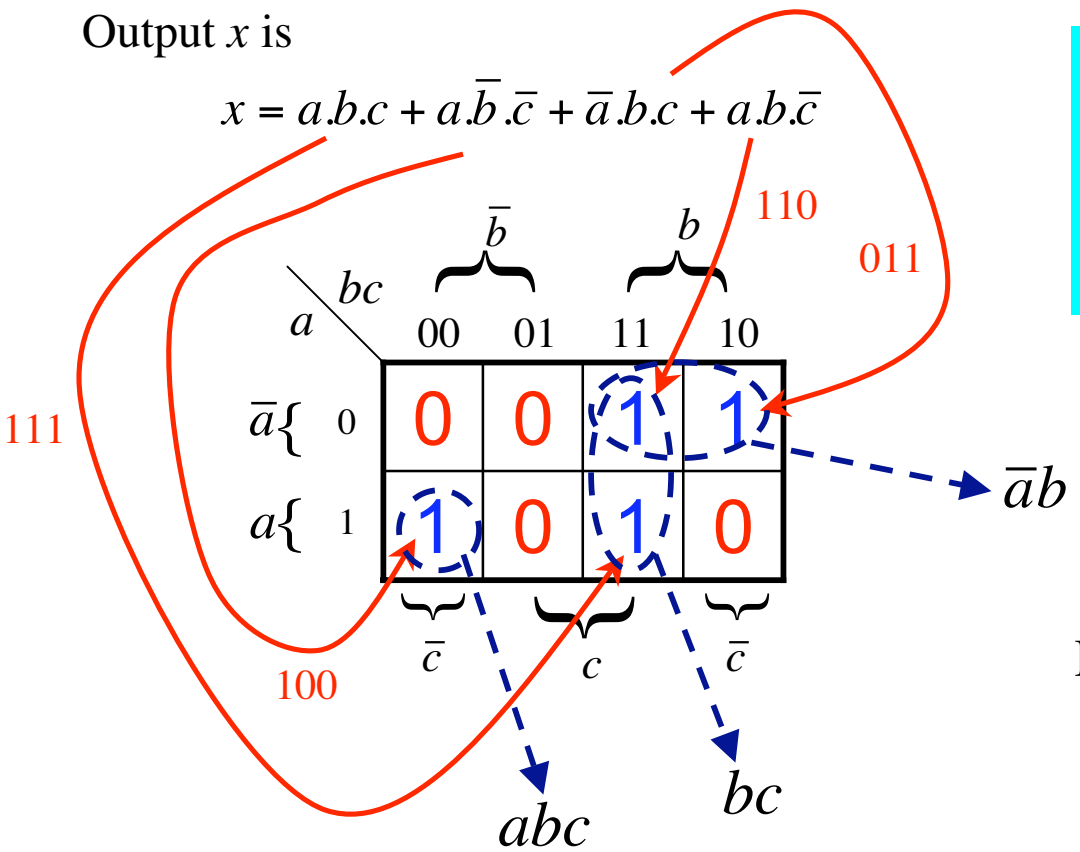
Therefore $Z = \overline{X} + \overline{Y}$ or $Z = \overline{X.Y}$

Karnaugh map (with 3 inputs)

Suppose we have three inputs a, b and c.

Output x is

$$x = a.b.c + a.\bar{b}.\bar{c} + \bar{a}.b.c + a.b.\bar{c}$$



Procedure:

1. Circle clusters of "1".
2. Determine the logic expressions for each cluster.
3. Add them up.

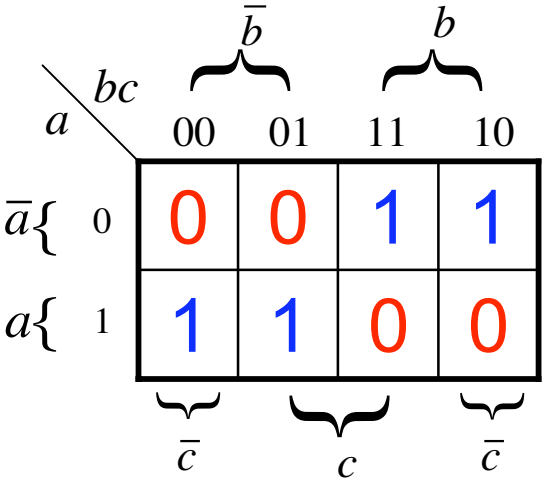
Hence, we get

$$x = a.b.c + b.c + \bar{a}.b$$

Example: Gray code

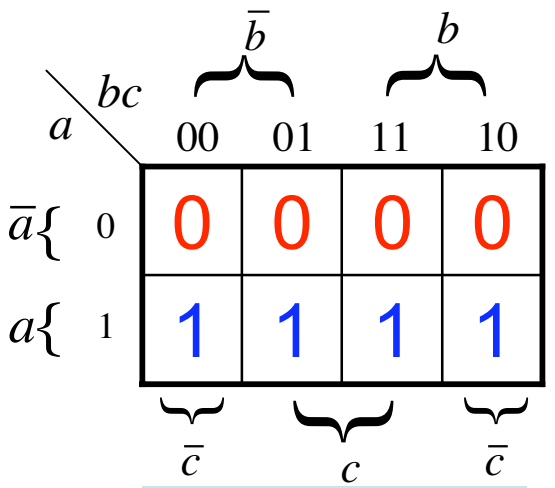
Binary			Gray code		
a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

y



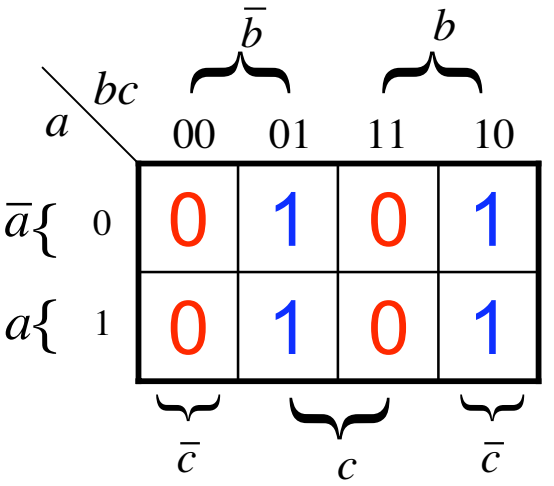
$$y = a\bar{b} + \bar{a}b$$

x



$$x = a$$

z



$$z = \bar{b}c + b\bar{c}$$

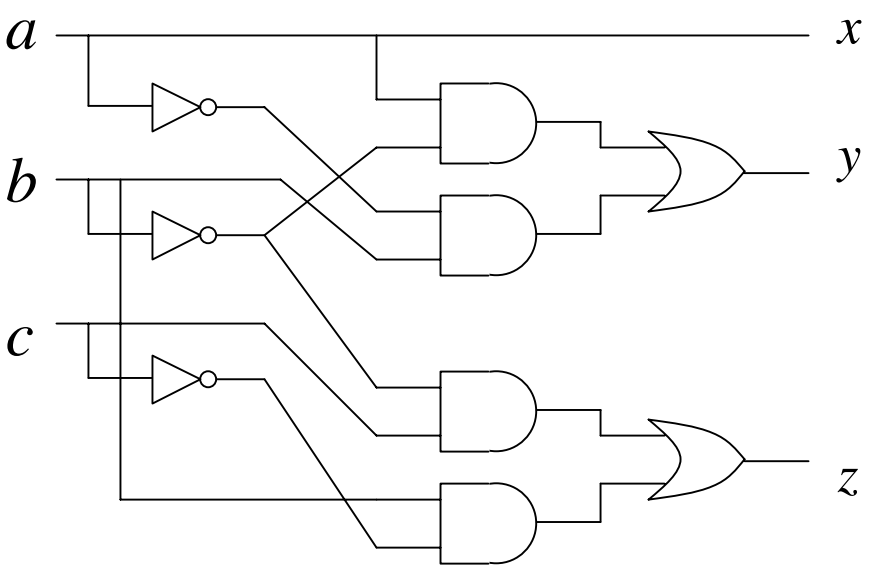
Example: Gray code

Binary			Gray code		
a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

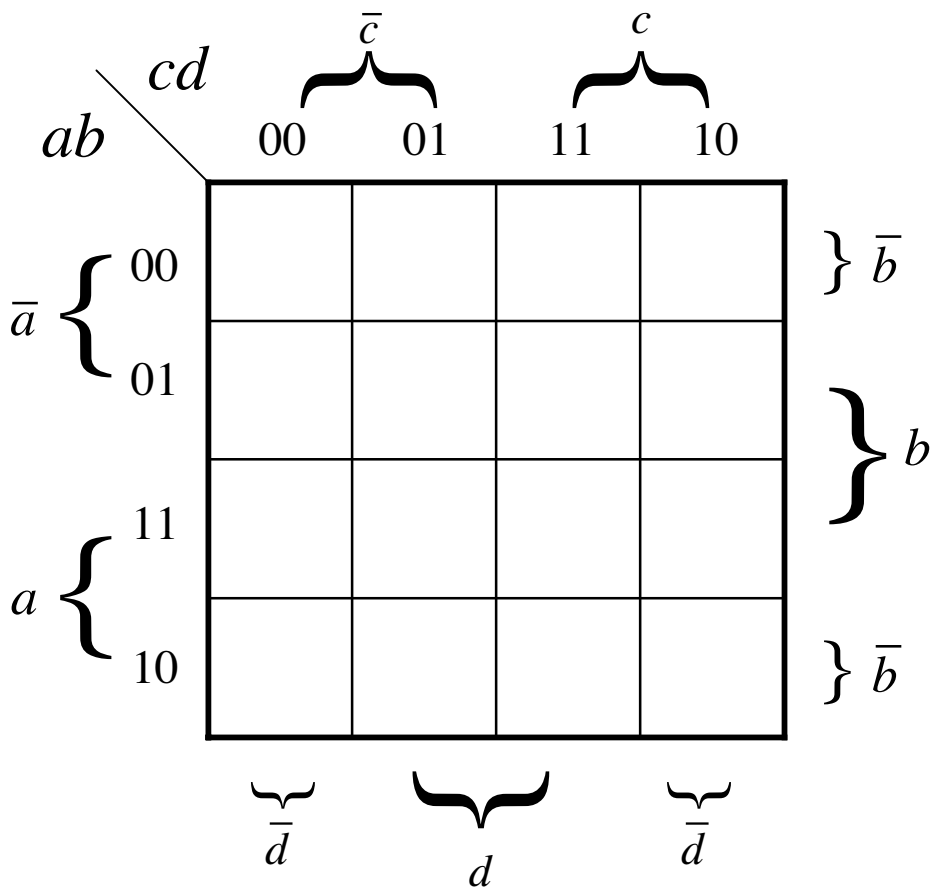
$$x = a$$

$$y = a\bar{b} + \bar{a}b$$

$$z = \bar{b}c + b\bar{c}$$



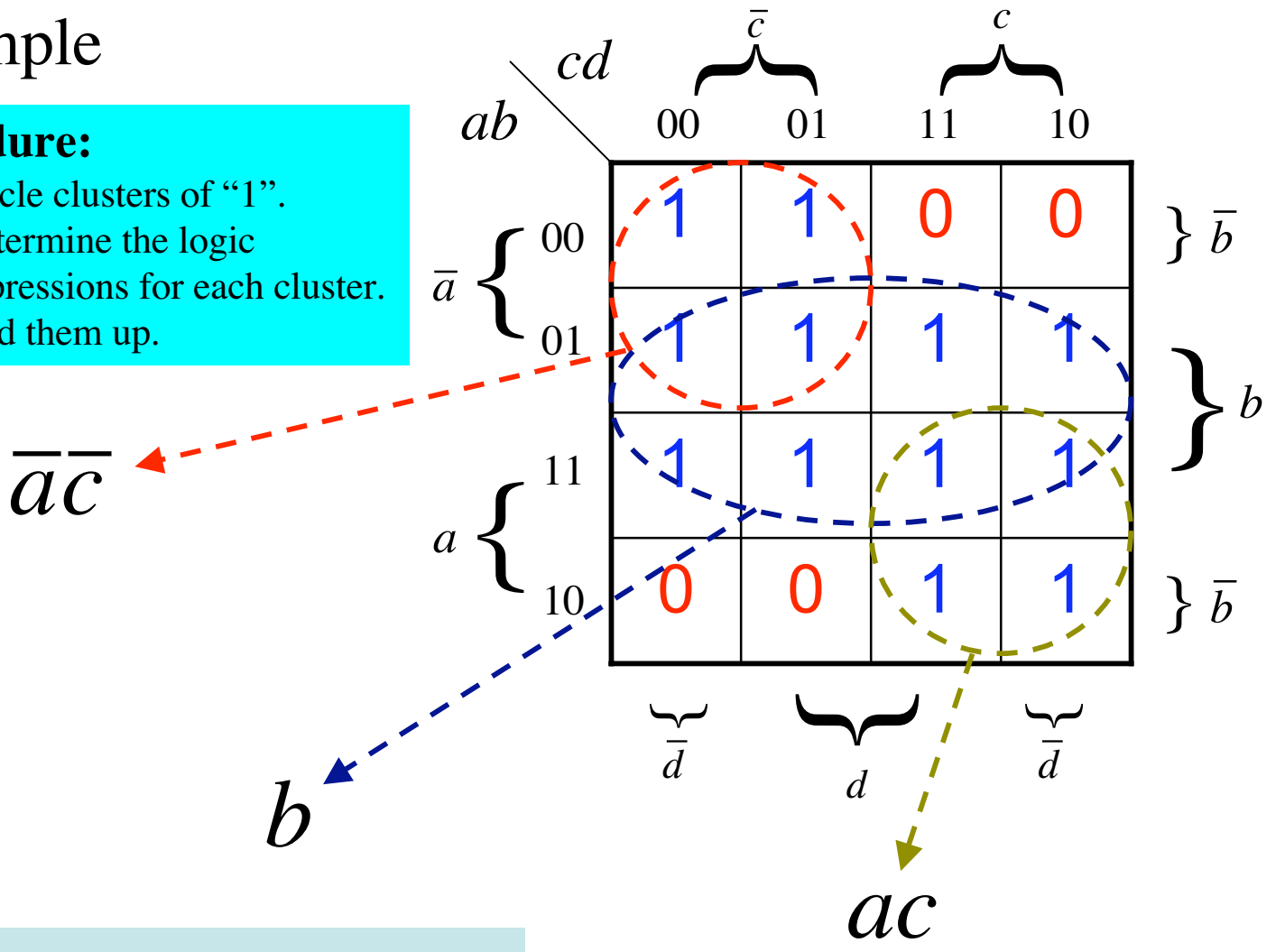
Karnaugh map (with 4 inputs)



Example

Procedure:

1. Circle clusters of "1".
2. Determine the logic expressions for each cluster.
3. Add them up.



$$x = \bar{a}\bar{c} + b + ac$$

Example

Derive the circuit for this combinational logic.

$$x = \overline{a}\,\overline{c} + b + ac$$