Computer Systems

Kasun Gunawardana

E-mail: kgg

University of Colombo School of Computing

Boolean Algebraic Expression Simplification

Boolean Algebra for Simplification

- Boolean Algebra is a mathematical branch.
- It has its own set of Axioms and laws.
- An expression can be manipulated using rules, laws and theorems.
- It can be used to simplify Boolean expressions.

Boolean Algebra - Postulates

- *Postulate (Axiom)* -
A thing suggested or assumed as true as the basis for mathematical reasoning.

0.0 = 0	**0+1 = 1**
0.1 = 0	**1+0 = 1**
1.0 = 0	**1+1 = 1**
1.1 = 1	$\overline{0} = 1$
0+0 = 0	$\overline{1} = 0$

Boolean Algebra – Laws

- Annulment Law
$A . 0 = 0$	$A + 1 = 1$
- Identity Law
$A + 0 = A$	$A . 1 = A$
- Idempotent Law
$A + A = A$	$A . A = A$
- Complement Law
$A.\overline{A} = 0$	$A + \overline{A} = 1$
- Commutative Law
$A . B = B . A$	$A + B = B + A$
- Double Negation Law
  - $\overline{\overline{A}} = A$

Boolean Algebra – Laws (Cont.)

- Distributive Law

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

- Identity Law

A + A = A	A . A = A

- Redundancy Law

A + A.B = A	A (A+B) = A

- Law

$\mathrm{A} + \overline{\mathrm{A}}.B = \mathrm{A} + \mathrm{B}$	$\mathrm{A}(\overline{\mathrm{A}} + B) = A.B$

- Law

$\mathrm{A}.\mathrm{B} + \overline{\mathrm{A}}.B = \mathrm{B}$	$(\mathrm{A} + \mathrm{B}).(\overline{\mathrm{A}} + B) = \mathrm{B}$

Boolean Algebra - Theorems

- De Morgan's Theorem
  - $\overline{(A.B)} = \bar{A} + \bar{B}$
  - $\overline{(A+B)} = \bar{A}.\bar{B}$

Simplification

- Simple expressions are always with less number of gates.
- A boolean function can be presented in SoM or PoM canonical forms.
- If the output column has more 1s than 0s then presenting the expression in PoM cannonical form would be cost effective.
- Otherwise the expression can be presented in SoM.
- However, there are ways to further simplify the expression.

Boolean Algebric Simplification

- Given expression can be further simplified using boolean algebric laws and theorems.
- Ex. $F = (x + \bar{y} + \bar{z}).(x + \bar{y}.z)$

$$= x.x + x.\bar{y}.z + x.\bar{y} + \bar{y}.\bar{y}.z + x.\bar{z} + \bar{y}.z.\bar{z}$$

$$= x(1 + \bar{y}.z + \bar{y} + \bar{z}) + \bar{y}.z$$

$$= x + \bar{y}.z$$

Exercise..!

- Simplify the following boolean expression

$$F(x,y,z) = x.\bar{y} + x.\bar{z} + y.\bar{z} + x.y.z + y.z$$

Answer

- $= \boldsymbol{x}.\overline{\boldsymbol{y}} + \boldsymbol{x}.\overline{\boldsymbol{z}} + \boldsymbol{y}.\overline{\boldsymbol{z}} + \boldsymbol{x}.\boldsymbol{y}.\boldsymbol{z} + \boldsymbol{y}.\boldsymbol{z}$
- $= x.\bar{y} + x.\bar{z} + x.y.z + y(\bar{z} + z)$
- $= x.\bar{y} + x.\bar{z} + x.y.z + y$
- $= x.\bar{y} + x.\bar{z} + y(x.y + 1)$
- $= x.\bar{y} + x.\bar{z} + y$
- $= (x.\bar{y} + y) + x.\bar{z}$
- $= (x + y) + x.\bar{z}$
- $= (x + x.\bar{z}) + y$
- $= x + y$

Exercise..!

- Simplify the following boolean expression using De Morgan's Theorem

$$F = \overline{(x.y + \bar{y}.z) + (x.z + \bar{x}.\bar{z})}$$

Answer

- $= \overline{(x.y + \bar{y}.z) + (x.z + \bar{x}.\bar{z})}$
- $= \overline{(x.y + \bar{y}.z)}.\overline{(x.z + \bar{x}.\bar{z})}$
- $= (\overline{x.y}.\overline{\bar{y}.z}).(\overline{x.z}.\overline{\bar{x}.\bar{z}})$
- $= \big((\bar{x} + \bar{y}).(y + \bar{z})\big).\big((\bar{x} + \bar{z}).(x + z)\big)$
- $= (\bar{x}.y + \bar{x}.\bar{z} + \bar{y}.y + \bar{y}.\bar{z}).(\bar{x}.x + \bar{x}.z + \bar{z}.x + \bar{z}.z)$
- $= (\bar{x}.y + \bar{x}.\bar{z} + \bar{y}.\bar{z}).(\bar{x}.z + \bar{z}.x)$
- $= \bar{x}.y.z + \bar{x}.y.\bar{z}.x + \bar{x}.\bar{z}.z + \bar{x}.\bar{z}.x + \bar{y}.\bar{z}.\bar{x}.z + \bar{y}.\bar{z}.x$
- $= \bar{x}.y.z + x.\bar{y}.\bar{z}$

Next…

Karnaugh Map

Computer Systems

Kasun Gunawardana

E-mail: kgg

University of Colombo School of Computing

Karnaugh Map

Karnaugh Map (K-Map)

- K-Map is a gate level minimization technique.
- A truth table is represented in an alternative diagram which is made up of cells.
- Each cell represents a Minterm in the truth table.
- A set of well defined rules to be applied for simplification.
- If rules are applied properly, it gurantees for the generation of simplest expression.

Karnaugh Map - Representation

- If a boolean funcation has n variables, then its truth table will have 2^n number of rows.
- Similarly its K-Map will have 2^n number of cells.
- Ex. 3 variable K-Map

Each cell is for a $minterm$.

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$				
$z = 1$				

Karnaugh Map - Characteristics

- Only one variable can be changed when considering two adjacent columns or rows.

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$				
$z = 1$				

K-Map: Simplification

- For the simplification we group cell content and find unchanged variables within groups.

$$F = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.\bar{z} + x.y.\bar{z}$$

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$				
$z = 1$				

K-Map: Simplification (Cont.)

- For the simplification we group cell content and find unchanged variables within groups.

$$F = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.\bar{z} + x.y.\bar{z}$$

Mapping

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$	0	0	1	1
$z = 1$	1	1	0	0

K-Map: Simplification (Cont.)

- For the simplification we group cell content and find unchanged variables within groups.

$$F = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.\bar{z} + x.y.\bar{z}$$

Grouping

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$	0	0	1	1
$z = 1$	1	1	0	0

K-Map: Simplification (Cont.)

- For the simplification we group cell content and find unchanged variables within groups.

$$F = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.\bar{z} + x.y.\bar{z}$$

Deriving

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$	0	0	1	1
$z = 1$	1	1	0	0

$x.\bar{z}$

$\bar{x}.z$

K-Map: Simplification (Cont.)

- For the simplification we group cell content and find unchanged variables within groups.

$$F = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.\bar{z} + x.y.\bar{z} = \boldsymbol{x.\bar{z} + \bar{x}.z}$$

Deriving

	$x = 0$ $y = 0$	$x = 0$ $y = 1$	$x = 1$ $y = 1$	$x = 1$ $y = 0$
$z = 0$	0	0	1	1
$z = 1$	1	1	0	0

$x.\bar{z}$

$\bar{x}.z$

SoP and PoS Expressions

- If the simplified expression is needed in the form of Sum of Products, then 1s are grouped.
- If the simplified expression is needed in the form of Product of Sums, then 0s are grouped.
  - Similar to deriving the expression from the truth table.

Next...

K-Map Grouping Rules

Computer Systems

Kasun Gunawardana

E-mail: kgg

University of Colombo School of Computing

K-Map Grouping Rules

K-Map Rules [1]

- Groups should contain only one type
  - Groups should contain only 1 if the expression is required in the form of Sum of Products.
  - Groups should contain only 0s if the expression is required in the form of Product of Sums.

y \ x	0	1
0	1	0
1	1	0

Correct

y \ x	0	1
0	1	0
1	1	0

Incorrect

K-Map Rules [2]

- Groups should be formed vertical or horizontal.
  - Expansion should be done in vertically or horizontally.
- Diagonal groups are not allowed.

y \ x	0	1
0	1	0
1	1	0

Correct

y \ x	0	1
0	0	1
1	1	0

Incorrect

K-Map Rules [3]

- Groups can only cover 2^n number of cells where $n \geq 0$.

cd \ ab	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	1	0
10	0	0	0	0

Correct

cd \ ab	00	01	11	10
00	1	1	1	0
01	0	0	0	0
11	1	1	1	1
10	1	0	0	0

Incorrect

K-Map Rules [4]

- Each group should be in its maximum size.

cd \ ab	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	1	1	0	0
10	1	1	0	0

Correct

cd \ ab	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	1	1	0	
10	1	1	0	0

Incorrect

K-Map Rules [4.1]

- Groups must be overlapped if it maximizes the groups' sizes.

cd \ ab	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	0	0	0
10	0	0	0	0

Correct

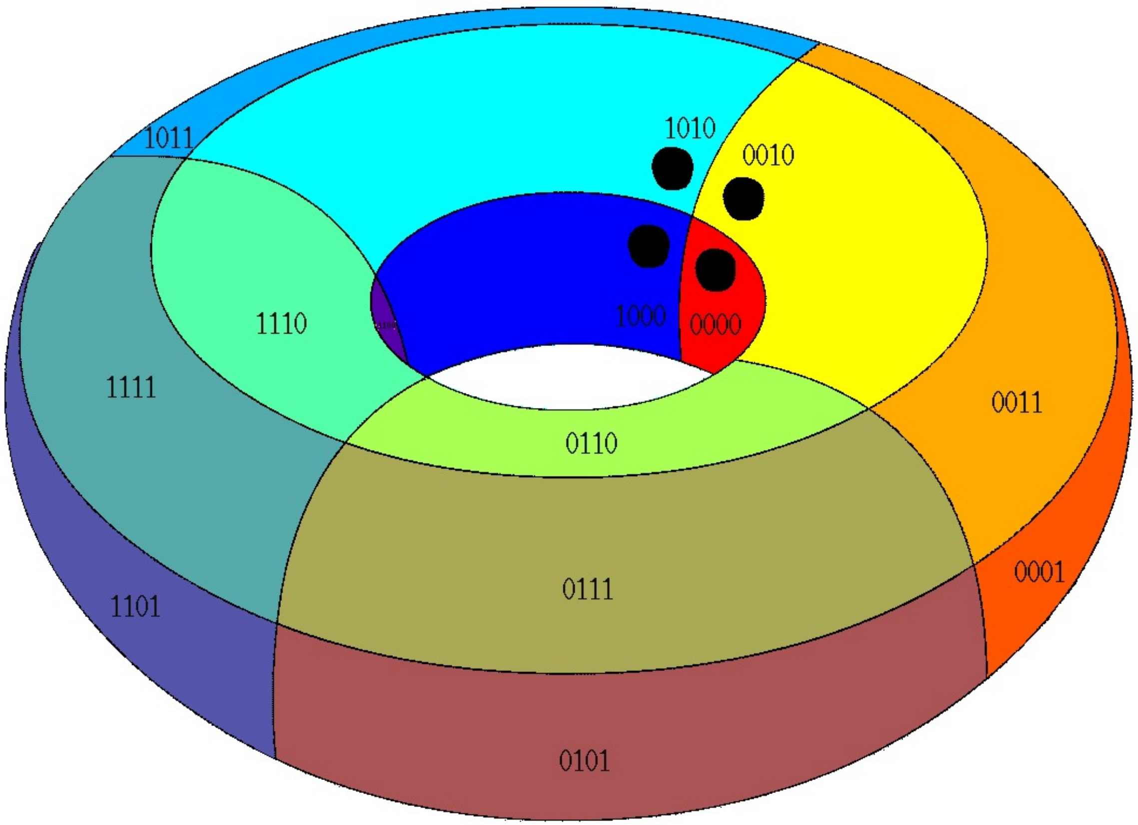
cd \ ab	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	0	0	0
10	0	0	0	0

Incorrect

K-Map Rules [4.2]

- Group must be formed or overlapped by considering the uppermost row and the lowermost row are adjacent.
- Similarly it is assumed that the leftmost column and the rightmost column are adjacent.
- Groups may formed around the table.

K-Map Rules [4.2] (Cont.)



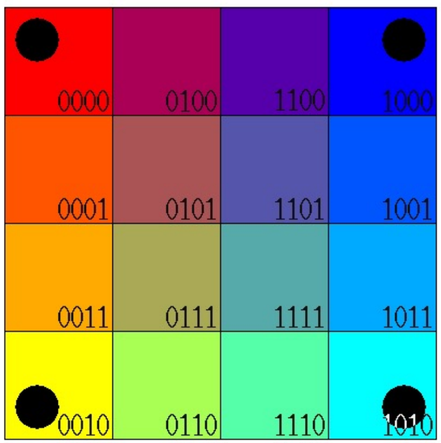


Image source - https://en.wikipedia.org/wiki/Karnaugh_map

K-Map Rules [4.2] (Cont.)

- Wrapping the groups around the table.

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Correct

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Incorrect

K-Map Rules [4.3]

- There should be as few groups as possible while maximizing the sizes of groups.

ab / cd	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	1	1	0	0
10	1	1	0	0

Correct

ab / cd	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	1	1	0	
10	1	1	0	0

Incorrect

Next…

Exercises on K-Map

Computer Systems

Kasun Gunawardana

E-mail: kgg

University of Colombo School of Computing

Exercises on K-Map

Exercise..

- Draw the K-Map for the following Boolean function and derive the simplified expression in Standard Sum of Products form by applying grouping rules.

- $F = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.c.\bar{d} + a.\bar{b}.\bar{c}.\bar{d} + a.\bar{b}.c.\bar{d}$

Answers (K-Map)

- K-Map for the function F,
- $F = \bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.c.\bar{d} + a.\bar{b}.\bar{c}.\bar{d} + a.\bar{b}.c.\bar{d}$

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Grouping)

- Group 1 (Blue)
- Unchanged: $\bar{a}\ and\ \bar{d}$
- Simplified term: $\bar{a}.\bar{d}$

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Grouping)

- Group 2 (Green)
- Unchanged: $\bar{b}\ and\ \bar{d}$
- Simplified term: $\bar{b}.\bar{d}$

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Grouping)

- Simplified expression: $\bar{a}.\bar{d} + \bar{b}.\bar{d}$
- Orignial expression: $\bar{a}.\bar{b}.\bar{c}.\bar{d} + \bar{a}.\bar{b}.c.\bar{d} + \bar{a}.b.\bar{c}.\bar{d} + \bar{a}.b.c.\bar{d} + a.\bar{b}.\bar{c}.\bar{d} + a.\bar{b}.c.\bar{d}$

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Standard PoS)

- What if we want to derive the simplified expression in Standard Product of Sums form?

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Standard PoS)

- If we want to derive the simplified expression in Standard Product of Sums form,
  - Group 0s by following grouping rules

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Answers (Standard PoS)

- Simplified expression would be F',
  - $F' = (\bar{a} + \bar{b}).(\bar{d})$

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Exercise..

- Derive the simplified Boolean algebric expression for the follownig K-Map in
  - Standard Sum of Products form
  - Standard Prodcut of Sums form

cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Answers

- Derive the simplified Boolean algebric expression for the follownig K-Map in
  - Standard Sum of Products form

cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Answers

- Derive the simplified Boolean algebric expression for the follownig K-Map in
  - Standard Prodcut of Sums form

cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Don’t Care Condition - x

- There are functions that output is not defined for its input patterns.
- These are called,
  - Incompletely sepcified functions
  - Incompletely Defined Functions
- These undefined/ unspecified input patterns are called Don’t Care Conditions.
- We can exploit these don’t care conditions in K-Map simplifications.

Don’t Care - Example

- Let’s Assume function F is defined as,

$$F(a,b,c,d) = \sum(1,3,5,7,9)$$

- F’s don’t care conditions are defined as,

$$G(a,b,c,d) = \sum(11,13)$$

Don’t Care – Example (Mapping)

$$F(a,b,c,d) = \sum(1,3,5,7,9)$$

$$G(a,b,c,d) = \sum(11,13)$$

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	1	X	1
11	1	1	0	X
10	0	0	0	0

Don’t Care - Example

$$F(a,b,c,d) = \sum(1,3,5,7,9)$$

$$G(a,b,c,d) = \sum(11,13)$$

cd \ ab	00	01	11	10
00	0	0	0	0
01	**1**	**1**	**X**	**1**
11	**1**	**1**	0	**X**
10	0	0	0	0

Don’t Care – Suboptimal Grouping

$$F(a, b, c, d) = \sum(1, 3, 5, 7, 9)$$

$$G(a, b, c, d) = \sum(11, 13)$$

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	1	X	1
11	1	1	0	X
10	0	0	0	0

Don’t Care – Optimal Grouping #1

$$F(a,b,c,d) = \sum(1,3,5,7,9)$$

$$G(a,b,c,d) = \sum(11,13)$$

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	1	X	1
11	1	1	0	X
10	0	0	0	0

$$\bar{c}d + \bar{a}d$$

Don’t Care – Optimal Grouping #2

$$F(a,b,c,d) = \sum(1,3,5,7,9)$$

$$G(a,b,c,d) = \sum(11,13)$$

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	1	X	1
11	1	1	0	X
10	0	0	0	0

$$\bar{b}d + \bar{a}d$$

Next...

How K-Map works [OPTIONAL]

Computer Systems

Kasun Gunawardana

E-mail: kgg

University of Colombo School of Computing

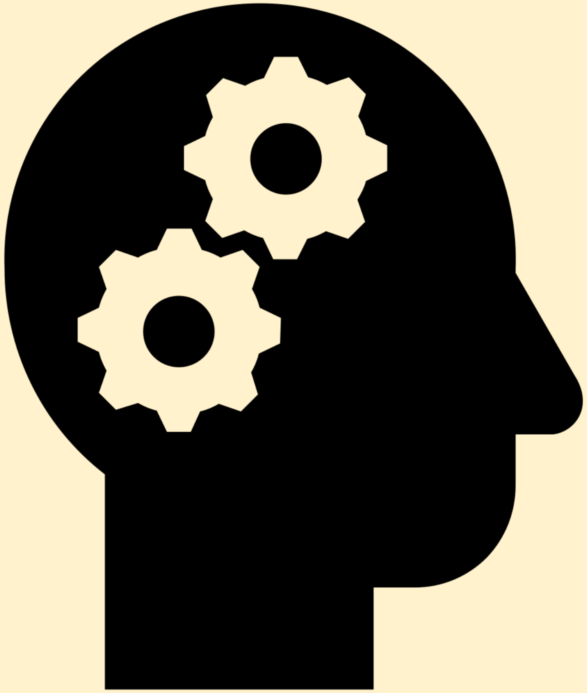
How K-Map works

[OPTIONAL]

Digging It Deep: K-Map

How does it work?

How does K-Map produce the simplest expression?



Questions to be Asked…

- Why do we group?
- Why groups are in 2^n?
- Why groups are formed vertical or horizontal?
- Why do we flip only one variable between two adjacent rows or columns?

Explanation

- K-Map ensures that a group recognizes a set of terms that has common variable states.

c \ ab	00	01	11	10
0	0	1	0	0
1	0	1	0	0

- Grouped terms: $\bar{a}.b.\bar{c} + \bar{a}.b.c$
- Here, both terms have a common state ($\bar{a}.b$) for variables a and b.
- The particular state is common for all possible states of the other variable.

Explanation (Cont.)

- When we have a common state for a set of variables over all the possible states of another variable, then the latter variable is irrelevant to the expression.

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

- By having a group of 2 cells we can recognize a single irrelevant variable (2 cells cover all possible states for a single variable).
- If we can group 4 cells then we can recognize two irrelevant variables (4 cells cover all possible states for two variables).
- This is why we form groups with 2^n number of cells.
- We are trying to eliminate variables as much as possible to make the expression simple.

Explanation (Cont.)

- Forming groups vertically or horizontally, ensures covering all the possible states for a given set of variables.
- Green Group Terms:

$$\overline{\boldsymbol{a}}.\boldsymbol{b}.\overline{\boldsymbol{c}}+\overline{\boldsymbol{a}}.\boldsymbol{b}.\boldsymbol{c}$$

Common state $(\overline{\boldsymbol{a}}.\boldsymbol{b})$ for all possible states of variable $\boldsymbol{c}$.

c \ ab	00	01	11	10
0	0	1	0	1
1	0	1	1	0

- Red Group: $\boldsymbol{a}.\overline{\boldsymbol{b}}.\overline{\boldsymbol{c}}+\boldsymbol{a}.\boldsymbol{b}.\boldsymbol{c}$

  Common state $(\boldsymbol{a})$ doesn't appear with all possible states of variable $\boldsymbol{b}$ and $\boldsymbol{c}$.
- Restriction on flipping 1 variable at a time also do the same.

Next…

Single Gate Type Circuits

Computer Systems

Kasun Gunawardana

E-mail: kgg

UCSC

University of Colombo School of Computing

Single Gate Type Circuits

Single Gate Type Circuits

- Circuits are preferred to be constructed in a single gate type.
- Manufacturing ICs with different types of gates are expensive.
- There can be unused gates in ICs if those ICs are manufactured with different types of gates.

Functional Completeness

- AND, OR and NOT operations are the primary functionalities.
- Therefore, any set of gates that can demonstrate all three functionalities is called *Functionally Complete Set*.
- Any Boolean expression can be constructed using a functionally complete set of gates.
- By its defintion, {AND,OR,NOT} is a functionally complete set.

Functional Completeness

A functionally complete set of logical connectives or Boolean operators is one which can be used to express all possible truth tables by combining members of the set into a Boolean expression.

~ Wikipedia

{AND, NOT}

- Is the set {AND, NOT} a functionally complete set?
- The misssig primary functionality is OR.
- Then {AND, NOT} should be able to demonstrate OR.

$$A + B = \overline{\overline{A + B}}$$
$$A + B = \overline{\bar{A} . \bar{B}}$$

- OR operation can be performed by AND and NOT.
- Therefore, {AND, NOT} is functionally complete.

{OR, NOT}

- Similarly, {OR, NOT} is a functionally complete set.
- Here, missing operation is AND
- Then {OR, NOT} should be able to demonstrate AND.

$$A.B = \overline{\overline{A.B}}$$
$$A.B = \overline{\bar{A} + \bar{B}}$$

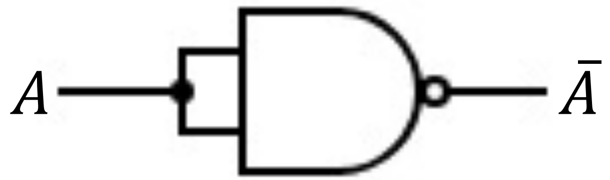
- AND operation can be performed by OR and NOT.
- Therefore, {OR, NOT} is functionally complete.

{NAND} , {NOR}

- NAND and NOR gates are identified as individually functionally complete.
- Thus, digital circuits can be implemented by single gate type (NAND or NOR).
- ICs come with several gates from single gate type.

NOT from NAND

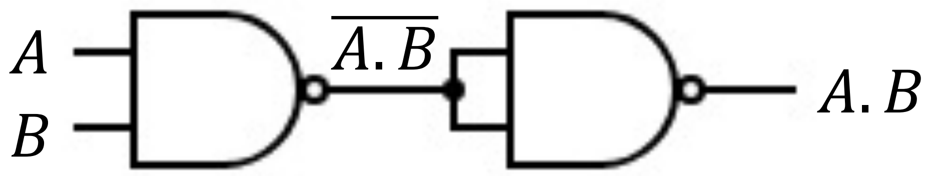
- NAND gate can be used as an inverter.



$$\overline{A.A} = \bar{A}$$

AND from NAND

- A set of NAND gates can be used as an AND gate.

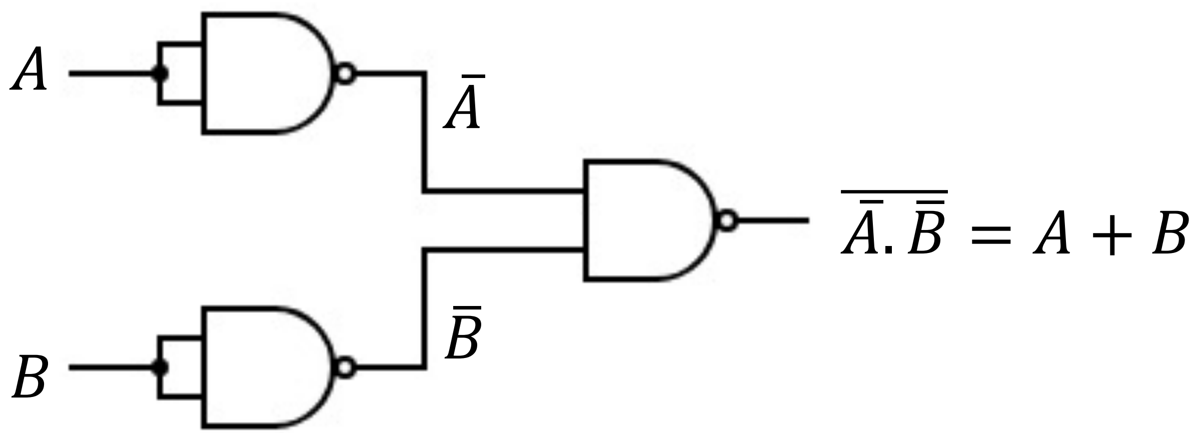


OR from NAND

- A set of NAND gates can be used as an OR gate.

$$A + B = \overline{\overline{A + B}}$$

$$A + B = \overline{\bar{A}.\bar{B}}$$



NAND as an Universal Gate

- NAND can implement XOR, NOR, XNOR.
- NAND can implement any Boolean function without any other gate type (Universal).
- NOR is also a Universal Gate.

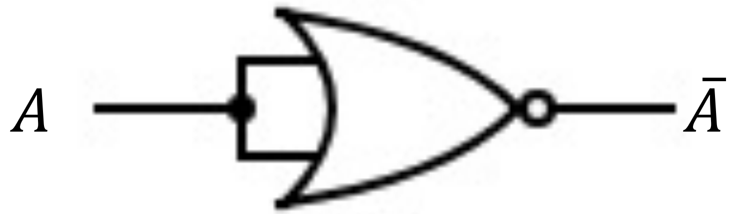
Exercise..

How can we use NOR gate,

- to implement NOT gate?
- to implement AND gate?
- to implement OR gate?

NOT from NOR

- NOR gate can be used as an inverter.



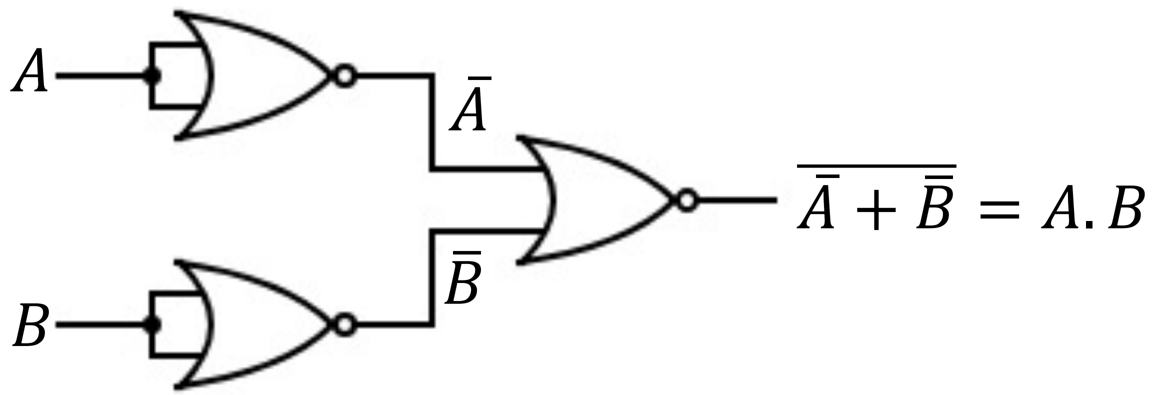
$$\overline{A + A} = \bar{A}$$

AND from NOR

- A set of NOR gates can be used to perform AND operation.

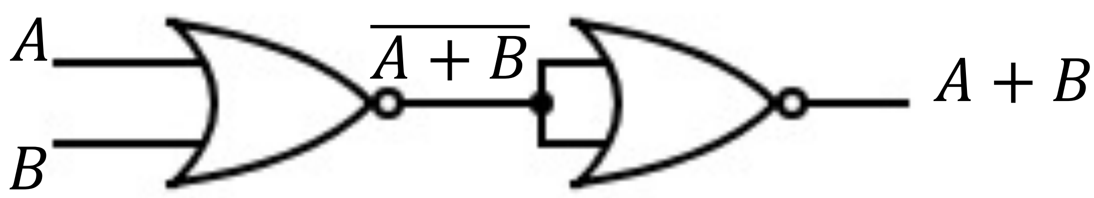
$$A.B = \overline{\overline{A.B}}$$

$$A.B = \overline{\bar{A} + \bar{B}}$$



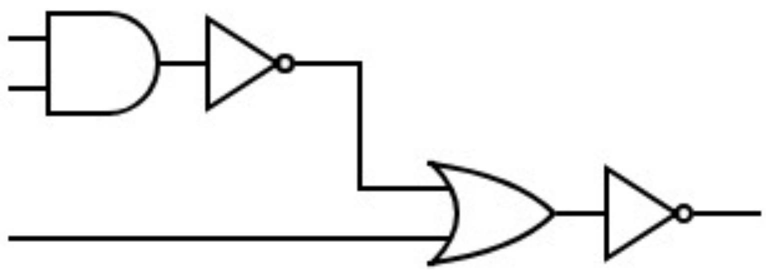
OR from NOR

- A set of NOR gates can be used to perform OR operation.



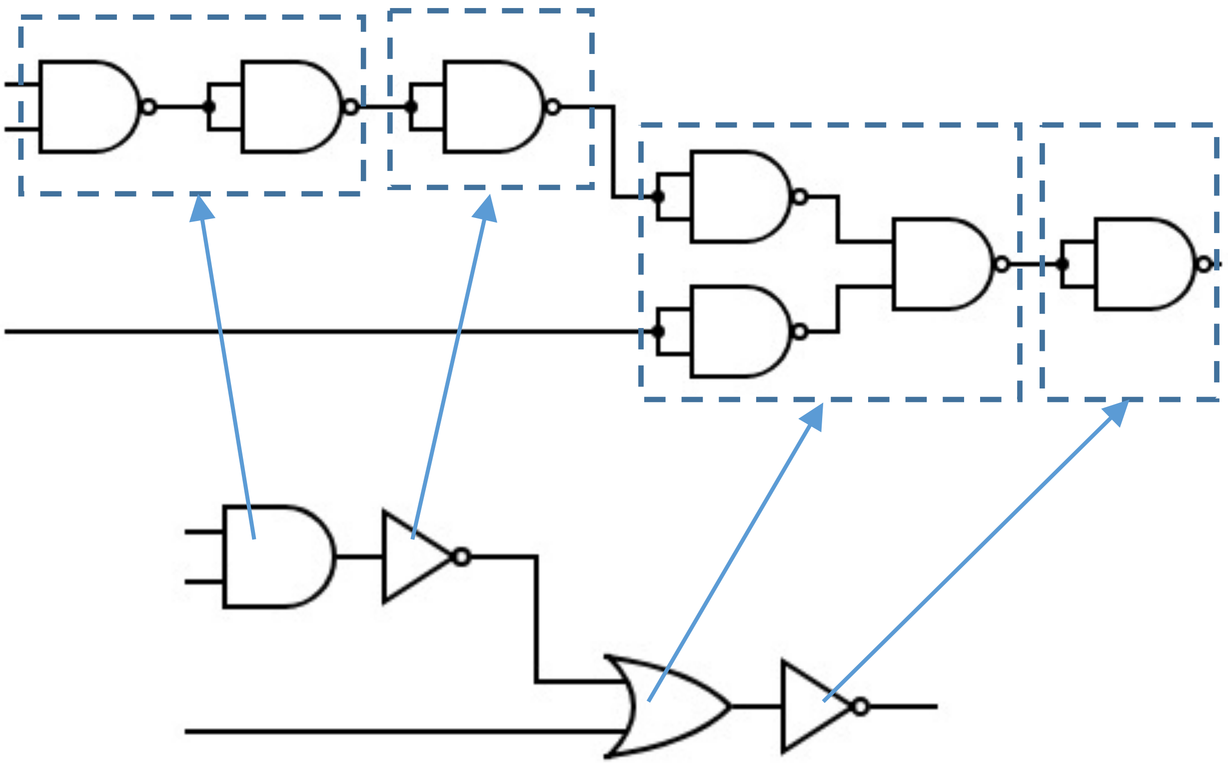
Exercise..

- Following circuit shows the gate arrangement for the boolean expression $F = \overline{\overline{x.y} + z}$
- Derive a single gate type circuit for it by substituting NAND gates.



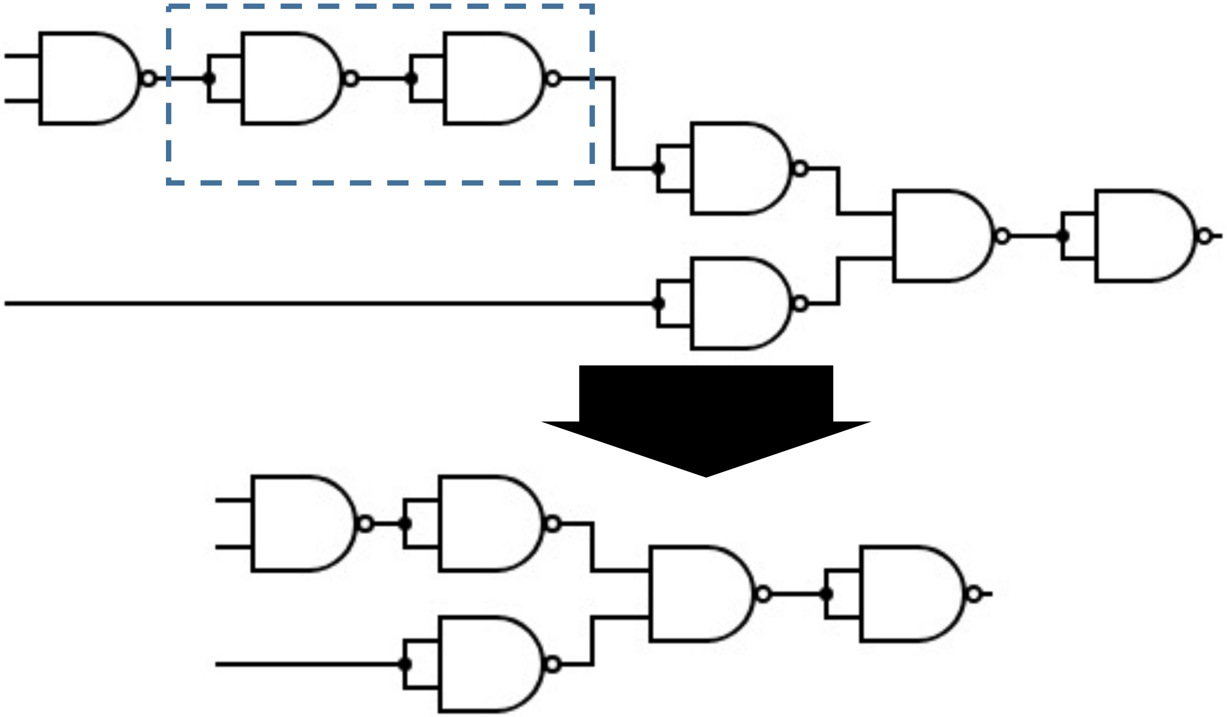
Answer

- Direct Substitution would construct a circuit as follows:



Answer

- Double negations can be removed and the circuit can be simplified further:



Thank You..!