FIT1047 - Week 2 hour 1

Introduction to computer systems, networks and security

MONASH University

Reference: https://www.alexandriarepository.org/syllabus/introduction-to-computer-systems-networks-and-security/
Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett

Topics for week 2

- Error detection
- Boolean algebra
- Logical circuits

Boolean logic and Boolean algebra

Boolean logic is a rather simple possible (but useful) logic.
Basic concepts:

- TRUE, FALSE
- AND, OR
- NOT

Usually, **TRUE** is represented by **1** and **FALSE** is represented by **0**.

TRUE and **FALSE** are values for statements.
Note that not all statements qualify:

- Today, the temperature is over 15 degrees Celsius.
- Today, the weather is good.
- Haggis tastes great.

Do not confuse binary and Boolean:

	Binary	Boolean
0	Zero	FALSE
1	One	TRUE

Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have **one** of **two** values
  - In formal logic, these values are **True/False**
  - In digital systems, these values are **on/off, 1/0 or high/low**
- Boolean expressions are created by performing operations on Boolean variables
  - Common Boolean operators include **AND, OR, NOT**

Truth Tables: **AND & OR**

A Boolean operator can be completely described using a *Truth Table*

- Truth tables for the Boolean operators AND and OR

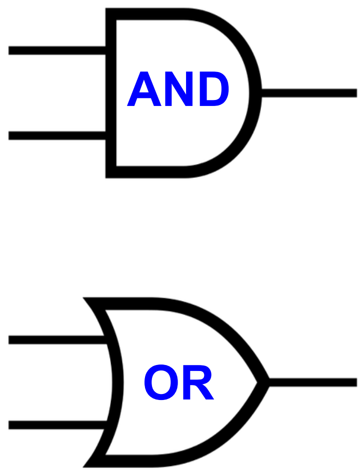
X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Examples for gates



- The **AND** operator can also be represented as a period (a full stop) or space, so **X.Y** or **XY**
- The **OR** operator can also be represented as a + represented as **X+Y**

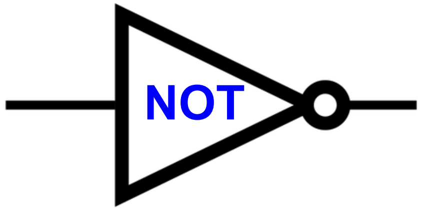
Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Truth Tables: **NOT**

- Truth table for the Boolean NOT operator

NOT x

x	$\overline{x}$
0	1
1	0



- The NOT operation is often designated by an **overbar**, a **prime mark**, an “**elbow**” or a “**squiggle**”, e.g.,

$\overline{A}$ A’   ¬A   ~A

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Boolean Functions

A Boolean function has:

- At least **one** Boolean variable (x, y,..)
- At least **one** Boolean operator (and, or, not,..)
- At least **one** input from the set {0,1} or {T,F}

➔ It produces an **output** that is also a member of the set {0,1} or {T,F}

Boolean Functions and Truth Tables

- Truth table for the Boolean function

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

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

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

To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Rules of Precedence

- The **NOT** operator has **highest priority**, followed by **AND** & then **OR**
  - This is how we chose the (shaded) function subparts in our table

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

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Boolean Algebra

- Digital computers contain circuits that implement Boolean functions
  - ✓ *The simpler we can make a Boolean function, the smaller the circuit that will result*
  - ✓ *Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits*
- With this in mind
  - We always want to reduce our Boolean functions to their simplest form
  - There are several Boolean identities that help us to do this

Laws of Boolean Algebra

- Identity Law
- Null Law (or Dominance Law)
- Idempotent Law
- Complement Law
- Commutative Law
- Associative Law
- Distributive Law
- Absorption Law
- DeMorgans Law
- Double Complement Law

Identity Law

AND Form	OR Form
$1A = A$	$0+A = A$

Null Law (Dominance Law)

AND Form	OR Form
$0A = 0$	$1+A = 1$

Idempotent Law

AND Form	OR Form
$AA = A$	$A+A = A$

Complement Law

AND Form	OR Form
$A\overline{A} = 0$	$A+\overline{A} = 1$

Commutative Law

AND Form	OR Form
$AB = BA$	$A+B = B+A$

Laws of Boolean Algebra

- Identity Law
- Null Law (or Dominance Law)
- Idempotent Law
- Complement Law
- Commutative Law
- Associative Law
- Distributive Law
- Absorption Law
- DeMorgans Law
- Double Complement Law

Associative Law

AND Form	OR Form
(AB)C = A(BC)	A+(B+C) = (A+B)+C

Distributive Law

AND Form	OR Form
A+(BC) = (A+B)(A+C)	A(B+C) = AB+AC

Absorption Law

AND Form	OR Form
A(A+B) = A	A+AB = A

DeMorgans Law

AND Form	OR Form
$\overline{AB} = \overline{A}+\overline{B}$	$\overline{(A+B)} = \overline{A} \times \overline{B}$

Double Complement Law

$\overline{\overline{A}} = A$

Laws of Boolean Algebra

- Identity Law
- Null Law (or Dominance Law)
- Idempotent Law
- Complement Law
- Commutative Law
- Associative Law
- Distributive Law
- Absorption Law
- DeMorgans Law
- Double Complement Law

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\overline{x} = 0$	$x + \overline{x} = 1$

(Inverse or Complement Law)

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \overline{x} + \overline{y}$	$\overline{(x+y)} = \overline{x}\,\overline{y}$
Double Complement Law	$\overline{(\overline{x})} = x$	

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Simplifying a Function

Apply Boolean algebra identities to simplify the following Boolean function:

$$(\overline{AB})A + A\bar{B}$$

SOLUTION:

de Morgan	$(\bar{A} + \bar{B})A + A\bar{B}$
Commutative	$A(\bar{A} + \bar{B}) + A\bar{B}$
Distributive	$A\bar{A} + A\bar{B} + A\bar{B}$
Inverse	$0 + A\bar{B} + A\bar{B}$
Identity	$A\bar{B} + A\bar{B}$
Idempotent	$A\bar{B}$

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

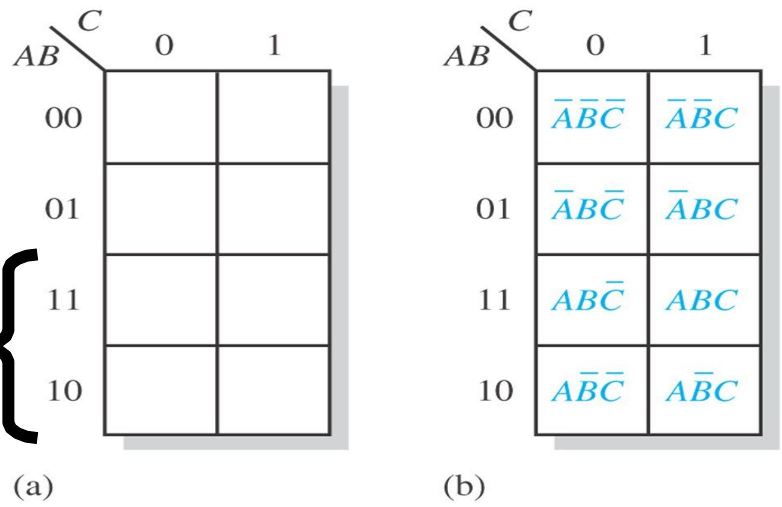
Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

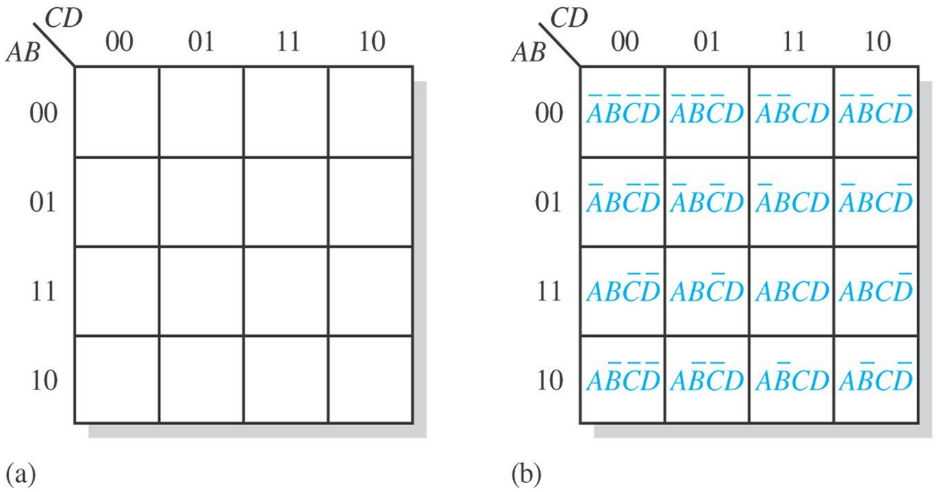
Simplifying using Karnaugh-Maps (K-map)

- ✓ Provides a method for simplifying Boolean expressions.
- ✓ It will produce the simplest Sums-Of-Products (SOP) and Product-Of-Sums (POS) expressions.
- ✓ Works best for less than 6 variables.
- ✓ Similar to a truth table ➔ it maps all possibilities.
- ✓ A Karnaugh map is an array of cells arranged in a special manner.
- ✓ The number of cells is $\mathbf{2^n}$ where **n** = number of variables

A 3-Variable Karnaugh Map:



A 4-Variable Karnaugh Map:



- ✓ **<u>Note:</u>** The order of these values they are reverse of the usual order.
- ✓ They are arranged this way so that only one variable changes at a time.

Simplifying using Karnaugh-Maps (K-map)

One of the most common forms of simplification in Boolean algebra is the following:

$A\overline{B} + AB = A(\overline{B} + B) = A$

The same with three variables:

$A\overline{B}C + ABC = AC$

Both functions are independent from the value of B.

Karnaugh-maps provide an easy graphical way to find minimal AND terms that are then combined with OR to get the complete function.

Truth table for A AND B

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Karnaugh-map for A AND B

		B	
		0	1
A	0	0	0
	1	0	1 =A.B

Simplifying using Karnaugh-Maps (K-map)

Let's look at a Karnaugh-map with 3 variables:

$\overline{B}A\overline{C} + AB\overline{C} + ABC + \overline{A}BC + \overline{A}B\overline{C}$

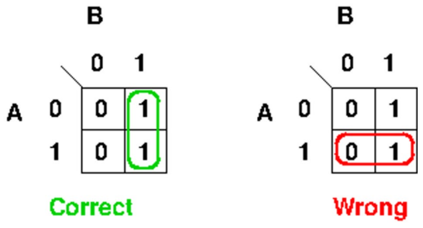
		BC			
		00	01	11	10
A	0	0	0	1	1
	1	1	0	1	1

- Find groups of 1s.
- The group of **four** 1s represents **B=1**. (Note: A=0/1, B=1, C=0/1)
- The wrapped group represents **A=1 AND C=0**. (Note: A=1, While B = 0/1, C=0)

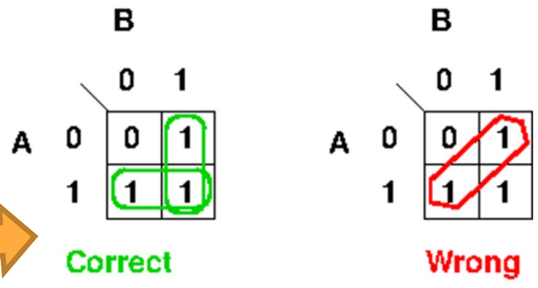
Simplified version: $B + A\overline{C}$

7 rules for working with Karnaugh-Maps (K-map)

☞ Rule 1: No group can contain a zero.

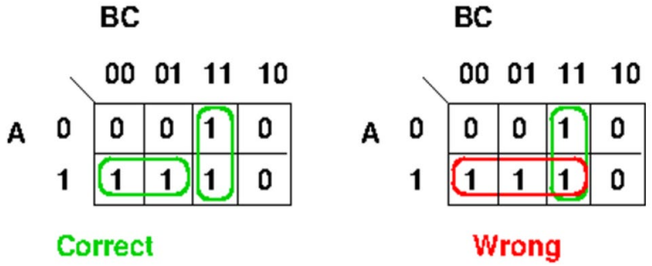


☞ Rule 2: Groups may be horizontal/vertical/square, but never diagonal.

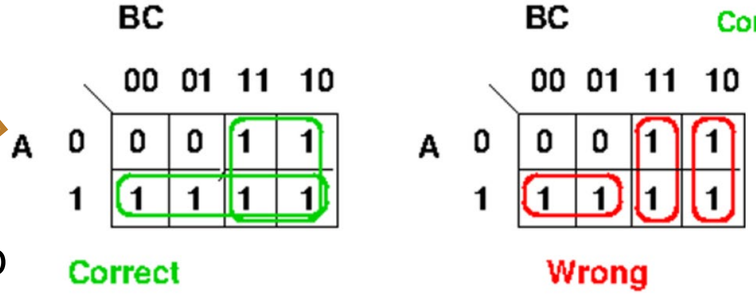


☞ Rule 3: Groups must contain 1, 2, 4, 8, 16, 32 ......(powers of 2) cells.

☞ Rule 4: **Each group must be as large as possible.**

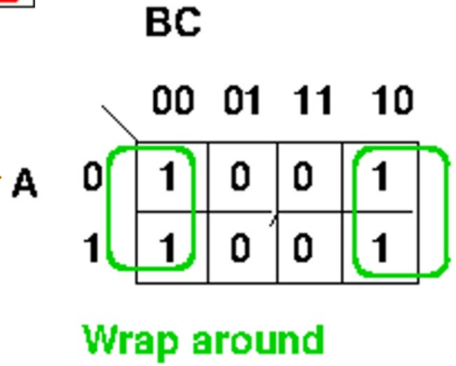


☞ Rule 5: **Groups can overlap.**



☞ Rule 6: Each 1 must be part of at least one group

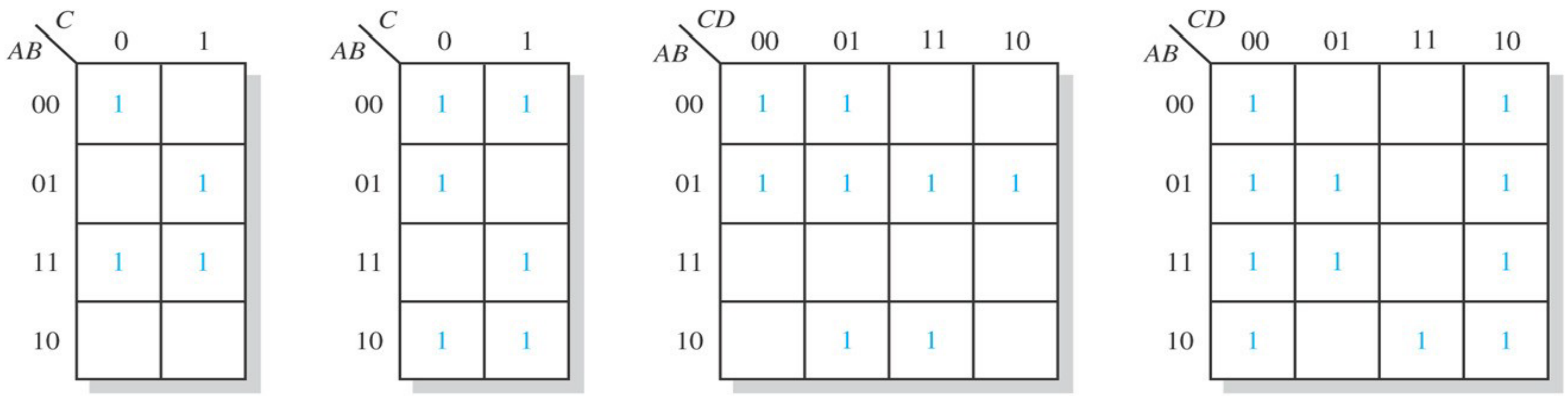
☞ Rule 7: Groups may wrap around the map.



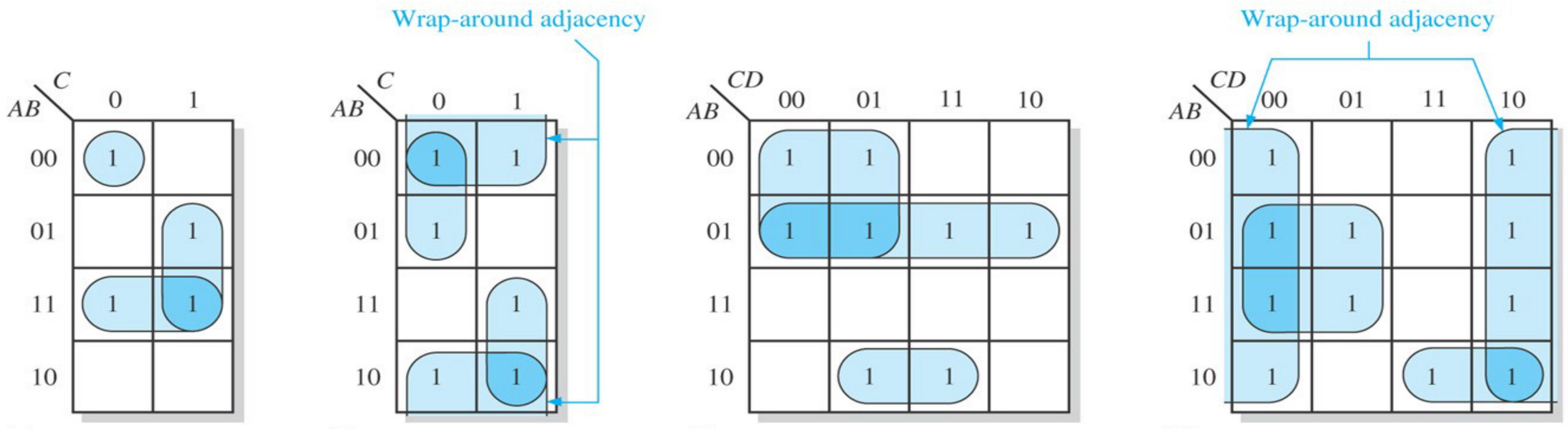
Karnaugh-maps are useful for smaller Boolean functions.
Automated algorithms for optimization are used for bigger functions.

Simplifying using Karnaugh-Maps (K-map)

Simplification example



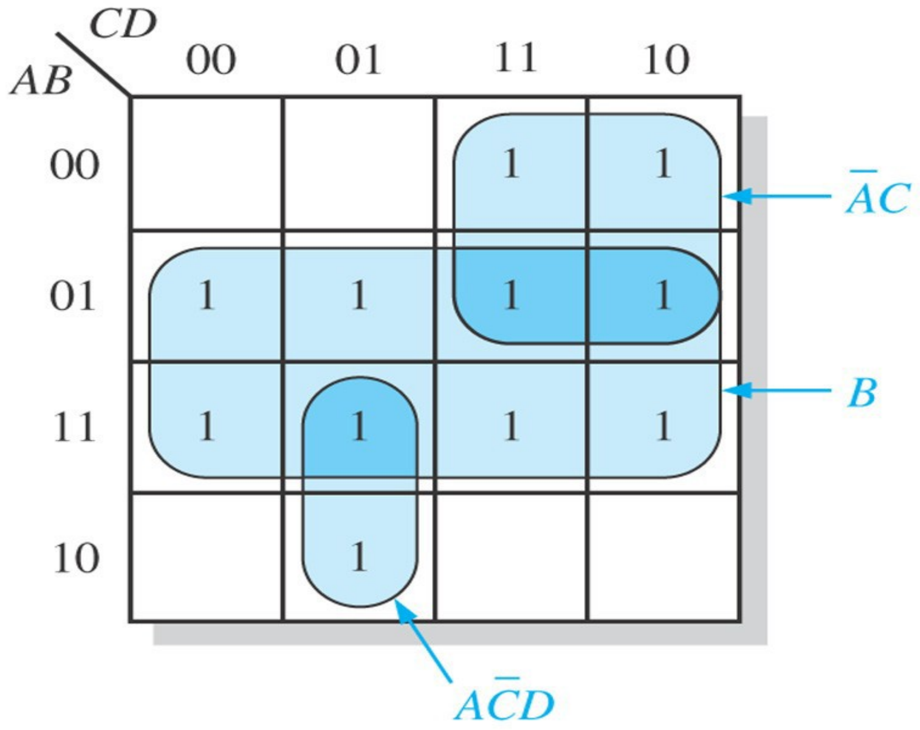
(a) (b) (c) (d)



(a) (b) (c) (d)

Simplifying using Karnaugh-Maps (K-map)

Simplification example:



$$F(A, B, C, D) == \bar{A}C + B + A\bar{C}D$$

Logic Gates and Truth Tables

Logic Gates

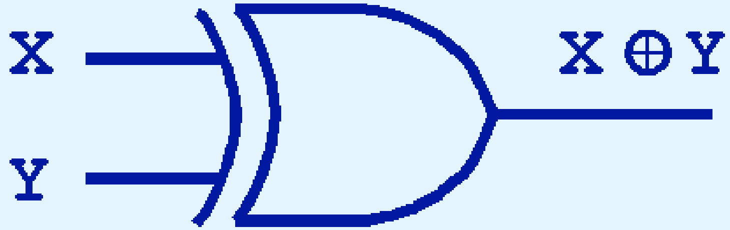
- The logic gate is the building brick of digital logic
- A logic gate is an electronic device that produces a result based on two or more input values
    - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit
- Integrated circuits contain collections of gates suited to a particular purpose

XOR

- The output of the XOR operation is True (1) only when the values of the inputs differ

X XOR Y

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0



Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

NAND and NOR (I)

NAND and NOR gates have special properties:

1. **NAND can be realised very efficiently.**
2. **All other gates can be build only using NAND gates.**

NAND is = $\overline{AB}$ or $\overline{XY}$

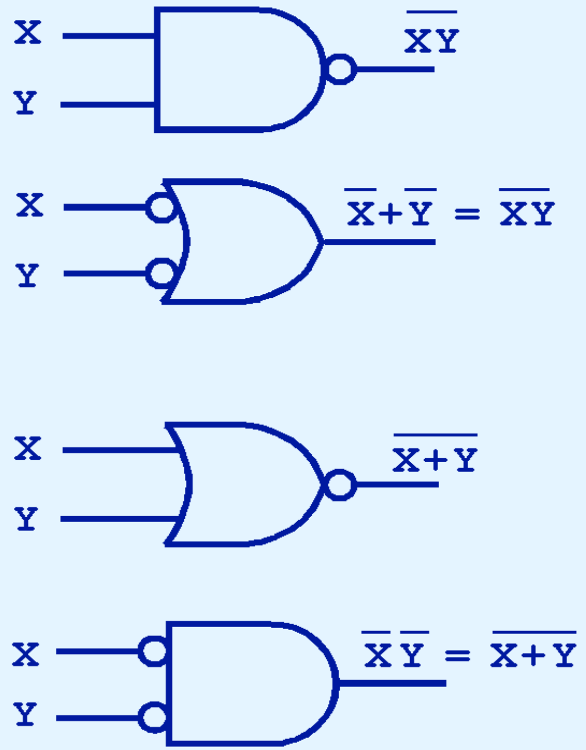
NOR is = $\overline{A+B}$ or $\overline{X+Y}$

X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

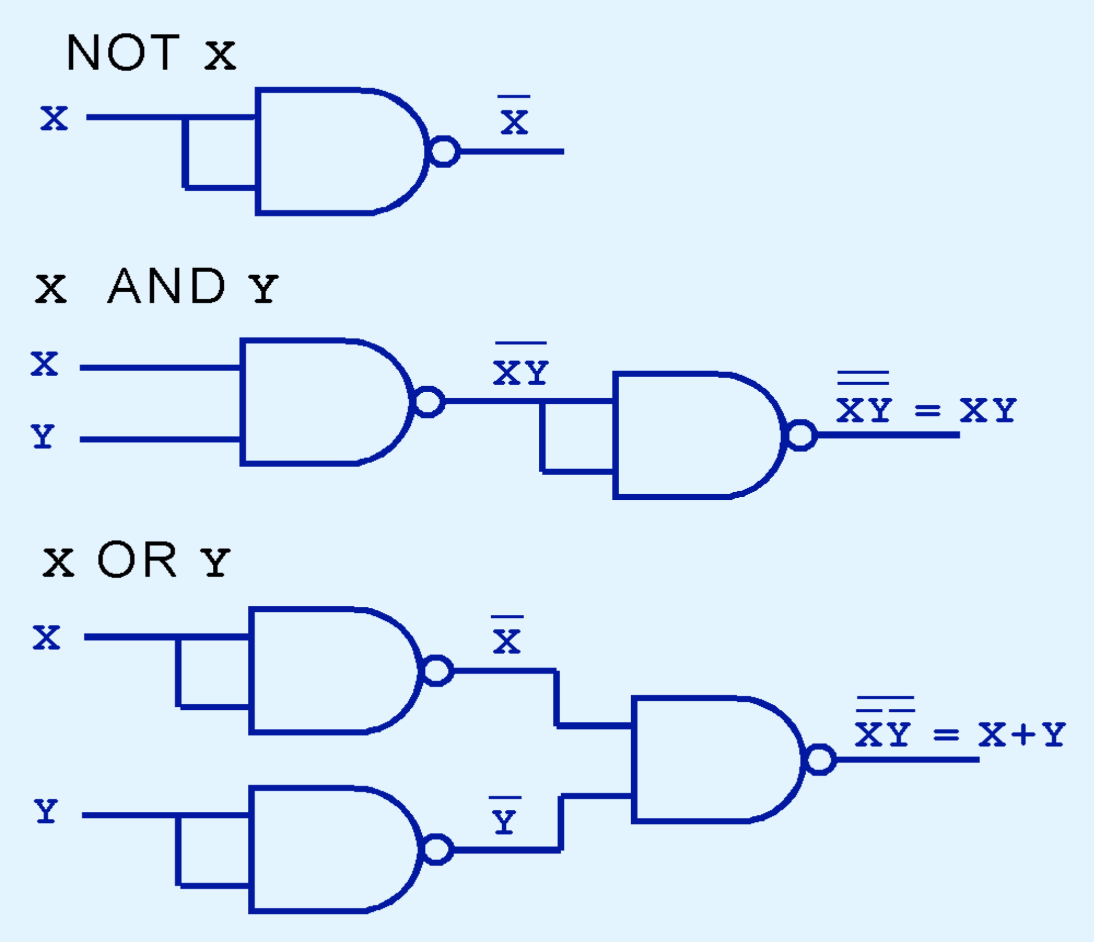


Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

NAND and NOR (II)

- Known as universal gates because they can be used to construct any gate

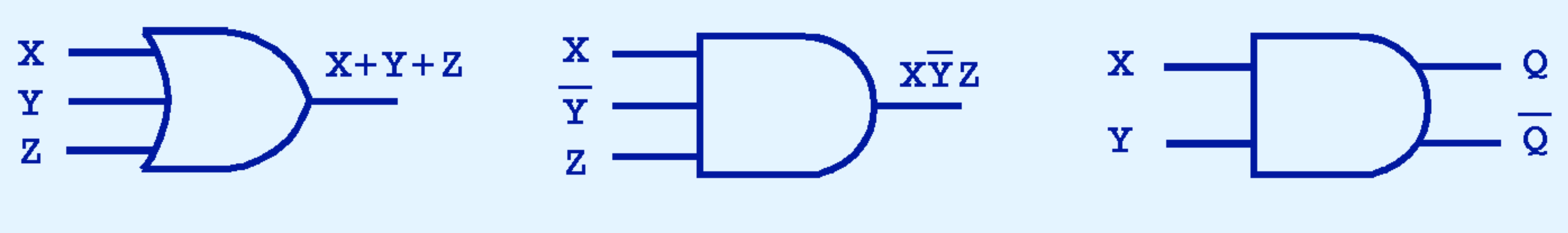
- Building **NOT** with NAND gates
- Building **AND** with NAND gates
- Building **OR** with NAND gates



Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Logic Gates: Inputs and Outputs

- Gates can have multiple inputs and more than one output
  - A second output can be provided for the complement of the operation



Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Summarize topics of first 2 weeks

- A little bit of history
- Vacuum tubes and transistors
- bit, byte, word (8bit/16bit/32bit/64bit)
- Numbering systems (base 2, base 10, base 16)
- Conversion between numbering systems
- Signed integer representations (sign/magnitude, 1's complement, 2's complement)

- Properties of 2's complement, adding 2's complement numbers
- Floating point representation
- Properties of floating point, precision, rounding
- Characters: ASCII and Unicode
- Error detection: Parity bits, Checksum, CRC

- Boolean Logic AND, OR, NOT, XOR, NAND, NOR
- Logic gates
- Simple logic circuits
- Boolean Algebra, Laws
- Karnaugh Maps/K-maps

End of Lecture Week-2 Hour 1

FIT1047 - Week 2 hour 2

Introduction to computer systems, networks and security

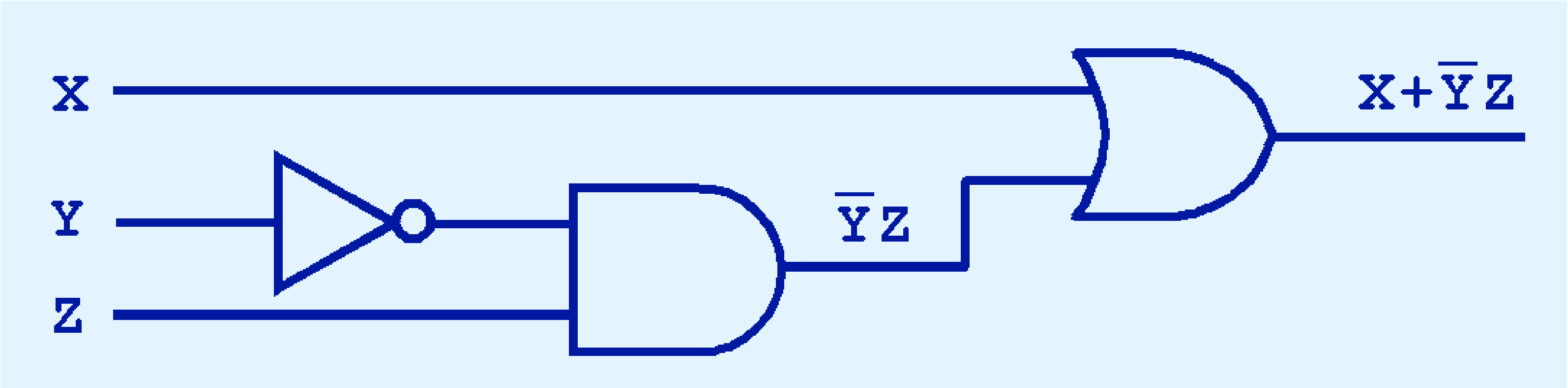
MONASH University

Reference: https://www.alexandriarepository.org/syllabus/introduction-to-computer-systems-networks-and-security/
Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett

Logic Gates and Boolean Functions

- Combinations of gates implement Boolean functions

Example $F(X,Y,Z) = X+\overline{Y}Z$



Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Simplifying a Function

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly
  - ✓ DeMorgan's law provides an easy way of finding the complement of a Boolean function
  - ✓ Recall DeMorgan's law states

$$\overline{(xy)} = \overline{x} + \overline{y} \quad \text{and} \quad \overline{(x+y)} = \overline{x}\,\overline{y}$$

  - DeMorgan's law can be extended to any number of variables

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Calculating the Complement of a Function

- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs
- Example
  - Find the complement of:

$$F(X,Y,Z) = (XY) + (\overline{X}Z) + (Y\overline{Z})$$

$$\begin{aligned}\overline{F}(X,Y,Z) &= \overline{(XY) + (\overline{X}Z) + (Y\overline{Z})} \\ &= \overline{(XY)}\,\overline{(\overline{X}Z)}\,\overline{(Y\overline{Z})} \\ &= (\overline{X}+\overline{Y})(X+\overline{Z})(\overline{Y}+Z)\end{aligned}$$

Reference: *Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.*

Canonical Form (I)

- There are numerous ways of stating the same Boolean expression
  - These “synonymous” forms are *logically equivalent*
  - Logically equivalent expressions have identical truth tables
  - In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form

Canonical Form (II)

There are **two canonical forms** for Boolean expressions

- ***Sum-of-products*** and ***product-of-sums***
  - Recall the Boolean product is the **AND** operation and the Boolean sum is the **OR** operation
- **Sum-of-products**: **ANDed** variables are **ORed** together
  - **For example: F(X,Y,Z) = XY + XZ + YZ**
- **Product-of-sums**: **ORed** variables are **ANDed** together
  - **For example: F(X,Y,Z) = (X+Y)(X+Z)(Y+Z)**

Converting to Sum of Products

We are interested in the values of the variables that make the function **True (=1)**

1. Using the Truth Table, make groups of **ANDed** variables (or negated variables), such that each group results in a True function value
2. OR together each group of variables

Converting to Sum of Products: Example

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

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$$
\begin{aligned}
F(x,y,z) = {} & \bar{x}y\bar{z} + \\
& \bar{x}yz + \\
& x\bar{y}\,\bar{z} + \\
& xy\bar{z} + \\
& xyz
\end{aligned}
$$

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Converting to Product of Sums

IDEA: use De Morgan Law to calculate the complement of the complement of the function

1. We are interested in the values of the variables that make the function **False (=0)**
2. **Complement** all the function values (function output)
3. Using the Truth Table, make groups of ANDed variables (or negated variables), such that each group results in a True value for the complement of the function (False value for the function)
4. OR together each group of variables

   → This yields the complement of the function
5. Take the complement of this complement

   → This yields the function itself

Converting to Product of Sums: Example

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

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$$\begin{aligned} F(x,y,z) &= \overline{\bar{x}\,\bar{y}\,\bar{z} + \bar{x}\,\bar{y}\,z + x\bar{y}z} \\ &= (\overline{\bar{x}\,\bar{y}\,\bar{z}})(\overline{\bar{x}\,\bar{y}\,z})(\overline{x\bar{y}z}) \\ &= (x+y+z)(x+y+\bar{z})(\bar{x}+y+\bar{z}) \end{aligned}$$

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

Calculating the Complement of a Function

- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs
- Example
  - Find the complement of:

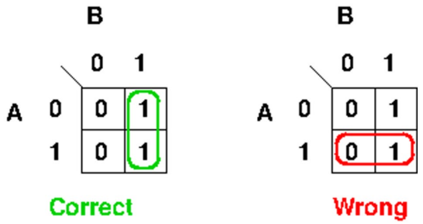
$$F(X,Y,Z) = (XY) + (\overline{X}Z) + (Y\overline{Z})$$

$$\begin{aligned}\overline{F}(X,Y,Z) &= \overline{(XY) + (\overline{X}Z) + (Y\overline{Z})} \\ &= \overline{(XY)}\,\overline{(\overline{X}Z)}\,\overline{(Y\overline{Z})} \\ &= (\overline{X}+\overline{Y})(X+\overline{Z})(\overline{Y}+Z)\end{aligned}$$

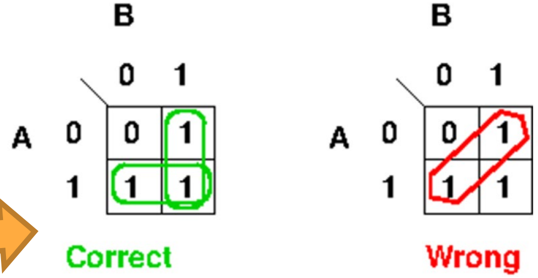
Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

7 rules for working with Karnaugh-Maps (K-map)

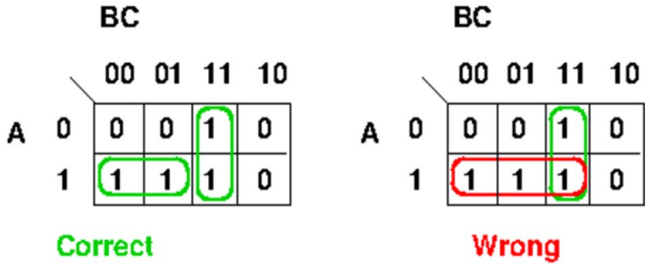
- Rule 1: No group can contain a zero.



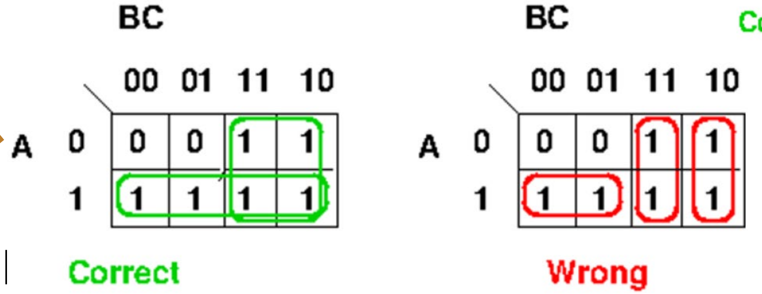
- Rule 2: Groups may be horizontal/vertical/square, but never diagonal.



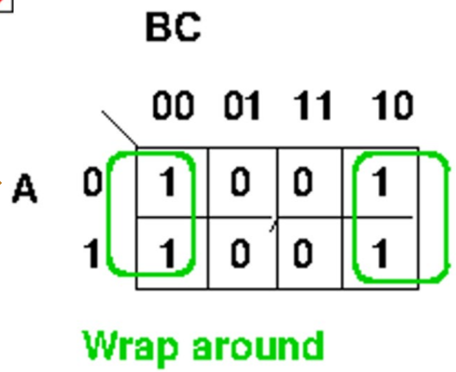
- Rule 3: Groups must contain 1,2,4,8,16,32 ……….(powers of 2) cells.
- Rule 4: Each group must be as large as possible.



- Rule 5: Groups can overlap.



- Rule 6: Each 1 must be part of at least one grou|
- Rule 7: Groups may wrap around the map.



Karnaugh-maps are useful for smaller Boolean functions.
Automated algorithms for optimization are used for bigger functions.

FIT1047 TUTORIAL 2 (week-2)

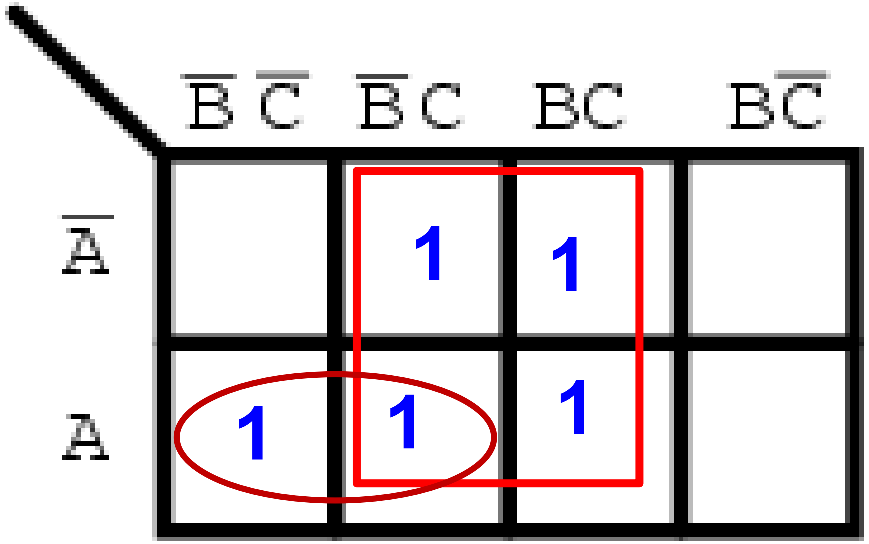
2d (Additional advanced task:) Use logisim and only AND, OR and NOT to build a circuit for the following function:

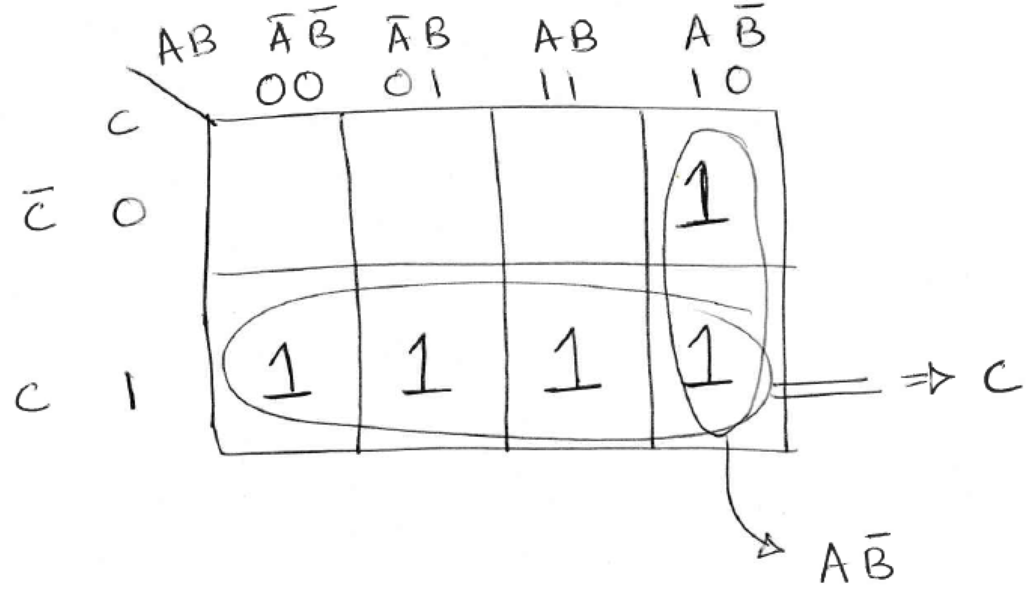
$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$

How many gates do you need? Simplify the function first, e.g. by using k-maps.

METHOD – II : Using K-maps

$= A\bar{B} + C$





$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$

$= C + A\bar{B}$

FIT1047 TUTORIAL 2 (week-2)

2d (Additional advanced task:) Use logisim and only AND, OR and NOT to build a circuit for the following function:

$$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

How many gates do you need? Simplify the function first, e.g. by using k-maps.

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

METHOD – I : Using Boolean Identities

$$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}C(\bar{B}+B) + A\bar{B}(\bar{C}+C) + ABC$$

$$= \bar{A}C + A\bar{B} + ABC$$

$$= \bar{A}C + A(\bar{B}+BC)$$

$$= \bar{A}C + A(\bar{B}+B)(\bar{B}+C)$$

$$= \bar{A}C + A(\bar{B}+C)$$

$$= \bar{A}C + A\bar{B} + AC$$

$$= C(\bar{A}+A) + A\bar{B}$$

$$\boxed{F(A,B,C) = C + A\bar{B}}$$

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

Reference: Linda Null, Julia Lobur. The essentials of computer organization and architecture. Fourth edition, 2015. Jones & Bartlett Learning.

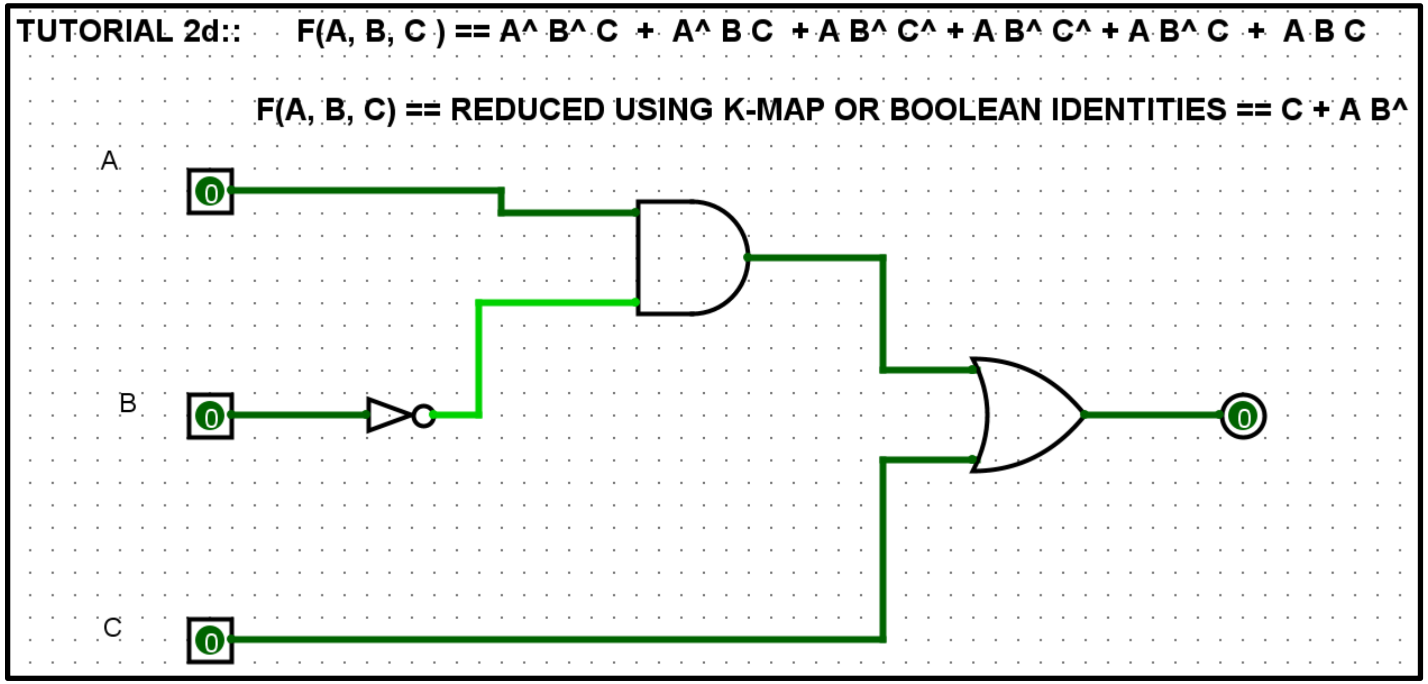
FIT1047 TUTORIAL 2 (week-2)

2d (Additional advanced task:) Use logisim and only AND, OR and NOT to build a circuit for the following function:

$$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC \quad = A\,\bar{B} + C$$

How many gates do you need? Simplify the function first, e.g. by using k-maps.

Simulate using logisim



Summarize topics of first 2 weeks

- A little bit of history
- Vacuum tubes and transistors
- bit, byte, word (8bit/16bit/32bit/64bit)
- Numbering systems (base 2, base 10, base 16)
- Conversion between numbering systems
- Signed integer representations (sign/magnitude, 1's complement, 2's complement)

- Properties of 2's complement, adding 2's complement numbers
- Floating point representation
- Properties of floating point, precision, rounding
- Characters: ASCII and Unicode
- Error detection: Parity bits, Checksum, CRC

- Boolean Logic AND, OR, NOT, XOR, NAND, NOR
- Logic gates
- Simple logic circuits
- Boolean Algebra, Laws
- Karnaugh Maps/K-maps

END of Lecture 2