# Tutorial: Overview of Boolean Algebra and Logic

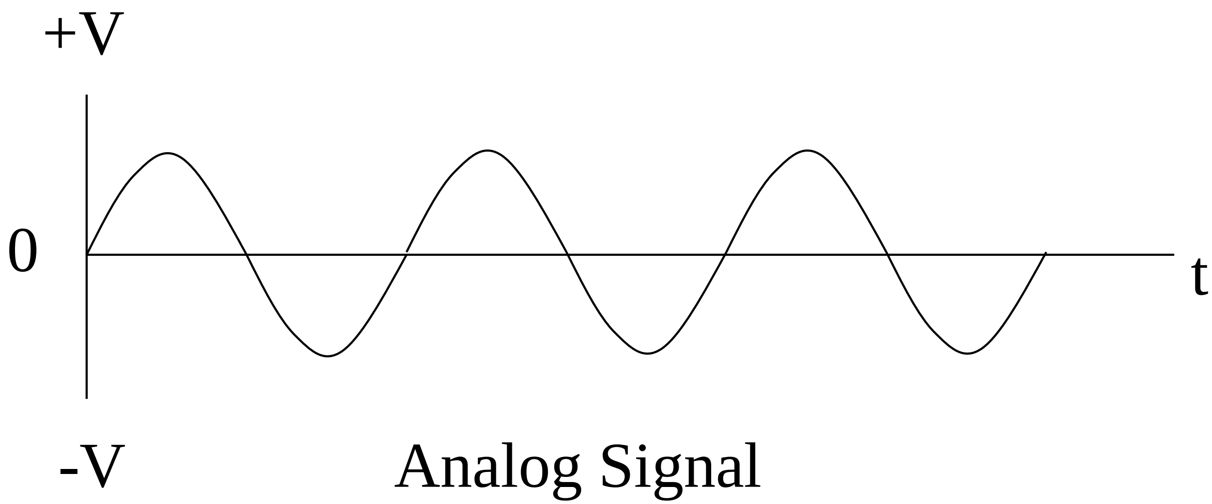
Yul Williams, D.Sc.

# Outline

- Digital Logic
- Boolean Algebra
- Boolean Functions
- DeMorgan’s Law
- Minterms and Maxterms
- Logic Minimization – The Karnaugh Map

# Digital Logic

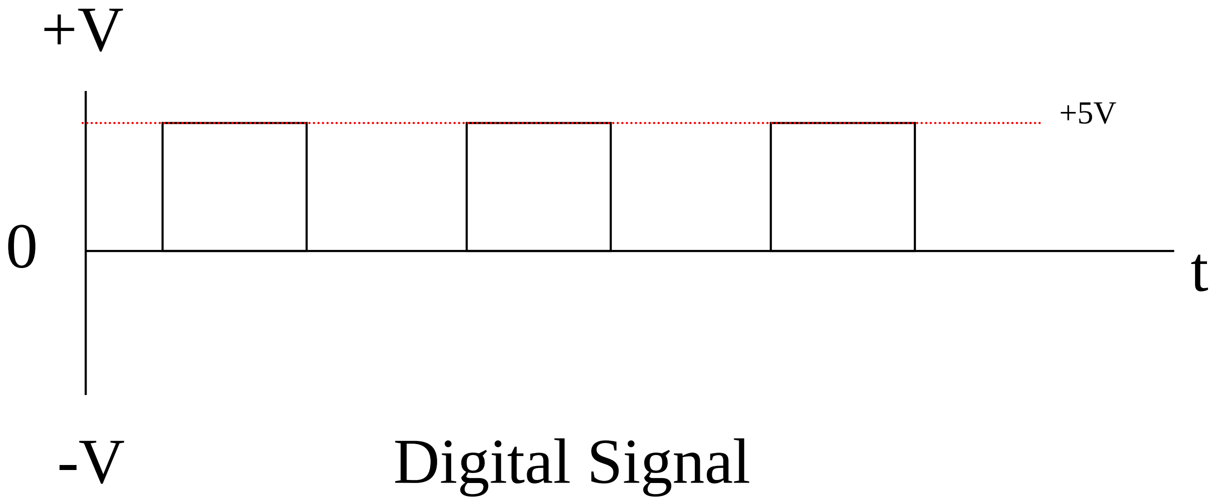
# Data Representation




Analog Signal

Analog signals are continuous and can take on a wide variety of values at specific intervals in time.

# Data Representation




Digital Signal

# Digital Systems

- In a digital system the physical signals may only assume discrete values.
- As shown in the previous figure, the digital signal may either be 0 volts or +5 volts
- Most components in a digital system assume only two discrete values

# Digital Logic Levels

If: V = 0 volts, then it is referred to as Logic ‘0’ or Logic “Lo”

If: V = 5 volts, then it is referred to as Logic ‘1’ or Logic “Hi”

# Boolean Algebra

It’s George’s Fault!!

# George Boole

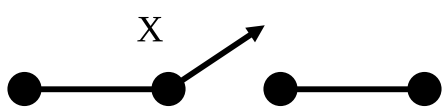
**(1815-1864)**

**Mathematician and logistician who developed ways of expressing logical processes using algebraic symbols, creating a branch of mathematics known as symbolic logic. Today, we call this Boolean Logic.**

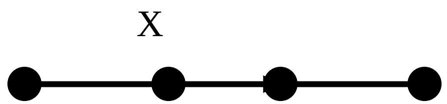
# Preliminaries

- Before we get into the design of computers and their components, we must understand the basics of Boolean algebra.
- The knowledge of Boolean algebra will enable you to specify and optimize your computer design(s)

# Switching Theory



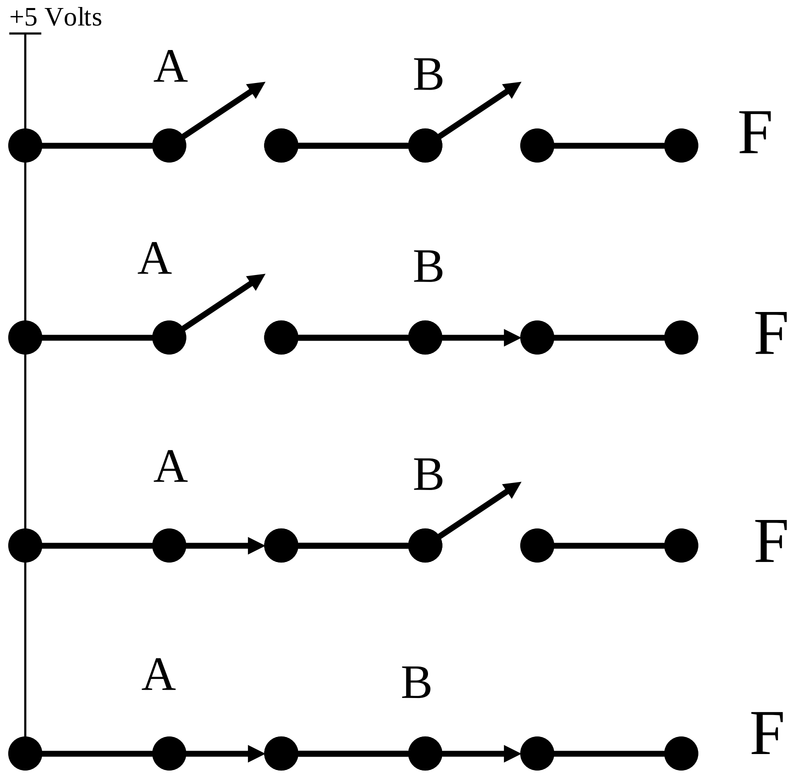

X = 0 → Switch open




X = 1 → Switch closed

# Switching Theory

## (The AND Gate)




A = 0, B = 0, F = 0

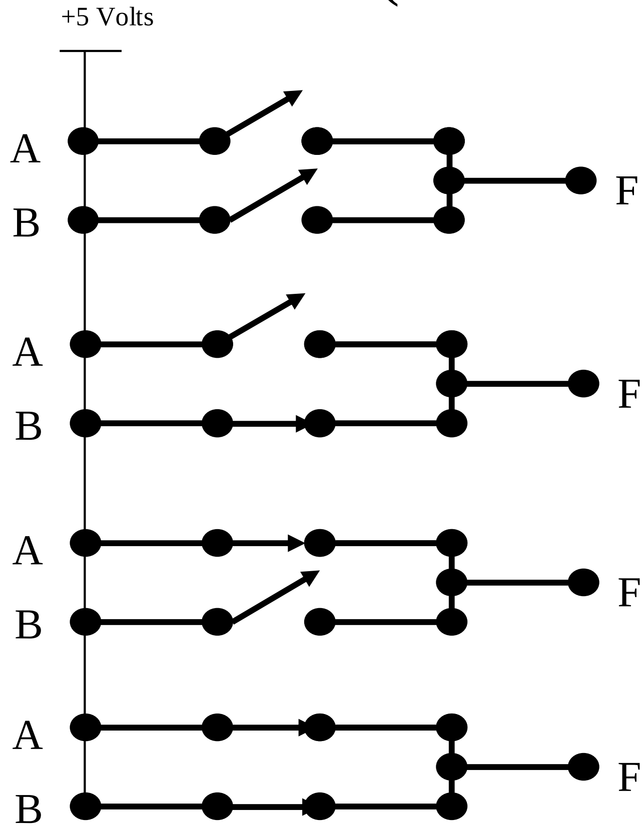
A = 0, B = 1, F = 0

A = 1, B = 0, F = 0

A = 1, B = 1, F = 1

# Switching Theory

## (The OR Gate)




A = 0, B = 0, F = 0

A = 0, B = 1, F = 1

A = 1, B = 0, F = 1

A = 1, B = 1, F = 1

# Boolean Functions

Mathematical and Symbolic Representations

# Symbolic Representations of Logic Functions

- There are more intuitive ways to represent logic functions than by the use of switches
- Each of the basic logic functions have symbolic representations that are universally understood
- These basic representations are referred to as logic gates

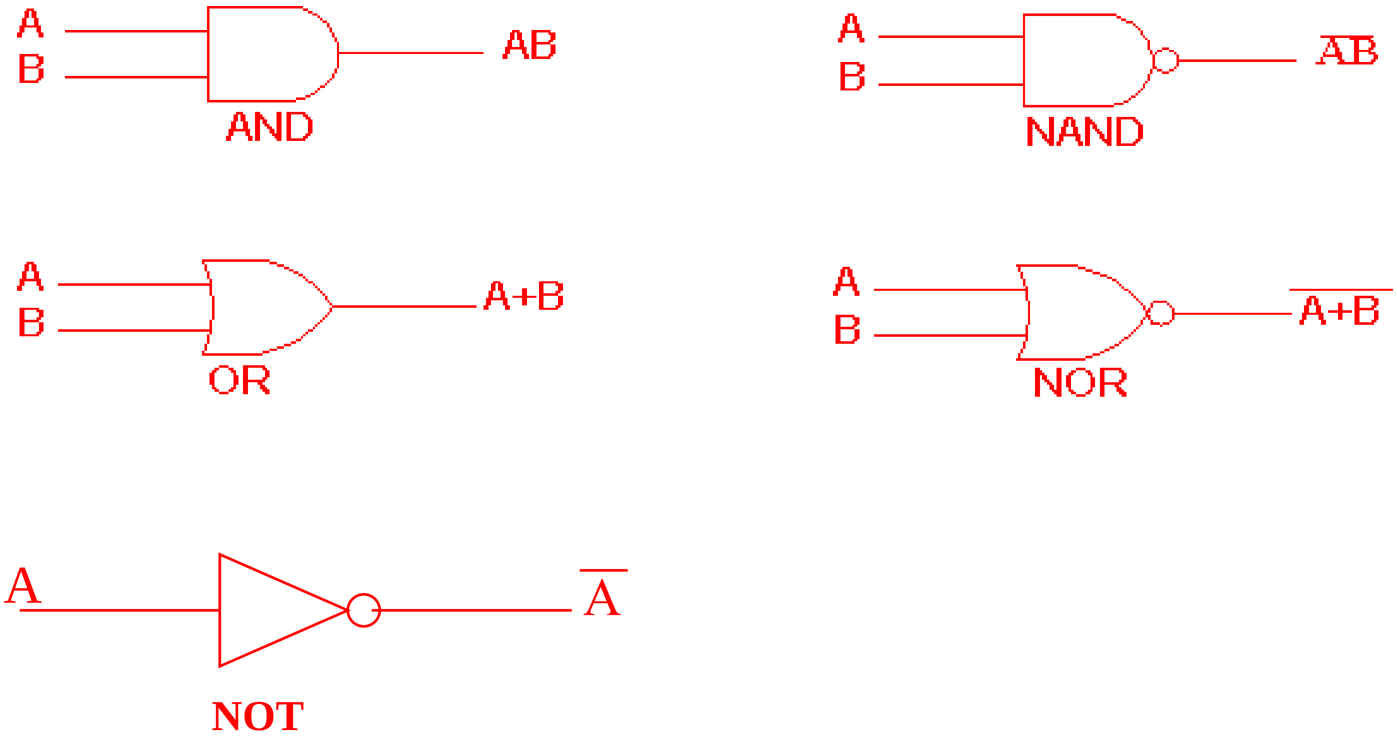
# Boolean Functions

- + → The OR function
- ' → The NOT function
- • → The AND function
- ⊕ →The XOR function

# Logic Gates

- Each Boolean function has a corresponding symbolic representation.
- These symbolic representations are commonly referred to as logic gates

# Logic Gates

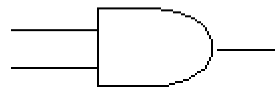

A
B
AB
AND
A
B
$\overline{AB}$
NAND
A
B
A+B
OR
A
B
$\overline{A+B}$
NOR
A
$\overline{A}$
NOT

# Truth Tables

- Truth tables specify values of a Boolean expression for all possible combinations of variables in the expression
- Each of the basic logic gates, for instance has a set of inputs. The respective truth tables for each gate shows the behavior of the gate's output for all possible input combinations.

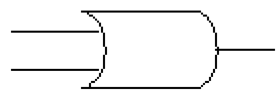
# Truth Tables for Logic Gates

AND



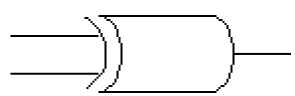
| A | B | A • B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR



| A | B | A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

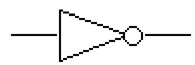
XOR



| A | B | $A \oplus B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Truth Tables for Logic Gates

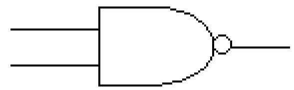
NOT



| $A$ | $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

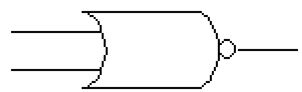
# Truth Tables for Logic Gates

NAND



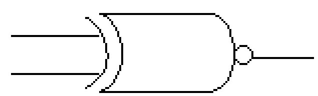
| A | B | $\overline{(A \bullet B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOR



| A | B | $\overline{(A + B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

XNOR



| A | B | $\overline{(A \oplus B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Exercise I

Draw the logic gates and generate the truth tables for the following expressions:

1. F = A + BC

2. F = AB' + A'B

3. F = A+B+AC

# Basic Theorems of Boolean Algebra

| | |
|---|---|
| $X + 0 = X$ | $X . 1 = X$ |
| $X + 1 = 1$ | $X . 0 = 0$ |

Idempotent Laws

| | |
|---|---|
| $X + X = X$ | $X . X = X$ |

Involution Law

$(X')' = X$

Laws of Complementarity

| | |
|---|---|
| $X + X' = 1$ | $X . X' = 0$ |

# Commutative, Associative and Distributive Laws

A lot of the laws of ordinary algebra are valid for Boolean algebra:

Commutative Law: $A+B = B+A$ $\quad AB = BA$

Associative Law: $(A+B)+C = A+(B+C)$ $\quad (AB)C = A(BC)$

Distributive Law: $A(B+C) = AB + AC$

Second Distributive Law: $A + BC = (A+B)(A+C)$

Note: This law does not hold for ordinary algebra.

# Simplification Theorems

These theorems are very helpful in simplifying Boolean expressions:

1. AB +AB' = A
2. (A+B)(A+B') = A
3. A + AB = A
4. A(A+B) = A
5. (A+B')B = AB
6. AB'+B = A+B

# Simplification Theorems Proof of Theorem 1

1. $AB + AB' = A$

$A(B + B') = A(1) = A$

# Simplification Theorems
# Proof of Theorem 2

2. $(A+B)(A+B') = A$

   $AA + AB' + AB + BB'$

   $A + A(B'+B) + 0$

   $A + A = A$

# Simplification Theorems
# Proof of Theorem 3

3. $A + AB = A$

   $A(1 + B)$

   $A(1) = A$

# Simplification Theorems
# Proof of Theorem 4

4. $A(A+B) = A$
   $AA + AB$
   $A + AB$
   $A(1+B)$
   $A(1) = A$

# Simplification Theorems
# Proof of Theorem 5

5. $(A+B')B = AB$
   $AB + BB'$
   $AB + 0 = AB$

# Simplification Theorems
# Proof of Theorem 6

6. $AB'+B = A+B$

$B+AB' = A+B$

$(B+A)(B+B')$

$(B+A)(1) = B+A = A+B$

# DeMorgan’s Law

# Augustus De Morgan

**( 1806 - 1871)**

He recognized the purely symbolic nature of algebra and he was aware of the existence of algebras other than ordinary algebra. He introduced De Morgan's laws and his greatest contribution is as a reformer of mathematical logic.

# Logic Equivalence

- DeMorgan's Law allows us to convert an AND function into an equivalent OR function (and vice versa)

DeMorgan's Law may be expressed by the following equivalent equations:

$$(ab)' = a' + b'$$

$$(a + b)' = a'b'$$

# DeMorgan's Law

- DeMorgan's Law (in general):

$$(a+b+c+\ldots)' = a'b'c'\ldots$$

$$(abc\ldots)' = a'+b'+c'+\ldots$$

# Minterms and Maxterms

# Minterms

- A minterm of *n* variables is a product of *n* literals in which each variable appears exactly once in either true or complimented form (but not both)

| ABC | Minterms | Designator |
|---|---|---|
| 000 | A'B'C' | $= m_0$ |
| 001 | A'B'C | $= m_1$ |
| 010 | A'BC' | $= m_2$ |
| 011 | A'BC | $= m_3$ |
| 100 | AB'C' | $= m_4$ |
| 101 | AB'C | $= m_5$ |
| 110 | ABC' | $= m_6$ |
| 111 | ABC | $= m_7$ |

A Boolean expression such as: F = ABC + A'B'C + AB'C is expressed as a sum of products. In addition, it may be expressed using the designator representation (or m-notation).

$F(A,B,C) = m_1 + m_5 + m_7$

OR

$F(A,B,C) = \sum m(1, 5, 7)$

When examining a truth table for a given expression, the minterms correspond to the 1's in F.

# Maxterms

- A maxterm of *n* variables is a sum of *n* literals in which each variable appears exactly once in either true or complimented form (but not both)

| ABC | Maxterms | Designator |
|---|---|---|
| 000 | A+B+C | $= M_0$ |
| 001 | A+B+C’ | $= M_1$ |
| 010 | A+B’+C | $= M_2$ |
| 011 | A+B’+C’ | $= M_3$ |
| 100 | A’+B+C | $= M_4$ |
| 101 | A’+B+C’ | $= M_5$ |
| 110 | A’+B’+C | $= M_6$ |
| 111 | A’+B’+C’ | $= M_7$ |

A Boolean expression such as: F = (A+B+C)(A’+B’+C)(A+B’+C) is expressed as a product of sums. In addition, it may be expressed using the designator representation (or M-notation).

$F(A,B,C) = M_0M_2M_6$

OR

$F(A,B,C) = \prod M(0, 2, 6)$

When examining a truth table for a given expression, the maxterms correspond to the 0’s in F.

# Logic Minimization

The Karnaugh Map

# Cost as a Function of Logic

- Each logic design will consist of a number of logic gates. Each gate that is a part of the resulting design adds cost to the design.
- The goal of logic minimization is to perform the logic function while minimizing the number of required gates (thus reducing the cost).
  - One way to solve this problem is to use the algebraic techniques that we just learned
  - Another way is to use a tool called the Karnaugh Map

# The Karnaugh Map

- Used to minimize logic functions
  – Logic functions may be minimized using an algebraic
- Many logic functions may be plotted on a Karnaugh map (sometimes called a K-map)
- Mapping functions containing more than 6 variables becomes difficult with Karnaugh maps

# 2-Variable K-Maps

Initial Function: F = AB + A'B

Truth Table for F:

| AB | F |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 11 | 1 |
| 10 | 0 |

K-Map

| B \ A | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |

B

F = B

Minimized Function

Algebraic Simplification of F

F = AB + A'B

= B(A + A')

= B(1)

= B

# The Gray Code

| | | | | |
|---|---|---|---|---|
| **0** | **0** | **0** | **00** | **000** |
| **1** | **1** | **1** | **01** | **001** |
| | | **-----** | **11** | **011** |
| | | **1** | **10** | **010** |
| | | **0** | | **-----** |
| | | | | **110** |
| | | | | **111** |
| | | | | **101** |
| | | | | **100** |

Notice that only a single digit is different from the number preceding or succeeding a specific number in the code sequence. These numbers are said to be adjacent to each other.

**Karnaugh maps are constructed using Gray codes . This is an important concept because this property helps to identify redundant terms in logic expressions so they may be systematically eliminated.**

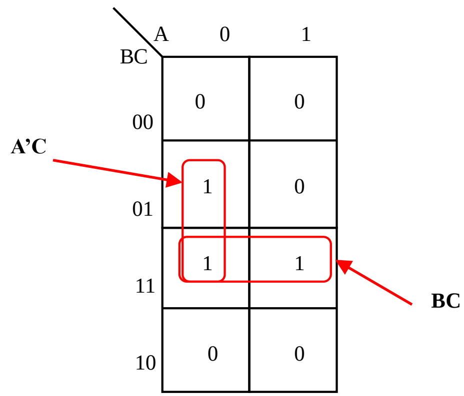
# 3-Variable K-Maps

Initial Function: $F = ABC + A'BC + A'B'C$

Truth Table for F:

| ABC | F |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 0 |
| 011 | 1 |
| 100 | 0 |
| 101 | 0 |
| 110 | 0 |
| 111 | 1 |

K-Map




$F = A'C + BC$

Minimized Function

Algebraic Simplification of F

$$
\begin{aligned}
F &= ABC + A'BC + A'B'C \\
&= C(AB + A'B + A'B') \\
&= C(AB + A' + B) \\
&= C(B(A + 1) + A') \\
&= C(B + A') \\
&= BC + A'C \\
&= A'C + BC
\end{aligned}
$$

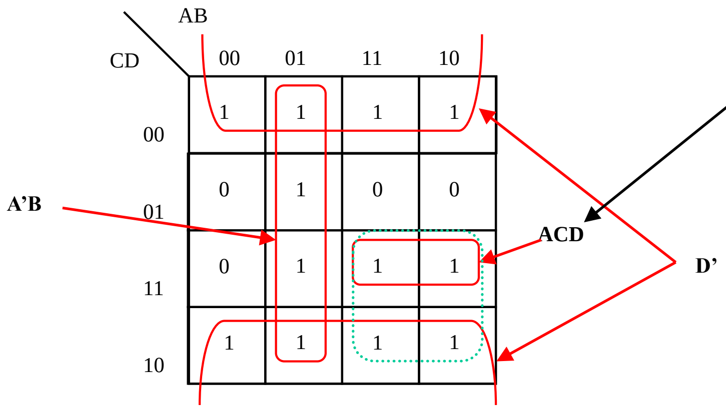
# 4-Variable K-Maps

Truth Table for F:

| ABCD | F |
|---|---|
| 0000 | 1 |
| 0001 | 0 |
| 0010 | 1 |
| 0011 | 0 |
| 0100 | 1 |
| 0101 | 1 |
| 0110 | 1 |
| 0111 | 1 |
| 1000 | 1 |
| 1001 | 0 |
| 1010 | 1 |
| 1011 | 1 |
| 1100 | 1 |
| 1101 | 0 |
| 1110 | 1 |
| 1111 | 1 |

Initial Function F:

F = A'B'C'D'+A'B'CD'+A'BC'D'+A'BC'D+A'BCD'+
A'BCD+AB'C'D'+AB'CD'+AB'CD+ABC'D'+ABCD'+
ABCD




The term ACD is a non-optimal grouping cause an extra term to be produced that adds no value to the valuation of the equation. Take the green grouping instead that yields the term AC.

F = AC + A'B + D'

Minimized Function

# Practice Questions

1. Prove the following:

   cd + cd’ = c

   (x+y)(x+y’) = x

   (a+x)(a+y) = a + xy

2. Draw the logic network for the following Boolean expressions:

   a + bc + d

   ab(c + d)

   (ab + b’c + a’c’)d

3. Draw the Karnaugh map and find the minimum sum of products for the function: F = b’c’+a’bd+abcd’+b’c

4. Draw the Karnaugh map and find the minimum product of sums for the function F in problem 3.