UCCD1133
Introduction to Computer Organisation and Architecture

Chapter 3
Basic Concept of Logic

Disclaimer

- This slide may contain copyrighted material of which has not been specifically authorized by the copyright owner. The use of copyrighted materials are solely for educational purpose. If you wish to use this copyrighted material for other purposes, you must first obtain permission from the original copyright owner.

Chapter 3-1

Introduction to concept of logic

Outline

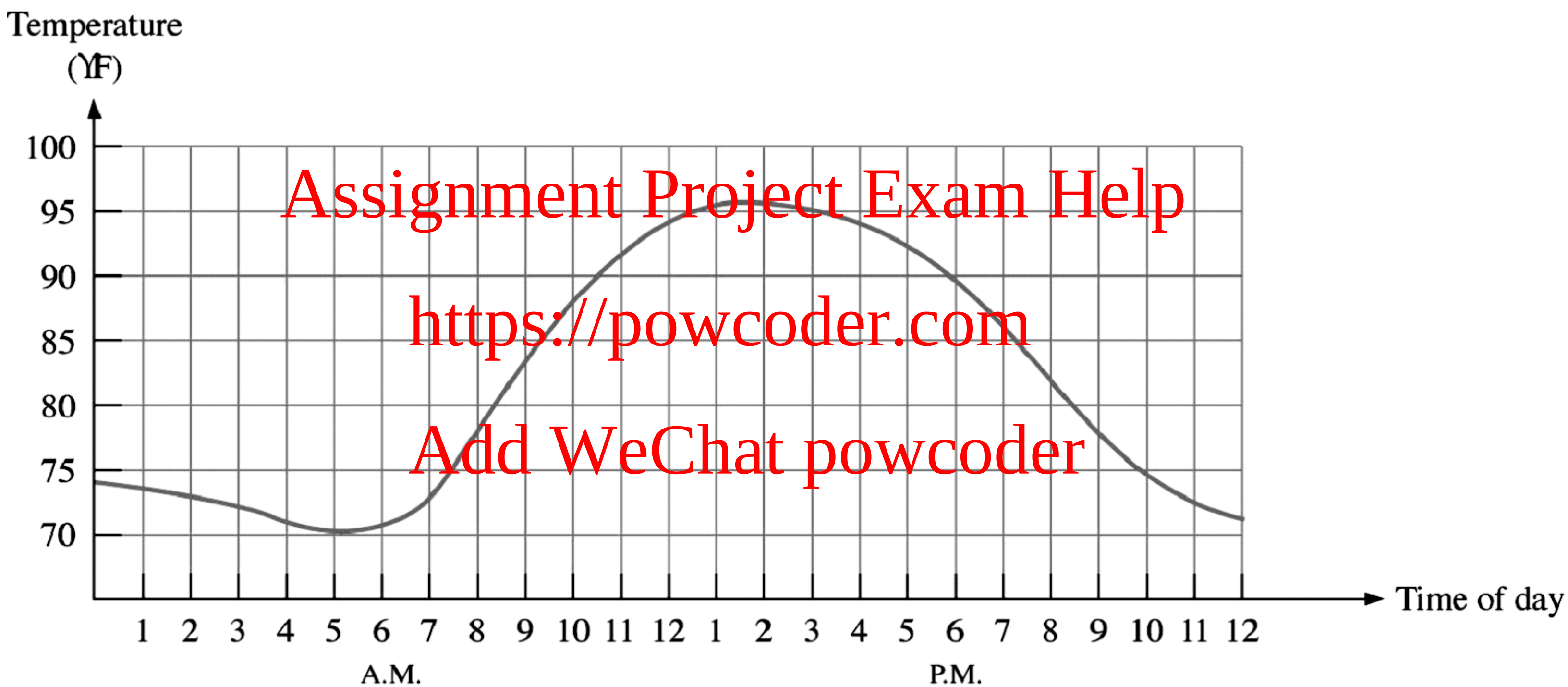
- Analogue and digital quantities
- Binary digits, signal voltage level and logic level
- Data transfer: serial and parallel

Digital versus Analogue

- Electronic circuits can be divided into two broad categories
  - *digital*
  - *Analogue*
- Most physical quantities are analogue in nature.
- However, the electronics inside a computer are *digital*.

Analogue and Digital Quantities

- An analogue quantity - having continuous values over time
- Example: position, velocity, acceleration, force, pressure, temperature and flow rate.

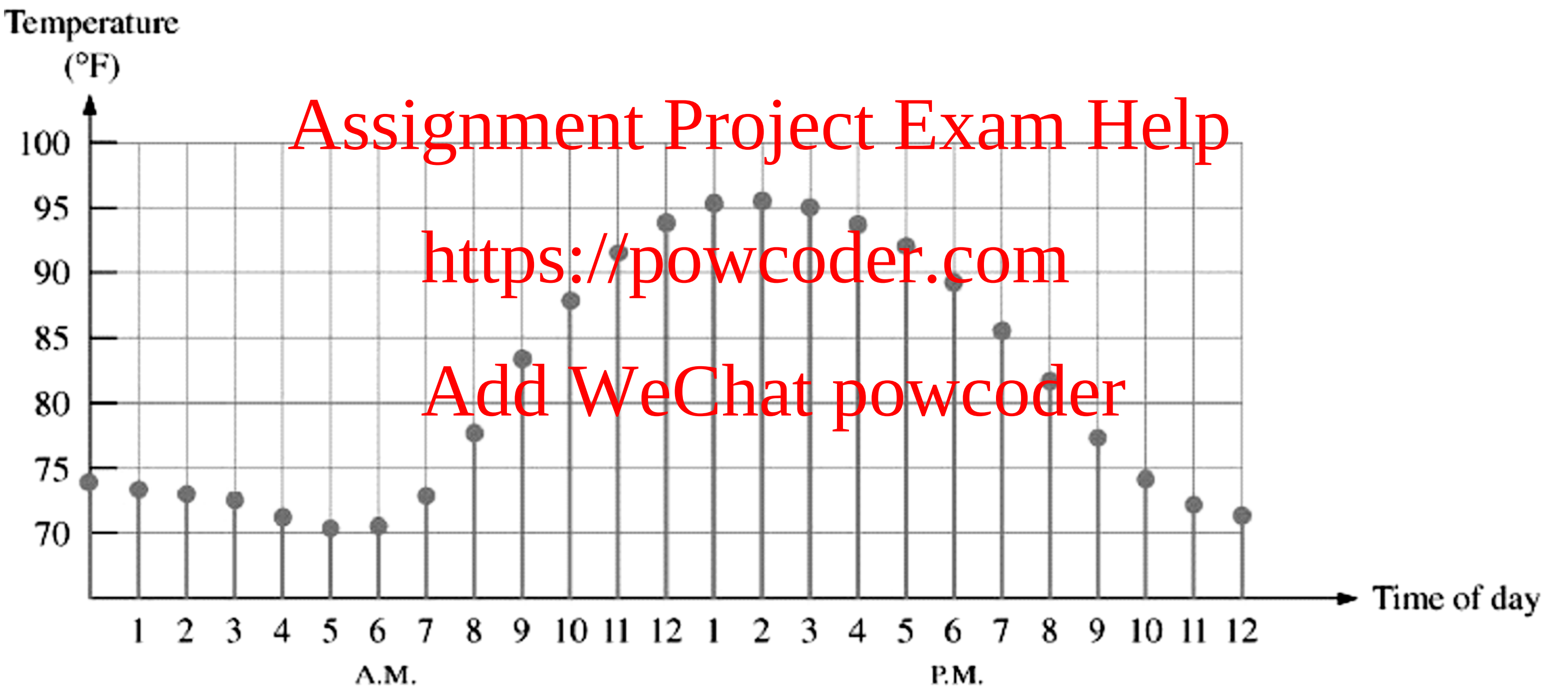


The air temperature values change over a continuous time (analogue quantity)

- No sudden jump in value, e.g. from 92 to 94 degrees.
- In between values are considered as infinite.

Analogue and Digital Quantities

- An digital quantity - having discrete set of values over time
- Example, the air temperature values are read at every hour – at discrete point in time.
  - Can assign a digital value or code to each dot to represent it - digitised.



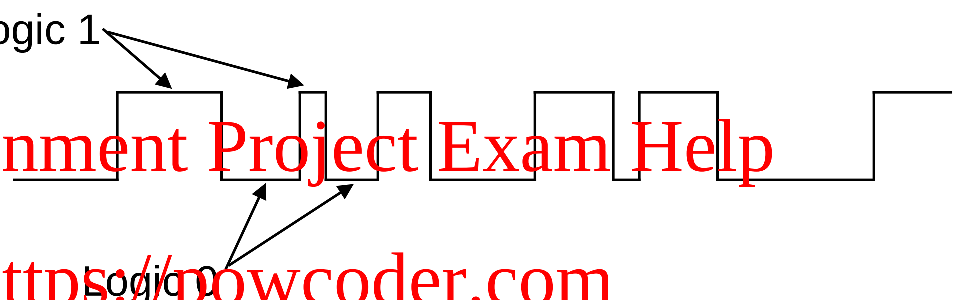
Sampled-value representation of the analogue quantity

- Digital representations of physical quantities are easier to be stored, transferred, and copied.

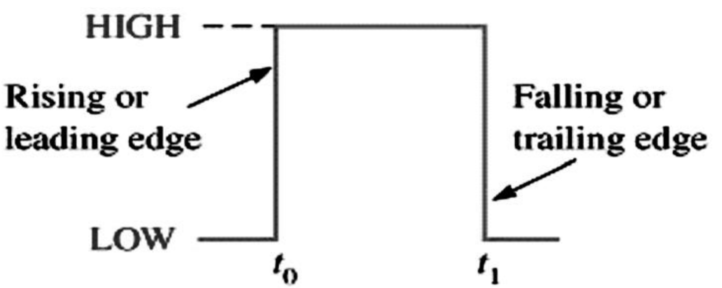
Binary Digits, Signal Voltage Level and Logic Level

- The fact that computers are *digital* is a key reason they use *binary* system.
- Binary system contains only two digits, 0 and 1
  - **Bits** comes from **B**inary dig**its** – 0 and 1.
- When apply in digital electronics, 0 and 1 correspond to logic 0 and logic 1

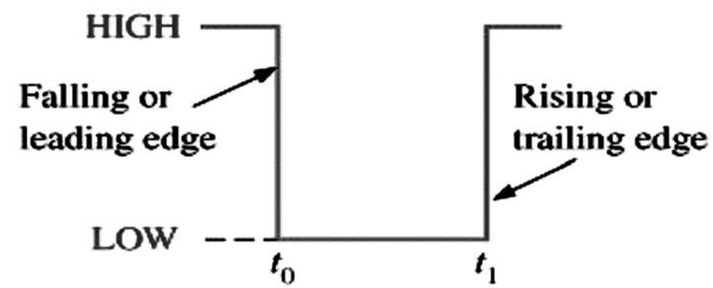
A digital signal/ waveform



- Voltage levels changing back and forth between the HIGH (H) and LOW (L) instantaneously



(a) The digital signal is normally LOW, then a positive-going pulse takes place.

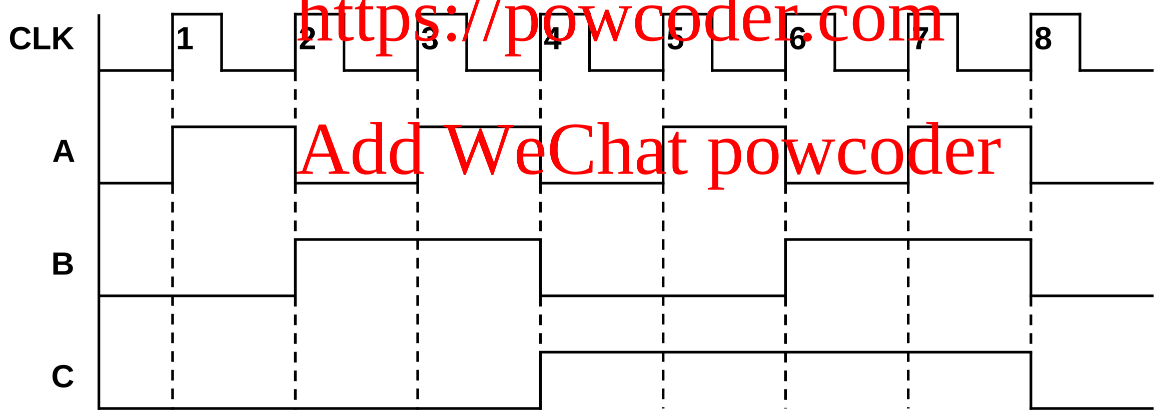


(b) The signal is normally HIGH, then a negative-going pulse takes place.

A Special Digital Signal: The Clock Signal

- Clock circuitry is commonly used in digital systems (analogous to a human heart) to generate periodic clock waveform.
  - To synchronize the generation of other waveforms
  - Thus, synchronizing the transfer activities within the digital system.
    - Creates synchronous digital system

- E.g. waveforms A, B and C are synchronized to the **clock**.
  - The transitions in waveforms A, B and C occur at the rising-edge of the clock



Data Transfer

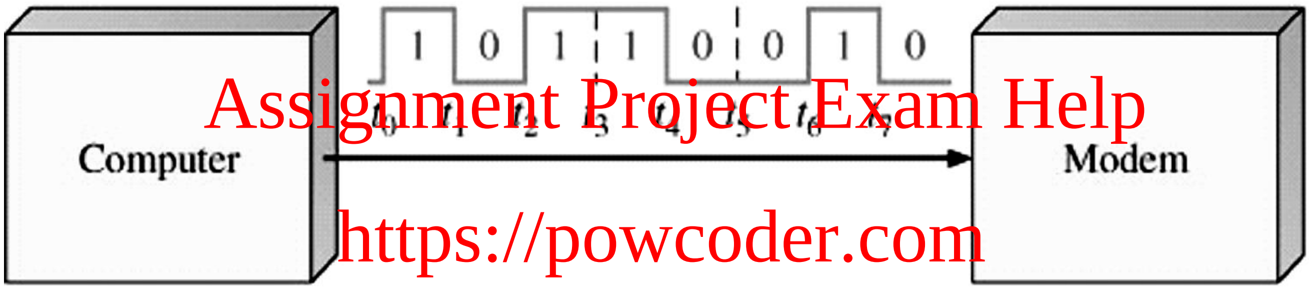
- Grouping of bits is commonly done to represent information in digital system
  - 4 bits => a **nibble**
  - 8 bits => a **byte**
  - 32 bits => a **word**
  - E.g. 2^8 can represent up to 256 information

- Single or group of bits are transferred as binary data from one circuit to another
  - Serial transfer
  - Parallel transfer

- Data transfer involves a transmitting circuit (source) and one or more receiving circuit (destination)

Serial Data Transfer

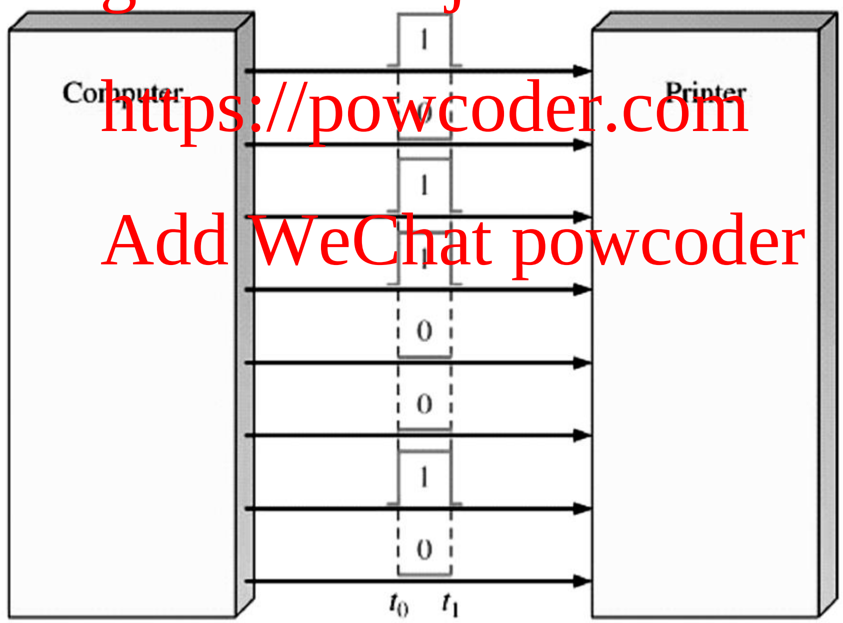
- Data is transferred one bit per clock cycle
  - Require only a single line
- Example, from one digital system (computer) to another (modem)
  - Total clock cycle required to transfer a byte of data: 8 cycles.



Serial transfer of 8 bits of binary data from computer to modem. Interval t_0 to t_1 is first.

Parallel Data Transfer

- Data is transferred as a group (e.g., a byte or word) per clock cycle
  - Require multiple lines
- Example, from the computer to a printer
  - 8 bits are transferred along 8 lines in a clock cycle
  - Total clock cycle required no matter how large the group: 1 cycle.



Chapter 3-2

Boolean algebra & truth table

Outline

- Laws and theorems of Boolean Algebra.
- Apply these laws and theorems to:
  - Simplify expressions
  - Convert any Boolean expression into a sum-of-product (SOP) form
  - Convert non-canonical form to canonical form
- Represent a Boolean expression by truth table.

Fundamentals of Boolean Algebra

- Boolean algebra is mathematics of logic.

- Unlike ordinary algebra; each **variable** represents a logical quantity which can take on only one of two values: 0 and 1.

- The **complement** is the inverse of a variable
  - E.g. the complement of U => U’ or $\bar{U}$

- Boolean algebra logical operators:
  - “ + ” => logical OR operator
  - “ . ” => logical AND operator
  - “ ’ ” or “ ¯ ” => logical NOT operator
  - E.g. on OR, AND and NOT computation

OR function with two variables = x + y	AND function with two variables = x.y	NOT function with a single variable = x'
0 + 0 = 0	0.0 = 0	0’ = 1
0 + 1 = 1	0.1 = 0	1’ = 0
1 + 0 = 1	1.0 = 0	
1 + 1 = 1	1.1 = 1	

- The operations of a Boolean algebra must adhere to certain laws and theorems

Laws of Boolean Algebra

- Law 1: Existence of 1 and 0 element
  - (a) $p + 0 = p$
  - (b) $p \cdot 1 = p$
- Law 2: Existence of complement
  - (a) $p + p' = 1$
  - (b) $p.p' = 0$
- Law 3: Commutativity
  - (a) $p + q = q + p$
  - (b) $p \cdot q = q \cdot p$

- Law 4: Associativity
  - (a) $p + (q + z) = (p + q) + z$
  - (b) $p \cdot (q \cdot z) = (p \cdot q) \cdot z$
- Law 5: Distributivity
  - (a) $p + (q \cdot z) = (p + q) \cdot (p + z)$
  - (b) $p \cdot (q + z) = p \cdot q + p \cdot z$

The laws of Boolean algebra can be used to further develop Theorems

Example

1. $a + b + c + 0 = a + b + c$ L1(a)
2. $a.b.c.1 = a.b.c$ L1(b)
3. $(W' + X' + Y' + Z')(W' + X' + Y' + Z)(W' + X' + Y + Z')(W' + X' + Y + Z)$
   $= ((W' + X' + Y') + Z' Z) ((W' + X' + Y) + Z' Z)$ L5(a)
   $= (W' + X' + Y') (W' + X' + Y)$ L2(b)
   $= (W' + X') + Y' Y$ L5(a)
   $= (W' + X')$ L2(b)

Theorems of Boolean Algebra

- **Theorem 1: Idempotency**
  - (a) x + x = x
  - (b) x.x = x

Example

1. x + x + x + x + x = x   T1(a)
2. x + a + a + x + x + a = x + a   T1(a)
3. x.x.x.x.x.x = x   T1(b)
4. x.a.x.x.a.a.a = x.a   T1(b)

- Let's prove T1(a)

x + x	= (x + x)1	[L1(b)]
	= (x + x)(x + x')	[L2(a)]
	= x + x.x'	[L5(a)]
	= x + 0	[L2(b)]
	= x	[L1(a)]

- **Theorem 2: Null element**
  - (a) x + 1 = 1
  - (b) x.0 = 0

Example

1. a + b + c + d + 1 = 1   T2(a)
2. a.b.c.d.0 = 0   T2(b)

- Let's prove T2(a)

x + 1	= (x + 1)1	[L1(b)]
	= 1(x + 1)	[L3(b)]
	= (x + x')(x + 1)	[L2(a)]
	= x + x' 1	[L5(a)]
	= x + x'	[L1(b)]
	= 1	[L2(a)]

- **Theorem 3: Involution**
  - (x')' = x

Theorems of Boolean Algebra

- **Theorem 4: Absorption**
  - (a) x + xy = x
  - (b) x(x + y) = x

Example

1. (X + Y) + (X + Y)Z = X + Y [T4(a)]
2. AB'(AB' + B'C) = AB' [T4(b)]

- Let's prove T4(a)

x + xy	= x1 + xy	[L1(b)]
	= x(1 + y)	[L5(b)]
	= x(y + 1)	[L3(b)]
	= x1	[T2(a)]
	= x	[L1(b)]

- **Theorem 5**
  - (a) x + x'y = x + y
  - (b) x(x' + y) = xy

Example

1. B + AB'C'D = B + AC'D [T5(a)]
2. (X + Y)((X + Y)' + Z) = (X + Y)Z[T5(b)]
3. wy' + wx'y + wxyz + wxz'
   = w(y' + x'y) + wx(yz + z')
   = w(y' + x') + wx(y + z')
   = wy' + wx' + wxz' + wxy
   ⋮
   = w

- Let's prove T5(a)

x + x'y	= (x + x')(x + y)	[L5(a)]
	= 1(x + y)	[L2(a)]
	= (x + y)1	[L3(b)]
	= (x + y)	[L1(b)]

Theorems of Boolean Algebra

- **Theorem 6: DeMorgan's Theorem**
  - To determine the complement of an expression
    - $(x + y)' = x'y'$
    - $(xy)' = x' + y'$
  - Generalized DeMorgan's Theorem
    - $(x + y + \dots z)' = x'y' \dots z'$
    - $(xy \dots z)' = x' + y' + \dots z'$

Example

$$
\begin{aligned}
(x + yz)' = (x + (yz))' &\neq ((x + y)z)' \\
&= x'(yz)' && [T6(a)] \\
&= x'(y' + z') && [T6(b)] \\
&= x'y' + x'z' && [L5(b)]
\end{aligned}
$$

- Useful in manipulating Boolean expressions into formats suitable for realization with specific types of logic gates
- Shortcut to apply DeMorgan's theorem is to invert the operators
  - E.g. $(x + yz)' = x'(y' + z')$.
  - Note $(x + yz)' \neq x'y' + z'$.

Theorems of Boolean Algebra

- **Theorem 7: Consensus**
  - (a) $xy + x'z + yz = xy + x'z$
  - (b) $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$

- Let’s prove T7(a).

$xy + x'z + yz$	$= xy + x'z + 1yz$	[L1(b)]
	$= xy + x'z + (x + x')yz$	[L2(a)]
	$= xy + x'z + xyz + x'yz$	[L5(b)]
	$= (xy + xyz) + (x'z + x'yz)$	
	$= xy + x'z$	[T4(a)]

Example

1. $AB + A'CD + BCD = AB + A'CD$ [T7(a)]

2.

$ABC + A'D + B'D + CD$	$= ABC + (A' + B')D + CD$	[L5(b)]
	$= ABC + (AB)'D + CD$	[T6(b)]
	$= ABC + (AB)'D$	[T7(a)]
	$= ABC + (A' + B')D$	[T6(b)]
	$= ABC + A'D + B'D$	[L5(b)]

Duality

- The dual of an expression is found by replacing
  - All (+) operators with (.).
  - All (.) operators with (+).
  - All ones with zeros.
  - All zeros with ones.

- Example 1

  p + p' = 1 [L2(a)]

  p.p' = 0 [L2(b)]

  L2(a) is the dual of L2(b).

- Example 2

  Find the dual of the expression x + (yz) = (x + y)(x + z).

  Solution 2

  x + (yz) = (x + y)(x + z)

  x(y + z) = (xy) + (xz)

- Do not alter the location of parenthesis when obtaining a dual.

Algebraic Representation of a Logic Function

variable
compliment

literal (e.g. 6 literals and 3 variables)

$$\overbrace{\underbrace{f(A,B,C)}_{\text{Logic function of 3 variables}} \quad = \quad \underbrace{AB + A'C + AC'}_{\text{Logic expression}}}^{\text{Logic equation}}$$

- Algebra – used as a mathematical representation of a logic function
  - Basically relates the inputs to output
  - How?
    - If any one or more terms AB, A'C or AC' are asserted, the output f will be asserted.
  - Evaluating f:
    - Example: if A = 1, B = 0, C = 0, then

      $f(1,0,0) = 1.0 + 1'.0 + 1.0'$
      $= 0 + 0 + 1$
      $= 1$

- Can also write:

$$Y \quad = \quad AB + A'C + AC'$$

Variables and logic function can be treated as signals

- C' is not a signal name.
  - It is an expression since ' is an operator.

Canonical Forms of Logic Expression

- A logic function can be expressed in a variety of algebraic forms.

  For example,

  $$Y = ab' + ac = a(b' + c) = a(a' + c + b')$$

- In general, a logic expression can be represented in the form of:
  - Sum-of-products (SOP).
  - Product-of-sum (POS).

- To eliminate the possible confusion, logic designers must learn to specify a Boolean function using *canonical* or standardised form

  – everyone will come up with the same expression.

Sum-of-Product (SOP) Forms

- SOP form
  - Example

A special product term called minterm (it has all the variables)

f(A, B, C) = A'BC + AB + C

Product term but not a minterm

- Canonical SOP form
  - f contains minterms only
  - Example

5 minterms

f(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC

- A *minterm* is a product involving all of the inputs to the function.

Converting Non-Canonical Form into Canonical Form

- Use of algebra method to convert non-canonical to canonical form

- Example 1

  Expand the following function to canonical SOP in minterm list form:

  $f(A,B,C) = AB + AC' + A'C$

  Solution 1

  $$\begin{aligned} f(A,B,C) &= AB + AC' + A'C \\ &= AB(C + C') + AC'(B + B') + A'C(B + B') \\ &= ABC + ABC' + AB'C' + A'BC + A'B'C \\ &= m7 + m6 + m4 + m3 + m1 \\ &= \sum m(1, 3, 4, 6, 7) \end{aligned}$$

- On the contrary,

  $f = ABC + ABC' + AB'C' + A'BC + A'B'C$

  can be simplified into

  $f = AB + AC' + A'C$

  using the previous theorems and laws.

Sum-of-Product (SOP) Forms

- Canonical form can also be written as *minterm list form*

  Example

  f(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC

  Solution

  From the truth table,

  f(A, B, C) = m1 + m3 + m5 + m6 + m7

  = $\sum$ m(1, 3, 5, 6, 7)

Inputs A B C	Minterm	Minterm Full List	
0 0 0	A'B'C'	m0	
0 0 1	A'B'C	m1	✓
0 1 0	A'BC'	m2	
0 1 1	A'BC	m3	✓
1 0 0	AB'C'	m4	
1 0 1	AB'C	m5	✓
1 1 0	ABC'	m6	✓
1 1 1	ABC	m7	✓

- The table shows all possible minterms of f, but only m1, m3, m5, m6 and m7 makes up f.
  - Each minterm is used to **detect** a specific code pattern.
- Careful with the ordering of the variables in the functional notation, f(A, B, C)
  - Example, even with the same minterm list, the following functions are not the same
    - f1(A, B, C) = $\sum$ m(1, 3, 5, 6, 7) *(A is the MSB, C is the LSB)*
    - f2(B, C, A) = $\sum$ m(1, 3, 5, 6, 7) *(B is the MSB, A is the LSB)*

Table Representation of a Logic Function

- A truth table is another type of representation of a logic function
  - It relates the inputs to output
  - How?
    - List the evaluated logic function for all the possible input combinations
      - E.g. the truth table for f(A,B,C) = AB + C

Input	Output
A B C	f(A,B,C)
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

- How to get from

  f(A,B,C) = AB + C => truth table?

Solution

Convert non-canonical to canonical form:

f(A,B,C)

= AB + C

= AB(C + C') + (A'B' + A'B + AB' + AB)C

= ABC + ABC' + A'B'C + A'BC + AB'C + ABC

= ABC + ABC' + A'B'C + A'BC + AB'C

- Minterm info can be directly transfer to truth table

- From the truth table
  - f(A, B, C) = $\sum$ m(1, 3, 5, 6, 7)
- Let's list out f' as well:
  - f'(A, B, C) = $\sum$ m(0, 2, 4)

  Each minterm will either appear in f or f'.
  - Meaning, f + f' = 1
    - ORing all minterms will yield a 1.

Optimization of Logic Function using Algebra Method

- Logic optimisation
  - Reduce redundant product or sum terms and literals.

- Let's use Algebra method to reduce a logic expression.

- Example 1

Reduce the canonical SOP of logic function f shown in the table to a simpler version.

Input	Output
A B C	f(A,B,C)
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

Solution 1

f = ∑ m(1, 3, 5, 6, 7)

= A'B'C + A'BC + AB'C + ABC' + ABC

= (A'B' + A'B + AB' + AB)C + ABC'

= ((A' + A)(B' + B))C + ABC'

= C + ABC'

= ABC' + C

= AB + C

Optimization of Logic Function using Algebra Method

- To obtain the reduce version of f', can either:
  - Perform the optimization process on f'.
  - Apply DeMorgan's theorem on the reduced f
- The result may or may not be the same.

- Example 2

By optimization process on f,

$f' = \sum m(0, 2, 4)$

$= A'B'C' + A'BC' + AB'C'$

$= (B' + B)\,A'C' + AB'C'$

$= (A' + AB')\,C'$

$= (A' + B')\,C'$

$= A'C' + B'C'$

Inputs A B C	Output f(A, B, C)	Complemented Output f'(A,B,C)
0 0 0	0	1
0 0 1	1	0
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	0

By applying DeMorgan's theorem on f,

$f = AB + C$

$f' = (AB + C)'$

$= (A' + B')\,C'$

$= A'C' + B'C'$ (same as above)

Limitations of Algebraic Method for Logic Optimisation

- Limitation of Boolean algebra method for logic optimization:
  - Solution is not guaranteed to be minimum.
    - Non systematic steps to reach a desired minimum solution.
    - Approach is heuristic.
      - Repeated search is based on intuition and experience.
        - Time consuming and error-prone
  - Impractical for large number of variables since relies heavily on the ability of the designer to use theorems and laws.
    - Slow and error-prone
    - Usage is limited to small number of variables.
  - The expression is often made complex due to expansion before it can be simplified.

- Other better methods that can overcome the limitations of Boolean algebra method
  - Karnaugh Map method
  - Quine-McCluskey method