5. Combinational Logic Circuits

- **The most common type of digital logic circuits**
- **Made up of combinations of logic gates**
- **At any point in time, the output logic level only depends on the present combination of logic levels at the N inputs**
    - $O(t) = f\,[\,I_1(t), I_2(t), I_3(t), \ldots, I_N(t)\,]$
- **It has no memory characteristics, unlike sequential circuits**
- **Relatively easy to analyse and design compared to sequential circuits**

Designing and Implementing a Combinational Logic Circuit

- **The function of a required logic circuit can be fully described by a truth table**
- **To design the circuit, we obtain the Boolean expression from the truth table**
- **The Boolean expression can then be implemented using a proper choice of logic gates**

A Boolean expression is also known as

- a Boolean equation
- a logic function

It fully describes, algebraically, the logic circuit's output in response to every possible input condition.

For simple circuits, the expression can usually be obtained by observation.

Forms of Boolean Expressions

- Canonical Form
  - Sum of minterms expression (SOm)
  - Product of maxterms expression (POM)

- Standard Form
  - Sum of products expression (SOP)
  - Product of sums expression (POS)

- **minterms**:
  - All possible combinations of a given set of Boolean variables formed by the **AND** operation

- **maxterms**:
  - All possible combinations of a given set of Boolean variables formed by the **OR** operation

A logic circuit with 2 inputs X and Y will have these 4 minterms and maxterms:

inputs		minterms		maxterms	
X	Y				
0	0	X’ • Y’	m0	X + Y	M0
0	1	X’ • Y	m1	X + Y’	M1
1	0	X • Y’	m2	X’ + Y	M2
1	1	X • Y	m3	X’ + Y’	M3

A minterm or maxterm uniquely describes the input combination at a given time instant

A logic circuit with 3 inputs X, Y and Z will have these 8 minterms and maxterms:

inputs			minterms		maxterms	
X	Y	Z				
0	0	0	X' • Y' • Z'	m0	X + Y + Z	M0
0	0	1	X' • Y' • Z	m1	X + Y + Z'	M1
0	1	0	X' • Y • Z'	m2	X + Y' + Z	M2
0	1	1	X' • Y • Z	m3	X + Y' + Z'	M3
1	0	0	X • Y' • Z'	m4	X' + Y + Z	M4
1	0	1	X • Y' • Z	m5	X' + Y + Z'	M5
1	1	0	X • Y • Z'	m6	X' + Y' + Z	M6
1	1	1	X • Y • Z	m7	X' + Y' + Z'	M7

For N-inputs, there will be 2^N minterms

e.g. 4 inputs: a, b, c, d

- 13 in decimal = 1101 in binary
- Then minterm m13 = a b c’ d
- maxterm M13 = a’ + b’ + c + d’
- 2 in decimal = 0010 in binary
- Then minterm m2 = a’ b’ c d’
- maxterm M2 = a + b + c’ + d

- **Minterms** are formed such that given a set of input conditions, only the corresponding minterm (but not the other minterms) will **yield a logic 1**.
- Eg. $x=0$, $y=1$, $z=1$ the corresponding minterm is m3, i.e. $x'yz$
- Substituting the values of x, y and z will result in $m3 = x'yz = 0' \bullet 1 \bullet 1 = 1$
- Notice that by arranging x, y, z together to form a 3-bit binary number (MSB=x, LSB=z), the decimal equivalent is used to denote the minterm number ($011_2 = 3_{10}$ in this example)

- **Maxterms** are formed such that given a set of input conditions, <u>only</u> the corresponding maxterm (but not the other maxterms) will **yield a logic 0**.

- eg. $x=1$, $y=0$, $z=1$ the corresponding maxterm is M5, i.e. $x' + y + z'$

- Substituting the values of x, y and z will result in $M5 = x' + y + z' = 1' + 0 + 1' = 0$

- Notice that arranging x, y, z together to form a 3-bit binary number (MSB=x, LSB=z), the decimal equivalent is used to denote the maxterm number ($101_2 = 5_{10}$ in this example)

The Sum of minterms expression

To write the sum-of-minterms Boolean expression from a truth table:

- **For each combination of the input variables that produces a logic 1 in the output, collect the corresponding minterms and OR them together**

The Product of maxterms Expression

- For each combination of the input variables that produces a logic 0 in the output, collect the corresponding maxterms and AND them together
- Conversion between the two forms is easy.
- Sum of minterms expression is associated with active HIGH output.
- Product of maxterms expression is associated with active LOW output.

Example: given the truth table, obtain the SOm and POM expressions for output F

inputs			Output F
X	Y	Z	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

inputs			minterms		maxterms		F
X	Y	Z					
0	0	0	X’ • Y’ • Z’	m0	X + Y + Z	M0	0
0	0	1	X’ • Y’ • Z	m1	X + Y + Z’	M1	1
0	1	0	X’ • Y • Z’	m2	X + Y’ + Z	M2	1
0	1	1	X’ • Y • Z	m3	X + Y’ + Z’	M3	0
1	0	0	X • Y’ • Z’	m4	X’ + Y + Z	M4	1
1	0	1	X • Y’ • Z	m5	X’ + Y + Z’	M5	0
1	1	0	X • Y • Z’	m6	X’ + Y’ + Z	M6	0
1	1	1	X • Y • Z	m7	X’ + Y’ + Z’	M7	1

$$F = X'Y'Z + X'YZ' + XY'Z' + XYZ$$
$$= \Sigma_{XYZ}(1, 2, 4, 7)$$

SOm

$$F = (X+Y+Z)\ (X+Y'+Z')\ (X'+Y+Z')\ (X'+Y'+Z)$$
$$= \Pi_{XYZ}(0, 3, 5, 6)$$

POM

Other ways of writing canonical expressions:

$F = \sum_{XYZ} (1, 2, 4, 7)$ **SOm**

$F(X,Y, Z) = \sum m\, (1, 2, 4, 7)$

$F(X,Y, Z) = m1 + m2 + m4 + m7$

$F = \prod_{XYZ} (0, 3, 5, 6)$ **POM**

$F(X,Y, Z) = \prod M\, (0, 3, 5, 6)$

$F(X,Y, Z) = M0 \bullet M3 \bullet M5 \bullet M6$

Interpretation of active High and active Low for the above example:

Value of Output F	Interpretation of output F	
	There is an **odd** number of 1's among the 3 inputs X, Y, Z	There is an **even** number of 1's among the 3 inputs X, Y, Z
1 (High)	**TRUE**	FALSE
0 (Low)	FALSE	**TRUE**

- **E.g. rename F as ODD, which is active High** (usually write Som expression)
- **E.g. rename F as EVEN*, which is active Low** (usually write PoM expression)

Standard Form of Boolean Expressions

- **SOP and POS**
- **Simplified expressions from the canonical forms**
- **Leads to simpler logic circuits**
- **Known as combinational circuit minimisation**
- **Minimise the number of gates (minumum number of product terms or sum terms)**
- **Minimise the number of inputs on each gate (minimum number of input variables in each product term and sum term)**

Sum of products (SOP) expression

example:

This is a sum-of-minterms expression:

$$f(x, y, z) = xyz' + xyz + x'y'z + xy'z$$

Simplifying, we get

$$\begin{aligned} f(x, y, z) &= xy\,(z' + z) + (x' + x)\,y'z \\ &= xy + y'z \end{aligned}$$

This is now a sum-of-products expression.

A product term need not contain all the input variables, unlike a minterm.

Product of sums (POS) expression

example:
This is a product-of-maxterms expression:

$$f(x, y, z) = (x+y'+z')(x+y'+z)(x'+y'+z)(x'+y+z)$$

Simplifying, we get

$$f(x, y, z) = (x + y')(x' + z)$$

This is now a product-of-sums expression.

A sum term need not contain all the input variables, unlike a maxterm.

These are neither SOP nor POS expressions:

$f = (xy)'z + xz'$ not a product term

$f = xy(x' + z)'$ not a sum term

$f = (xy + z)(x' + y)$ not a sum term

Use the standard form (i.e. SOP or POS) wherever possible.

Obtaining Simplified Standard Expressions from the Canonical Form

- **Different methods for Boolean expression simplification**
  - **Algebraic method**
  - **Karnaugh map (K-map)**
  - **Quine-McCluskey method (Q-M method or tabulation method)**
  - **Heuristic methods, e.g. Espresso-II**

Simplified Boolean expressions yield

- Simpler circuits with fewer logic gates
- Fewer connections
- Lower cost
- Improved reliability

Algebraic method

- Use Boolean theorems
- Requires experience and skills

Examples:

simplify $Z = ABC + AB'(A'C')'$

$$\boxed{Z = AB' + AC}$$

simplify $X = (A' + B)(A + B + D)D'$

$$\boxed{X = BD'}$$

Karnaugh Map

- Graphical method
- Easier to use than algebraic method
- Based on the Boolean theorems

  AB + AB' = A(B + B') = A (for SOP)

  (A+B)(A+B') = A (for POS)

- Truth table gives value of output X for each combination of input values. K-map gives the same info in a different format.

- **K-map squares are labelled such that adjacent squares differ only in one variable.**

- **SOP expression for output X can be obtained by ORing together those squares that contain a 1.**

- **Can also obtain POS expression by ANDing together those squares that contain a 0.**

- **Note the correspondence with SOm (think 1) and POM (think 0).**

Truth Table to K-Map Conversion [2 Inputs]

$X = A'B' + AB$

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

K-map

X	B=0	B=1
A=0	1	0
A=1	0	1

Truth Table to K-Map Conversion [3 Inputs]

$$X = A'B'C' + A'B'C + A'BC' + ABC'$$

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

K-map

X	C=0	C=1
A=0,B=0	1	1
A=0,B=1	1	0
A=1,B=1	1	0
A=1,B=0	0	0

Truth Table to K Map Conversion [4 Inputs]

A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

$$X = A'B'C'D + A'BC'D + ABC'D + ABCD$$

C, D

X (A, B)	0 0	0 1	1 1	1 0
0 0	0	1	0	0
0 1	0	1	0	0
1 1	0	1	1	0
1 0	0	0	0	0

Truth Table to K Map Conversion [4 Inputs]

A	B	C	D	dec
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

C, D

A, B

X	0 0	0 1	1 1	1 0
0 0	0	1	3	2
0 1	4	5	7	6
1 1	12	13	15	14
1 0	8	9	11	10

Kmap: Looping 2 “neighbouring” 1’s

C, D

A, B

X	0 0	0 1	1 1	1 0
0 0	0	1	0	1
0 1	0	1	0	0
1 1	0	1	1	0
1 0	1	0	0	1

A’C’D

B’CD’

AB’D’

ABD

Kmap: Looping 4 “neighbouring” 1’s

C, D

A, B

X	0 0	0 1	1 1	1 0
0 0	1	1	1	1
0 1	0	1	1	0
1 1	0	1	1	0
1 0	1	0	0	1

A’B’

BD

B’D’

Kmap: Looping 8 "neighbouring" 1's

D

X	C, D: 0 0	0 1	1 1	1 0
A, B: 0 0	0	1	1	0
0 1	0	1	1	0
1 1	0	1	1	0
1 0	0	1	1	0

Note these rules on Kmap for simplification:

- Loop 1's to obtain SOP expression.
- Only 2^N number of "neighbouring" 1's can be looped together.
- No looping along diagonal.
- All 1's must be looped.
- Use the biggest loops and the fewest loops.
- A square(s) may be looped more than once.
- Each loop of 1's will yield a product term.

- 0's can also be looped in a similar manner to obtain POS expression.
- Each loop of 0's will yield a sum term.

Example: Simplify the Boolean expression for output Z using K-map

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Example (cont)

Step 1 : Convert truth table to K-map
Step 2 : Loop adjacent 1's to get SOP

	CD			
Z	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	1	1	1	1

AB

BCD

A

$$Z = A + BCD$$

Example (cont)

Alternatively

Step 2 : Loop adjacent 0's to get POS

	CD			
Z	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	1	1	1	1

AB

A+B

A+C

A+D

Z = (A + B)(A+C)(A+D)

Don’t care conditions

- Some logic circuits can be designed such that there are certain input conditions for which there are no specified output levels.

- This is possible because
  - These input conditions will not occur
  - It does not matter whether the output is 0 or 1

- The designer is free to make the output for any “don’t care” condition to be 0 or 1 in order to produce the simplest output expression.

Example: Design a logic circuit whose input is a BCD digit, and whose output goes HIGH if the input is smaller than 6_{10}

A	B	C	D	Z
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

X denotes “don’t care”

Example (cont):

Design 1: SOP

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

	CD			
Z	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

AB

X denotes “don’t care”

Example (cont):

Design 2: SOP

$$Z = \color{red}{A'B'} + \color{blue}{BC'}$$

	CD			
Z	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

AB

X denotes "don't care"

Example (cont):

Design 3: SOP

Z = B'C + A'C'

	CD			
Z	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

AB

X denotes "don't care"

Example (cont):

Design 4: POS

$$Z = (A'+B+C)(A+B'+C')$$

	CD			
Z	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

AB

X denotes "don't care"

Example (cont):

Design 5: POS

$$Z = A' (B'+C')$$

Z	CD 00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

AB

X denotes "don't care"

Don’t cares

- “Don’t cares” can be looped in a similar way on the Karnaugh map.
- When looped with 1’s to write SOP expression, the “don’t care” is treated as 1. Those “don’t cares” not looped are treated as 0.
- Conversely, when looped with 0’s to write POS expression, the “don’t care” is treated as 0. Those “don’t cares” not looped are treated as 1.
- “Don’t cares” should only be looped if it helps to simplify a Boolean expression (i.e. helps to form a bigger loop).

Summary: Designing a Combinational Logic Circuit

1. **From problem specifications, derive the relationship between the output(s) and the inputs. This can be expressed in the form of a truth table.**

2. **Obtain the Boolean expression that relates the desired output to the inputs. It can either be in SOP or POS form, although SOP is usually used.**

3. **Simplify the expression using either algebraic, K-map or QM method.**

4. **Implement the circuit from the simplified expression. Certain restrictions may need to be taken into account, eg. use only 2-input NAND gates.**

Enable/Disable Circuits

Enable

A logic circuit is said to be enabled if the output is allowed to change in response to changes in the inputs.

Disable/Inhibit

A logic circuit is said to be disabled/inhibited if the output is not allowed to change in response to changes in the inputs.
The output is either fixed at 0 (typically for active High output) or 1 (typically for active Low output).

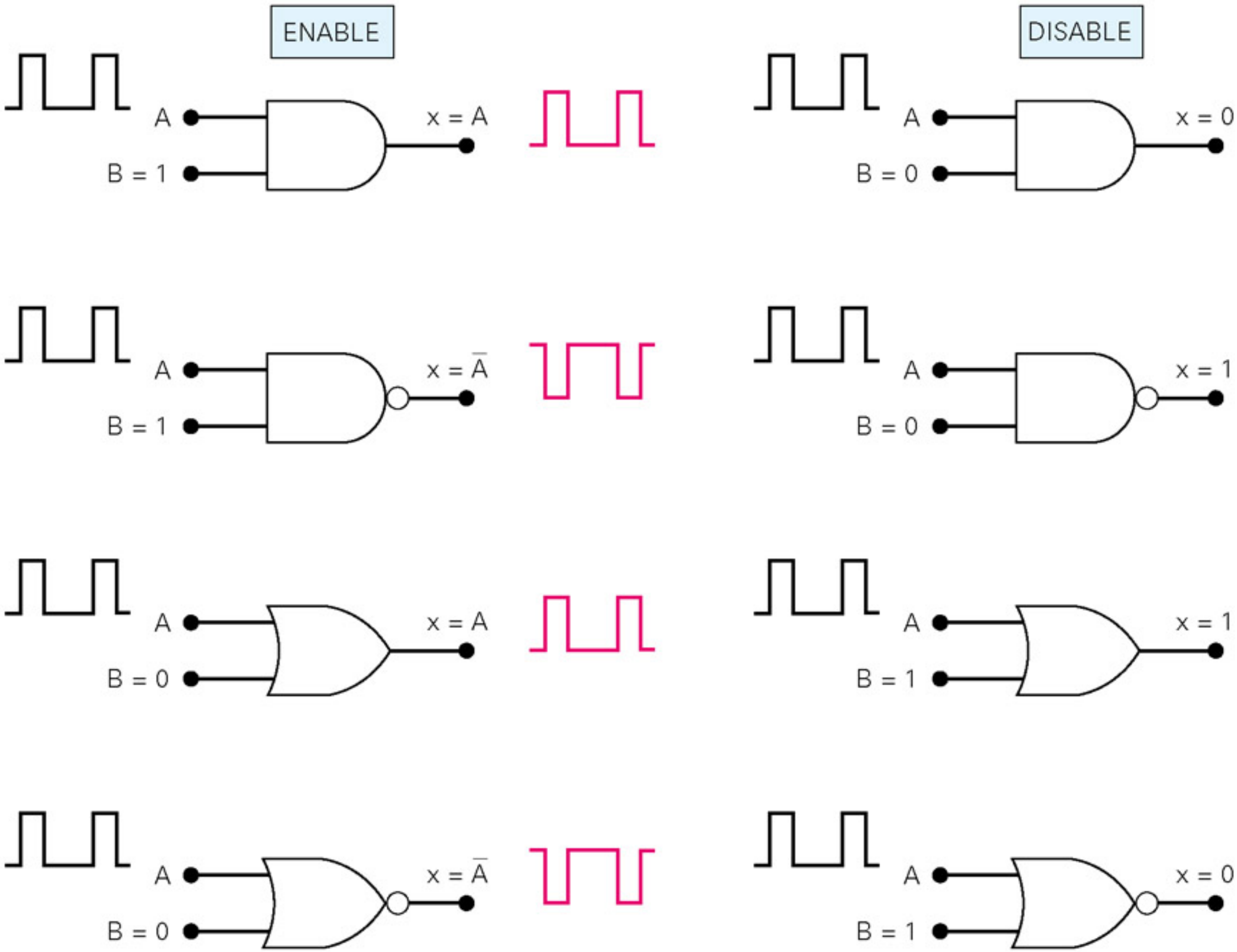

Fig. 4-26 (Tocci 10th ed.)

Enable/Disable Example

Inputs		Output	Effect of output
Child Safe	Unlock	Open	
0	X	0	Door closed
1	0	0	Door closed
1	1	1	Door opened

X: “don’t care”, i.e. can be 0 or 1

- **Circuit is enabled when ChildSafe=1; the output Open changes with input Unlock.**
- **Circuit is disabled when ChildSafe=0; Open is stuck at 0.**