Week 2- T1 2020

Combinational Circuit Analysis

ELEC2141: Digital Circuit Design

UNSW SYDNEY

Summary

Number systems: binary, octal, hexadecimal

Binary codes: BCD, Gray

Logical gates: AND, OR, NOT

Boolean functions and expression simplification

Duality and the complement of a function

Overview

Standard/canonical form of Boolean expression

Minterms and maxterms

Sum of products/product of sums

Gate input cost

Karnaugh maps

Multi-level optimization

NAND/NOR gates

3 state buffers

Reading: Mano - Chapter 2, 2.3-2.7

UNSW SYDNEY

Standard forms

Standard forms are standard ways to express Boolean functions

They contain product terms (Minterms) and sum terms (Maxterms)

They result in more desirable expression for circuit implementation

Minterms are AND terms with every variable present in either true or complement form

Maxterms are OR terms with every variable present in either true or complement form

Examples

Minterms

A function with *n* variables has 2^n minterms

The literals are listed in the same order for all minterms (usually alphabetically)

X	Y	Z	Product Term	symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\bar{X}\bar{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{X}Y\bar{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\bar{Y}\bar{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\bar{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\bar{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

UNSW SYDNEY

Minterms

A variable in a minterm is complemented for 0 and is not complemented for 1

The symbol index corresponds to the binary combination of the variables

X	Y	Z	Product Term	symbol	m_o	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	m_o	1	0	0	0	0	0	0	0
0	0	1	$\bar{X}\bar{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{X}Y\bar{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\bar{Y}\bar{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\bar{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\bar{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

UNSW SYDNEY

Sum-of-minterms canonical form

Function expressed as a logical sum (OR) of minterms

Sum of all minterms where the function value is 1

Also written as

$$F(X,Y,Z) = \sum m(0,2,5,7)$$

Example

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

UNSW SYDNEY

Maxterms

A function with *n* variables has 2^n maxterms

The literals are listed in the same order as for the minterms with a corresponding maxterm

X	Y	Z	Sum Term	symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

UNSW SYDNEY

Maxterms

A variable in a maxterm is complemented for 1 and is not complemented for 0

The symbol index corresponds to the binary combination of the variables

X	Y	Z	Sum Term	symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

UNSW SYDNEY

Product-of-maxterms canonical form

Function expressed as a logical product (AND) of all maxterms

Product of all maxterms where the function value is 0

Also written as

$$F(X,Y,Z) = \prod M(1,3,4,6)$$

Example

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

UNSW SYDNEY

Problems

Find the sum of minterms and product of maxterms canonical form of

X	Y	Z	G
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

X	Y	Z	H
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Minterms and maxterms relationship

From DeMorgan's theorem

$$m_i = \overline{M_i}$$

X	Y	Z	Product Term	Sum Term
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	$\boldsymbol{X+Y+Z}$
0	0	1	$\bar{X}\bar{Y}Z$	$\boldsymbol{X+Y+\bar{Z}}$
0	1	0	$\bar{X}Y\bar{Z}$	$\boldsymbol{X+\bar{Y}+Z}$
0	1	1	$\bar{X}YZ$	$\boldsymbol{X+\bar{Y}+\bar{Z}}$
1	0	0	$X\bar{Y}\bar{Z}$	$\boldsymbol{\bar{X}+Y+Z}$
1	0	1	$X\bar{Y}Z$	$\boldsymbol{\bar{X}+Y+\bar{Z}}$
1	1	0	$XY\bar{Z}$	$\boldsymbol{\bar{X}+\bar{Y}+Z}$
1	1	1	XYZ	$\boldsymbol{\bar{X}+\bar{Y}+\bar{Z}}$

Function complements

The complement of a function expressed as a sum of minterms is constructed by selecting the minterms missing in the expression

Alternatively, the complement of a function expressed as a sum of minterms is simply the product of maxterms with the same indices

UNSW SYDNEY

Function complement example

Find complement expressions for the function:

$$G(X,Y,Z) = \sum m(1,3,5,7)$$

As a sum of minterms:

As a product of maxterms:

X	*Y*	*Z*	*G*
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	0
1	1	1	1

Problems

Find the complement expression for *G* and *H*

X	Y	Z	G
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

X	Y	Z	H
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Sum-of-Products (SOP)

In Sum of Products (SOP) form, equations are written as an OR of AND terms

Similar to sum of minterms but does not need to contain all variables in every term

Simplified form of canonical sum of minterms

Can be directly implemented as a two-level circuit

Sum-of-Products implementation

SOP example:

$$F = \bar{Y} + \bar{X}Y\bar{Z} + XY$$

Product-of-Sums (POS)

In Product of Sums (POS) form, equations are written as an AND of OR terms

Simplified form of canonical sum of maxterms

Can also be implemented as a two-level circuit

POS example:

$$F = X(\bar{Y} + Z)(X + Y + \bar{Z})$$

Cost criteria

Need a method to measure the complexity of a logic circuit

The *Gate-Input Cost (GIC)* is defined as the number of inputs to the gates in the implementation

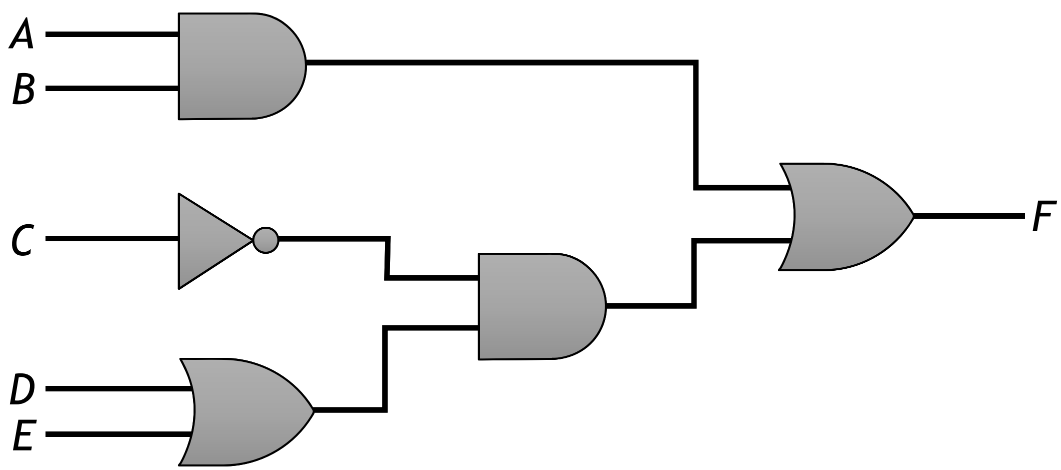
Can also be calculated directly from the equation corresponding to the circuit

- Here only count distinct terms and complemented literals

Gate-input cost

Example:

$$F = AB + \bar{C}(D + E)$$



Gate-input cost

It can also be found from a SOP/POS equation. The GIC is the sum of

1. All literal appearance(s)
2. The number of terms (excluding single literals)
3. The number of distinct complemented single literals

Examples

$$F = ABCD + \bar{A}\bar{B}\bar{C}\bar{D}$$

Examples

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

Examples

$$H = AC\overline{D} + ABD + \bar{A}\bar{B}C$$

Problems

$$F = \bar{A}\bar{B}\bar{C}\bar{D} + AB + AC$$

$$G = (\bar{Y} + X)(\bar{Y} + Z)(\bar{X} + \bar{Y})$$

Two-level circuit optimization

Boolean functions directly dictates the logic circuit implementation

It is important to device a way that leads to the simplest logic circuit implementation

Simplification can lead significant cost and performance improvements

Although Boolean expressions can be simplified by algebraic manipulation, the procedure is awkward and it is hard to confirm if the simplest expression is obtained

A map method is often used to arrive at optimized Boolean equations

Karnaugh map or K-map

The map method is commonly referred as Karnaugh map or K-map

K-map provides systematic and straight forward procedure to simplify Boolean functions up to four variables

Consists of squares each of which corresponds to each minterm (a row of a truth table) of the Boolean function

The Boolean function is identified in the map by those squares for which it has value 1

Two variable K-map

The optimized expression obtained from K-map is always expressed in sum of products or product of sums forms

Thus, the expression is realizable using two-level logic circuit implementation

Two-variable K-map:

$Y \backslash X$	0	1
0	m_0	m_2
1	m_1	m_3

Three variable K-map

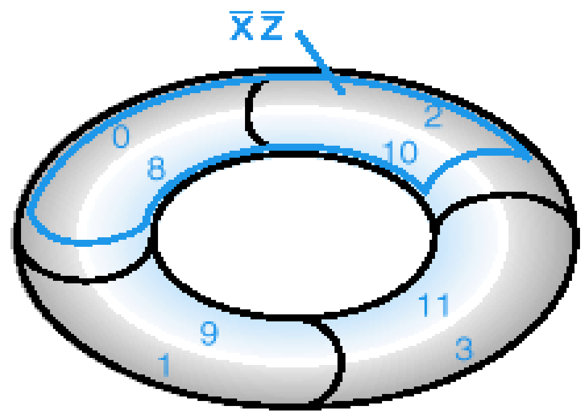
$Z \backslash XY$	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

Adjacent squares to east, west, north and south of a given square represent binary combinations that differ by only one bit, i.e. gray code format

Two squares can be adjacent even if they do not physically appear adjacent in the K-map, e.g. m_0 and m_2, m_4 and m_6 in a three variable K-map

Four variable K-map

YZ \ WX	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}



Minterms are adjacent if differ only in one literal, i.e. complemented in one and uncomplemented in another

m_o, m_4, m_{12}, m_8 are adjacent to m_2, m_6, m_{14}, m_{10}
m_o, m_1, m_3, m_2 are adjacent to m_8, m_9, m_{11}, m_{10}

Two adjacent squares (minterms) can be combined to form a product term with one less variable

Using K-maps

1. Enter the function on the K-map

The function may be given in the form of truth table, a sum of minterms or a SOP expression

Done by placing '1' in the squares where the function has a logical '1' for the corresponding minterm

Example 1: $F = \sum m(0,1,3)$ or $F = \bar{A} + AB$

Using K-maps

2. Identify collections of squares on the K-map to be considered in the simplified expression

Each collection of squares with logical 1s forms a rectangle that represents a simplified expression

The goal is to find the fewest of such rectangles that cover all the squares marked with 1s

Example 1: $F = \sum m(0,1,3)$ or $F = \bar{A} + AB$

Using K-maps

3. Obtain the SOP expression corresponding to the constructed rectangles in the map

Example 2: $G = \sum m(1,2)$

Using K-maps

Example 3: $F(A,B,C) = \sum m(0,1,2,3,4,5)$

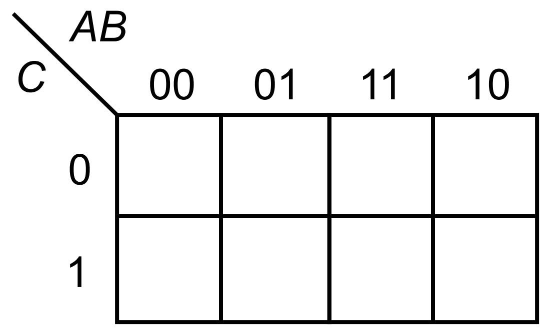
$C \backslash AB$	00	01	11	10
0				
1				

Example 4: $G(A,B,C) = \sum m(0,2,4,5,6)$

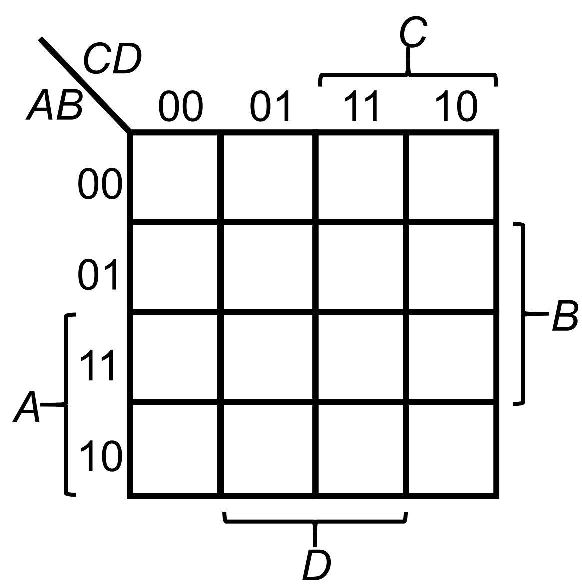
$C \backslash AB$	00	01	11	10
0				
1				

Using K-maps

Example 5: $H(A,B,C) = \sum m(1,3,4,5,6)$

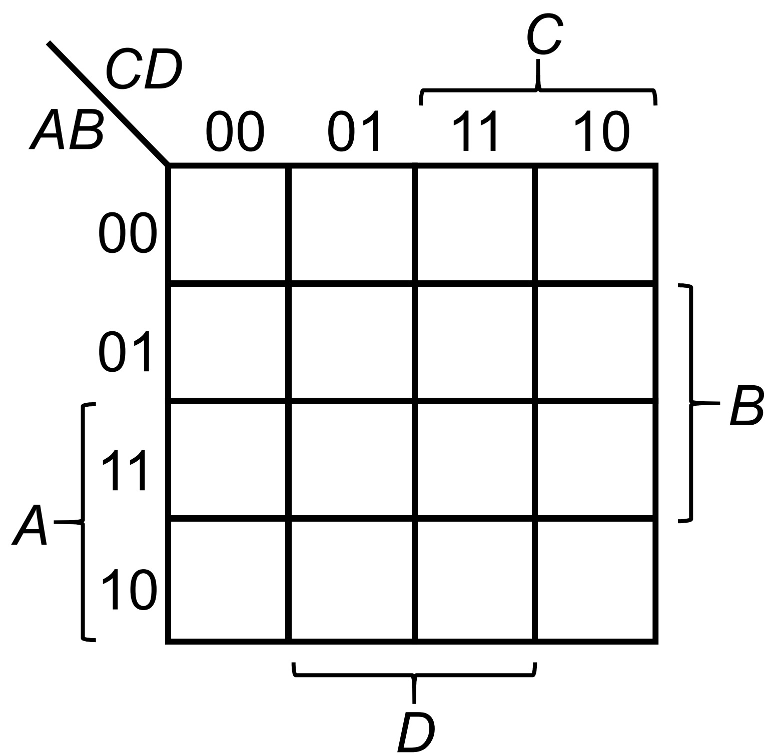


Example 6: $F(A,B,C,D) = \sum m(0,1,2,4,5,6,8,9,10,12,13)$



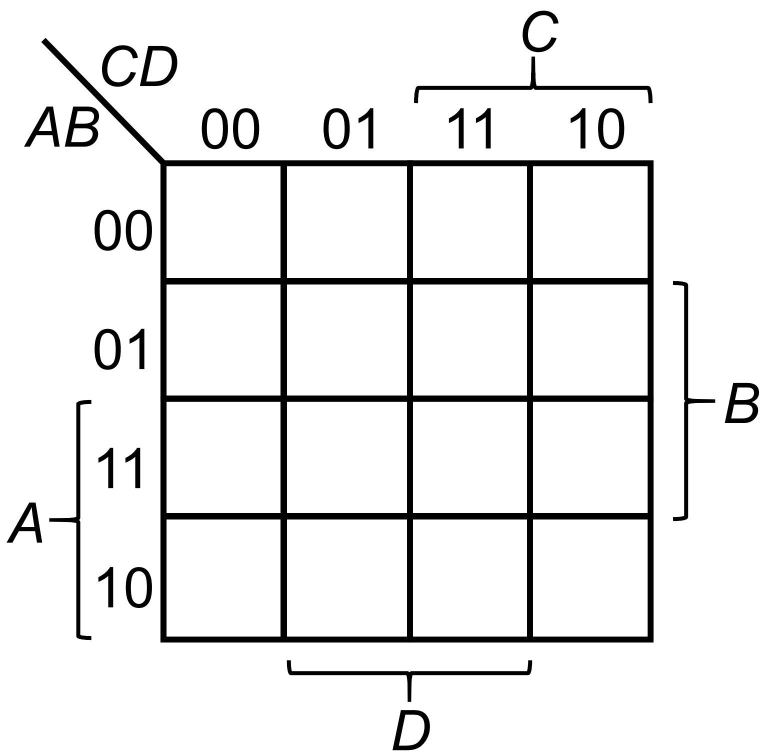
Using K-maps

Example 7: $G(A,B,C,D) = \bar{A}\bar{C}\bar{D} + \bar{A}D + \bar{B}C + CD + A\bar{B}\bar{D}$



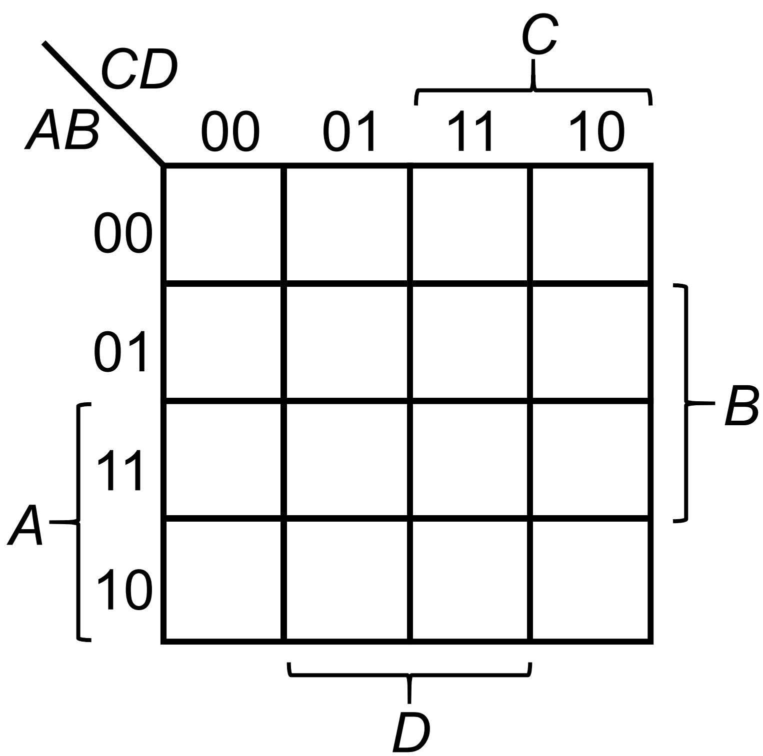
Problem

$$F(A,B,C,D) = \sum m(0,2,8,9,10,15)$$



Problem

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



Terminology

Implicant: A product term where the function has the value 1 for all minterms in that product term

In other words, all rectangles on a map made up of squares containing 1s correspond to implicants

Prime implicant: If the removal of any literal from an implicant P results in a product term that is not an implicant of the function

In other words, prime implicant corresponds to a rectangle containing as many squares as possible

Essential prime implicant: If a miniterm of a function is only included in one prime implicant

In other words, an essential prime implicant is a prime implicant containing at least a square with a 1 not included in any other prime implicants.

Map manipulation

Example 1:

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	0	0	1
10	0	0	0	0

Example 2:

AB \ CD	00	01	11	10
00	1	0	0	0
01	0	1	0	1
11	1	1	1	0
10	0	0	1	1

Selection rule

Selection rule: Minimize the overlap among prime implicants as much as possible.

Example: $F(A,B,C,D) = \sum m(0,1,2,4,5,10,11,13,15)$

AB \ CD	00	01	11	10
00	1	1	0	1
01	1	1	0	0
11	0	1	1	0
10	0	0	1	1

Optimization

Find *all* prime implicants

Include *all* essential prime implicants in the solution

Include other prime implicants to cover all minterms not yet covered

Try to minimize the overlap between the prime implicants

Optimization example

Example: $F(A, B, C, D) = \sum m(0,3,5,9,10,14,15)$

$AB \backslash CD$	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	0	0	1	1
10	0	0	0	1

Problem

Simplify $F(A,B,C,D) = \sum m(0,2,4,5,10,11,13,15)$

AB \ CD	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	0	0	1	1
10	0	0	0	1

Product-of-sums optimization

Find a SOP expression for the complement of the function by grouping 0's instead of 1's

Invert the function back and apply DeMorgan's Theorem

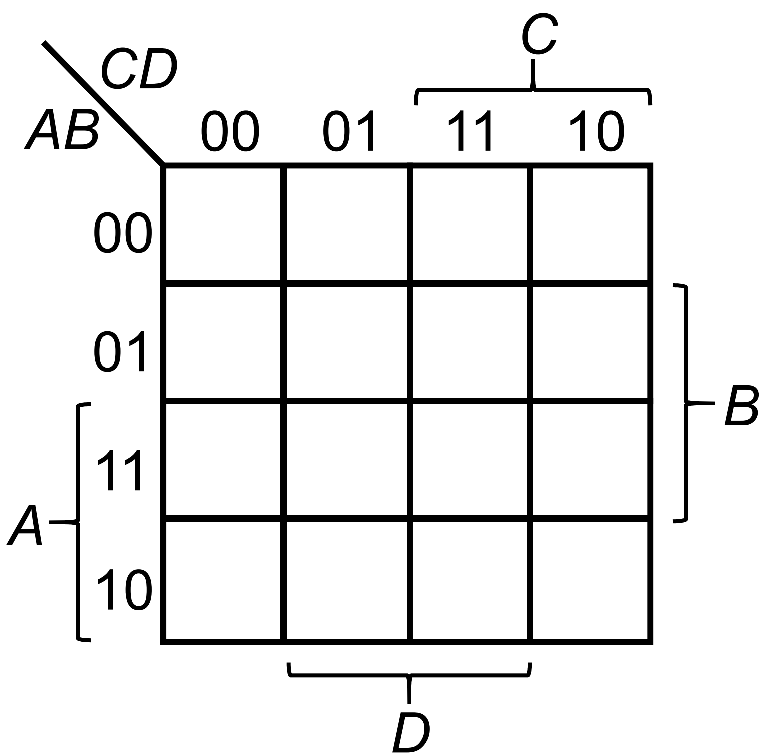
POS optimization example

Example: $F(A,B,C,D) = \sum m(0,1,2,5,8,9,10)$

AB \ CD	00	01	11	10
00	1	1	0	0
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Problem

$$G(A,B,C,D) = \prod M(0,1,4,8,9,12,15)$$



Don’t care conditions

The output of a Boolean function for a particular combinations of input variable can be unspecified

When a particular input combination never occurs, then output is unspecified. For example, in a BCD code, input combinations from 1010 to 1111 will never occur

This is an example of a case where a particular input combination is expected to occur, but we do not care about its output

The unspecified outputs and their corresponding minterms/maxterms of the function are referred as don’t care conditions

Simplification with don’t cares

Don’t care conditions can be used on a map to provide further simplification of the function.

Don’t care conditions are represented by “X” in K-maps or truth tables

In choosing adjacent squares to simplify the function in a map, the don’t care minterms/maxterms may be used with either 1’s or 0’s squares.

Don't care example

Example: $F(A,B,C,D) = \sum m(1,3,7,11,15)$

$$d(A,B,C,D) = \sum m(0,2,5)$$

AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

C B A D

Multiple-level optimization

Multiple-level circuits are circuits with more than two gate levels

Multiple-level circuits can have reduced gate input cost compared to two-level circuits, but tend to have longer propagation delay

Multiple-level optimization is performed by applying transformations to circuits

No systematic procedure exists – may not get optimum solution

Factoring

Factoring is finding a factored form from either SOP or POS expressions

Example: gate input count = 17

$$G = ABC + ABD + E + ACF + ADF$$
$$= AB(C + D) + E + AF(C + D)$$
$$= (AB + AF)(C + D) + E$$
$$= A(B + F)(C + D) + E$$

GIC = 9

Savings of almost half the gate inputs!

UNSW SYDNEY

Decomposition

Decomposition is the expression of a function as a set of new functions

Example: GIC = 26

$$G = A\bar{C}E + A\bar{C}F + A\bar{D}E + A\bar{D}F + BCD\bar{E}\bar{F}$$

Factor first:

$$G = A\left(\bar{C}E + \bar{C}F + \bar{D}E + \bar{D}F\right) + BCD\bar{E}\bar{F}$$
$$= A(\bar{C} + \bar{D})(E + F) + BCD\bar{E}\bar{F}$$

Decomposition

After factoring:

$$G = A(\bar{C} + \bar{D})(E + F) + BCD\bar{E}\bar{F}$$

Define:

$$X_1 = CD \qquad \text{GIC} = 2$$

$$X_2 = E + F \quad \text{GIC} = 2$$

Rewrite *G* as:

$$G = A\overline{X_1}X_2 + BX_1\overline{X_2} \qquad \text{GIC} = 14$$

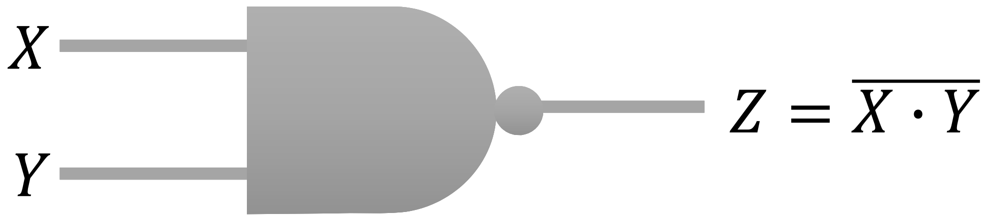
Savings of 12 gate inputs!

Example

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

$$G = (A + B)(C + D)$$

Other gate types – NAND gate



X	Y	$Z = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0

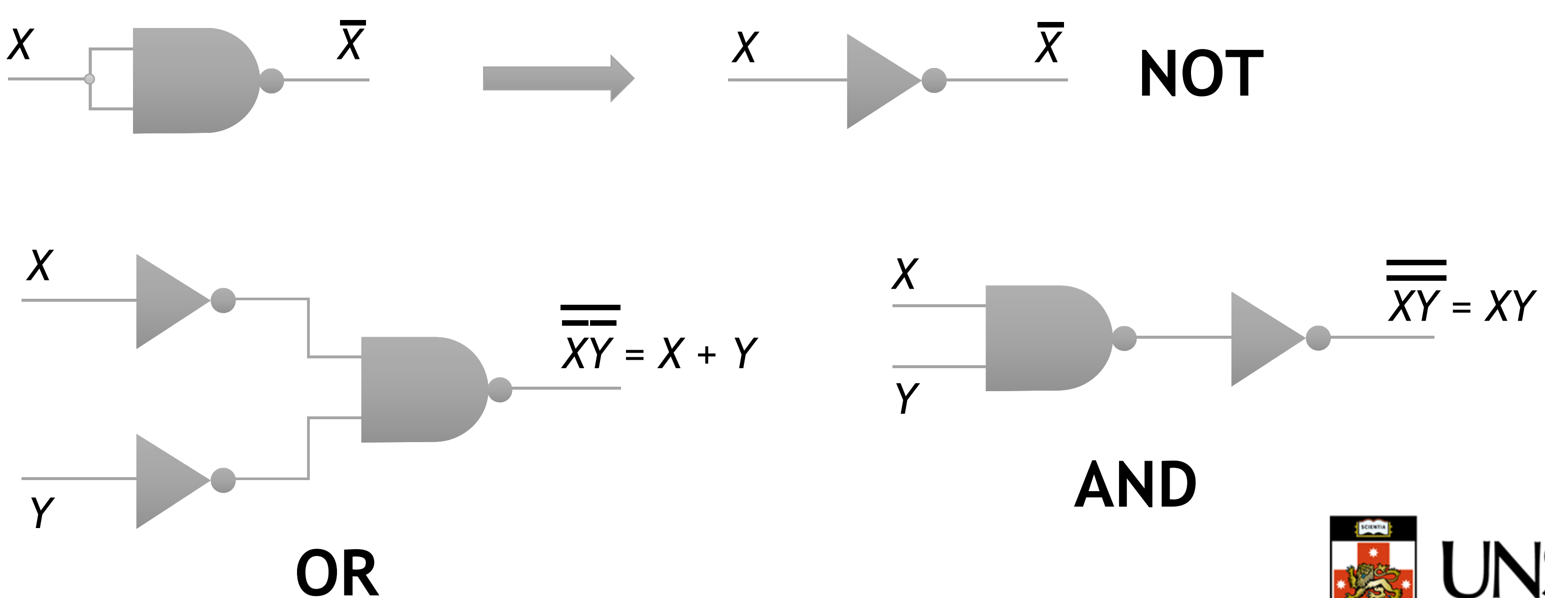
NAND represents NOT-AND, i.e. the AND function with a NOT applied to the result

By applying DeMorgan's Theorem, the NAND function can also be expressed as

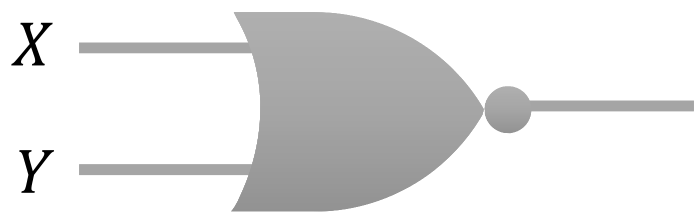
NAND as a universal gate

The NAND gate is the natural implementation for CMOS technology in terms of chip area and speed

The NAND gate is a *universal gate* – a gate type that can implement any Boolean function



NOR gate



$Z = \overline{X + Y}$

X	Y	$Z = \overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0

NOR represents NOT-OR, i.e. the OR function with a NOT applied to the result

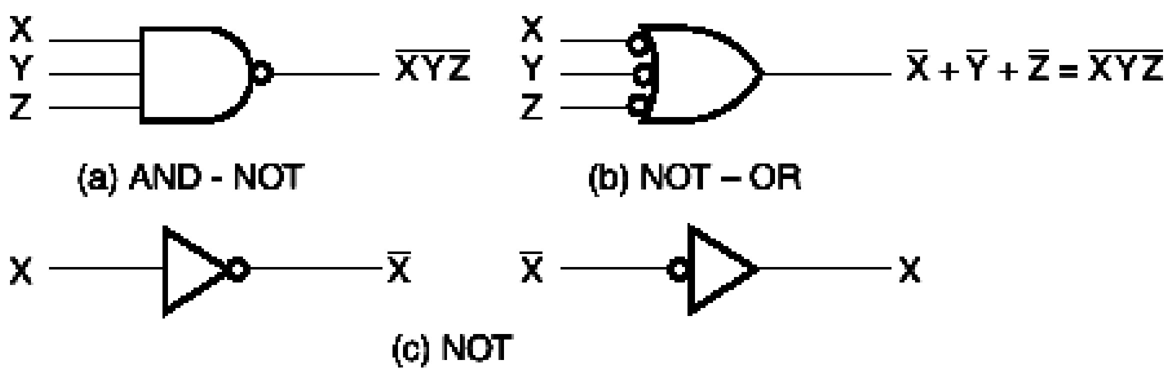
The NOR gate is another *universal gate*

By applying DeMorgan's Theorem, the NOR function can also be expressed as

NAND only implementation

As the NAND gate is a universal gate, all other gates can be replaced with NAND gates to have NAND only implementations

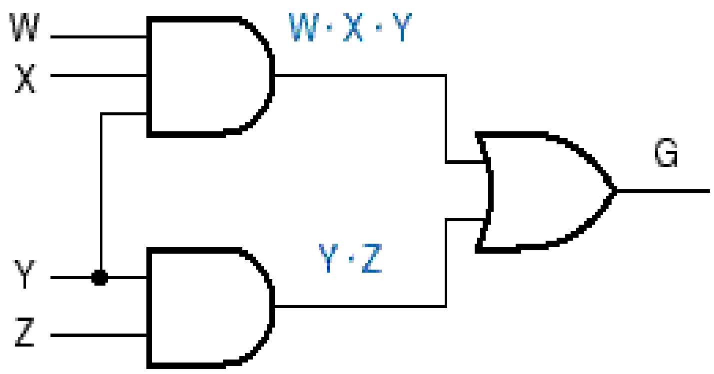
Use the alternative symbols below to change from AND-OR circuit to a NAND circuit



Multi-level NAND circuits

1. Find the simplified SOP circuit
2. Convert all AND gates to NAND gates with AND-NOT graphic symbols
3. Convert all OR gates to NAND gates with NOT-OR graphic symbols
4. Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance

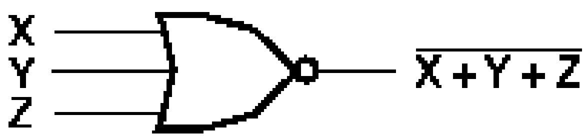
Example



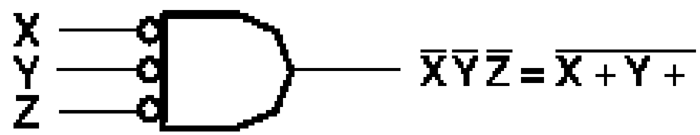
NOR only implementation

Similarly as the NOR gate is a universal gate, all other gates can be replaced with NOR gates to have NOR only implementations

Use the alternate symbols below from AND-OR circuit obtain the NOR only implementation

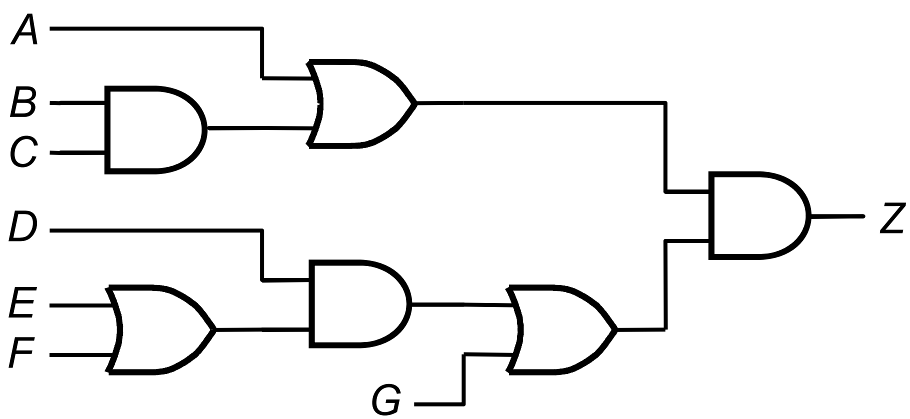


(a) OR – NOT



(b) NOT – AND

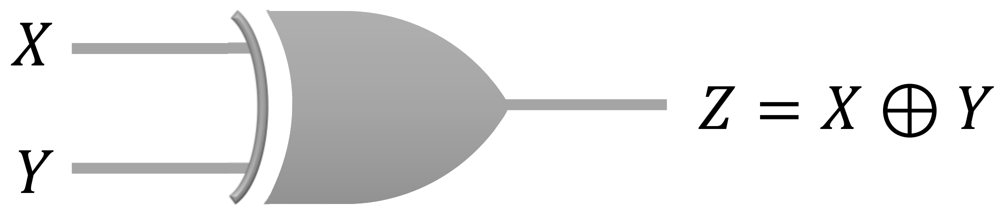
Example



Multi-level NOR circuits

1. Find the simplified SOP circuit
2. Convert all OR gates to NOR gates with OR-NOT graphic symbols
3. Convert all OR gates to NOR gates with NOT-AND graphic symbols
4. Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance

Exclusive-OR gate



X	Y	$Z = X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

The Exclusive-OR (XOR) gate outputs 1 if one of its inputs are 1, but not both

The XOR is denoted by $\oplus$ and is defined as

$$X \oplus Y = \bar{X}Y + X\bar{Y}$$

XOR Identities

The following identities apply to XOR

1. $X \oplus 0 = X$	2. $X \oplus 1 = \bar{X}$
3. $X \oplus X = 0$	4. $X \oplus \bar{X} = 1$
5. $X \oplus \bar{Y} = \overline{X \oplus Y}$	6. $X \oplus \bar{Y} = \overline{X \oplus Y}$

The XOR operation is commutative

$$A \oplus B = B \oplus A$$

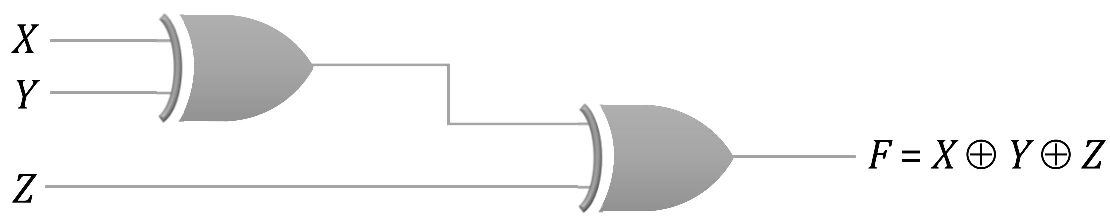
And associative

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Cascading XOR gates

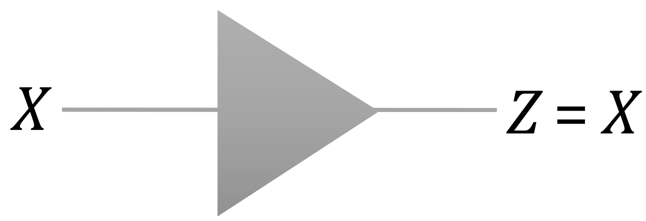
An XOR gate can only have two inputs

The XOR operation on three or more variables is known as an *odd function*, implemented by



Similarly, XNOR operation on three or more variables is known as an *even function* and is implemented by adding an inverter to the output of the odd function

Buffer



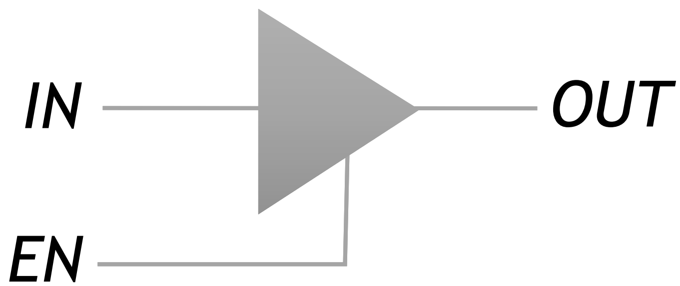
X	$Z = X$
0	0
1	1

A *buffer* is a gate with the function

$$F = X$$

It acts as an electrical amplifier used to improve circuit voltage levels and increase the speed of circuit operation

3-State buffer



EN	IN	OUT
0	X	Z
1	0	0
1	1	1

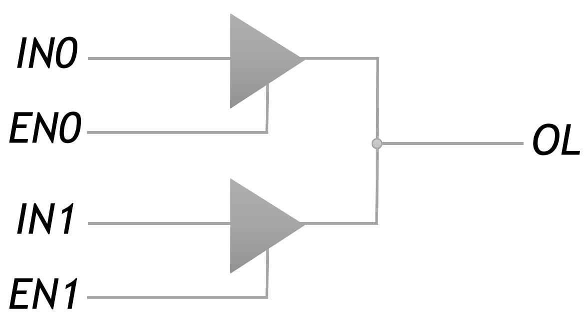
3-State Buffer adds a third logic value – High-Impedance – denoted as *Hi-Z* or just *Z*

The input *EN* (enable) acts as the control line of the buffer

High-impedance outputs

A Hi-Z value behaves as an open-circuit, i.e. the output appears to be disconnected

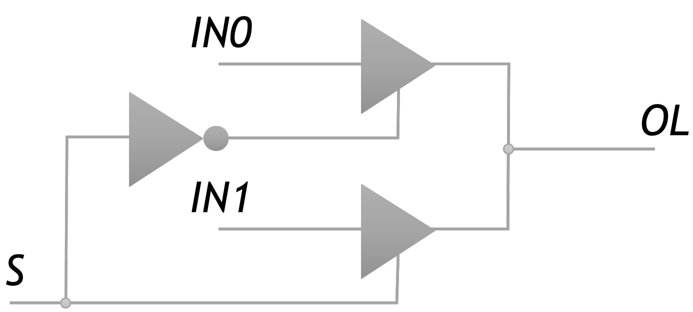
Two or more Hi-Z capable gates can hence be connected to the same output line:



EN1	EN0	IN1	IN0	OL
0	0	X	X	
0	1	X	0	
0	1	X	1	
1	0	0	X	
1	0	1	X	
1	1	0	0	
1	1	1	1	
1	1	0	1	
1	1	1	0	

3-State buffers as a multiplexer

Used to ensure no electric clashes in the circuit



S	IN1	IN0	OL
0	X	0	
0	X	1	
1	0	X	
1	1	X	

This circuit acts as a multiplexer with control input *S* (select)