EES4725
Digital Circuits and FPGA Design

Chua Dingjuan
elechuad@nus.edu.sg

Lecture 2

BOOLEAN ALGEBRA

Postulates, Theorems, Laws, AND, OR, NOT, XOR, Minterm, Maxterm, SOP/POS, CSOP/CPOS

Boolean Algebra

- Boolean algebra is a two-valued type of switching algebra
- Switching algebra represents bistable electrical switching circuits (On or Off)
- There are two main operators (***AND, OR***)
  - Binary operators (two arguments involved)
    - ***AND*** → "**.**"
    - ***OR*** → "**+**"
  - Plus, one unary operator (only one argument involved)
    - ***NOT*** → "$\overline{\ }$" (*Complement* operator represented by an overbar)
- Boolean algebra satisfies six Huntington postulates

The Three Operators in Two-Valued Boolean Algebra (*B={0,1}*)

OR: $A + B$

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

AND: $A \cdot B$

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

NOT: $\overline{A}$

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

$$A = 0 \rightarrow \overline{A} = 1$$
$$A = 1 \rightarrow \overline{A} = 0$$

Priority: NOT has highest precedence, followed by AND and OR

NOT(A · B + C) = NOT((A · B) + C)

Theorems of Boolean Algebra

#	Theorem		
1	$A + A = A$	$A \cdot A = A$	Tautology Law
2	$A + 1 = 1$	$A \cdot 0 = 0$	Union Law
3	$\overline{(\bar{A})} = A$		Involution Law
4	$A + (B + C)$ $= (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	Associative Law
5	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$	De Morgan's Law
6	$A + A \cdot B = A$	$A \cdot (A + B) = A$	Absorption Law
7	$A + \bar{A} \cdot B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$	
8	$AB + A\bar{B} = A$	$(A + B)(A + \bar{B}) = A$	Logical adjacency
9	$AB + \bar{A}C + BC$ $= AB + \bar{A}C$	$(A + B)(\bar{A} + C)(B + C)$ $= (A + B)(\bar{A} + C)$	Consensus Law

Duality (OR and AND, 0 and 1 can be interchanged)

Boolean Functions and Truth Table

- A Boolean function expresses the logical relationship between binary variables.
- It can be evaluated by determining the binary value of the expression for all possible values of the variables

Truth table is a tabular technique for listing all possible combinations of input variables and the values of function for each combination

$$F_1 = A + B$$

A	B	F_1
0	0	0
0	1	1
1	0	1
1	1	1

Example : Prove the De Morgan's Law:

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

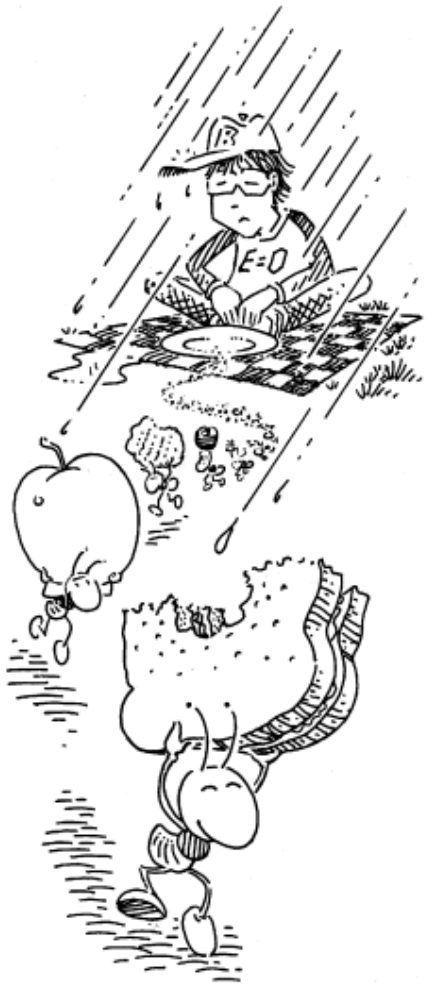
A	B	$\overline{A+B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

A	B	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

But how do we use these ideas?

Design Example



Ben Bitdiddle is having a picnic. He won't enjoy it if it rains or if there are ants. Design a circuit that will output TRUE only if Ben enjoys the picnic.

Solution

Inputs : A (Ants), B (Rain)

Output : E (Ben's Enjoyment)

Boolean Algebra Version

E = ?

Ben enjoys his picnic *if* there is no rain **and** no ants :

E = 1 *if* $A = 0$ **and** $B = 0$

Truth Table Version

A	B	*E*
0	0	1
0	1	0
1	0	0
1	1	0

E = ?

Minterm and Maxterm – cont.

A	B	F	Minterm	Maxterm
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	0	$A \cdot B$	$\bar{A} + \bar{B}$

minterm $\mathrm{A.B} = 1$ for A = 1 and B = 1

maxterm $\bar{A} + \bar{B} = 0$ for A = 1 and B = 1

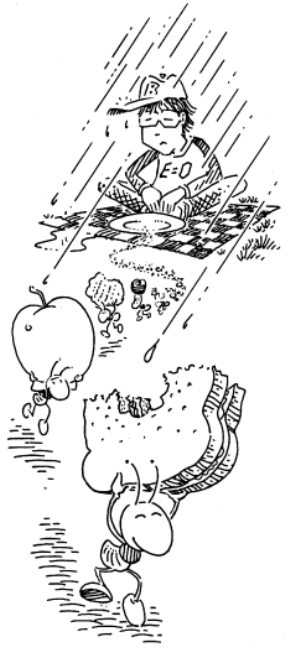
Minterm

- ***Minterm*** is a *product term* that contains all variables in the function
- AND all the variables
- If the variable in truth table is "0", take its complement in the minterm
- Minterm is equal to 1 for that set of given input

Maxterm

- ***Maxterm*** is a *sum term* that contains all variables in the function
- OR all the variables
- If the variable in truth table is "1", take its complement in the maxterm
- Maxterm is equal to 0 for that set of given input

Truth Table → CSOP or CPOS



A Boolean function is in canonical form if it is expressed as

- a ***sum*** of ***minterms*** (Canonical Sum Of Products - CSOP) or
- a ***product*** of ***maxterms*** (Canonical Product Of Sums - CPOS)

Using Ben's Bitdiddle's truth table :

A	B	E	minterm	maxterm
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	0	$A \cdot B$	$\bar{A} + \bar{B}$

A truth table expressed in either CSOP or CPOS. (How should we choose?)

- CSOP → sum the minterms that make output = 1

$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$

- CPOS → product the maxterms that make output = 0

$\boldsymbol{E_2(A,B)} = (A+\bar{B}) \cdot (\bar{A}+B) \cdot (\bar{A}+\bar{B})$

Are the above two functions equivalent? How do we check?

SOP and POS → Truth Table

Are the following two Boolean functions same?

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

Let's use truth tables to check:

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

A	B	E_1
0	0	1
0	1	0
1	0	0
1	1	0

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

A	B	E_2
0	0	1
0	1	0
1	0	0
1	1	0

SOP: If any *PRODUCT* in SOP is "1", the function is "1". Otherwise, the function is "0"

POS: If any *SUM* in POS is "0", the function is "0". Otherwise, the function is "1"

SOP and POS are different ways to present the same Boolean function

Example-1: Non-Canonical → Canonical Form via Truth Table

Example: For the given Boolean function below, find a canonical *minterm* and *maxterm* expression.

1) obtain the truth table from the given function

2) find *minterm* or *maxterm* expression from truth table (CSOP or CPOS)

$F(x, y, z) = \bar{x} + y\bar{z}$ ← Non Canonical form

Truth table:

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Canonical *minterm* expression:

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

(only contains the *minterms* that make the function = 1)

Canonical *maxterm* expression:

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

(only contains the *maxterms* that make the function = 0)

Example-2: Non-Canonical → Canonical Form via Postulates and Theorems

Example: For the given Boolean functions below, convert it to canonical *minterm or maxterm* expression.

(*Using postulates/theorem to expand the given function to canonical form)

SOP → CSOP:
(CSOP – Canonical SOP)

$$F(x,y,z) = \bar{x}y + xz$$
$$= \bar{x}y \cdot 1 + x \cdot 1 \cdot z$$
$$= \bar{x}y(z + \bar{z}) + x(y + \bar{y})z$$
$$= \bar{x}yz + \bar{x}y\bar{z} + xyz + x\bar{y}z$$

For missing literals, complete minterms through postulates:
$A \cdot 1 = A \quad and \quad A + \bar{A} = 1$

SOP → CPOS:
(CSOP – Canonical POS)

1) express SOP as POS
   a) complement twice
   b) apply De Morgan's law
   c) expand
   d) re-apply De Morgan's law
2) for missing literals, complete maxterms through distribution postulate

$$F(x,y,z) = \bar{x}y + xz \quad 1)$$
$$= \overline{\overline{\bar{x}y + xz}} \quad a$$
$$= \overline{(x + \bar{y})(\bar{x} + \bar{z})} \quad b$$
$$= \overline{x\bar{x} + x\bar{z} + \bar{x}\bar{y} + \bar{y}\bar{z}} \quad c$$
$$= (x + y)(\bar{x} + z)(y + z) \quad d$$
$$= (x + y + z) \cdot (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z) \quad 2)$$
$$\cdot (x + y + z) \cdot (\bar{x} + y + z)$$

Use distribution postulate:
$A + (\boldsymbol{BC}) = (A + \boldsymbol{B})(A + \boldsymbol{C})$
(A = incomplete sum,
$\boldsymbol{C}$=NOT($\boldsymbol{B}$)=missing literal)

$x+y = (x + y) + \boldsymbol{z}\bar{\boldsymbol{z}}$
$= (x + y + \boldsymbol{z})(x + y + \bar{\boldsymbol{z}})$ →

Pre-Lab Preparation Q1 Q2

LOGIC GATES

AND, OR, NOT, XOR gates

Outline

Logic gate introduction

- AND/NAND, OR/NOR, NOT/Buffer, XOR/NXOR
- different levels of description (Boolean, truth table, graphical, Verilog)

Implementation of Boolean function using gates

- different levels of description (Boolean, graphical, Verilog)

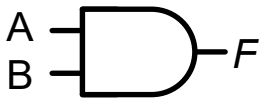
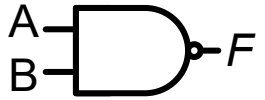
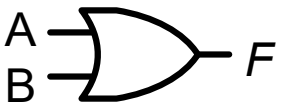
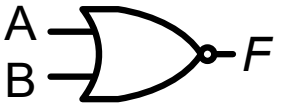
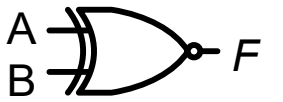
Design simplification by algebra manipulation

Commercial logic gates

Logic Gate Introduction

Logic gates are digital circuits that implement the Boolean operations.

Basic Logic Gates:

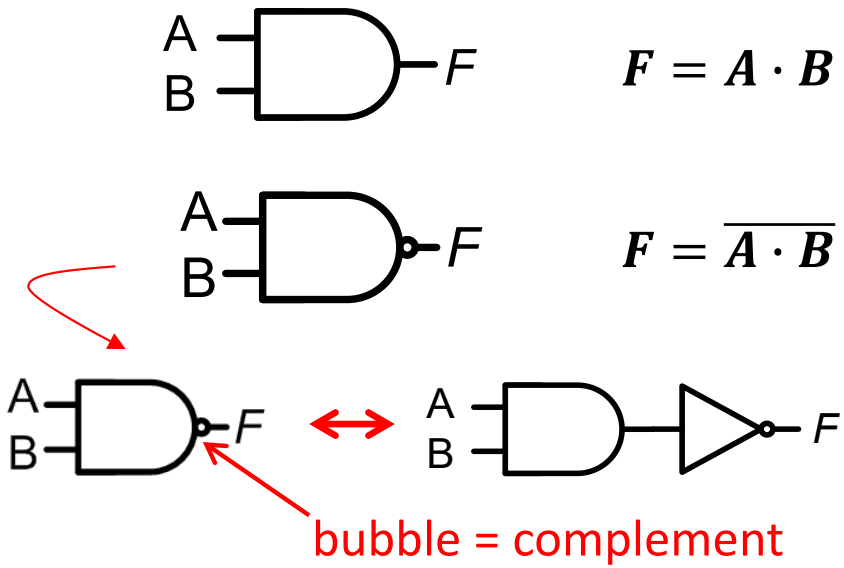
Gate	Symbol	Function (*F*)	Verilog Operator	Gate	Symbol	Function (*F*)	Verilog Operator
AND		$A \cdot B$	F = A & B	NAND		$\overline{A \cdot B}$	F = ~(A & B)
OR		$A + B$	F = A | B	NOR		$\overline{A + B}$	F = ~(A | B)
NOT		$\bar{A}$	F = ~A	Buffer		A	F = A
XOR		$A \oplus B$	F = A ^ B	XNOR		$\overline{A \oplus B}$	F = ~(A ^ B)

Verilog Bit-wise Operator Precedence : ~, &, ^, |

不同取1

不同input. 输出为1

AND and NAND Gates



Truth Table (AND, NAND):

		AND	NAND
A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

AND

F is TRUE only when both ***A*** and ***B*** are TRUE

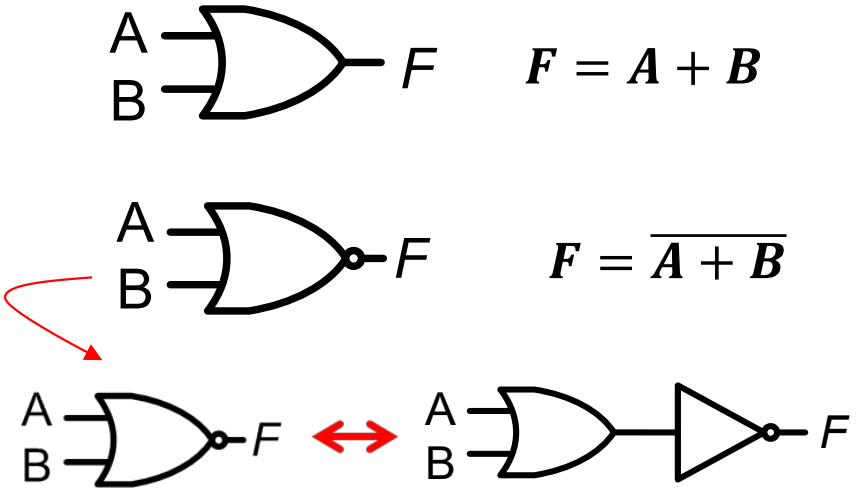
```
module andgate(A, B, F);
    input A, B;
    output F;
    assign F = A & B;
endmodule
```

NAND

- ***F*** is FALSE only if both ***A*** and ***B*** are TRUE

```
module nandgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A&B)
endmodule
```

OR and NOR Gates



Truth Table (OR, NOR):

		OR	NOR
A	B	$A + B$	$\overline{A + B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

OR

- ***F*** is FALSE only when both ***A*** and ***B*** are FALSE

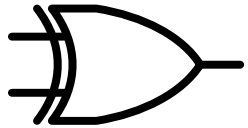
```
module orgate(A, B, F);
    input A, B;
    output F;
    assign F = A | B;
endmodule
```

NOR

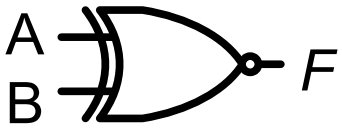
- ***F*** is TRUE only if both ***A*** and ***B*** are FALSE

```
module norgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A | B);
endmodule
```

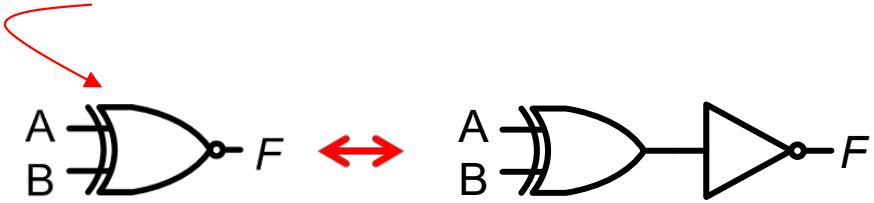
XOR and XNOR Gates



$$F = A\overline{B} + \overline{A}B = A \oplus B$$



$$F = \overline{A \oplus B}$$



Truth Table (XOR, XNOR):

		XOR	XNOR
A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	0	1

XOR

- ***F*** is TRUE if $A \neq B$

XNOR

- ***F*** is TRUE if $A = B$

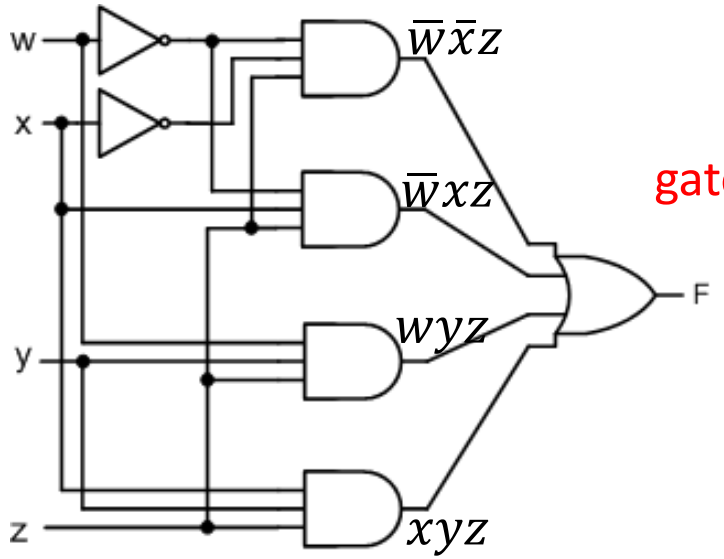
```
module xorgate(A, B, F);
      input A, B;
      output F;
      assign F = A ^ B;
endmodule
```

```
module xnorgate(A, B, F);
      input A, B;
      output F;
      assign F = ~(A ^ B);
endmodule
```

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(w,x,y,z) = \bar{w}\bar{x}z + \bar{w}xz + wyz + xyz$$



gate count = 7

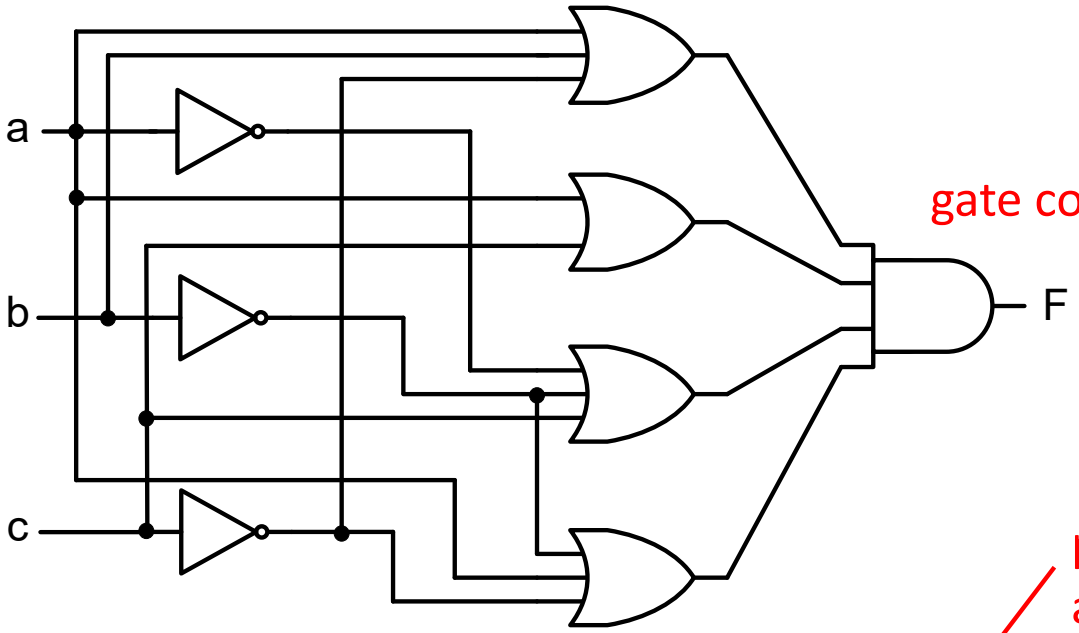
parentheses (~w & x & z) not needed in SOP, as precedence order is ~, &, ^, |

```
module func(w,x,y,z,F);
      input w, x, y, z;
      output F;
      assign F = ~w & ~x & z | ~w & x & z | w & y & z | x & y & z;
endmodule
```

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c) = (a + b + \bar{c})(a + c)(\bar{a} + \bar{b} + c)(a + \bar{b} + \bar{c})$$



gate count = 8

parentheses (~a | ~b | c) needed in POS, as precedence order is ~, &, ^, |

```
module func(a,b,c,F);
    input a, b, c;
    output F;
    assign F = (a | b | ~c) & (a | c) & (~a | ~b | c) & (a | ~b | ~c);
endmodule
```

Pre-Lab Preparation Q3

Boolean Function Simplification using Algebra Manipulation

- To reduce the hardware cost, the Boolean function can be simplified before implemented using logic gates
- A simplified Boolean Function contains a minimal number of terms such that no other expression with fewer literals and terms will represent the original function
- Simplification can be done by
  - Algebra manipulation using postulates and theorem
  - Karnaugh Map

假设

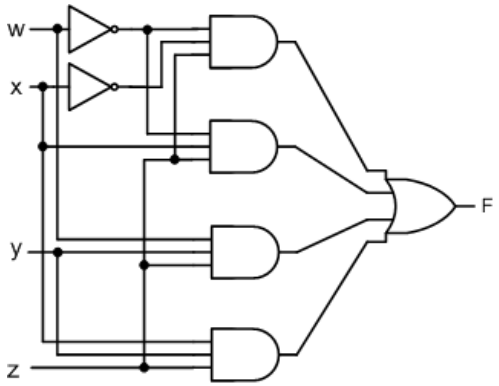
Boolean Function Simplification

(Relook at the first examples on Slide 7):

$$F(x,y,z) = \bar{w}\bar{x}z + \bar{w}xz + xyz + wxy$$
$$= \bar{w}z(\bar{x} + x) + w(xy) + z(xy) \quad \leftarrow (A + \bar{A} = 1)$$
$$= \bar{w}z + w(xy) + z(xy) \quad \leftarrow (A + \bar{A} = 1)$$
$$= \bar{w}z + wxy \quad \leftarrow (AB + \bar{A}C + BC = AB + \bar{A}C)$$ - consensus

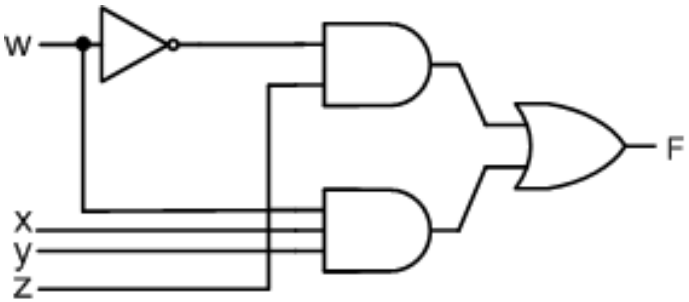
一次

Before simplification:



gate count = 7

After simplification:



gate count = 4

(43% reduction!)

Some Guidelines for Simplification of Boolean Function (in SOP)

Three most used theorems:

(1) $AB + A\bar{B} = A$ (Logical adjacency)

(2) $A + \bar{A} \cdot B = A + B$

(3) $AB + \bar{A}C + BC = AB + \bar{A}C$ (Consensus)

Apply (1) until it cannot be applied further

Apply (2) until it cannot be applied further

Go back to (1) and then (2) until they can no longer be applied

Apply (3) until it cannot be applied further

Go back to (1), (2) and then (3) until none of them can be applied

It can then be assumed that the function is simplified

Empirical: the result is usually close to minimal, but may not be the minimal

Cumbersome: other methods are much easier and quicker

LOGIC MINIMIZATION

Karnaugh Maps

Karnaugh Map (K-Map)

K-map is a diagram that consists of a number of squares

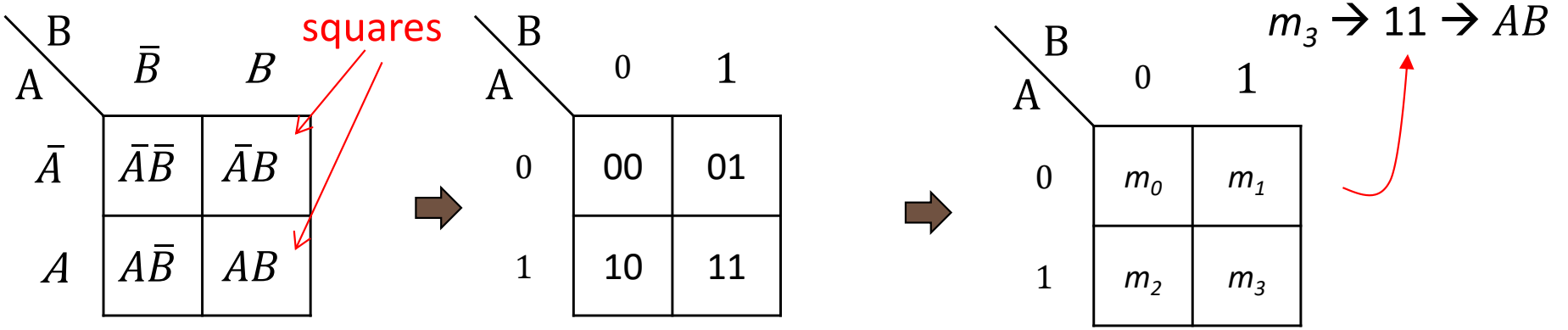
Each square represent one minterm (or maxterm) of a Boolean function

The Boolean function (SOP) can be expressed as a sum of minterms in the map

n-variables Boolean function has maximum 2^n minterms

Two-variable K-map:
(maximum 4 minterms)

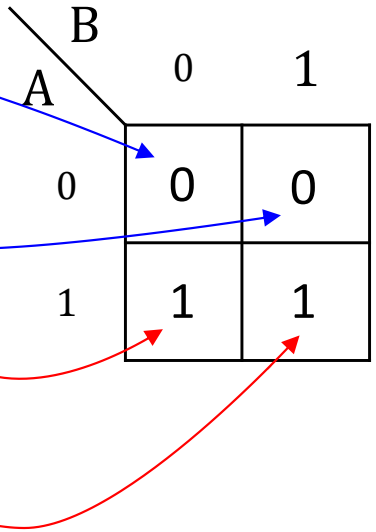
$m_0 \rightarrow 00 \rightarrow \overline{A}\,\overline{B}$
$m_1 \rightarrow 01 \rightarrow \overline{A}\,B$
$m_2 \rightarrow 10 \rightarrow A\,\overline{B}$
$m_3 \rightarrow 11 \rightarrow AB$



"0" → Literal **with** overbar
"1" → Literal **without** overbar

Truth table → K-map

A	B	F
0	0	0
0	1	0
1	0	1
1	1	1



K – map is a two-dimensional truth table

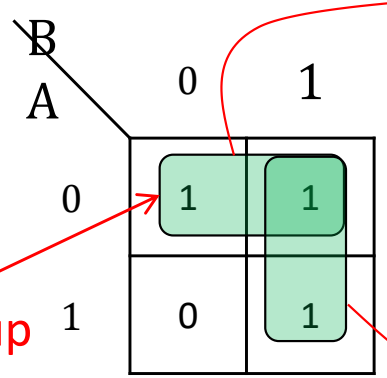
Each row of in truth table corresponds to one square in the k-map

If the term in a row is a *minterm* of the function (*F=1*), place a “1” in the corresponding square of the K-map, otherwise (*maxterm*), place a “0”.

Boolean function simplification using K-map

Boolean function (SOP) simplification using K-map

Simplify: $F = \bar{A}B + AB + \bar{A}\bar{B}$



2-cell group

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

eliminated

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

eliminated

MSOP:

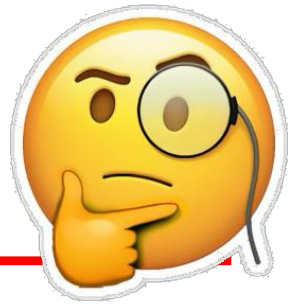
$F = \bar{A} + B$

Alternatively,

$$F = \bar{A}B + AB + \bar{A}\bar{B}$$
$$= \bar{A} + AB$$
$$= \bar{A} + B$$

*The variable that changes value in the group is eliminated, or the variable that doesn't change value in the group remains

Minimization (MSOP) using K-Map (cont.)



***** K-Map Grouping Rules *** :**

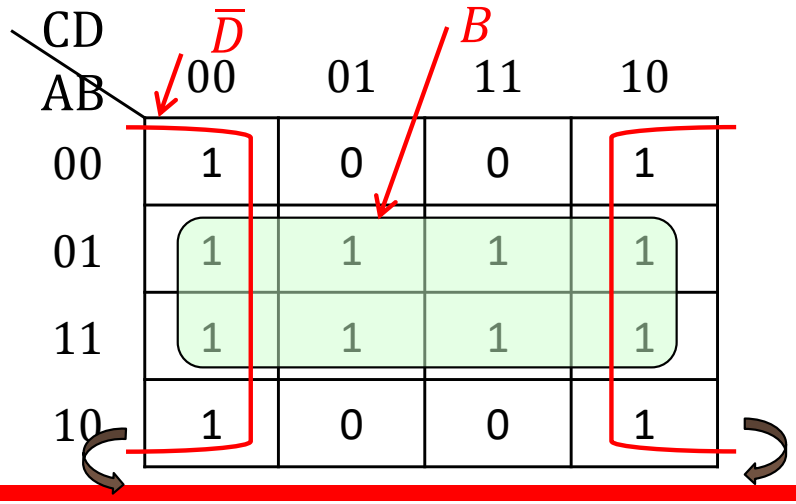
- Group all squares that contains "1".
- Groups must be either horizontal or vertical (diagonal is invalid), but can wrap around edges of the map.
- Group size is always 2^n , that is, 2, 4, 8, …
- Group should be as large as possible (contains as many as squares with "1" as possible)
- Simplified term retains variables that don't change value.
- A 1 in may be circled multiple times if doing so allows fewer circles to be used.
- Minimum essential number of groupings to cover all the "1"s.

Four-variables:

$$F = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D$$
$$+\bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D$$
$$+ABCD + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$$

$$\downarrow$$

$$F_{MSOP} = B + \overline{D}$$



Application Time!

Three-variables:

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

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

$F_{MSOP} = \bar{A} + B\cdot\bar{C}$

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

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

$F_{MSOP} = \bar{B} + \bar{A}B\cdot\bar{C} + A\cdot\bar{C}$

Four-variables:

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

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

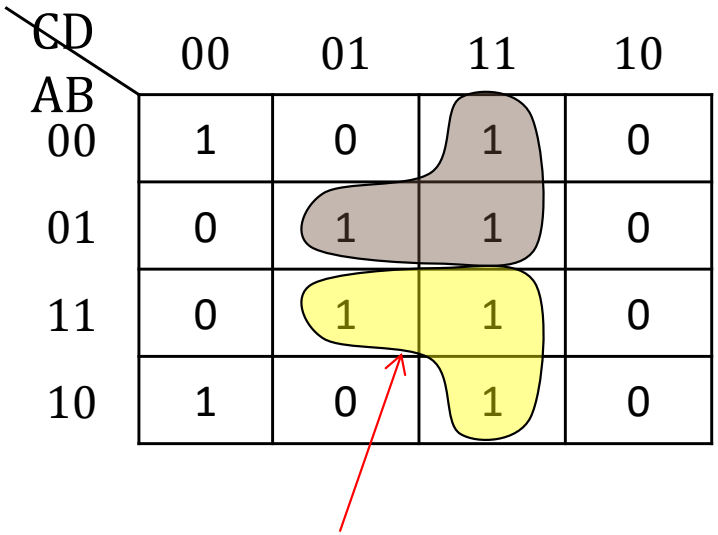
$F_{MSOP} = BD + C\bar{D} + \bar{B}\bar{C}\bar{D}$

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

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

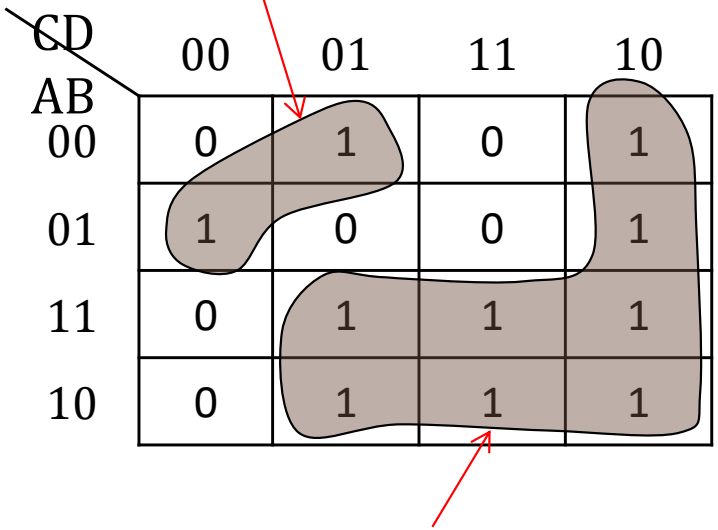
$F_{MSOP} = BD + \bar{B}\cdot\bar{D}$

Invalid groupings



Squares in the group are not in power of two

Two variable change value



not horizontal or vertical

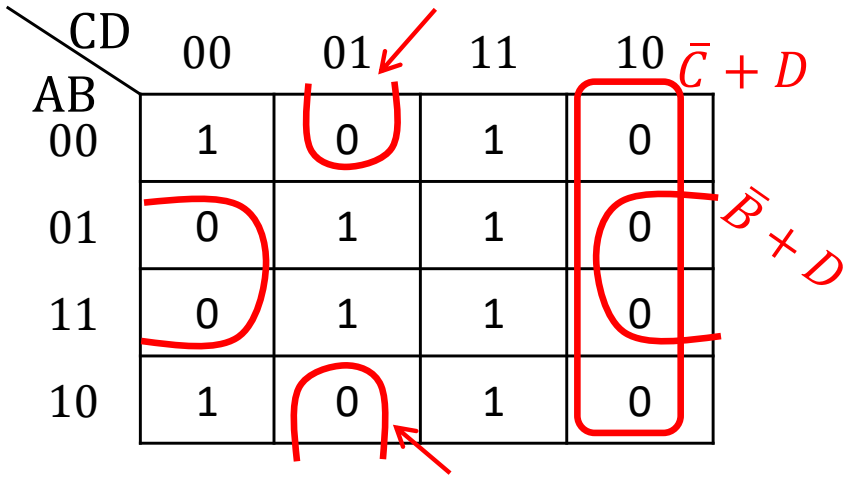
Minimization (MPOS) using K-Map

Boolean function in POS:

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

$$F_{MPOS} = (B + C + \bar{D}) \cdot (\bar{C} + D) \cdot (\bar{B} + D)$$

maxterm-input correspondence: complement literals if 1



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

POS simplification using K-map:

Group the squares that only contains "0"

Form an OR term (sum) for each group, instead of a product

Value "1", instead of "0", represent complement of the variable

Follow similar grouping rules for SOP

Either SOP or POS can be used to implement the Boolean function, depending on which gives more efficient implementation.

summarizing: proceed as SOP, but group 0's instead of 1's (square = maxterm)
+ complement the values in row-col. to find maxterm associated with square

Don’t-care condition

So far we assume that all combination of the input variables of a Boolean function are valid (for example, 3-variable Boolean function has 8 different input combinations that makes the function equal to 0 or 1)

There are applications in which some variable combinations never appear.

One of such examples is the BCD code

- 4-bit BCD code can have 16 values
- However, 1010 – 1111 are never used, or

  $A\bar{B}C\bar{D}, A\bar{B}CD,\ AB\bar{C}\bar{D}, AB\bar{C}D, ABC\bar{D},$ and ABCD never occur

These conditions are called don’t-care conditions.

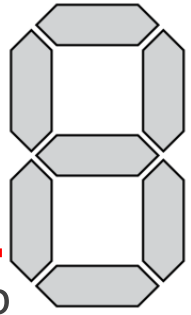
Don’t-care condition is marked with “X” in K-map

For minimization, X can take either “1” or “0”.

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

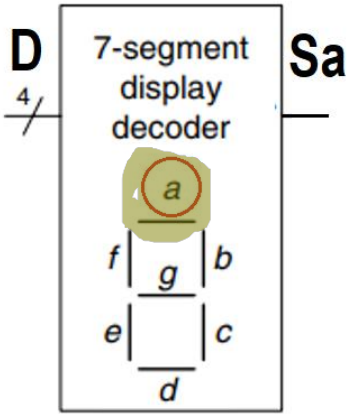
Design Example - 7-Seg Decoder

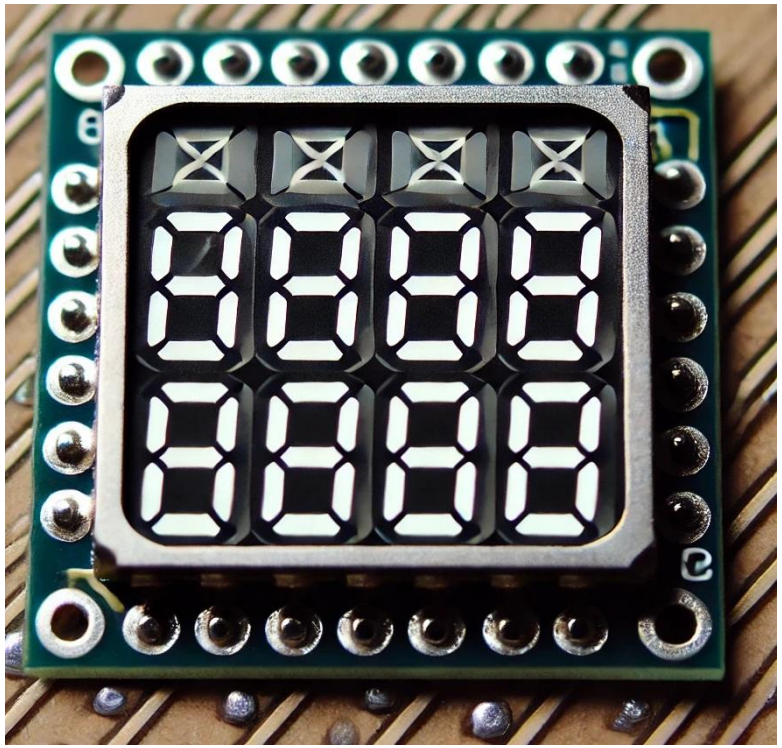


A 7-segment display decoder takes a 4-bit input, $\mathbf{D_{3:0}}$, and produces seven outputs to control LEDs, to display a digit from 0 to 9. The LEDs are named ***a*** through ***g***, or $\mathbf{S_a} - \mathbf{S_g}$.

Write a truth table for the output **Sa** and derive the MSOP.

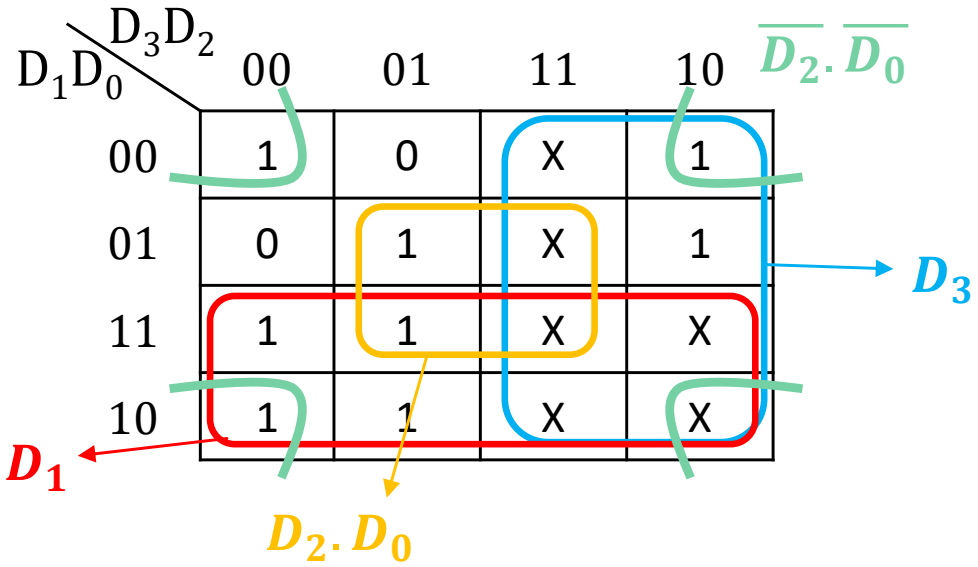
You may assume illegal input values (10–15) will never appear and use don't cares.





D3	D2	D1	D0	Sa
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
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

Design Example - 7-Seg Decoder



$$S_a = D_1 + D_0 \cdot D_2 + D_3 + \overline{D_2} \cdot \overline{D_0}$$

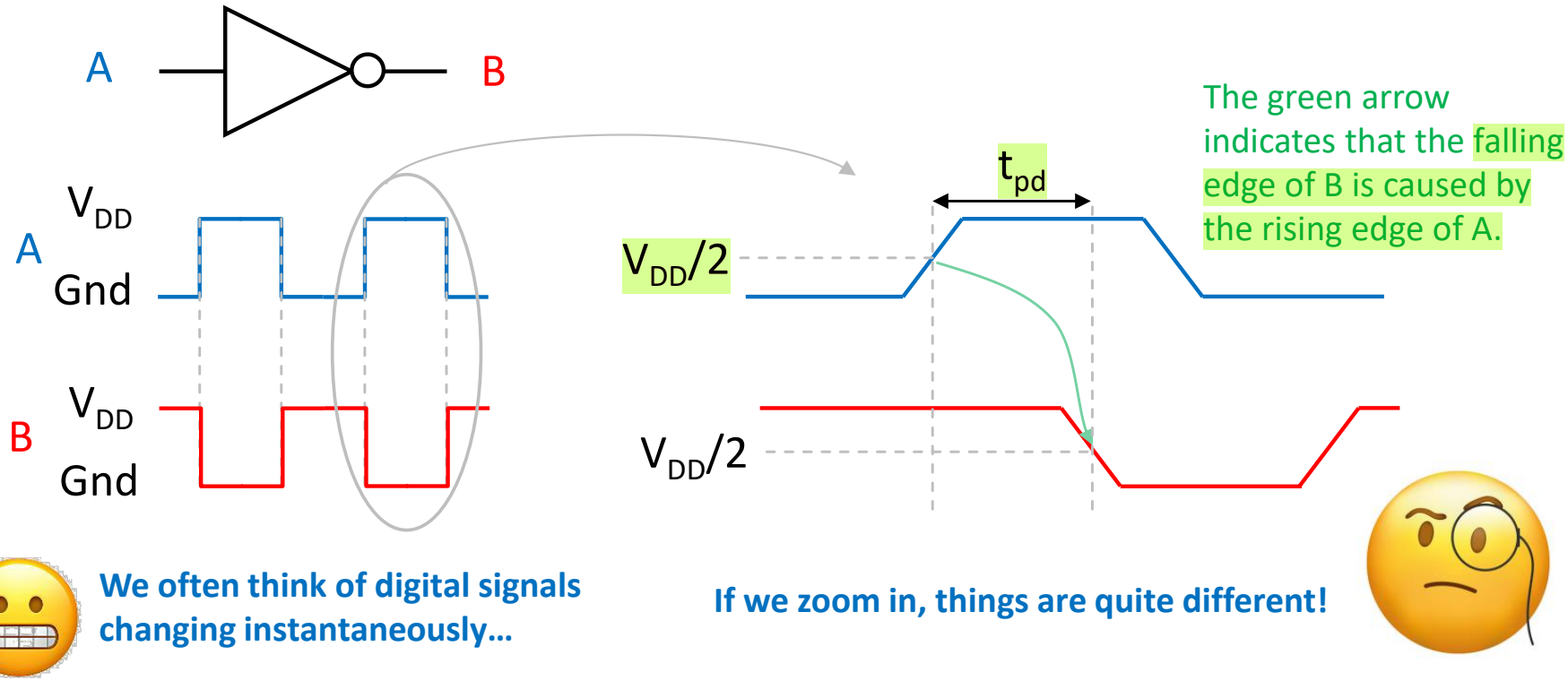
- We often assume that all combinations of input are valid (e.g. 3 inputs = 8 different input combinations that makes the function equal to 0 or 1)
- There are applications in which some variable combinations never appear.
- These conditions are called don't-care conditions.
- Don't-care condition is marked with "X" in K-map
- For minimization, X can take either "1" or "0".

Pre-Lab Preparation Q4 Q5 Q6

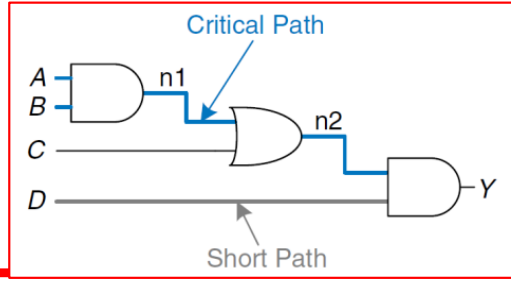
Timing

Propagation Delays

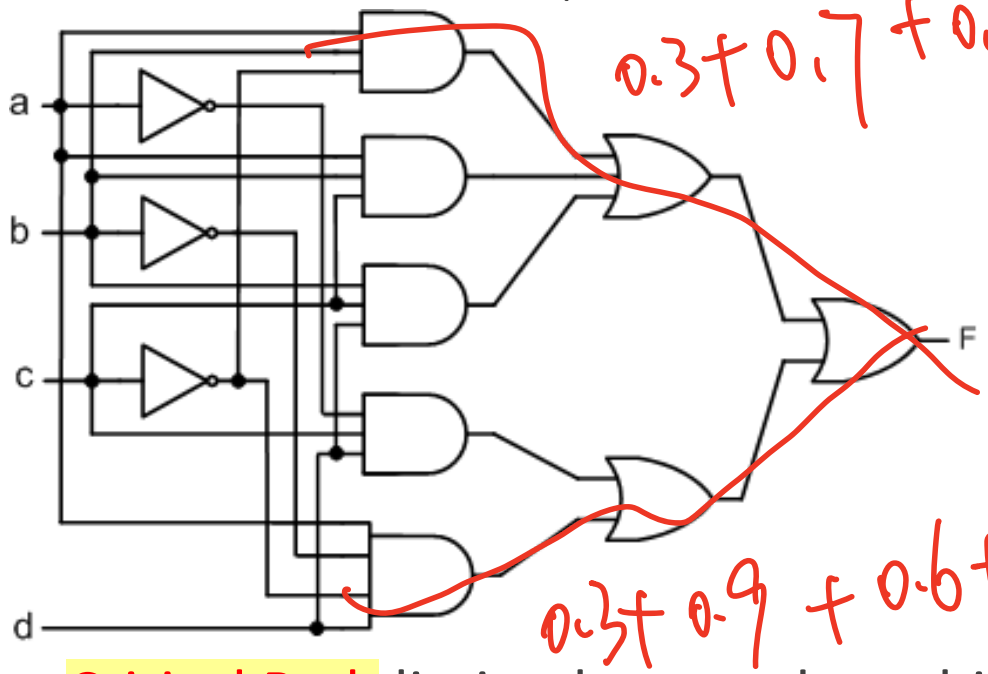
- Apart from the # gates ($), we are also concerned with **speed**!
- Logic gates take time to change its output → Propagation Delay, t_{pd} 传播
- t_{pd} = from 50% of change in i/p to 50% of induced change in o/p
- Usually in range of sub-ns! (Artix-7 FPGAs Link)



Critical Path



- Assuming the below t_{pd}s, which path below is the slowest?



Gate	t_{pd}
NOT	0.3ns
2-input OR	0.6ns
3-input OR	0.8ns
3-input AND	0.7ns
4-input AND	0.9ns

- The Critical Path limits the speed at which the **entire** circuit operates
  →eg. the slowest path in circuit!

- Total t_{pd} of a combinational circuit is sum of t_{pd} through each element on the critical path.

Which is the critical path here?

Summary

- Boolean Algebra:
  - Postulates, theorems and 3 binary operators: AND, OR and NOT
  - Truth table and Boolean function evaluation using truth table
  - Minterm and maxterm
  - Boolean function in SOP or POS form from truth table
- Logic Gates:
  - AND and NAND gates, OR and NOR gates, XOR and XNOR gates
  - Boolean function implementation using logic gates
  - Boolean function simplification using algebra postulates and theorems
- K-Maps
  - SOP / POS simplification
  - Don't-care condition
  - Minimal SOP (MSOP) and POS (MPOS)
- Verilog
  - Modeling of logic gates in Verilog