

Outline

3.1 Design Procedure

3.2 Examples

- Adder
- Subtractor
- Carry-look ahead adder
- Code converter
- Magnitude comparator
- 7-segment display

3.3 Timing Hazard

3.1 Design Procedure

1. State the problem / specification for the design.
2. Determine the number of input variables and output variables.
3. Formulate truth tables / Boolean functions between inputs and outputs.
4. Simplification / minimization for the logic functions.
5. Design and draw the logic circuit diagram.

3.2.1 Adder

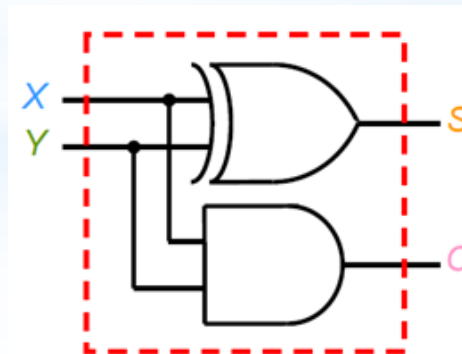
1-bit Half-Adder

- As known as **half adder (HA)**

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S(X, Y) = \sum m(1, 2) \\ = X \oplus Y$$

$$C(X, Y) = XY$$



1-bit Full-Adder (FA)

Carry out

x	y	c_i	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

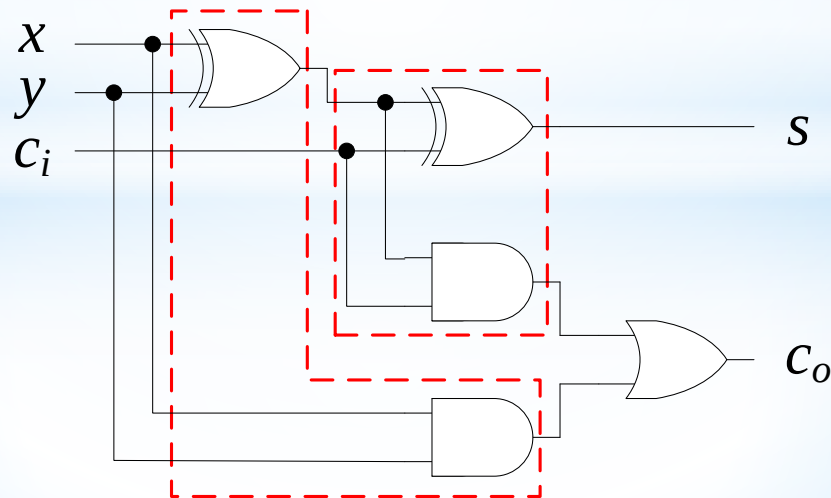
Sum

	yc_i			
x		1		1
	1		1	

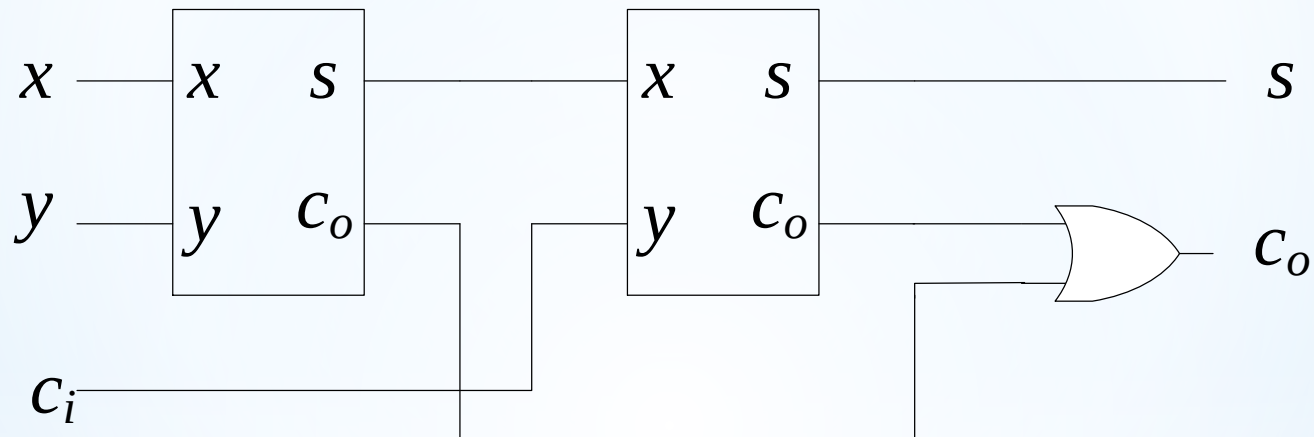
	yc_i			
x			1	
		1	1	1

$$\begin{aligned}
 s &= \bar{x}\bar{y}c_i + \bar{x}y\bar{c}_i + x\bar{y}\bar{c}_i + xyc_i \\
 &= \bar{x}(\bar{y}c_i + y\bar{c}_i) + x(\bar{y}\bar{c}_i + y c_i) \\
 &= \bar{x}(y \oplus c_i) + x(\overline{y \oplus c_i}) \\
 &= x \oplus y \oplus c_i
 \end{aligned}$$

$$\begin{aligned}
 c_o &= yc_i + xc_i + xy \\
 &= xy + x(y + \bar{y})c_i + y(x + \bar{x})c_i \\
 &= xy + xyc_i + x\bar{y}c_i + \bar{x}yc_i \\
 &= xy(1 + c_i) + (x\bar{y} + \bar{x}y)c_i \\
 &= xy + (x \oplus y)c_i
 \end{aligned}$$



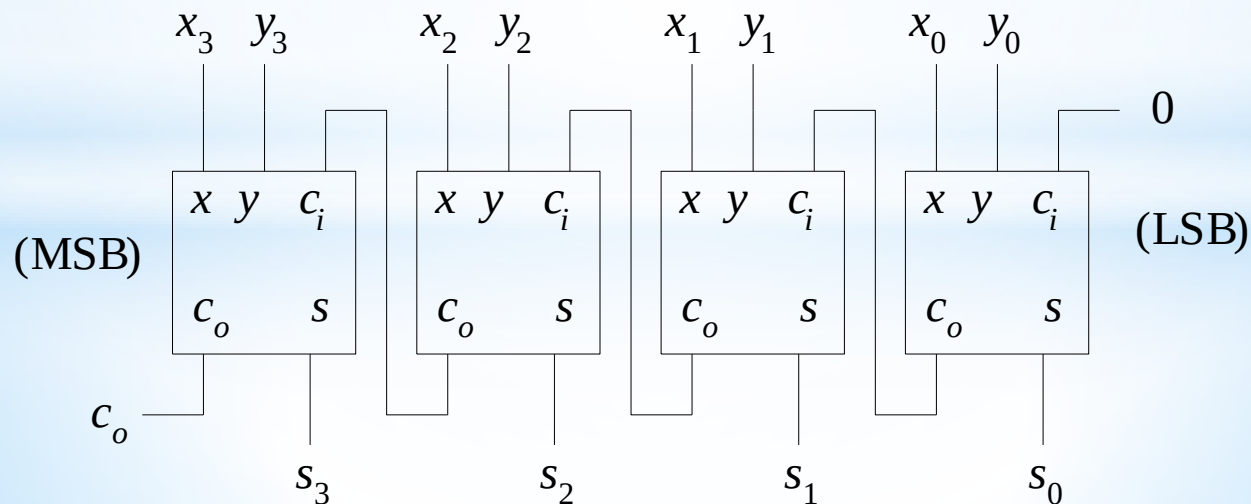
1-bit Full-Adder (FA)



Arrangement of 2 half-adders to form a full-adder

Ripple Carry Adder (Parallel adder)

- A single full-adder is capable of adding two one-bit numbers and an input carry.
- To add binary numbers with more than one bit, additional full-adders must be employed.
- The carry out from previous stage will be fed into the carry input of the next stage.
- A ripple carry adder is formed.



3.2.2 - Subtractor

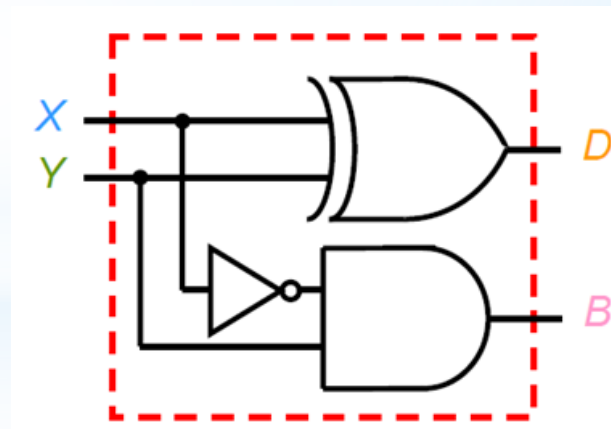
1-bit Half-Subtractor

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The outputs are
 D (difference bit) and
 B (borrow out bit)

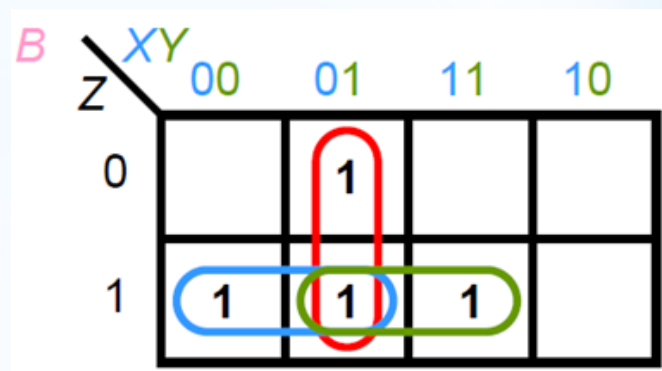
$$D(X, Y) = \sum m(1, 2) \\ = X \oplus Y$$

$$B(X, Y) = X'Y$$

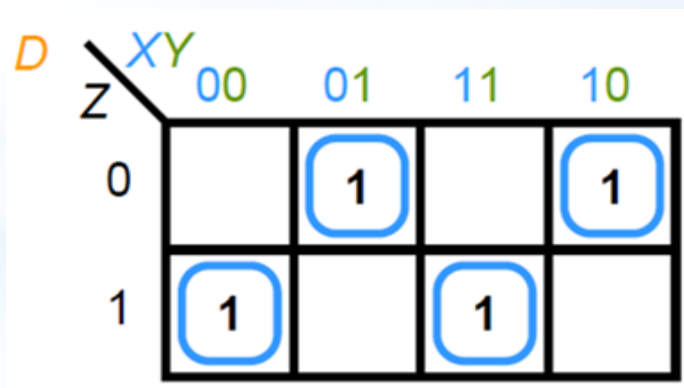


Full Subtractor

X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



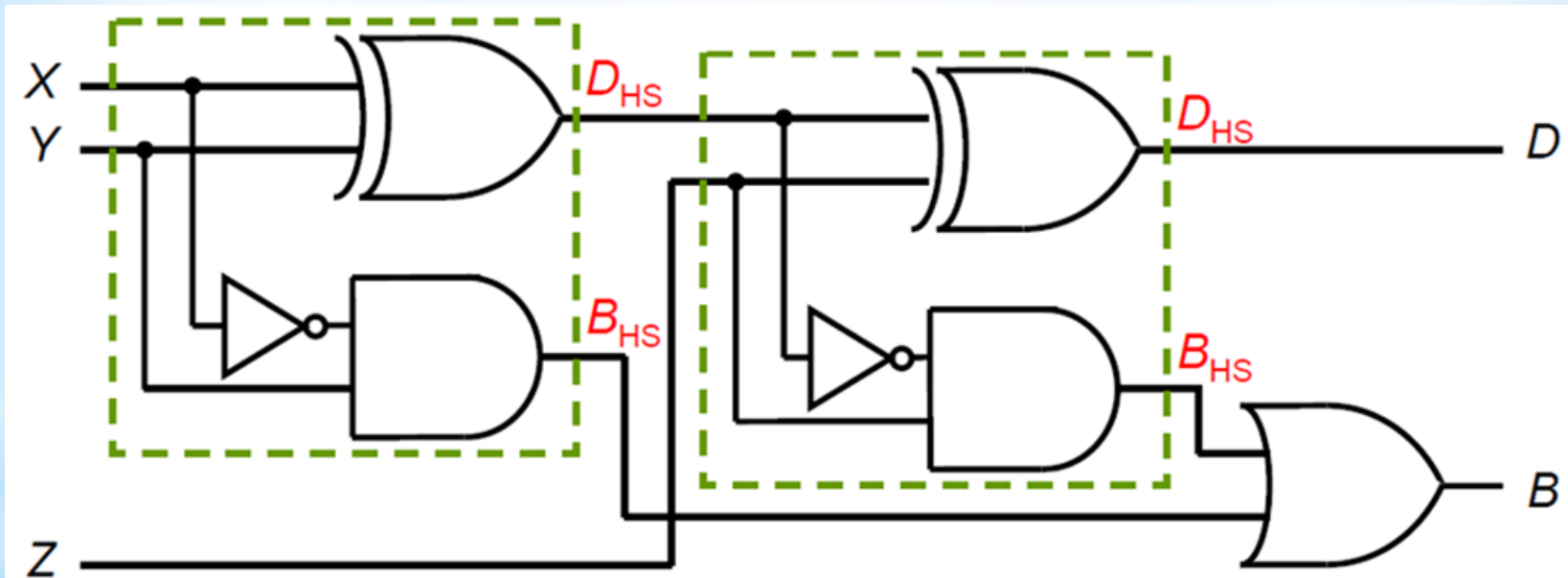
$$\begin{aligned}
 B &= X'Y + YZ + X'Z, \\
 &= X'Y + Z(X'Y' + XY) \\
 &= X'Y + Z(X \oplus Y)'
 \end{aligned}$$



$$D = X \oplus Y \oplus Z$$

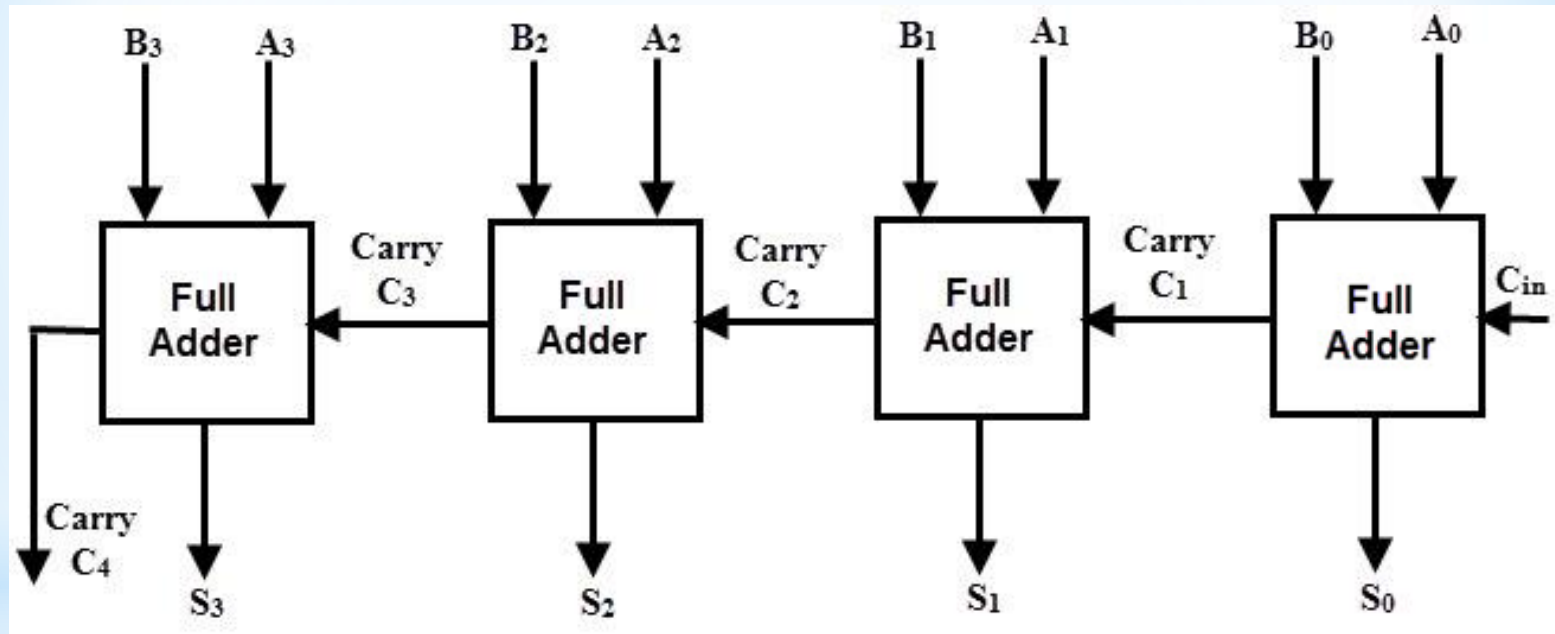
Full Subtractor

$$D = X \oplus Y \oplus Z, B = X'Y + Z(X \oplus Y)'$$

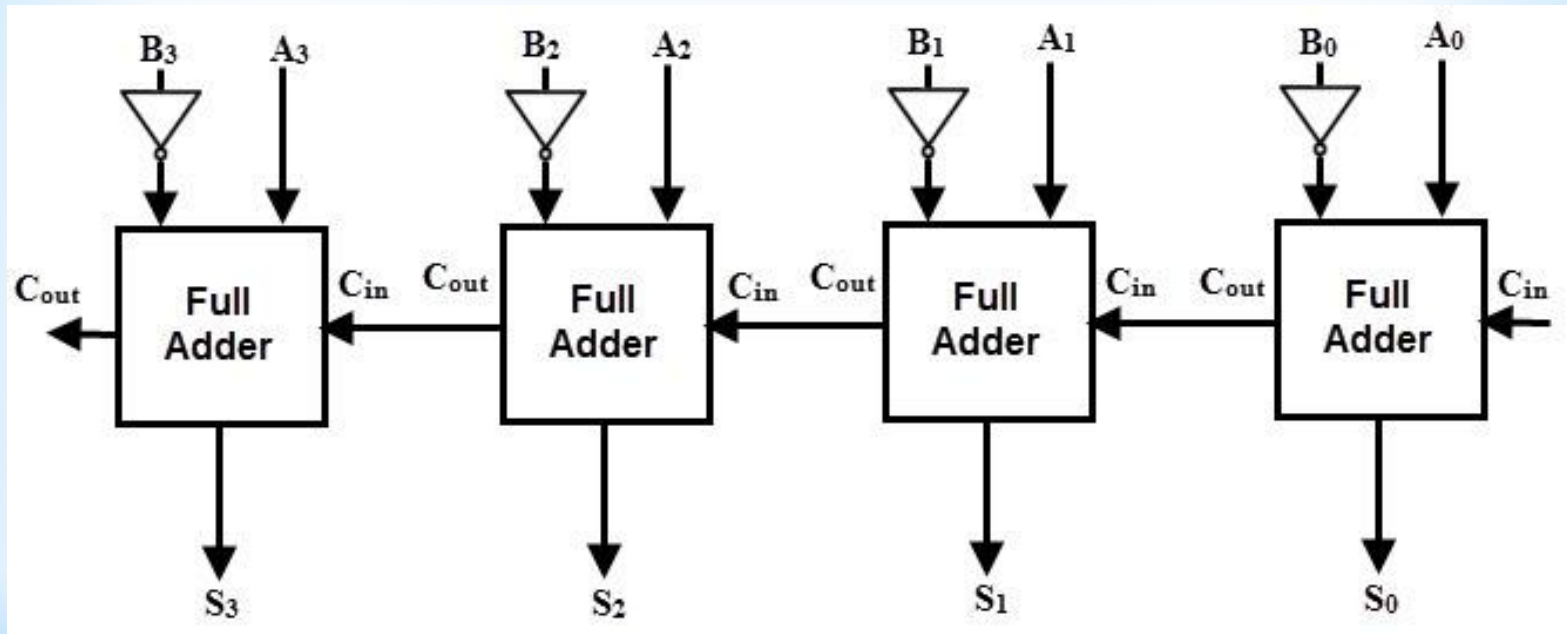


Like FA, FS is built by two HSs

4-bit parallel adder with 1-bit 4 FA modules

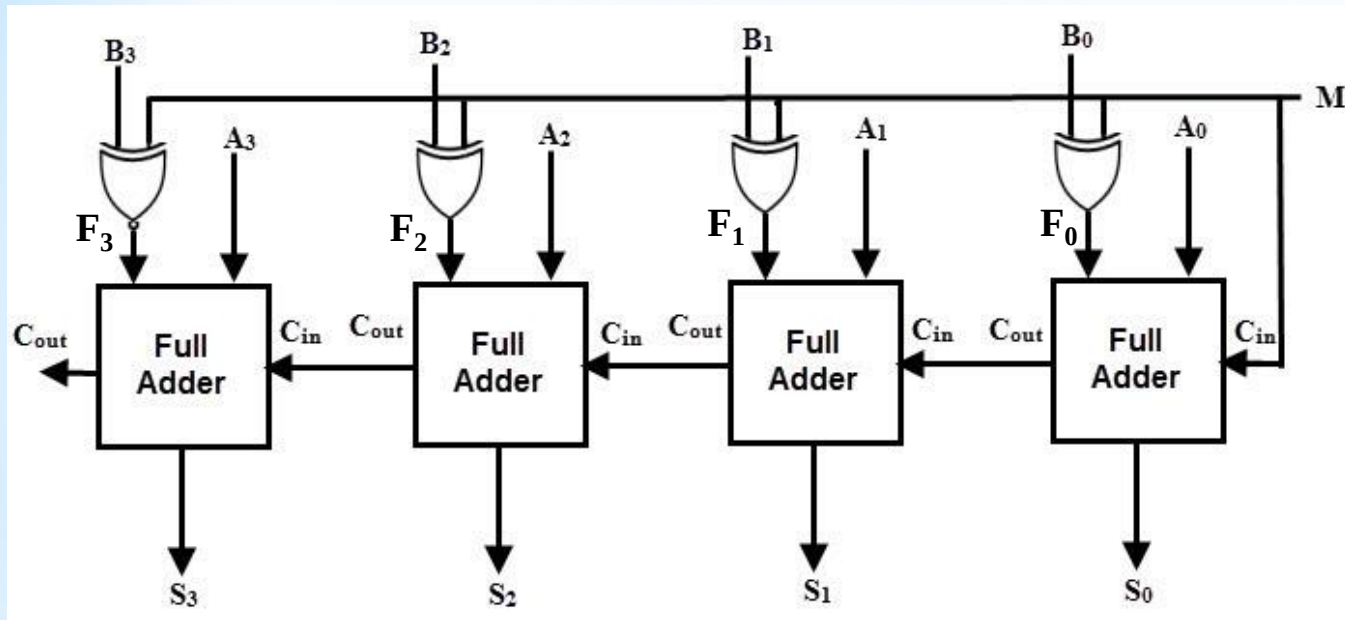


4-bit parallel subtractor with 1-bit 4 FA modules



$$A - B = A + (2\text{'s of } B) = A + B' + 1$$

4-bit Adder/Subtractor



$M = 0 \rightarrow \text{Adder}$

$M = 1 \rightarrow \text{Subtractor}$

M	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$$F = B \oplus M$$

3.2.3 Carry Look Ahead Adder

- The carry output of each full-adder stage is connected to the carry input of the next higher stage
- A time delay thus occurs because the sum and the carry output of each stage cannot be produced until the input carry appears
- **Assuming** that the delay for generating the carry output is m nsec
- For a N -bit adder, the total delay will be up to Nm nsec
- Serious delay problem if N is a large number
- **Solution: Calculate the carry bits beforehand, then construct carry-look-ahead adder**

Carry generate (g) and carry propagate (p)

x	y	c_i		c_o
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1	→	1 p
1	0	0		0
1	0	1	→	1 p
1	1	0		1 g
1	1	1	→	1 g

Conditions for generating g and p

From the table, we obtain

$$g = xy \quad \text{and} \quad p = x \oplus y$$

More importantly, the carry output can be obtained by:

$$c_o = g + pc_i$$

Derivation of a 4-bit carry look-ahead adder:

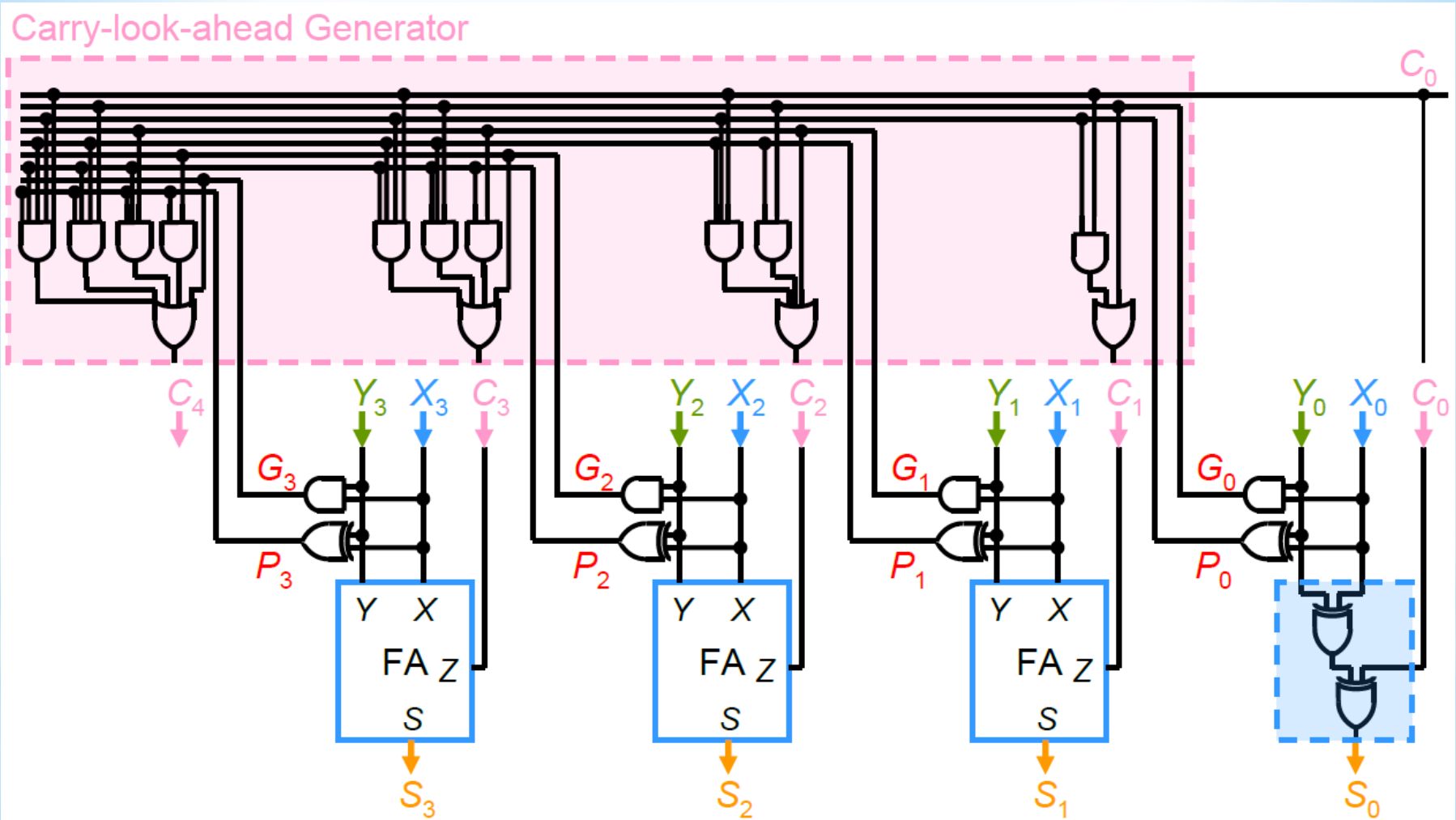
$$\text{Full-adder 0: } c_{o0} = g_0 + p_0 c_{i0}$$

$$\begin{aligned} \text{Full-adder 1: } c_{i1} &= c_{o0} \\ c_{o1} &= g_1 + p_1 c_{i1} \\ &= g_1 + p_1 c_{o0} \\ &= g_1 + p_1(g_0 + p_0 c_{i0}) \\ &= g_1 + p_1 g_0 + p_1 p_0 c_{i0} \end{aligned}$$

$$\begin{aligned} \text{Full-adder 2: } c_{i2} &= c_{o1} \\ c_{o2} &= g_2 + p_2 c_{o1} \\ &= g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_{i0}) \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{i0} \end{aligned}$$

$$\begin{aligned} \text{Full-adder 3: } c_{i3} &= c_{o2} \\ c_{o3} &= g_3 + p_3 c_{o2} \\ &= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{i0}) \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{i0} \end{aligned}$$

4-bit Carry Look Ahead Adder



3.2.4 Code Converter

State the case

Design a circuit to convert the BCD to the Excess-3 code

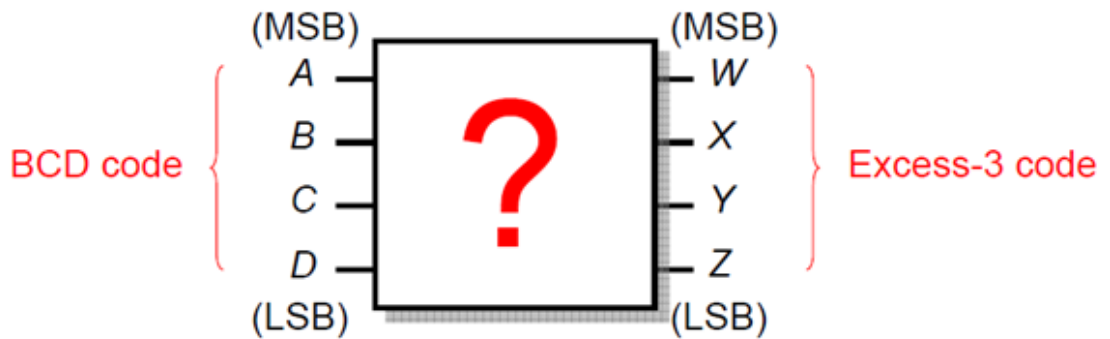
- A, B, C, D are the input of BCD.
- W, X, Y, Z are the output of Excess-3 code.
- The output functions are:

$W(A, B, C, D)$

$X(A, B, C, D)$

$Y(A, B, C, D)$

$Z(A, B, C, D)$



Decimal digit	Input (8421 code)				Output (Excess-3 code)			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X
Unused	X	X	X	X	X	X	X	X

Unused outputs can consider as DON'T CARE.

- There are four variables (A, B, C, D) in the functions
- Each output variable depends on 4 variables
- So we need 4 four-variable K-maps

$$W(A, B, C, D) = \sum m(5, 6, 7, 8, 9) + \sum d(10, 11, 12, 13, 14, 15)$$

$$X(A, B, C, D) = \sum m(1, 2, 3, 4, 9) + \sum d(10, 11, 12, 13, 14, 15)$$

$$Y(A, B, C, D) = \sum m(0, 3, 4, 7, 8) + \sum d(10, 11, 12, 13, 14, 15)$$

$$Z(A, B, C, D) = \sum m(0, 2, 4, 6, 8) + \sum d(10, 11, 12, 13, 14, 15)$$

K-map for W:

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

K-map for X:

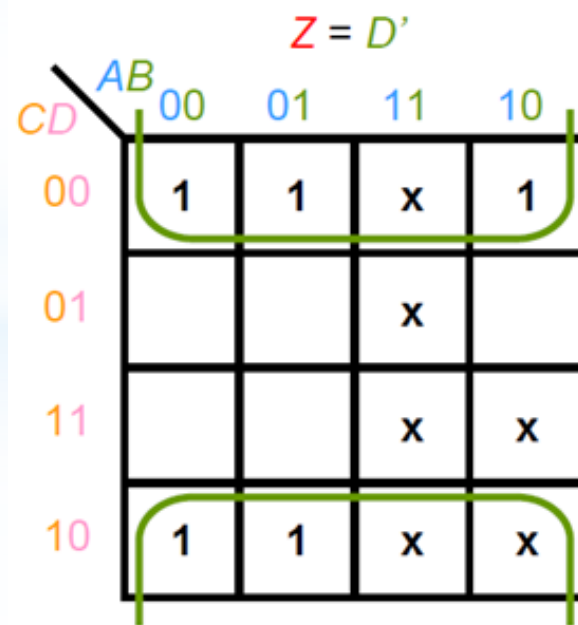
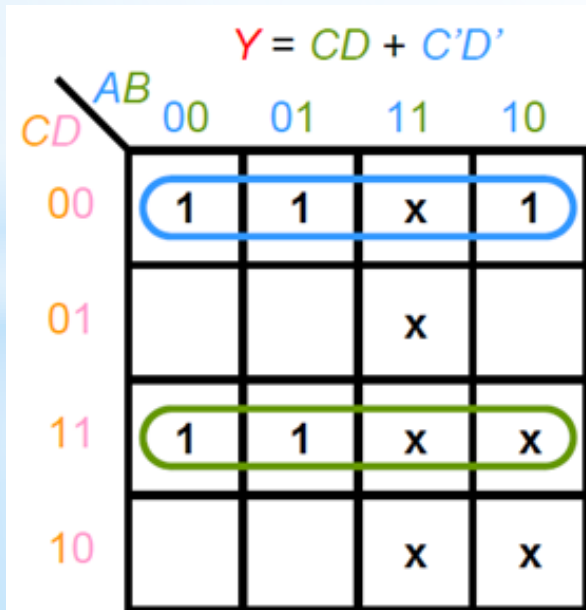
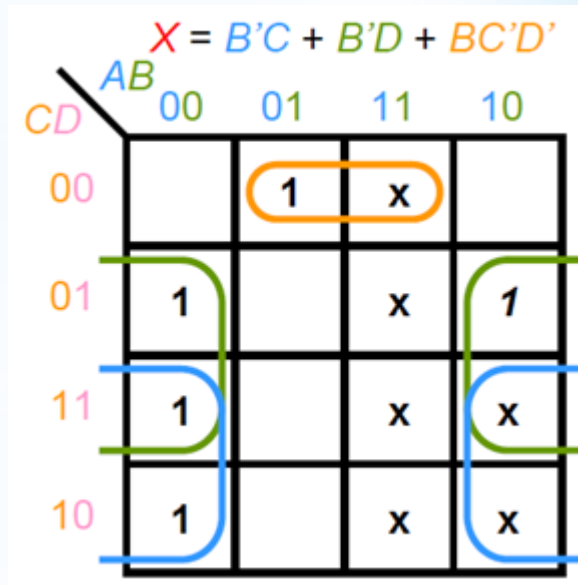
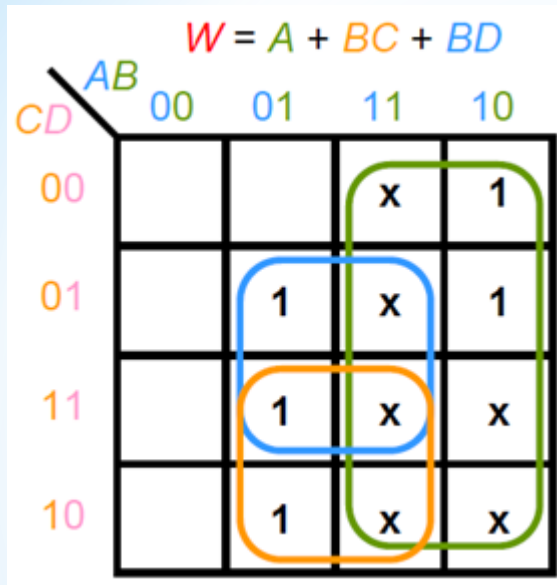
CD \ AB	00	01	11	10
00		1	x	
01	1		x	1
11	1		x	x
10	1		x	x

K-map for Y:

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

K-map for Z:

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



Logic functions and circuit

From previous K-maps,

$$W = A + BC + BD$$

$$X = B'C + B'D + BC'D'$$

$$Y = CD + C'D'$$

$$Z = D'$$

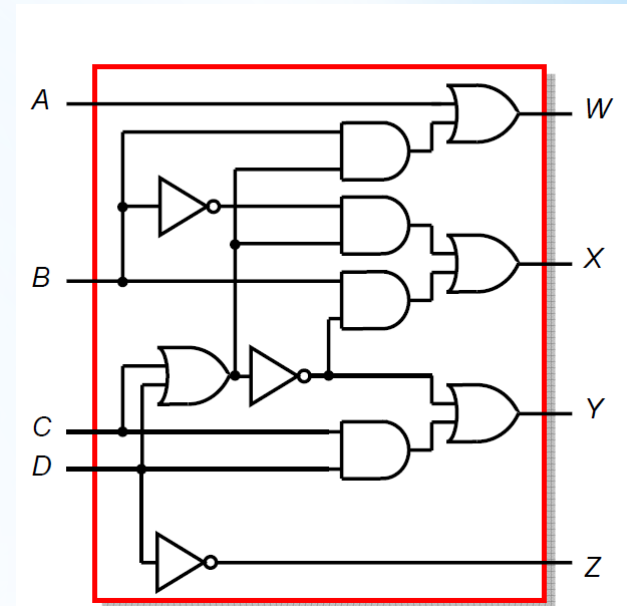
We further optimize them (optional)

$$W = A + BC + BD = A + B(C + D)$$

$$X = B'C + B'D + BC'D' = B'(C + D) + B(C + D)'$$

$$Y = CD + C'D' = CD + (C + D)'$$

DeMorgan's Law



3.2.5 Magnitude Comparator

A comparator is used to compare the magnitude of two numbers.

The output will be one of the three possibilities,
small (*s*), equal (*e*) and greater (*g*)

For two 2-digit numbers, *A* and *B*, we have the following description:

$$z = \begin{cases} g, & \text{if } A > B \\ e, & \text{if } A = B \\ s, & \text{if } A < B \end{cases}$$

Finally, we construct the following truth table:

input				output				input				output		
a_1	a_0	b_1	b_0	z_2	z_1	z_0		a_1	a_0	b_1	b_0	z_2	z_1	z_0
0	0	0	0	0	1	0		1	0	0	0	1	0	0
0	0	0	1	0	0	1		1	0	0	1	1	0	0
0	0	1	0	0	0	1		1	0	1	0	0	1	0
0	0	1	1	0	0	1		1	0	1	1	0	0	1
0	1	0	0	1	0	0		1	1	0	0	1	0	0
0	1	0	1	0	1	0		1	1	0	1	1	0	0
0	1	1	0	0	0	1		1	1	1	0	1	0	0
0	1	1	1	0	0	1		1	1	1	1	0	1	0

The output ($z_2 z_1 z_0$) can be obtained by using k-map:

	b_1b_0			
a_1a_0				
	1			
	1	1		1
	1	1		

$$z_2 = a_1\bar{b}_1 + a_0\bar{b}_1\bar{b}_0 + a_1a_0\bar{b}_0$$

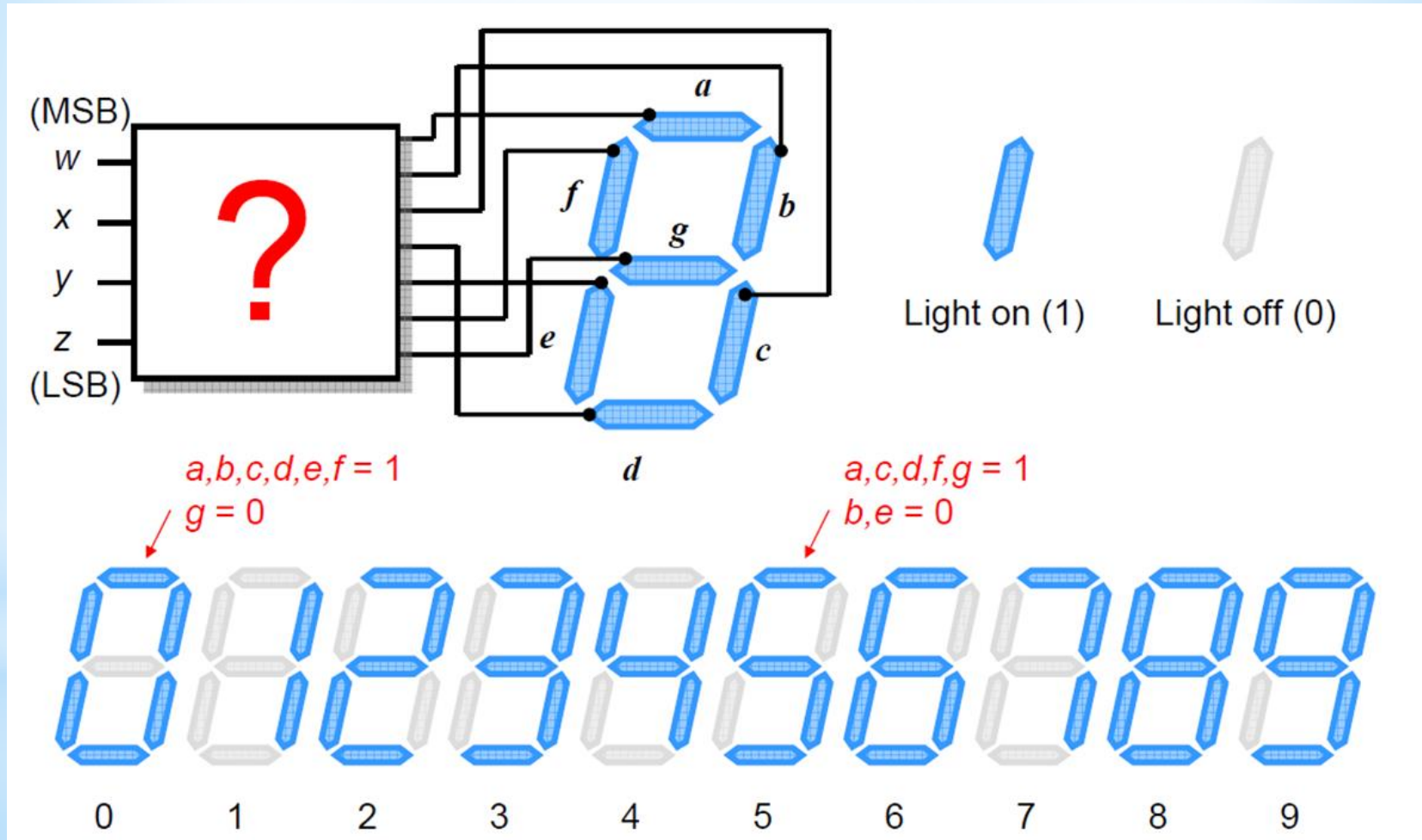
	b_1b_0			
a_1a_0	1			
		1		
			1	
				1

$$z_1 = (a_1b_1 + \bar{a}_1\bar{b}_1)(a_0b_0 + \bar{a}_0\bar{b}_0)$$

	b_1b_0			
a_1a_0		1	1	1
			1	1
			1	

$$z_0 = \bar{a}_1b_1 + \bar{a}_0b_1b_0 + \bar{a}_1\bar{a}_0b_0$$

3.2.6 7-segment display



Input BCD				Seven-Segment Display						
<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

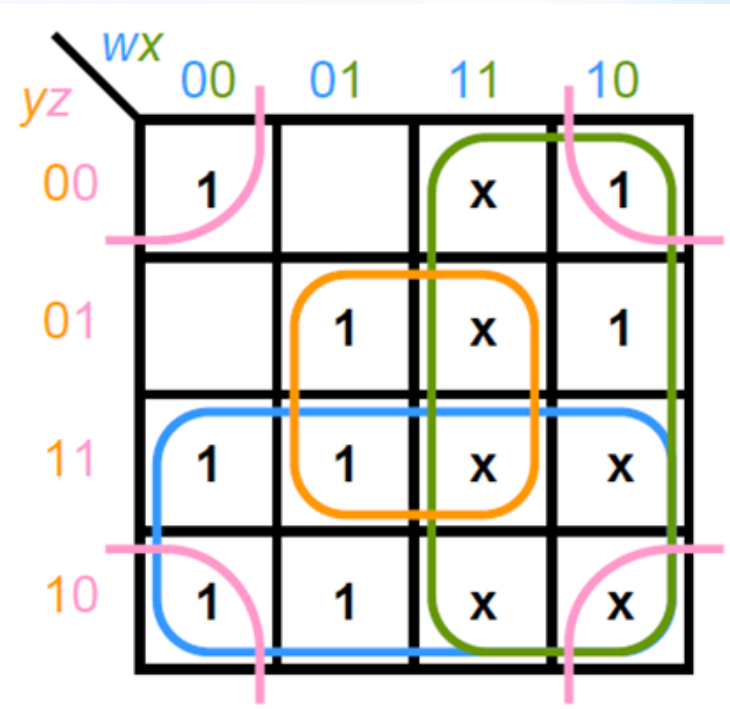
Note: the corresponding outputs of those unused BCD code should either be 0 or don't care

- Number of variables (input)?
- Number of functions (output)?
- Number of K-map?

- Example of functions:
- $a(w, x, y, z) = \sum m(0, 2, 3, 5, 6, 7, 8, 9) + \sum d(10, 11, 12, 13, 14, 15)$

K-maps of segment-a

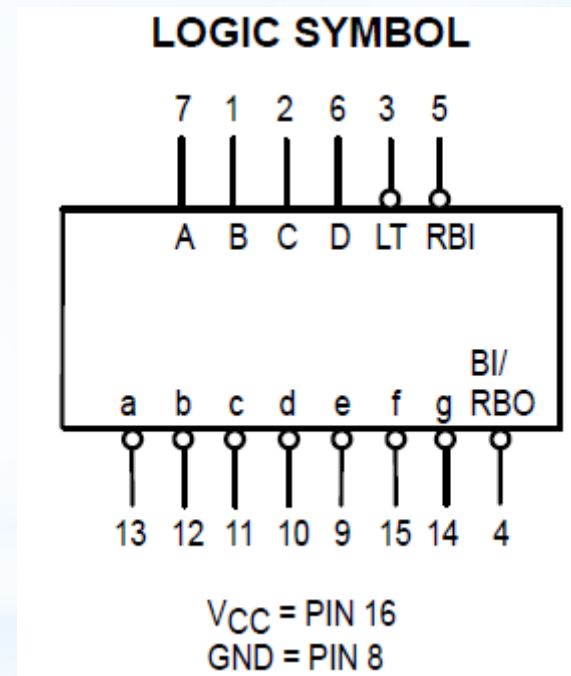
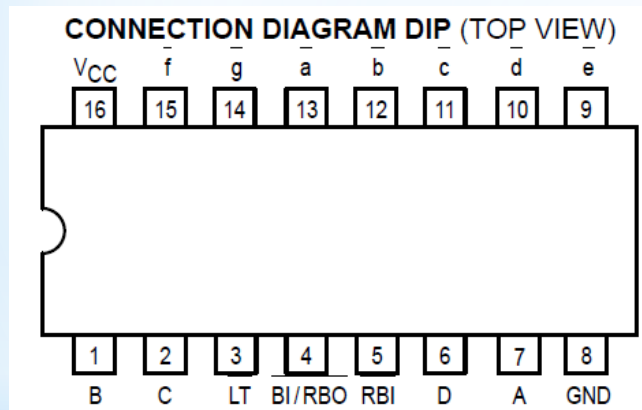
	wx			
	00	01	11	10
yz	00	1		x
	01		1	x
	11	1	1	x
	10	1	1	x

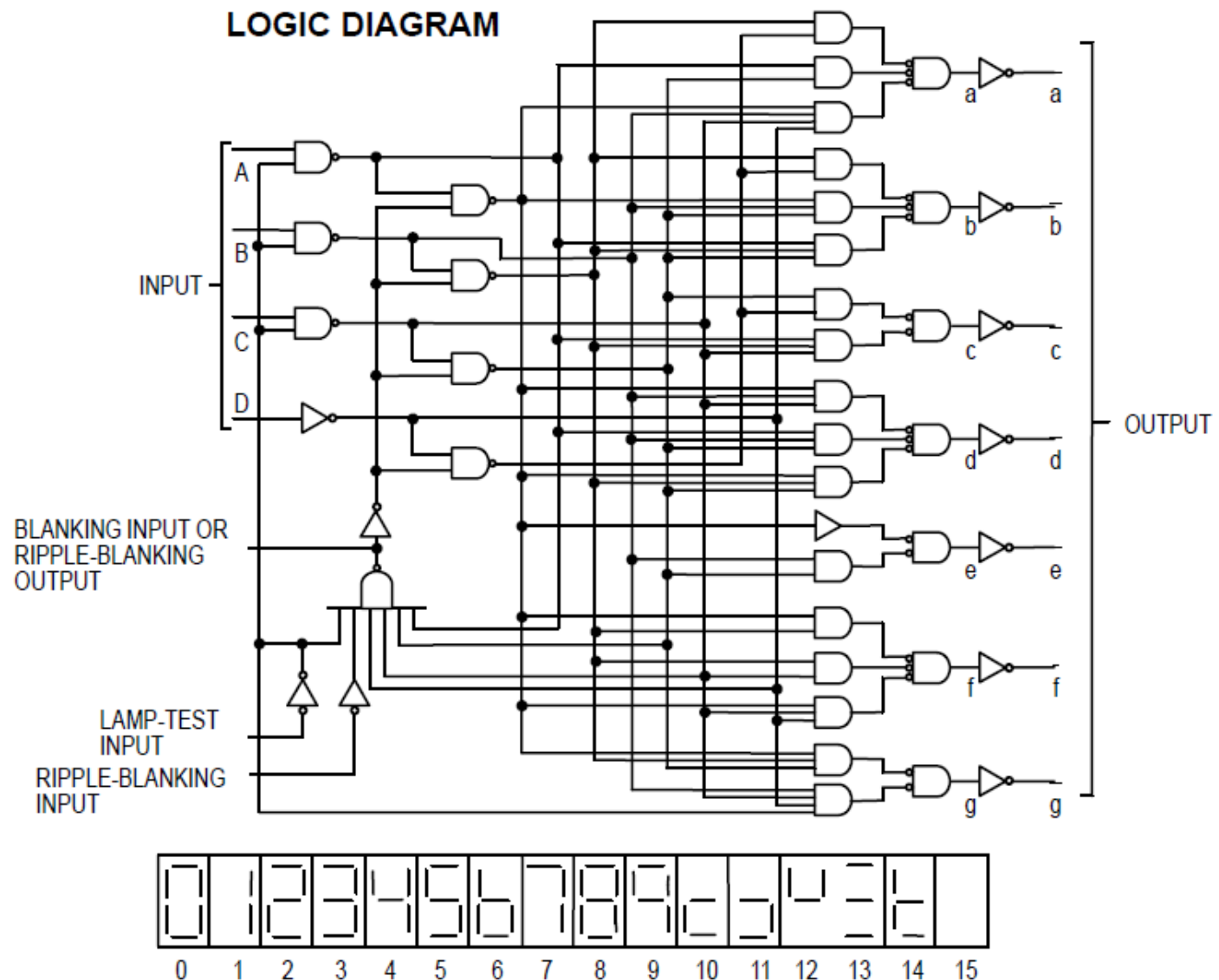


$$a(w,x,y,z) = w + y + xz + x'z'$$

Exercise: find the simplified expression of b , c , d , e , f and g

BCD TO 7-SEGMENT DECODER/DRIVER 74LS47





NUMERICAL DESIGNATIONS — RESULTANT DISPLAYS

INPUTS							OUTPUTS								
DECIMAL OR FUNCTION	$\overline{\text{LT}}$	$\overline{\text{RBI}}$	D	C	B	A	$\overline{\text{BI/RBO}}$	$\overline{\text{a}}$	$\overline{\text{b}}$	$\overline{\text{c}}$	$\overline{\text{d}}$	$\overline{\text{e}}$	$\overline{\text{f}}$	$\overline{\text{g}}$	NOTE
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	A
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	A
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	B
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	C
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	D

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

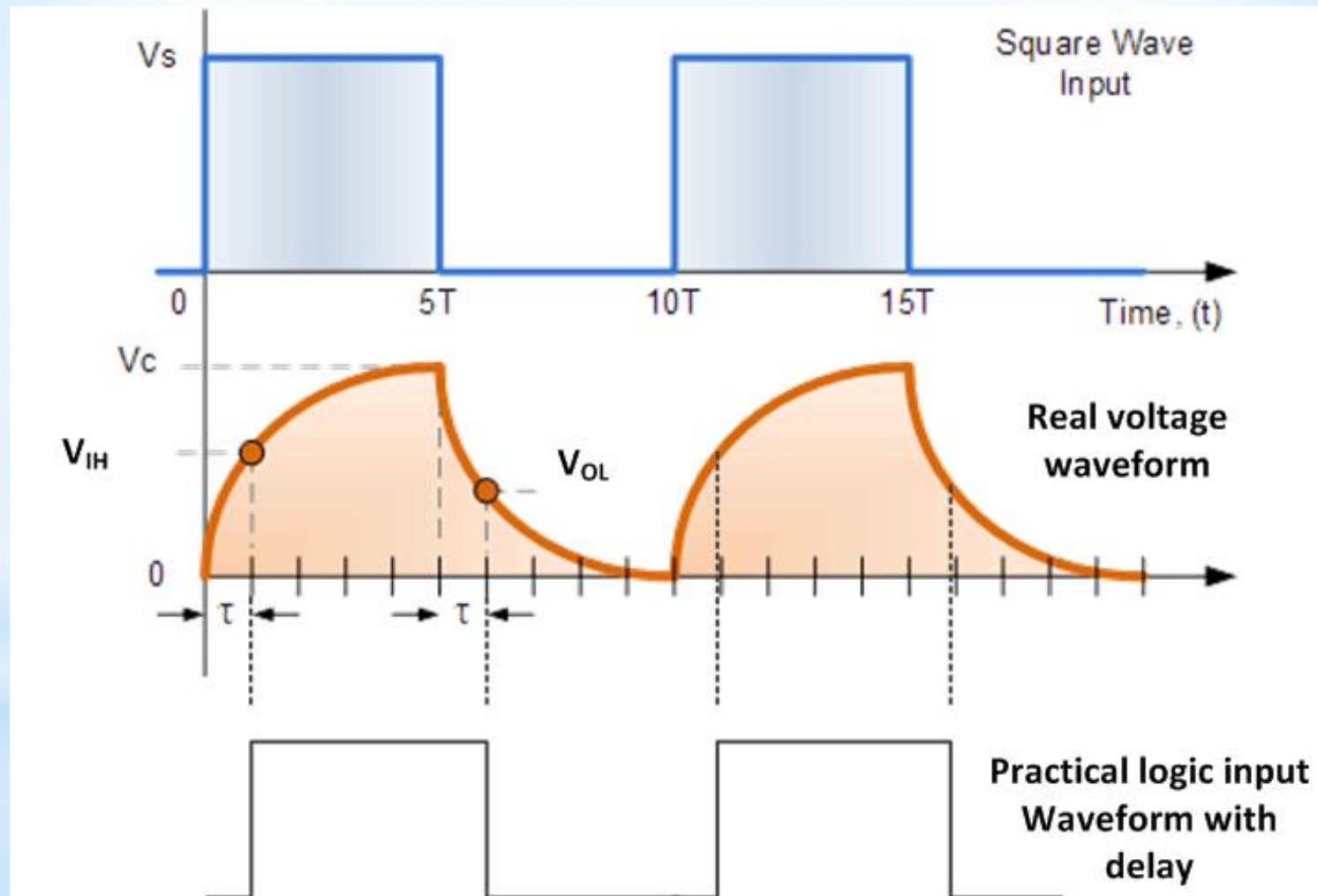
NOTES:

- (A) BI/RBO is wire-AND logic serving as blanking input (BI) and/or ripple-blanking output (RBO). The blanking out (BI) must be open or held at a HIGH level when output functions 0 through 15 are desired, and ripple-blanking input (RBI) must be open or at a HIGH level if blanking of a decimal 0 is not desired. X = input may be HIGH or LOW.
- (B) When a LOW level is applied to the blanking input (forced condition) all segment outputs go to a LOW level regardless of the state of any other input condition.
- (C) When ripple-blanking input (RBI) and inputs A, B, C, and D are at LOW level, with the lamp test input at HIGH level, all segment outputs go to a HIGH level and the ripple-blanking output (RBO) goes to a LOW level (response condition).
- (D) When the blanking input/ripple-blanking output (BI/RBO) is open or held at a HIGH level, and a LOW level is applied to lamp test input, all segment outputs go to a LOW level.

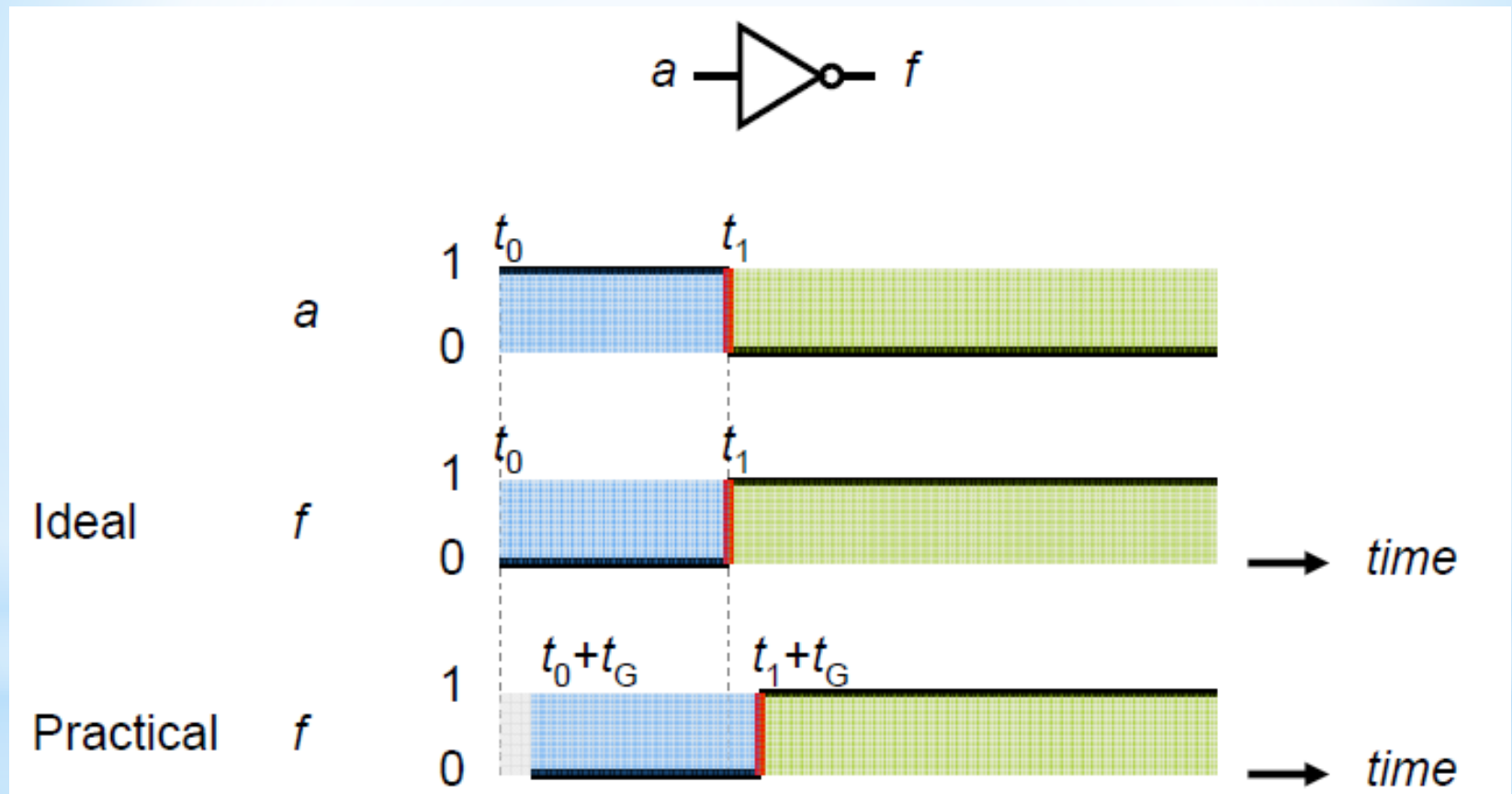
3.3 Timing Hazard

- Logic devices (gates or other more complex circuits) are essentially made from semiconductor
- IC input impedance and PCB copper track inductance cause charging and discharging effect (similar to RC circuit response)
- The “real” input and output voltages are not a perfect step function
- Practical logic input and output waveforms exhibit “delay” nature

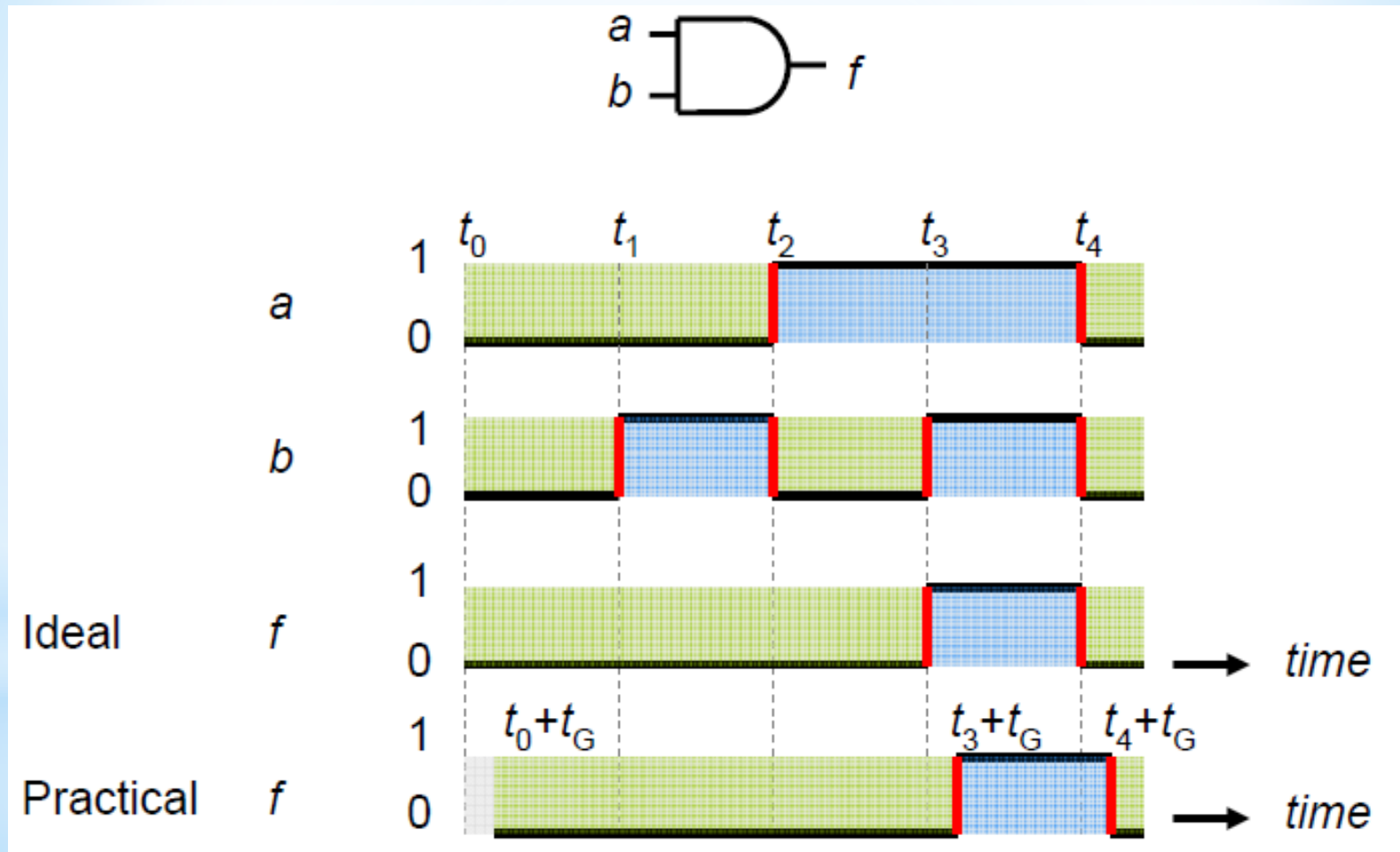
I/P and O/P waveform



Timing Diagram of NOT Gate

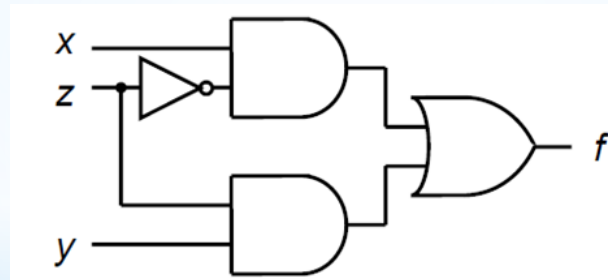


Timing Diagram of AND Gate



Gate delay

- Output waveform is shifted t_G time units
- The length of time for an input change, to result in the corresponding output change.



- The propagation delay could cause undesirable events:
timing hazard: **timing hazards**

Timing Hazards

- The circuit output may produce a short pulse (which should not be happened) at the transient time caused by delay in propagation of signals
- Short pulse also known as **glitch**
- Two kinds of hazards: **static** and **dynamic**

Static and Dynamic Hazards



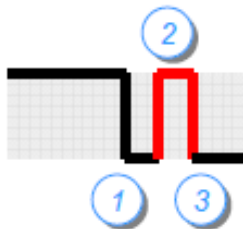
Static-0 hazard: the output may momentarily go to 1 when it should *remain 0*



Static-1 hazard: the output may momentarily go to 0 when it should *remain 1*

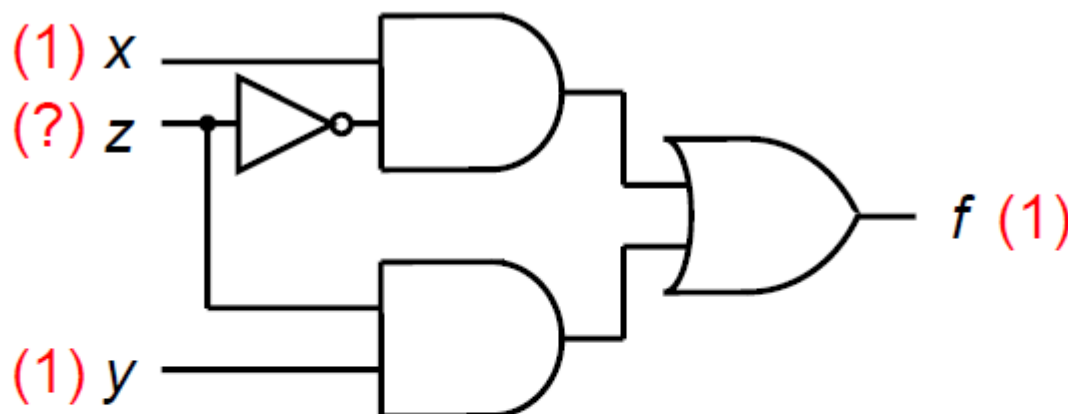


Dynamic hazard: The output changes three or more times when it should change from 1 to 0 or from 0 to 1 *only once*



Example

■ $f(x, y, z) = xz' + yz$

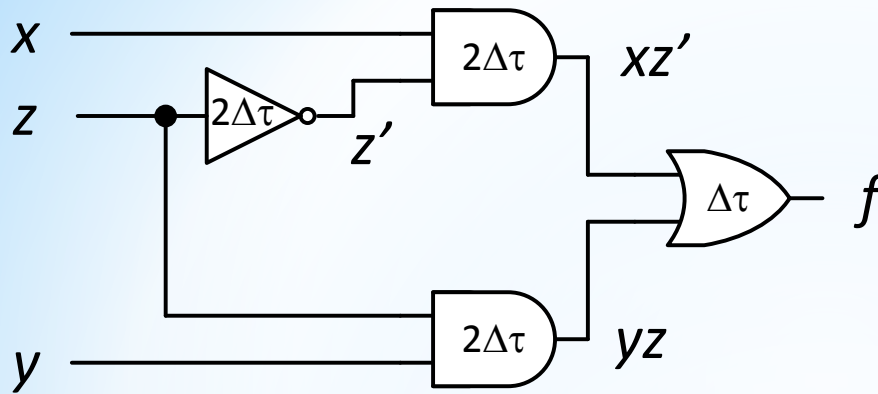


■ If given that ($x = 1$ AND $y = 1$)

■ $f(x, y, z) = (1)z' + (1)z$

■ $= z' + z = 1$

■ The output f will always equal to 1 no matter what the value of z is (i.e. independent of z)



Assuming that the delay of the gates are different as shown above.

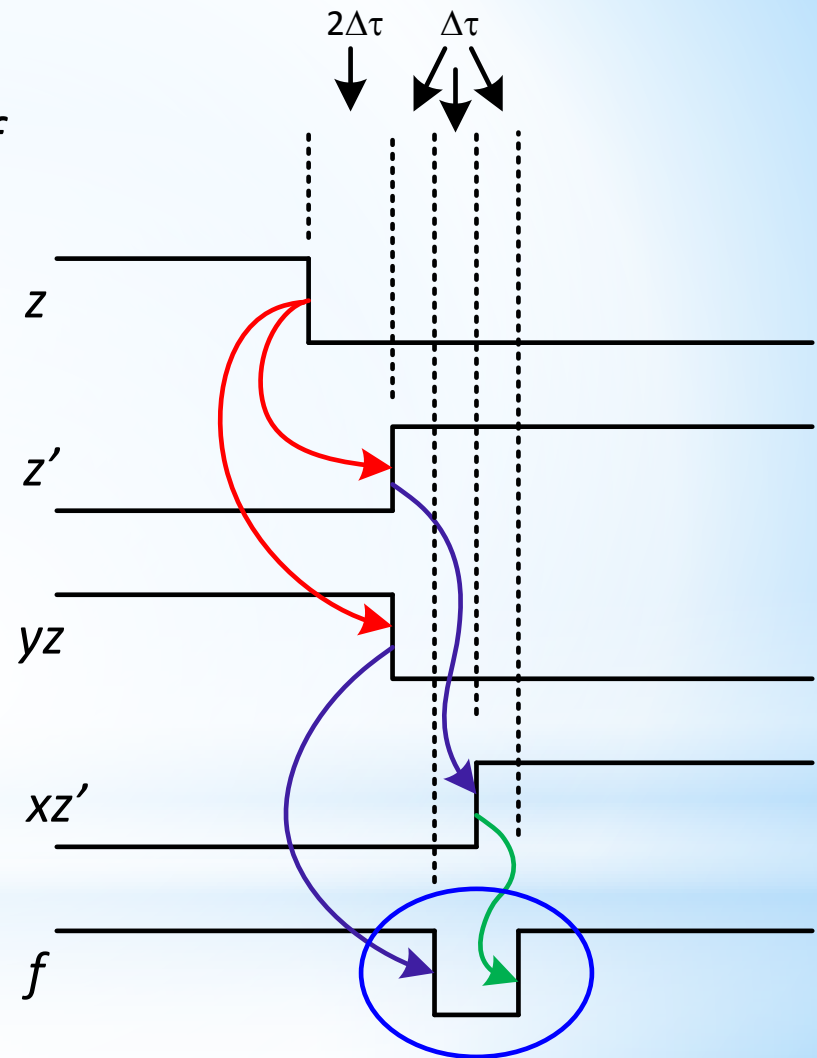
With the initial input condition:

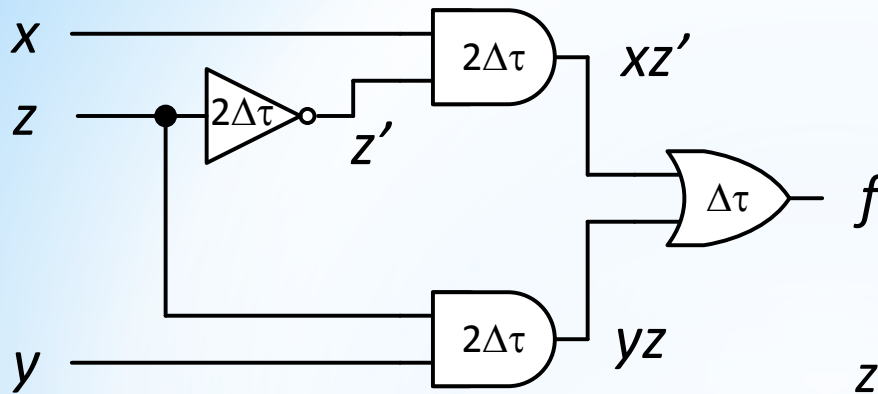
$$xyz = 111$$

Now, change z from “1” to “0”.

Then the associated gate outputs will be change after their corresponding delays.

An unwanted glitch will appear at the output - **static-1 hazard !!!**





Consider a different case

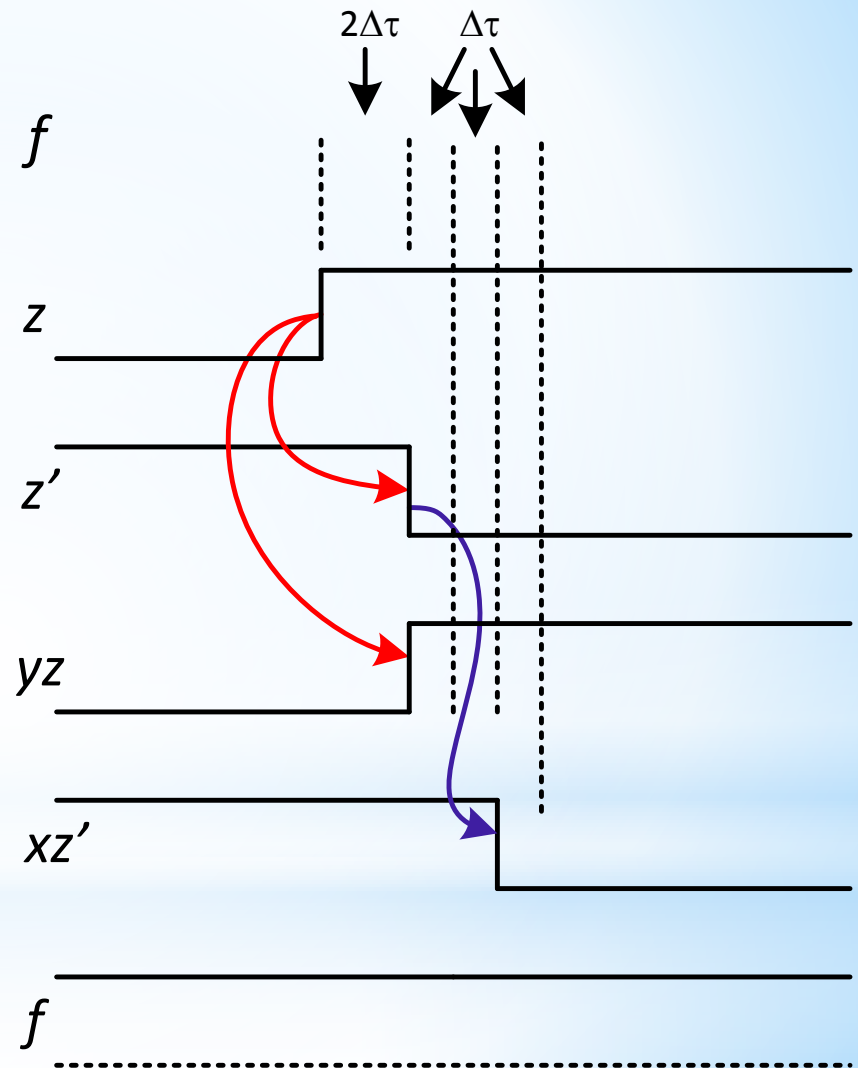
With the initial input condition:

$$xyz = 110$$

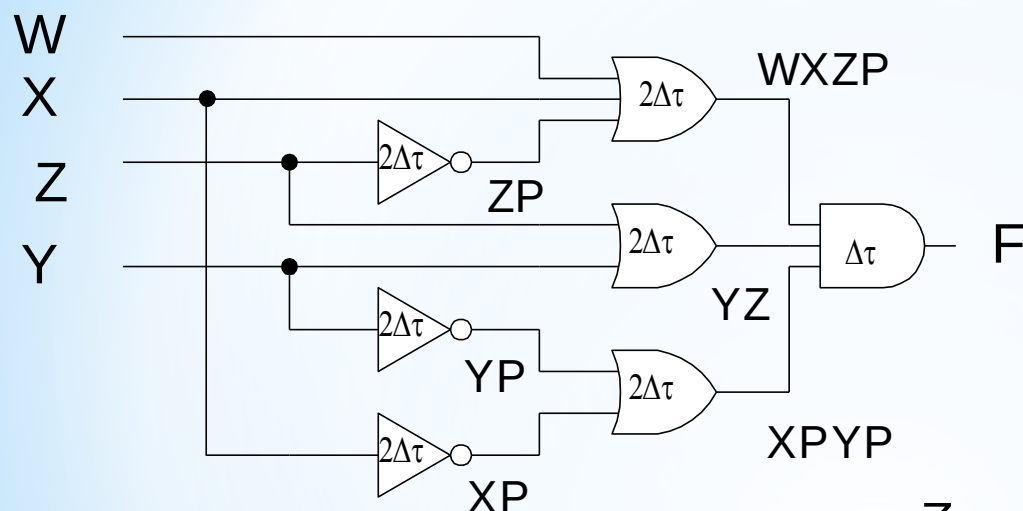
Now, change z from “0” to “1”.

NO glitch !!!

Hazard only occurs when input of xzy change from 111 to 110, NOT for the case of from 110 to 111.



Example of static-0 hazard



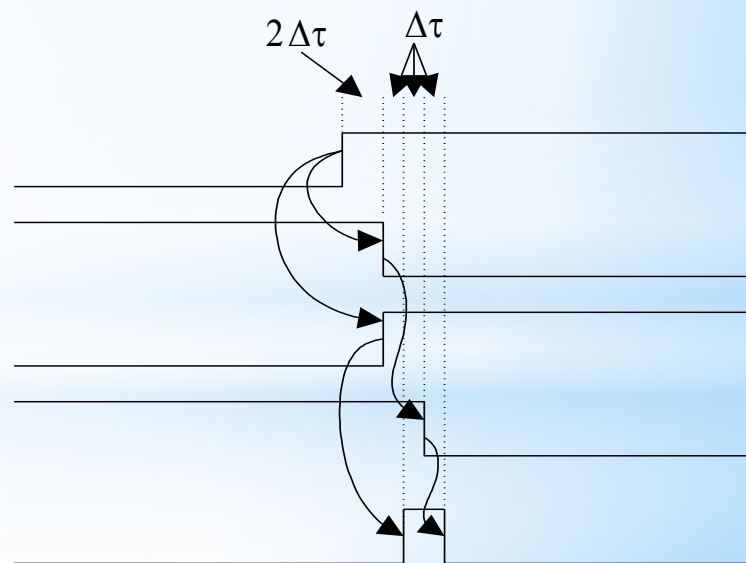
With the initial input condition:

$$wxyz = 0000$$

Now, change **z** from “0” to “1”.

Static-0 hazard occurs !!!

Z
ZP
YZ
WXZP
F

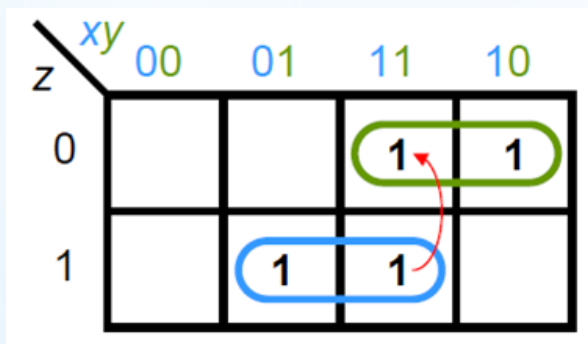


Timing hazard elimination

- The occurrence of the hazard can be detected by inspecting the K-map of the particular circuit.
- Eliminating a hazard is to enclose the two minterms in question with another product term that overlaps both groupings.
- The removal of hazards requires the addition of redundant gates to the circuit.

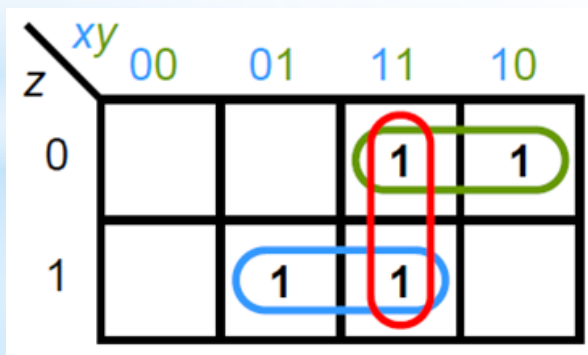
K-map for eliminating timing hazard

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



The optimal grouping

Static-1 hazard occurs when input change from 111 to 110



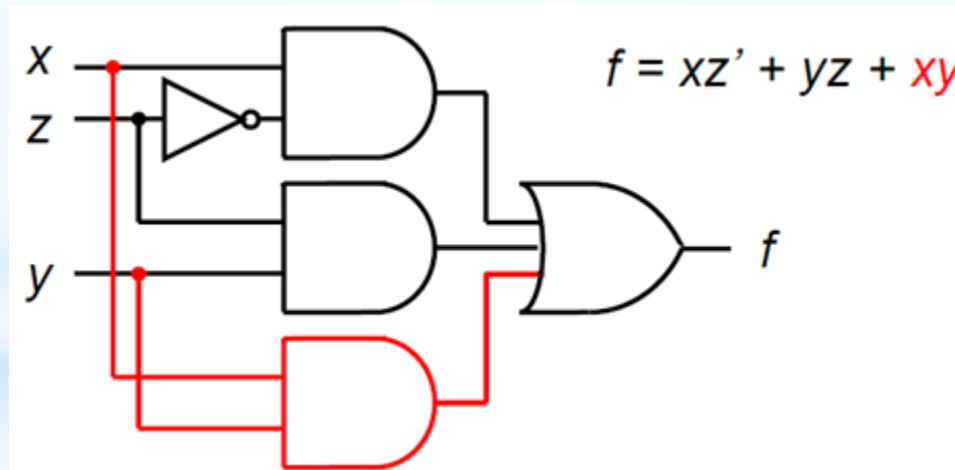
Include an redundant product term

$$f = xz' + yz + xy$$

Now the hazard is removed!

Use K-map to eliminate Hazards

- To eliminate the hazard, group the two minterms that cause the hazards
- An additional redundant product term (gate) is introduced



Hazard-free realization

- General idea: to include all PIs
- Work for both K-map and QM-method

Please try to develop a hazard-free circuit for the function

$$f(a, b, c) = \Sigma m(0, 2, 4, 5)$$

Steps:

- Minimize the function f
- Realize f to a hazard-free circuit