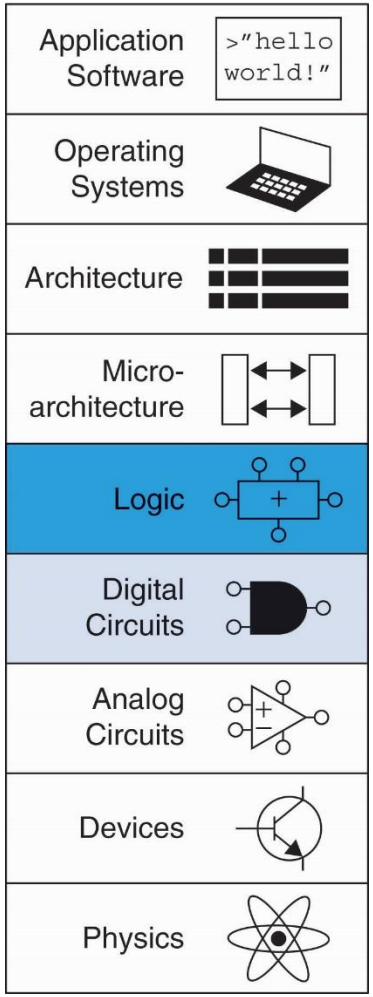
Chapter 2 :: Combinational Logic Design

***Digital Design and Computer Architecture*, 2nd Edition**

David Money Harris and Sarah L. Harris

Chapter 2 :: Topics

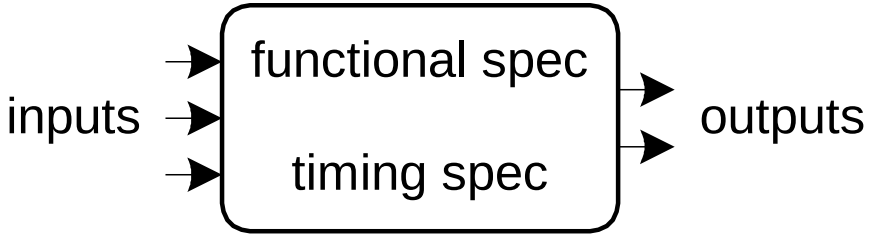
- Introduction
- Boolean Equations
- Boolean Algebra
- From Logic to Gates
- Multilevel Combinational Logic
- X's and Z's, Oh My
- Karnaugh Maps
- Combinational Building Blocks
- Timing



Introduction

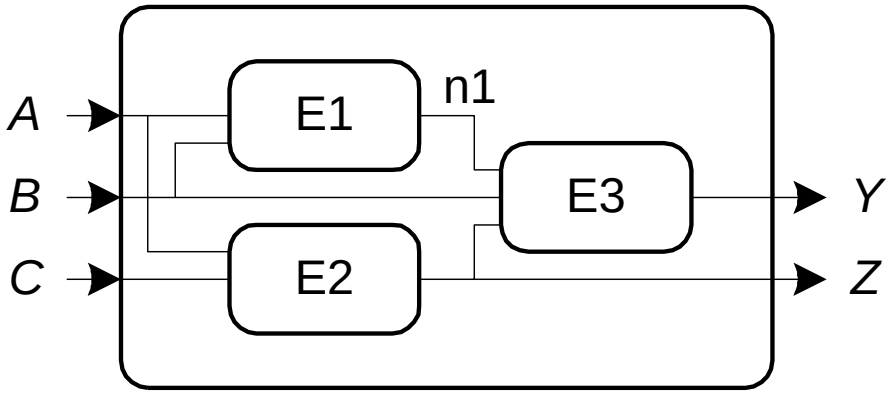
A logic circuit is composed of:

- Inputs
- Outputs
- Functional specification
- Timing specification



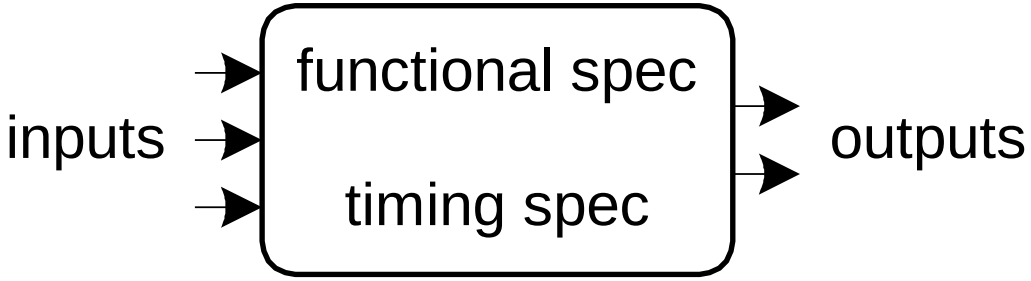
Circuits

- Nodes
  - Inputs: *A*, *B*, *C*
  - Outputs: *Y*, *Z*
  - Internal: n1
- Circuit elements
  - E1, E2, E3
  - Each a circuit



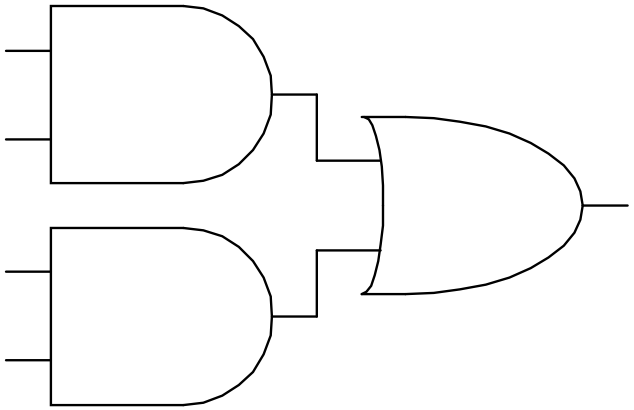
Types of Logic Circuits

- Combinational Logic
  - Memoryless
  - Outputs determined by current values of inputs
- Sequential Logic
  - Has memory
  - Outputs determined by previous and current values of inputs



Rules of Combinational Composition

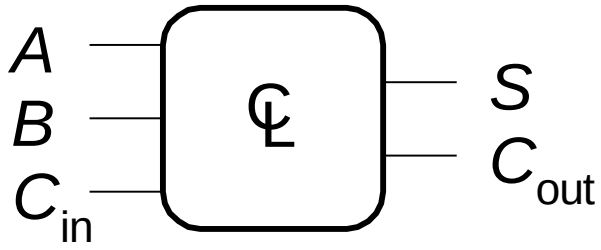
- Every circuit element is itself combinational
- Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element
- The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once
- Example:



2.2 Boolean Equations (부울식)

- Functional specification of outputs in terms of inputs
- Example:

$$S = \mathrm{F}(A, B, C_{\text{in}})$$
$$C_{\text{out}} = \mathrm{F}(A, B, C_{\text{in}})$$



$$S = A \oplus B \oplus C_{\text{in}}$$
$$C_{\text{out}} = AB + AC_{\text{in}} + BC_{\text{in}}$$

Equations (기본 논리식의 표현)

- 기본적인 불 대수식은 AND, OR, NOT을 이용하여 표현
- AND 식은 곱셈의 형식으로 표현하고, OR 식은 덧셈의 형식으로 표현
- NOT 식은 $\overline{X}$ 또는 X'로 표현
- 완전한 논리식은 입력 항목들의 상태에 따른 출력을 결정하는 식

• *X*=0 and *Y*=1 일 때 출력을 1로 만들려는 경우 출력 논리식

$$F = \overline{X}Y$$

• *X*=0 or *Y*=1 일 때 출력을 1로 만들려는 경우 출력 논리식

$$F = \overline{X} + Y$$

• (*X*=0 and *Y*=1) or (*X*=1 and *Y*=0) 일 때 출력을 1로 만들려는 경우 출력 논리식 $F = \overline{X}Y + X\overline{Y}$

Sum-of-Products (SOP) Form (곱의 합 형태)

- All Boolean equations can be written in SOP form
- Each row in a truth table has a **minterm**(함수에 모든 변수를 포함)
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- The function is formed by ORing the minterms for which the output is TRUE
- Thus, a sum (OR) of products (AND terms)

  (1 단계는 AND항(곱의 항, product term)으로 구성되고,)

  (2 단계는 OR항(합의 항, sum term)으로 만들어진 논리식)

A	*B*	*Y*	minterm	minterm name
0	0	0	$\overline{A}\,\overline{B}$	m_0
0	1	1	$\overline{A}\,B$	m_1
1	0	0	$A\,\overline{B}$	m_2
1	1	1	$A\,B$	m_3

$$Y = \mathrm{F}(A, B, C) = \overline{A}B + AB = \Sigma(1, 3)$$

Product-of-Sums (POS) Form

- All Boolean equations can be written in POS form
- Each row in a truth table has a **maxterm**
- A maxterm is a sum (OR) of literals
- Each maxterm is FALSE for that row (and only that row)
- The function is formed by ANDing the maxterms for which the output is FALSE
- Thus, a product (AND) of sums (OR terms)

  (1단계는 OR항(합의 항, sum term)으로 구성되고,)

  (2 단계는 AND항(곱의 항, product term)으로 만들어진 논리식)

A	*B*	*Y*	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

$$Y = \mathrm{F}(A, B, C) = (A + B)(\overline{A} + B) = \Pi(0, 2)$$

Some Definitions

- Minterm: product that includes all input variables
  ABC', AB'C, A'BC'

- Maxterm: sum that includes all input variables
  (A'+B'+C), (A'+B+C'), (A+B'+C)

Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won't eat lunch ($\overline{E}$)
  - If it's not open ($\overline{O}$) or
  - If they only serve corndogs (C)
- Write a truth table for determining if you will eat lunch (E).

O	*C*	*E*
0	0	0
0	1	0
1	0	1
1	1	0

SOP & POS Form

- SOP – sum-of-products

O	*C*	*E*	minterm
0	0	0	$\overline{O}\ \overline{C}$
0	1	0	$\overline{O}\ C$
1	0	1	$O\ \overline{C}$
1	1	0	$O\ C$

$$E = OC'$$
$$= \Sigma(2)$$

- POS – product-of-sums

O	*C*	*E*	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$E = (O+C)(O+C')(O'+C')$$
$$= \Pi(0, 1, 3)$$

2.3 Boolean Algebra (부울 대수)

- Set of axioms and theorems to **simplify** Boolean equations
  (부울식을 간단히 하기 위해 부울 대수를 사용한다.)

- Like regular algebra, but in some cases simpler because variables can have **only two values** (1 or 0)

- Axioms and theorems obey the principles of **duality**:
  - ANDs and ORs interchanged, 0's and 1's interchanged

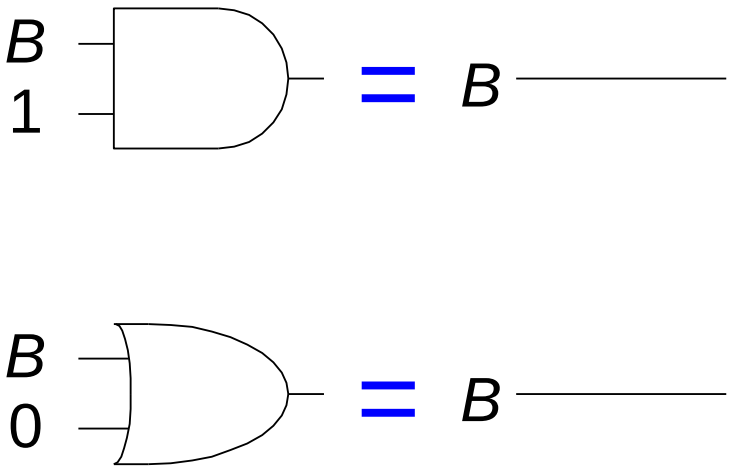
Boolean Axioms and Theorems

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

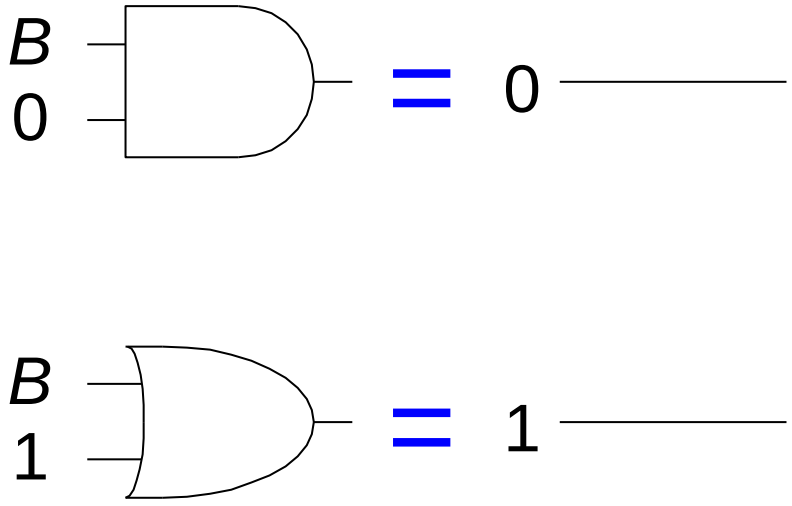
T1: Identity Theorem

- $B \bullet 1 = B$
- $B + 0 = B$



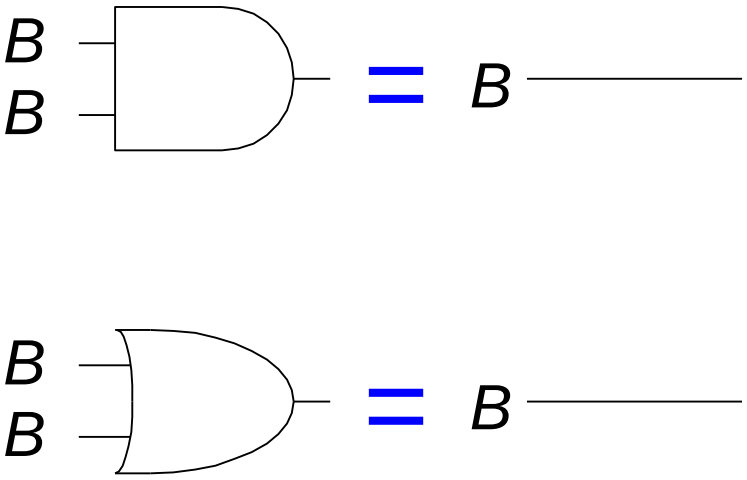
T2: Null Element Theorem

- $B \bullet 0 = 0$
- $B + 1 = 1$



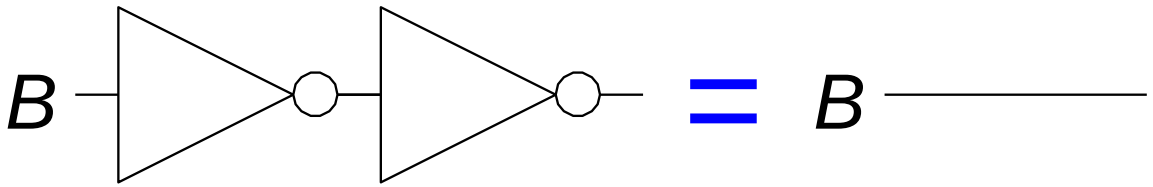
T3: Idempotency Theorem

- $B \bullet B = B$
- $B + B = B$



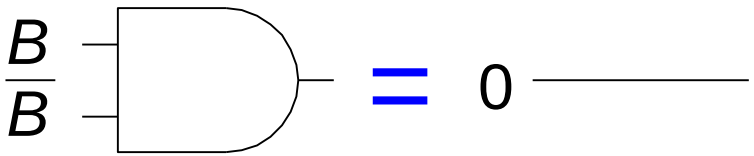
T4: Identity Theorem

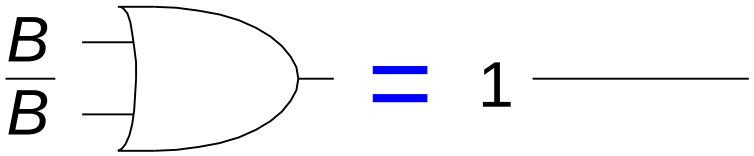
- $\overline{\overline{B}} = B$



T5: Complement Theorem

- $B \bullet \overline{B} = 0$
- $B + \overline{B} = 1$





Boolean Theorems of Several Variables

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + B \bullet D = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D) = B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D) = (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots} = (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

Simplifying Boolean Expressions: Example 1

- $Y = A\overline{B} + AB$

  $= A(B + \overline{B})$ T8

  $= A(1)$ T5’

  $= A$ T1

Simplifying Boolean Expressions: Example 2

- $Y = A(AB + ABC)$

$= A(AB(1 + C))$	T8
$= A(AB(1))$	T2’
$= A(AB)$	T1
$= (AA)B$	T7
$= AB$	T3

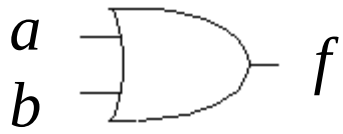
Examples

- 진리표로부터 SOP(sum-of-products(최소항식)) 형태로 논리식을 표현하고 식을 간소화 하시오. 또한 게이트 회로를 설계 하시오.

입력		출력
a	b	f
0	0	0
0	1	1
1	0	1
1	1	1

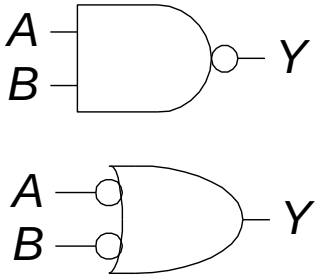
$$\Rightarrow \quad f = \bar{a}b + a\bar{b} + ab$$

$$\begin{aligned} f &= a'b + ab' + ab + ab \\ &= a(b'+b) + b(a'+a) \\ &= a+b \end{aligned}$$



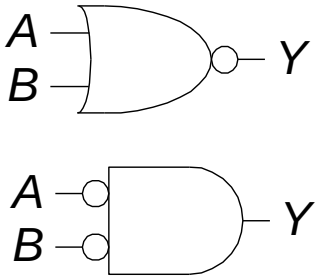
DeMorgan's Theorem

- $Y = \overline{AB} = \overline{A} + \overline{B}$



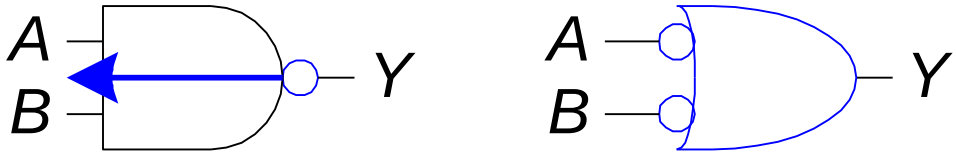
Bubble Pushing

- $Y = \overline{A + B} = \overline{A} \bullet \overline{B}$

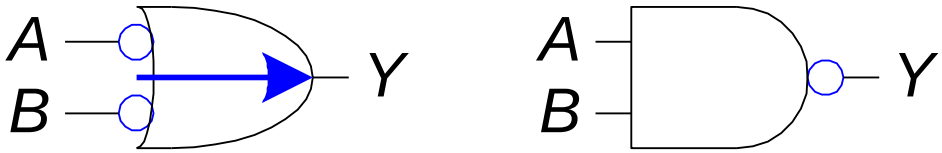


2.5.2 Bubble Pushing (버블 추가)

- Pushing bubbles backward (from the output) or forward (from the inputs) changes the body of the gate from AND to OR or vice versa.
- Pushing a bubble from the output back to the inputs puts bubbles on all gate inputs.

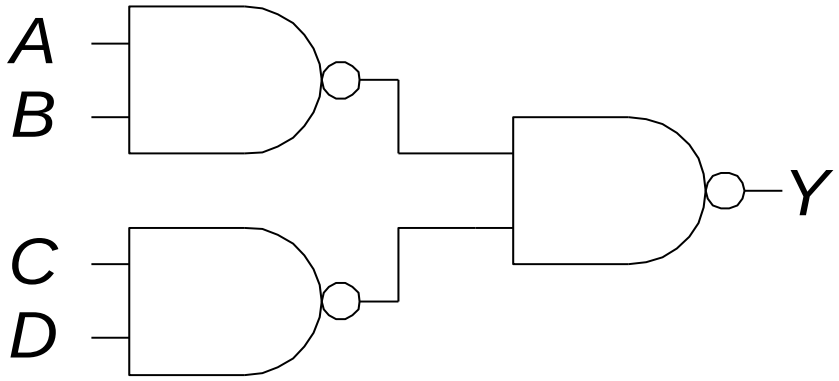


- Pushing bubbles on *all* gate inputs forward toward the output puts a bubble on the output and changes the gate body.



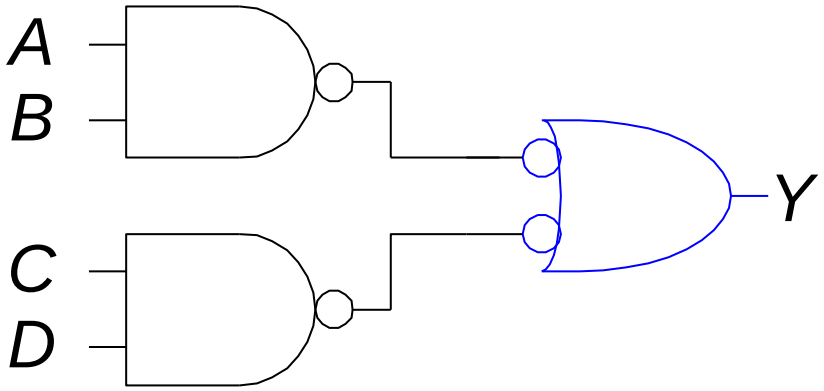
Bubble Pushing

- What is the Boolean expression for this circuit?



Bubble Pushing

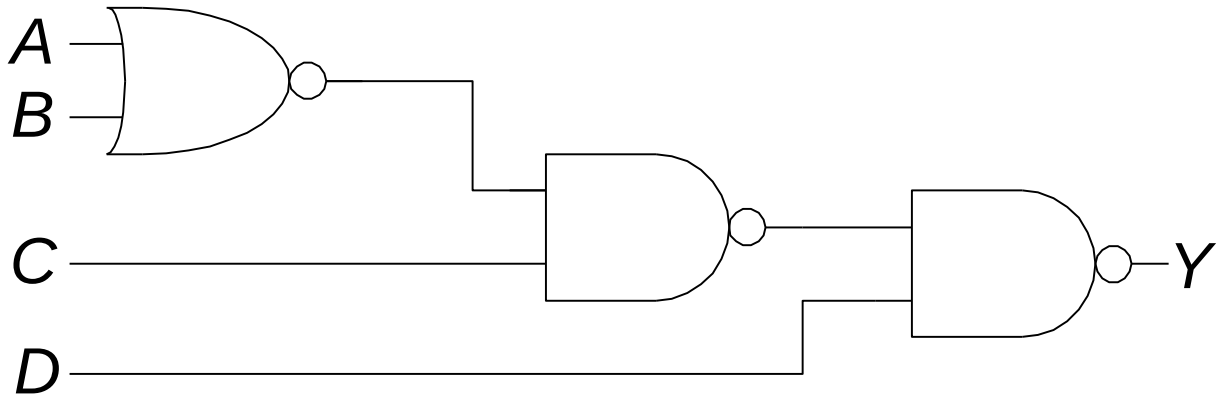
- What is the Boolean expression for this circuit?



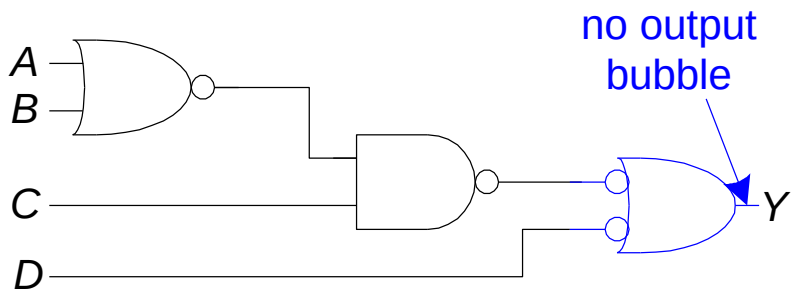
$$Y = AB + CD$$

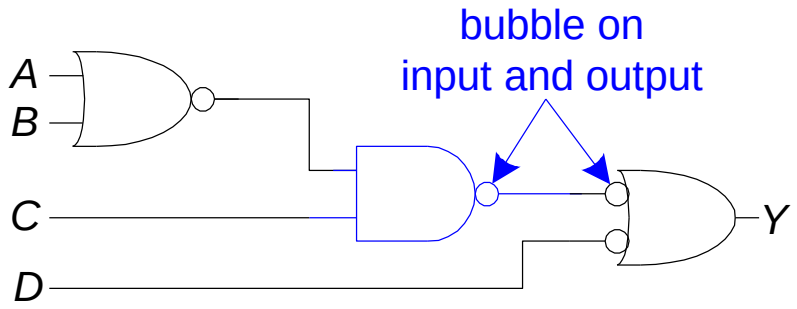
Bubble Pushing Rules

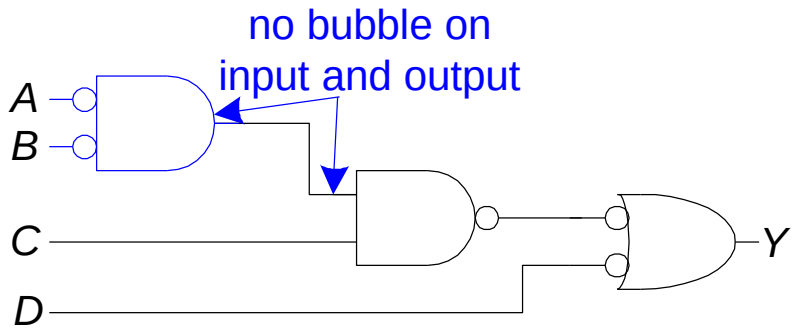
- Begin at the output of the circuit and work toward the inputs. (출력부분에서 시작하여 입력 부분으로 진행)
- Push any bubbles on the final output back toward the inputs.
- Working backward, draw each gate in a form so that bubbles cancel. (버블이 없어지도록 각 게이트 변경)



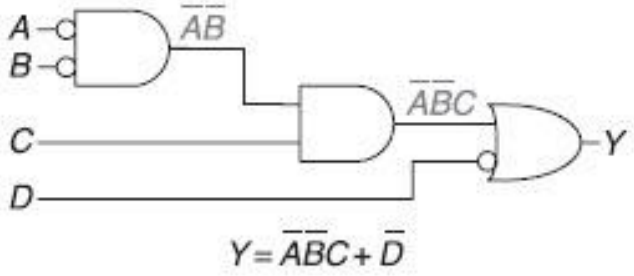
Bubble Pushing Rules





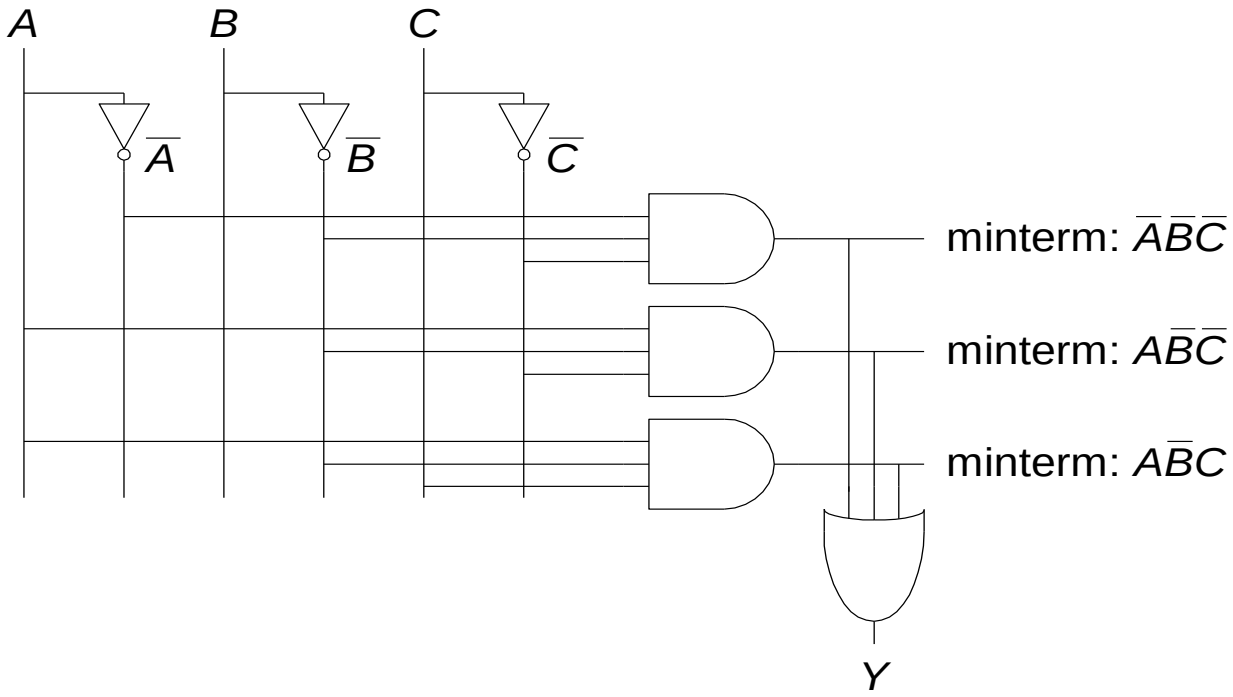


$Y = \overline{A}\,\overline{B}C + \overline{D}$



2.4 From Logic to Gates (논리에서 게이트로)

- Two-level logic: ANDs followed by Ors
- Example: $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$
- Schematic(개요도)
  - 서로 연결된 선과 요소를 보여주는 digital 회로의 diagram



Circuit Schematic Rules (개요도 작성 규칙)

- Inputs are on the left (or top) side of a schematic
  (입력은 개요도의 왼쪽이나 위쪽에 있다.)

- Outputs are on the right (or bottom) side of a schematic
  (출력은 개요도의 오른쪽이나 아래쪽에 있다.)

- Whenever possible, gates should flow from left to right
  (개요도는 왼쪽에서 오른쪽으로 흐른다.)

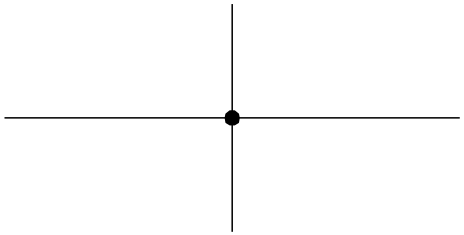
- Straight wires are better to use than wires with multiple corners
  (가능한 꺽이선보다는 곧은 직선을 사용한다.)

Circuit Schematic Rules (cont.)

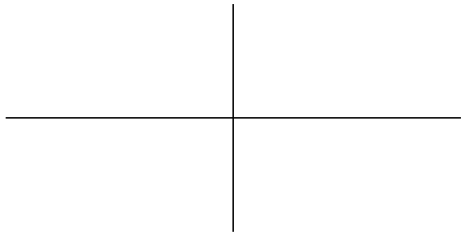
- Wires always connect at a T junction
  (선은 항상 T 접합으로 연결한다.)
- A dot where wires cross indicates a connection between the wires
- Wires crossing *without* a dot make no connection

wires connect at a T junction

wires connect at a dot

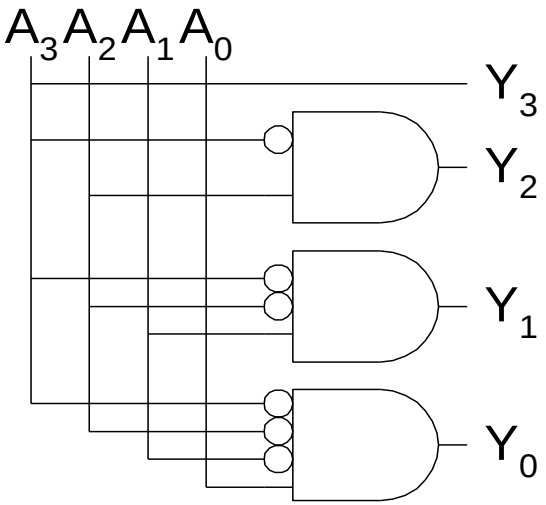


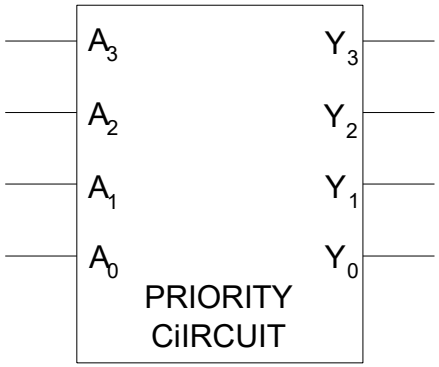
wires crossing without a dot do not connect



Priority Circuit Hardware (예제 2.7)

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



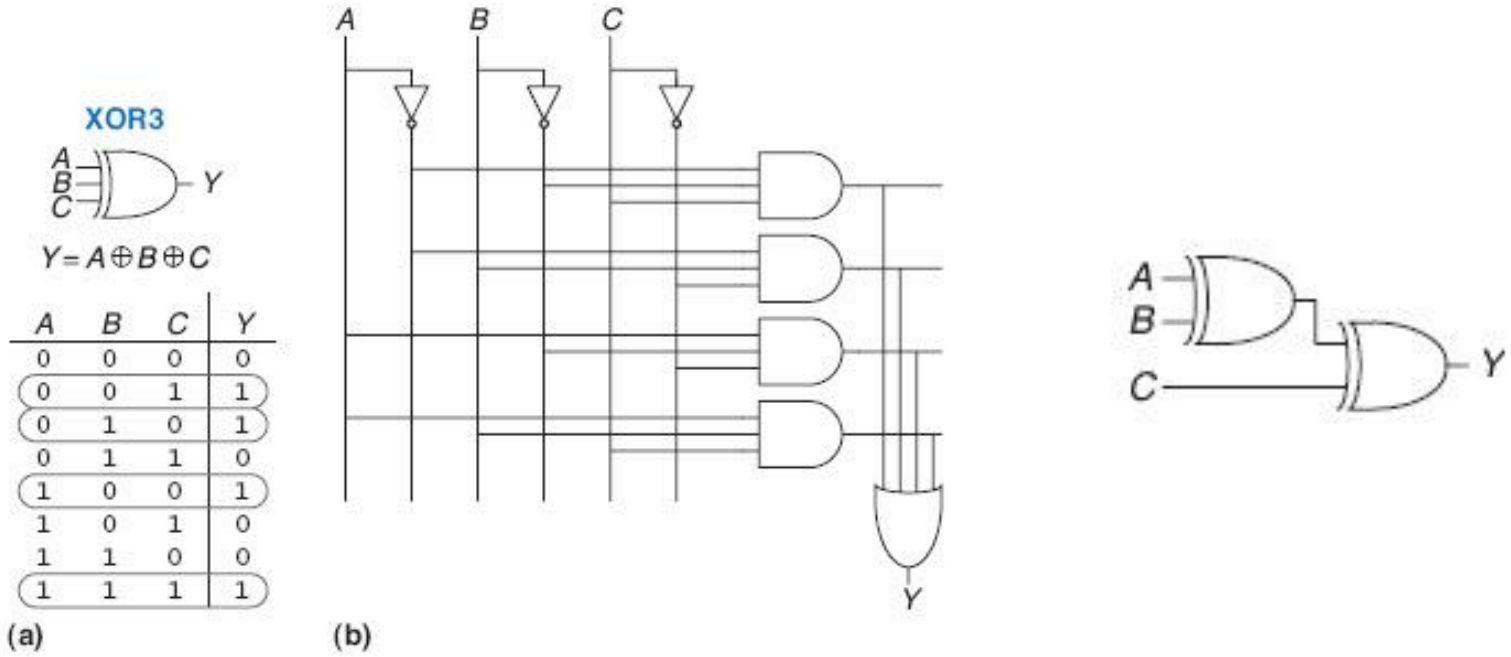


진리표에서의 X는
Don't care(무관항)

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	x	0	0	1	0
0	1	x	x	0	1	0	0
1	x	x	x	1	0	0	0

2.5 Multiple Output Circuits(다단계 조합논리)

- 하드웨어 축소
  - 다단계 조합논리는 더 적은 하드웨어를 사용할 수 있다.
  - $A \oplus B \oplus C = (A \oplus B) \oplus C$



2.6 Don’t Cares

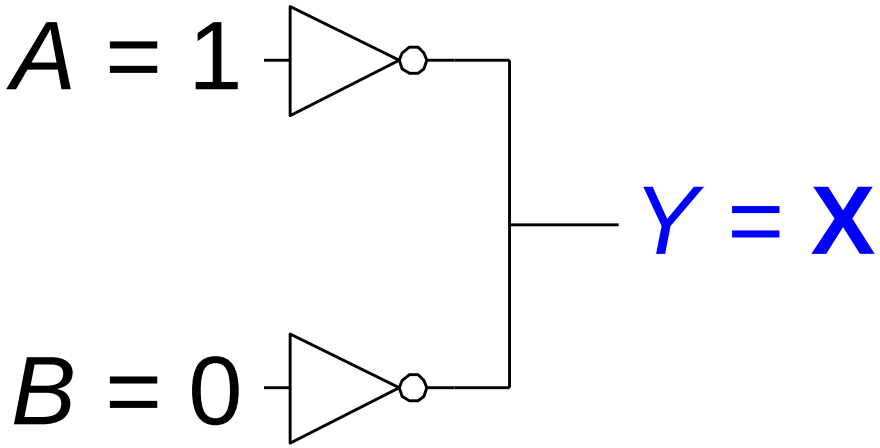
- 진리표에서의 X는 Don’t care(무관항)

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	x	0	0	1	0
0	1	x	x	0	1	0	0
1	x	x	x	1	0	0	0

Contention: X

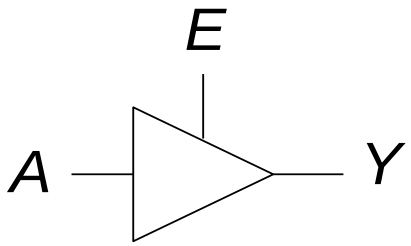
- 회로에서의 X는 Contention(경쟁)
  - circuit tries to drive the output to 1 and 0
  - 금지값 or 잘 알려지지 않은 값



Floating: Z

- Floating(부동값) Z : high나 low로 구동되지 않음.
- Said to be Floating, high impedance, open, high Z
- Output is not connected to the input

Tristate Buffer



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

2.7 Karnaugh Maps (K-Maps)

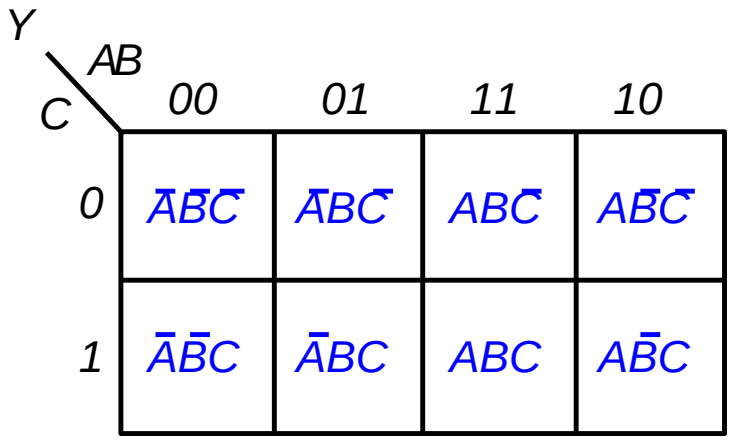
- Boolean expressions can be minimized by combining terms
- K-maps minimize equations graphically
- (부울식을 간단히 하기 위하여 그래프로 나타내는 방법)
- $PA + P\overline{A} = P$

A	B	C	Y
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y: C \ AB	00	01	11	10
0	**1**	0	0	0
1	**1**	0	0	0

Y: C \ AB	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

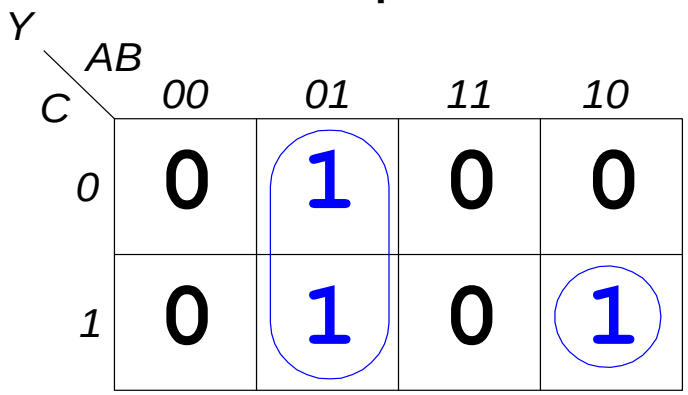
K-map Example



Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	0
1	1	1	0

K-Map



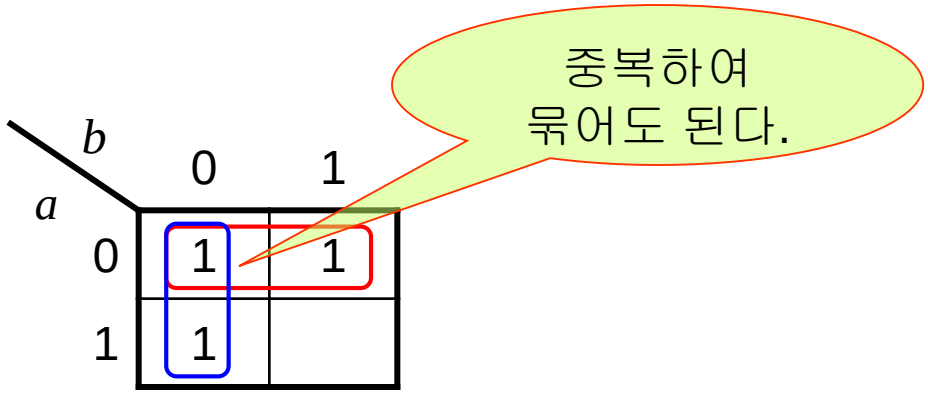
$$Y = AB + A\bar{B}C$$

K-map Rules

- Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction. (각각의 원은 2의 제곱형태의 크기를 가진 직사각형 블록으로)
- Each circle must be as large as possible (각각의 원은 가능한 크게)
- A circle may wrap around the edges of the K-map (원은 맵의 가장자리를 둘러 쌀 수 있고)
- A one in a K-map may be circled multiple times (1은 여러 원에 묶일 수도 있고)
- A “don't care” (X) is circled only if it helps minimize the equation (X는 식을 최소화하다면 묶고)

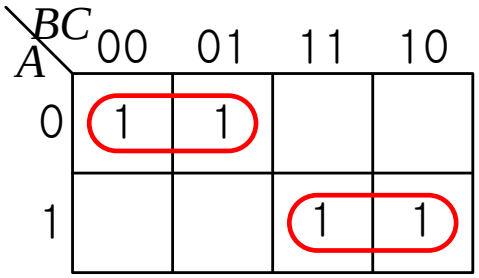
Karnaugh Maps 간략화 예

a	b	f
0	0	1
0	1	1
1	0	1
1	1	0

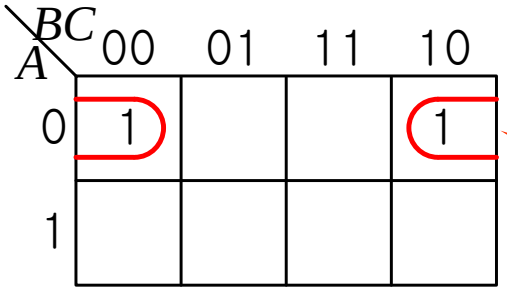


$$f = \bar{a} + \bar{b}$$

Karnaugh Maps 간략화 예

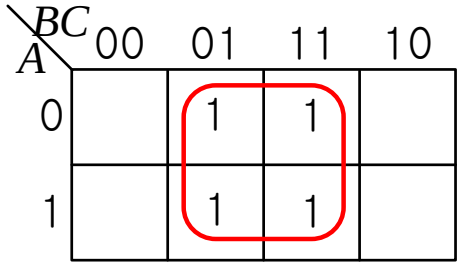


$F = \overline{A}\overline{B} + AB$

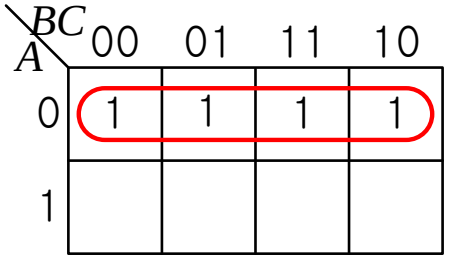


양쪽 끝은 연결되어 있다.

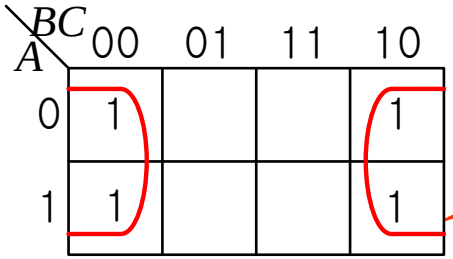
$F = \overline{A}\overline{C}$



$F = C$



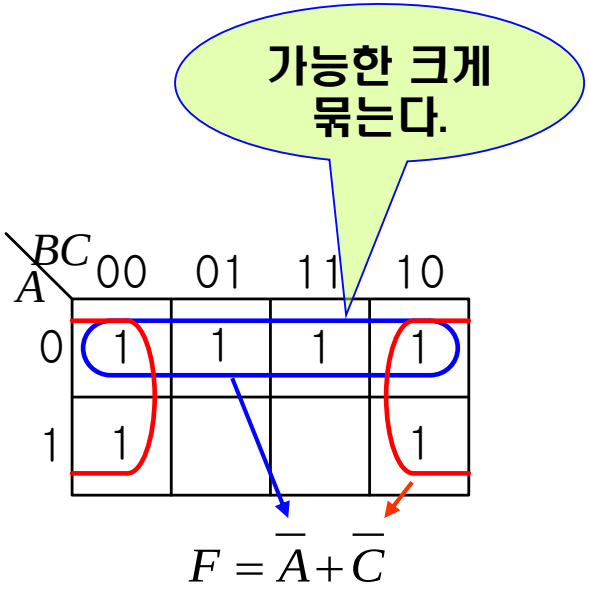
$F = \overline{A}$

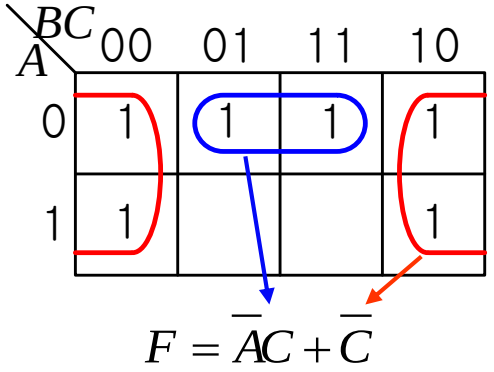


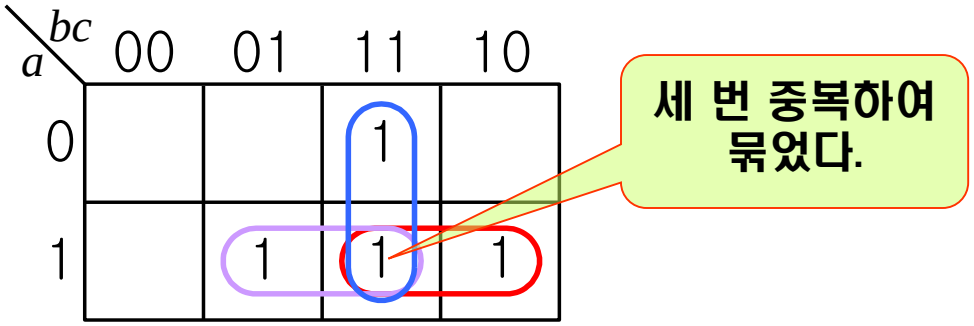
양쪽 끝은 연결되어 있다.

$F = \overline{C}$

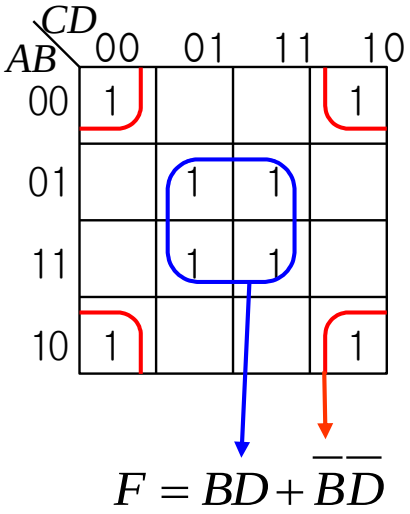
Karnaugh Maps 간략화 예



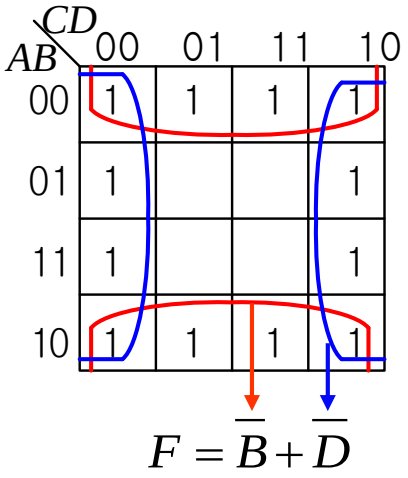




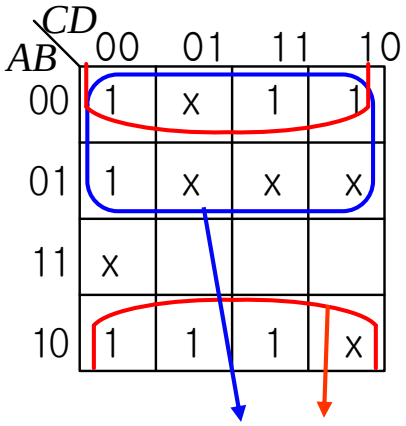
Karnaugh Maps 간략화 예



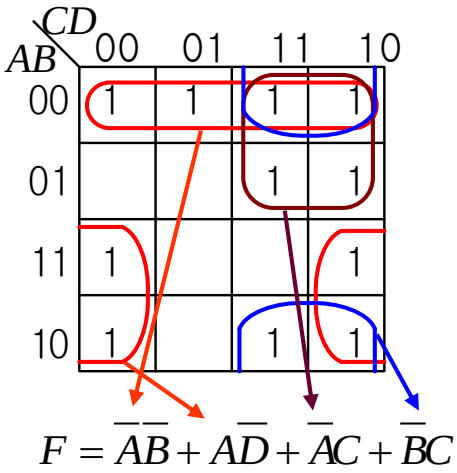
$$F = BD + \overline{B}\,\overline{D}$$



$$F = \overline{B} + \overline{D}$$

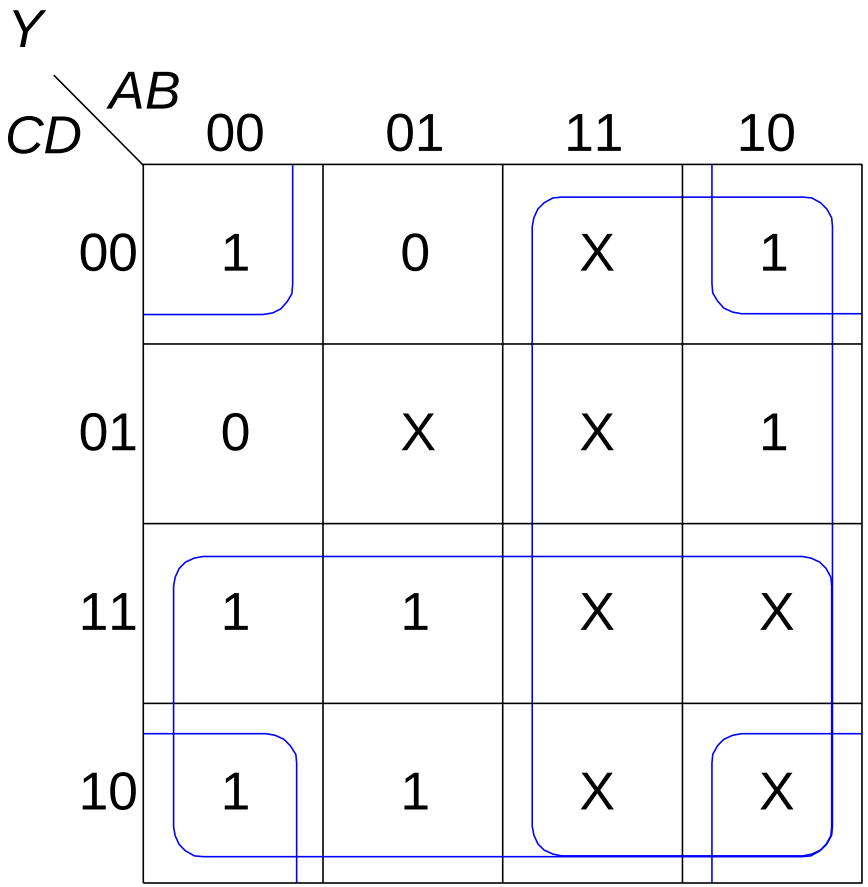


$$F = \overline{A} + \overline{B}$$



$$F = \overline{A}\,\overline{B} + A\overline{D} + \overline{A}C + \overline{B}C$$

K-Maps with Don't Cares

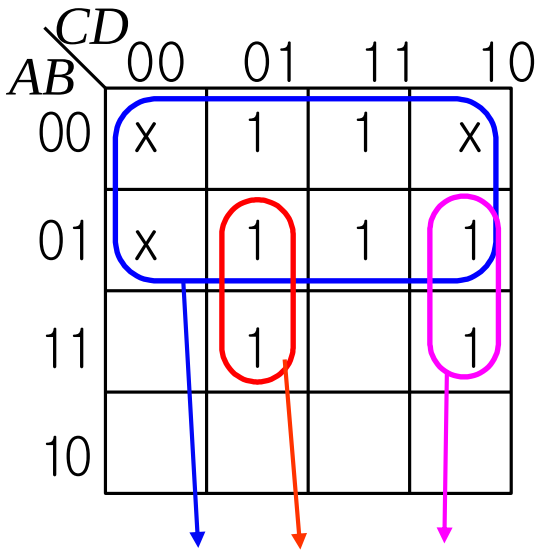


$$Y = A + \overline{B}\,\overline{D} + C$$

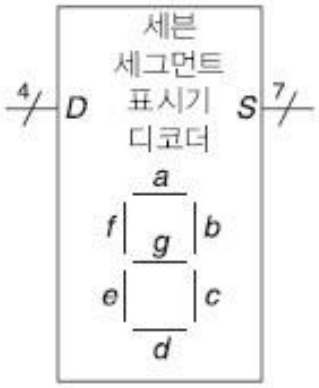
K-Maps 예제

- 다음 진리표로부터 카르노 맵을 작성하고 간략화하여라.

A B C D	F
0 0 0 0	x
0 0 0 1	1
0 0 1 0	x
0 0 1 1	1
0 1 0 0	x
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
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	1
1 1 1 1	0



예제 **2.10) 7-**세그먼트 디코더



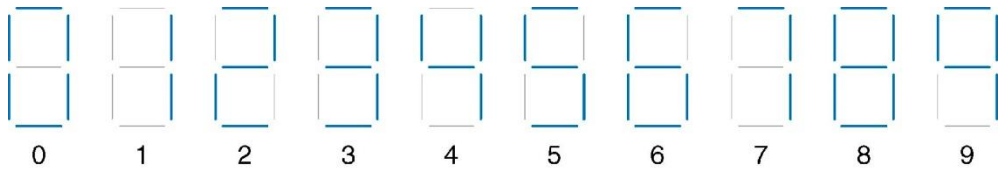


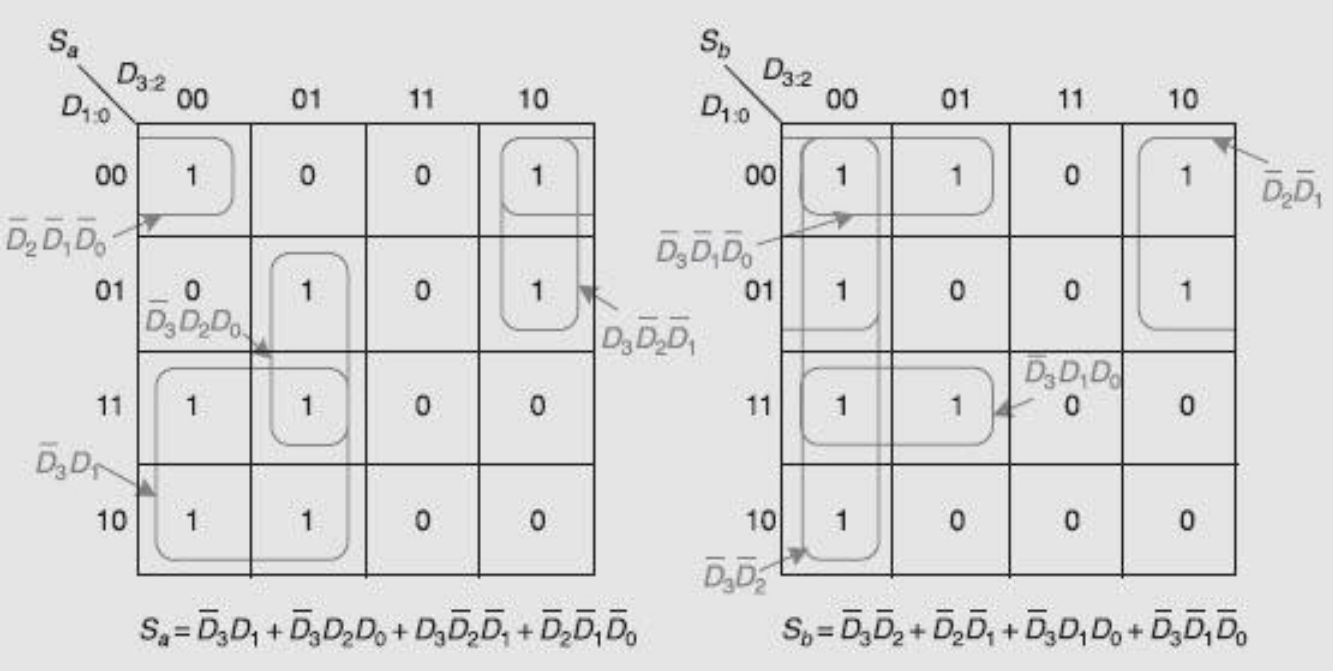
Table 2.6 Seven-segment display decoder truth table

$D_{3:0}$	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

예제 **2.10**

Table 2.6 Seven-segment display decoder truth table

$D_{3:0}$	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1					
0100	0	1					
0101	1	0					
0110	1	0					
0111	1	1					
1000	1	1					
1001	1	1					
others	0	0					



2.8 Combinational Building Blocks

- full-adder (전가산기) : 그림 2.5 참조
- Multiplexers
- Decoders

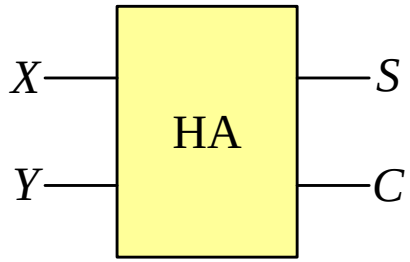
Half-Adder

- Half-Adder (반가산기)란?
  - 두 개의 2진수 한자리를 입력(X, Y)하여 합(Sum)과 자리올림수(Carry)를 얻는 덧셈 논리회로
  - 반가산기는 더하기 수행 시 자리올림을 무시
  - 예를 들어 5+6의 내용을 입력하면 반가산기의 출력은 올림자리는 무시하고 1이 출력됨
  - 자리올림을 포함하는 계산을 위해서는 전가산기를 사용해야 함

Half-Adder 논리식과 진리표

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

입력		출력	
X	***Y***	***S***	***C***
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

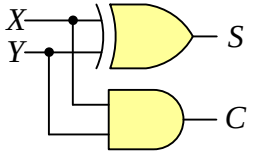


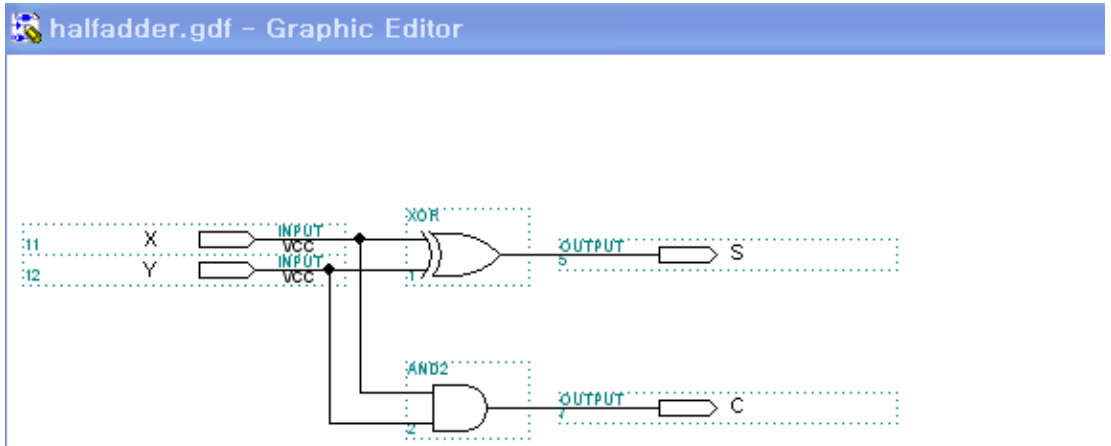
$$S = \overline{X}Y + X\overline{Y} = X \oplus Y$$

$$C = X \cdot Y$$

Half-Adder 논리회로

- 논리회로 작성 및 컴파일
- Timing Simulation 결과 파형





Full-Adder

- Full-Adder(전가산기)란?
  - 두 개의 2진수 입력(X, Y)과 하위 자리에서 발생한 자리올림수($Carry_{in}$) 를 포함하여 한 자리수 2진수 세 개의 입력 비트들의 덧셈을 수행하여 합(Sum)과 자리올림수($Carry_{out}$)를 얻는 논리회로

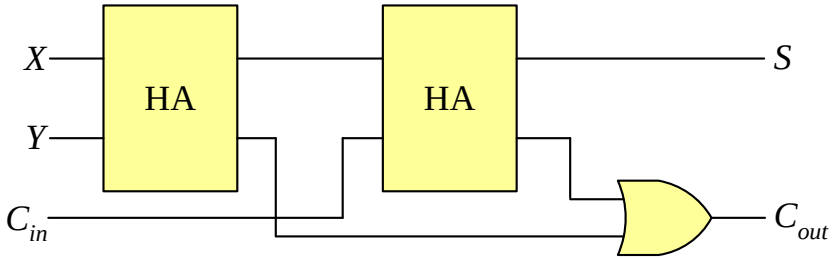
Full-Adder 논리식과 진리표

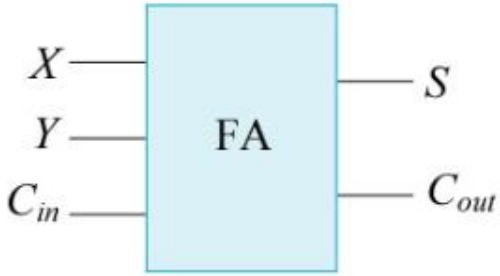
입력			출력	
X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) 진리표

$$S = X \oplus Y \oplus C_{in}$$
$$C_{out} = XY + (X \oplus Y) C_{in}$$

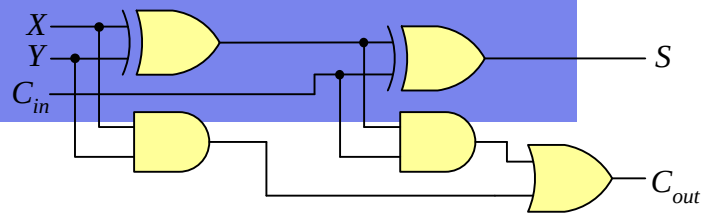
(b) 논리식



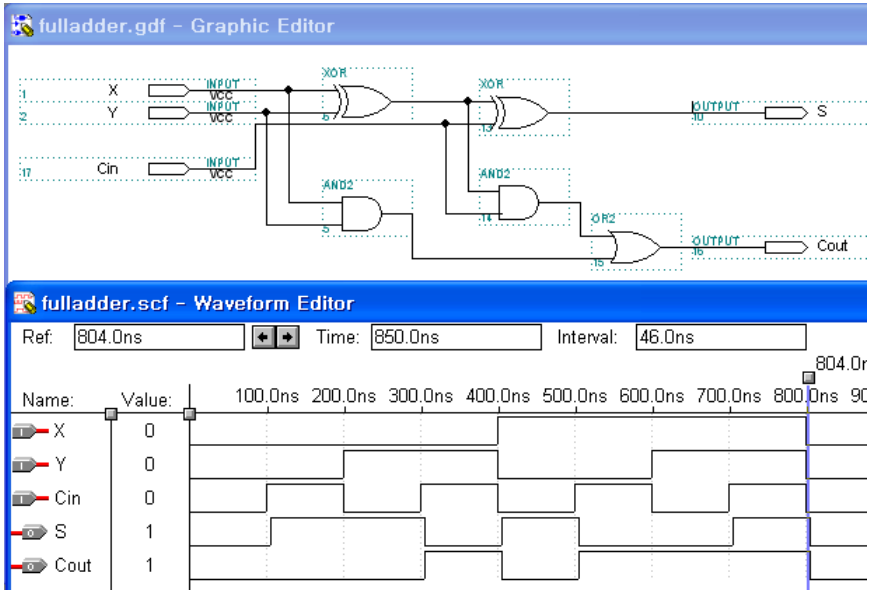


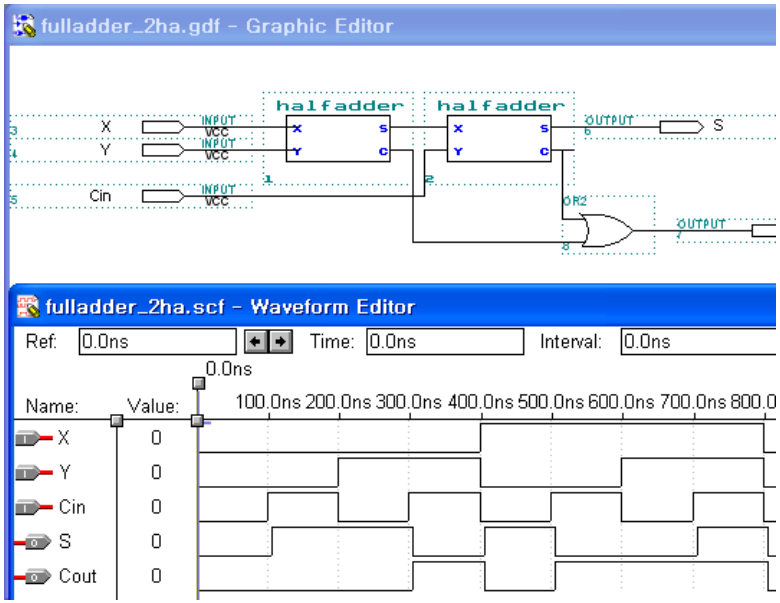
(c) 논리기호

Full-Adder 논리회로



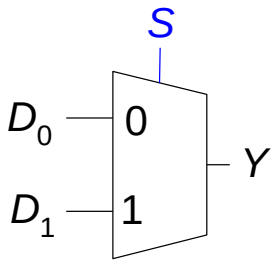
- 논리회로 작성 및 컴파일
- Timing Simulation 결과 파형
  - 조합논리회로사용, HalfAdder 두 개 사용)





2.8.1 Multiplexer (Mux)

- Selects between one of N inputs to connect to the output.
  (여러 가지 가능한 입력 중에서 출력을 선택)
- $\log_2 N$-bit select input – control input
- Example: 2:1 Mux

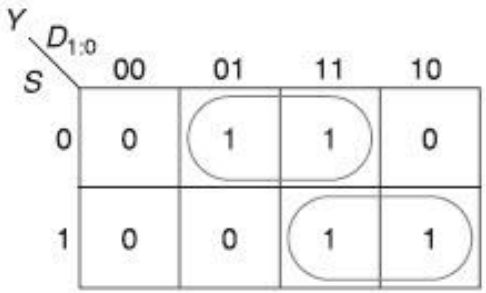


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

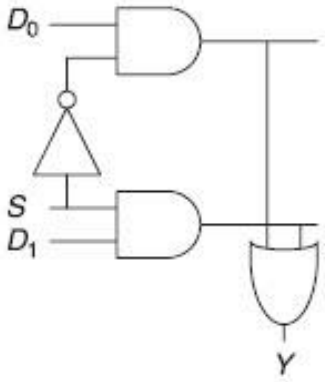
S	Y
0	D_0
1	D_1

2:1 Mux

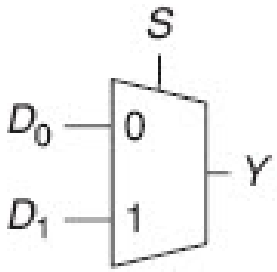
- **Logic gates**
  - Sum-of-products form

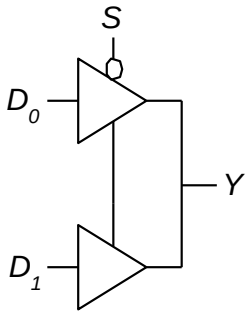


$Y = D_0\bar{S} + D_1 S$

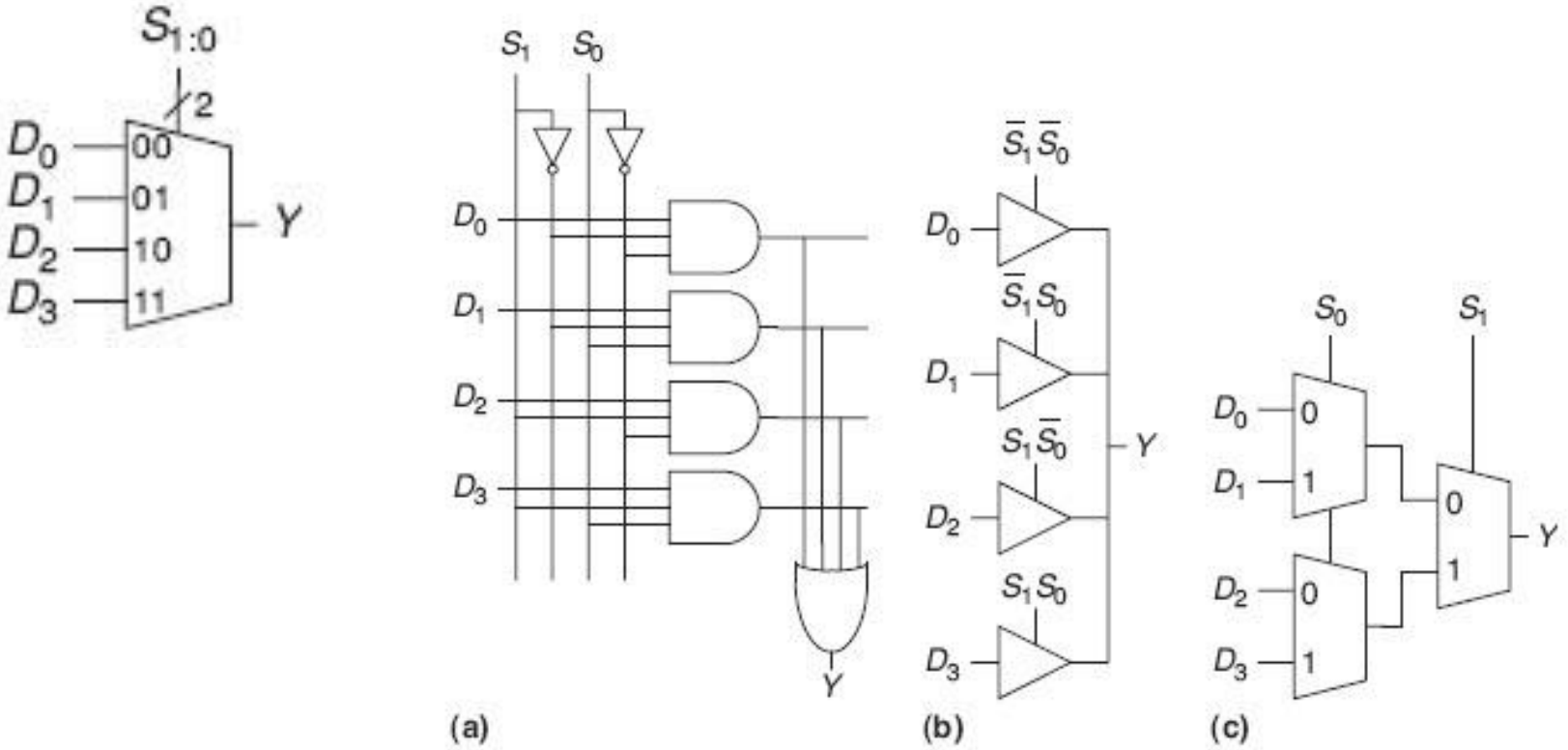


- **Tristates**
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input





4:1 Mux

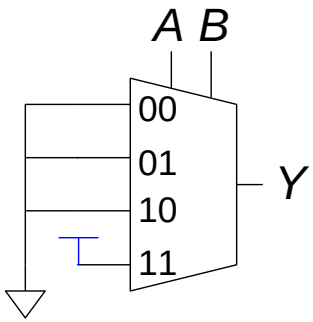

$S_{1:0}$
2
D_0
D_1
D_2
D_3
00
01
10
11
Y
S_1
S_0
(a)
$\bar{S}_1\bar{S}_0$
$\bar{S}_1 S_0$
$S_1\bar{S}_0$
$S_1 S_0$
(b)
(c)

Logic using Multiplexers

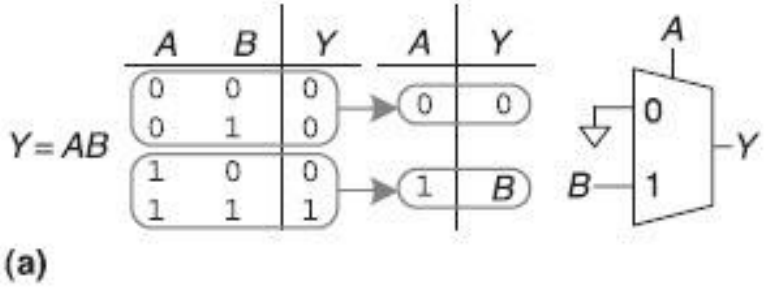
- Using the mux as a lookup table

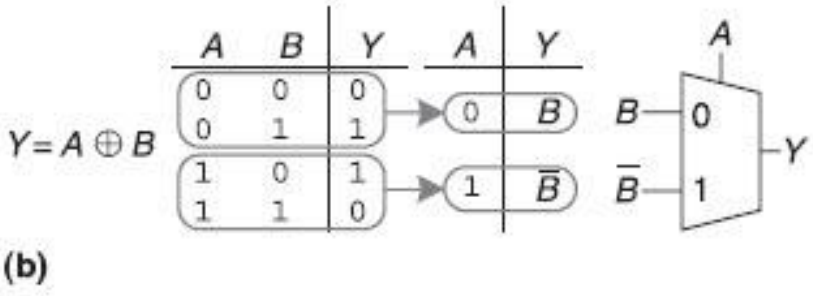
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$



- Reducing the size of the mux



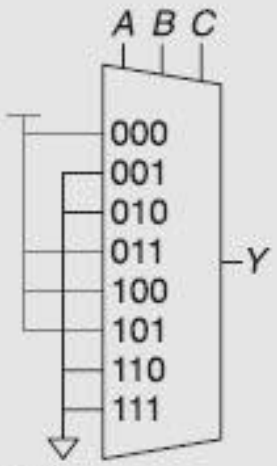


예제 **2.12, 13**

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC$

(a)



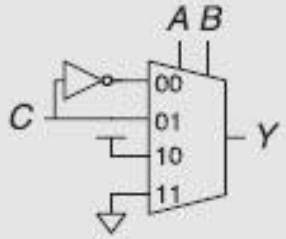
(b)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(a)

A	B	Y
0	0	$\overline{C}$
0	1	C
1	0	1
1	1	0

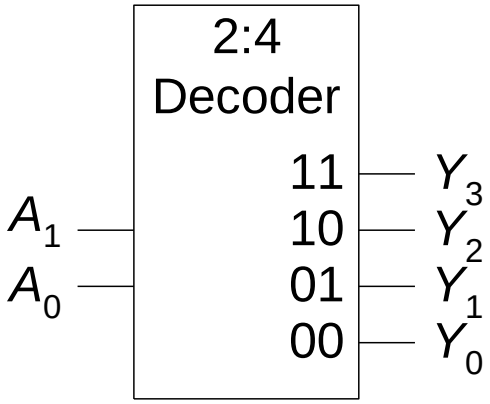
(b)



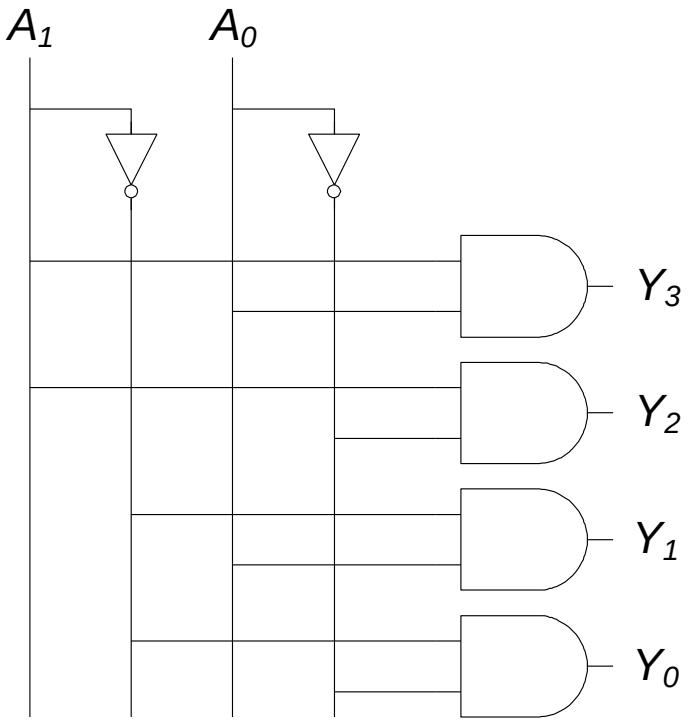
(c)

2.8.2 Decoders

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once

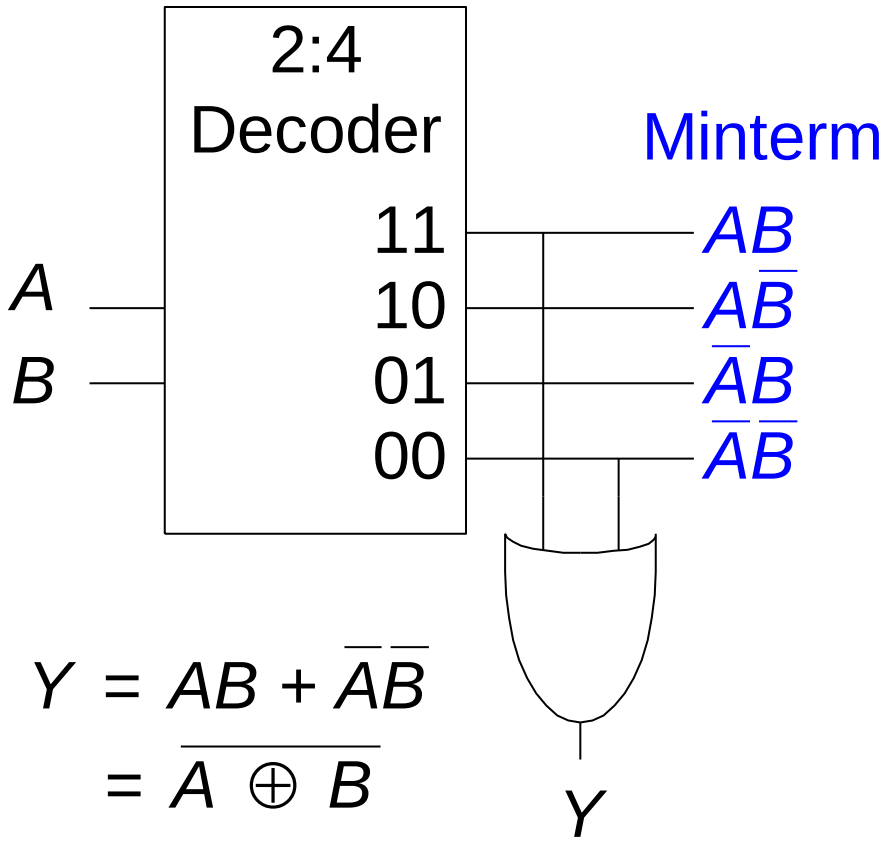


A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Logic using Decoders

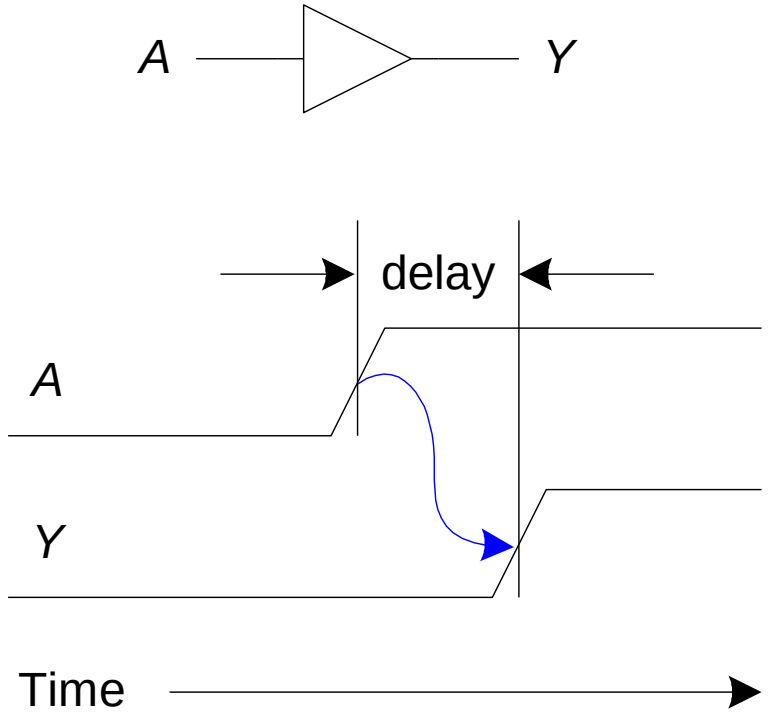
- OR minterms
- XNOR



$$Y = AB + \overline{A}\overline{B}$$
$$= \overline{A \oplus B}$$

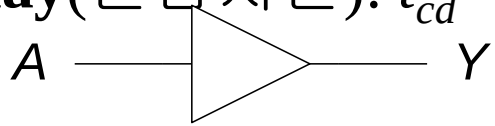
2.9 Timing

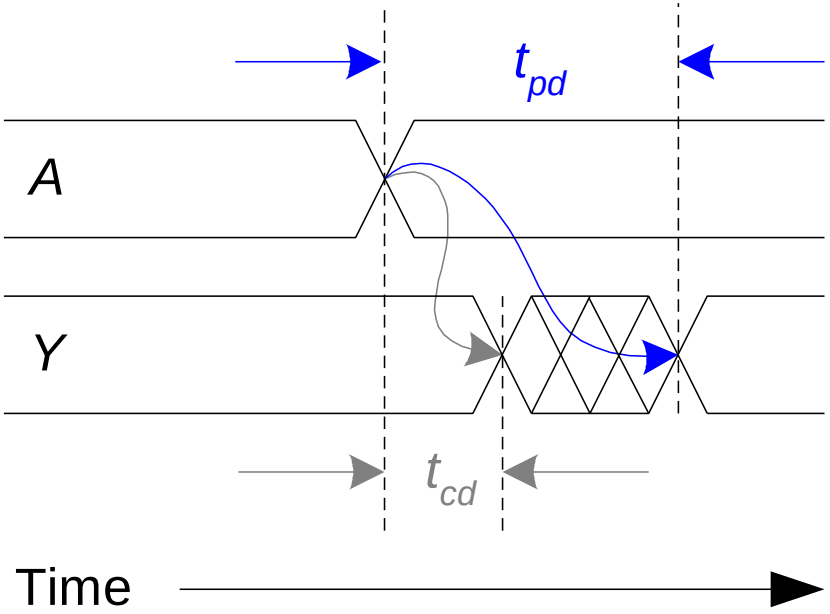
- Delay between input change and output changing
- One of the biggest challenges in circuit design: making the circuit fast



Propagation & Contamination Delay

- **Propagation delay**(전파지연): t_{pd} = max delay from input to output
- **Contamination delay**(혼합지연): t_{cd} = min delay from input to output



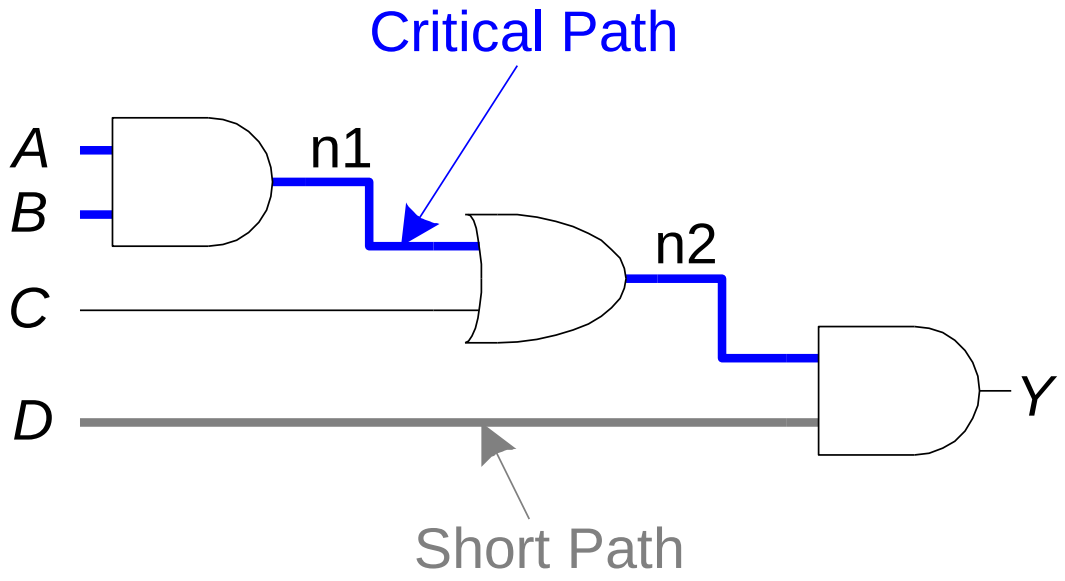


Propagation & Contamination Delay

- Delay is caused by
  - Capacitance and resistance in a circuit
  - Speed of light limitation

- Reasons why t_{pd} and t_{cd} may be different:
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold

Critical and Short Paths



Critical (Long) Path: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

Short Path: $t_{cd} = t_{cd\_AND}$

Glitch, Why Understand Glitches?

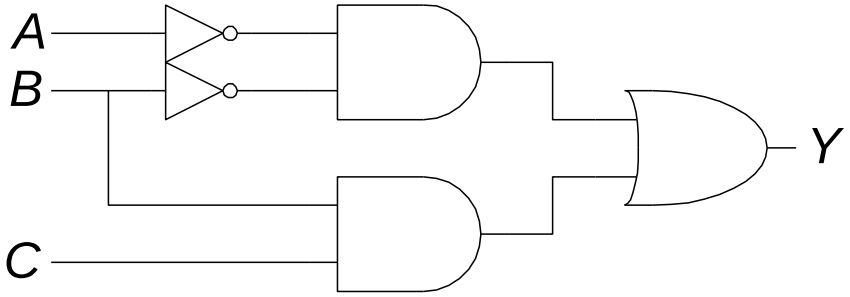
- A **glitch** occurs when a single input change causes multiple output changes
  (글리치는 하나의 입력변화가 다수의 출력변화의 원인이 될 때 발생)

- Glitches don't cause problems because of **synchronous design** conventions (which we'll talk about in Chapter 3)

- But it's important to **recognize a glitch** when you see one in simulations or on an oscilloscope
  (글리치의 존재를 인식하는 것이 중요하다)

- **Can't get rid of all glitches** – simultaneous transitions on multiple inputs can also cause glitches

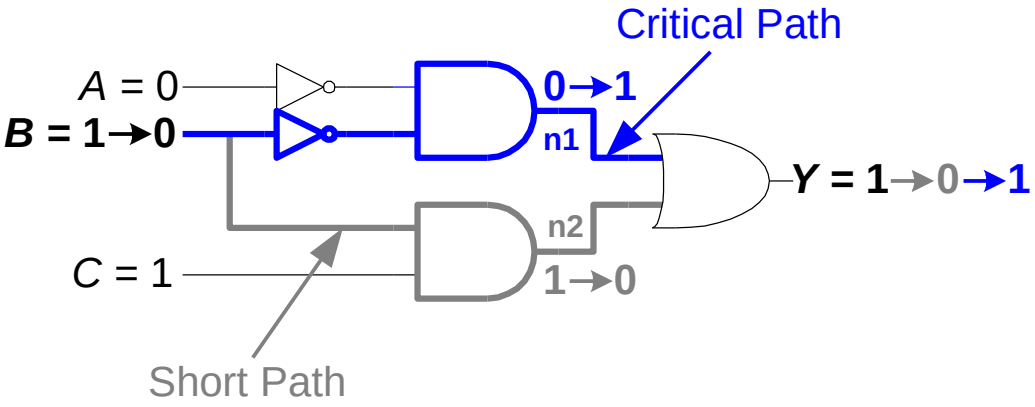
Glitch Example

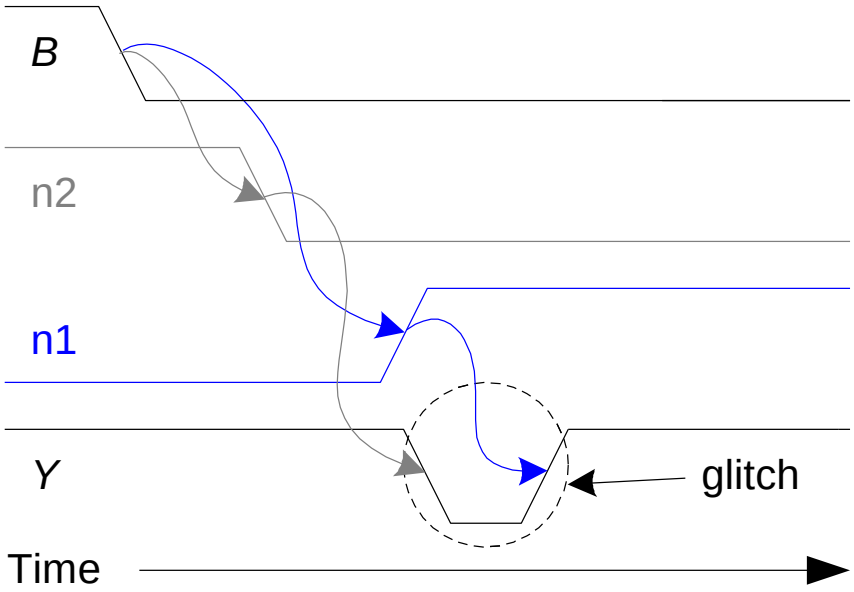


Y	AB = 00	01	11	10
C = 0	1	0	0	0
1	1	1	1	0

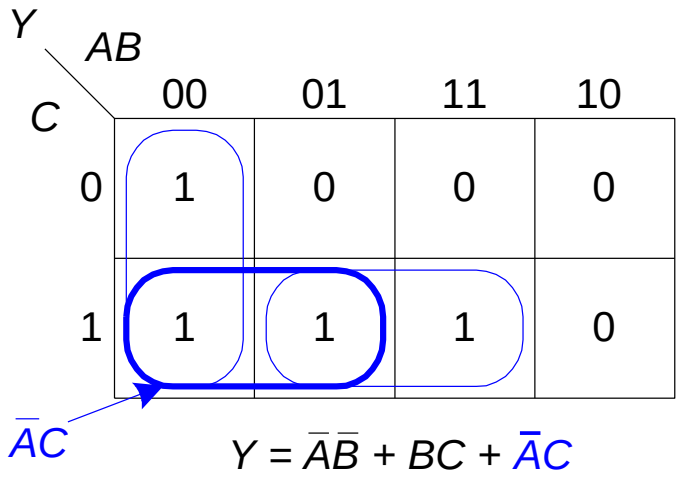
$$Y = \bar{A}\bar{B} + BC$$

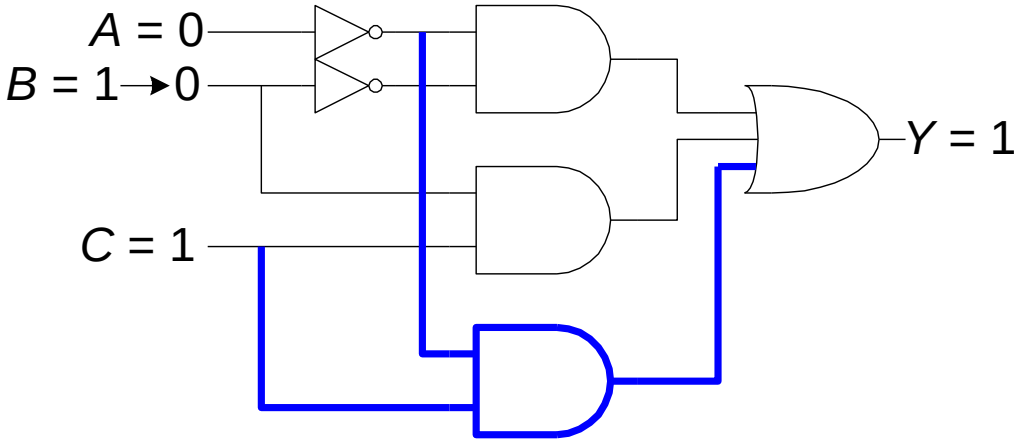
Glitch Example (cont.)





Glitch Example (cont.)





Q & A