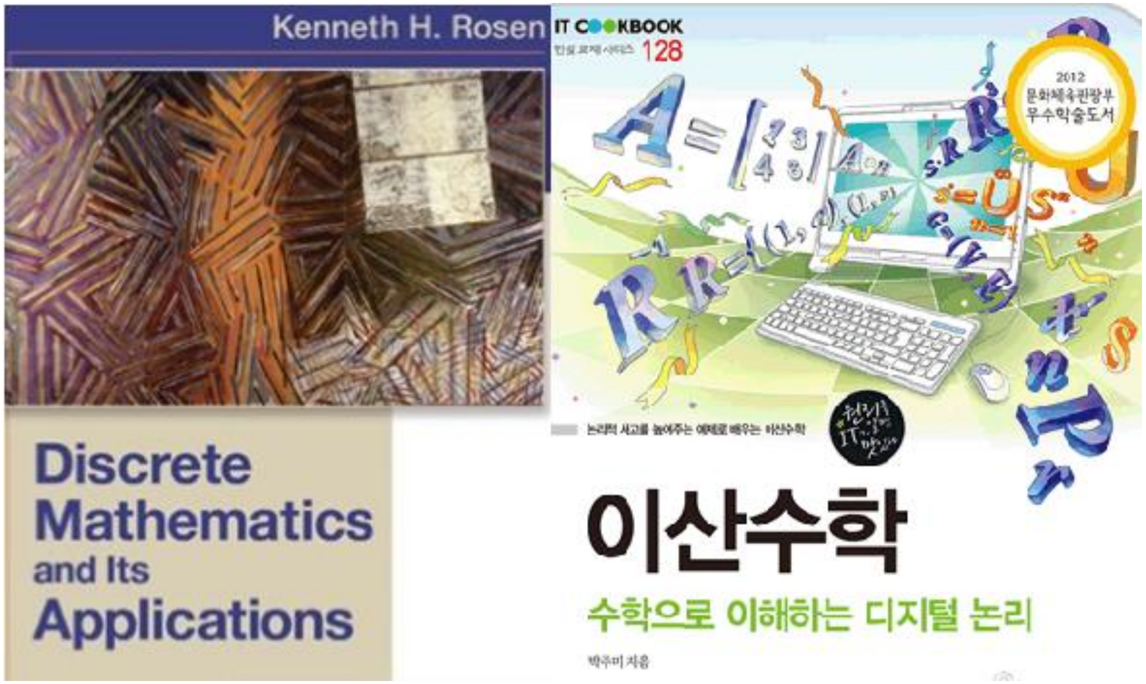
이산수학

Discrete Mathematics



Chapter 11:

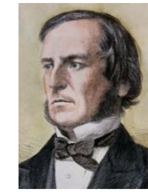
부울대수와

논리 게이트

Soongsil University : Kim Chang Wook

Lecture Note : ***Digital Design***, *with RTL Design, VHDL, and Verilog*, 2nd Edition, by Frank Vahid, **Discrete Mathematics** and Its **Applications**, 7E By Kenneth H. Rosen

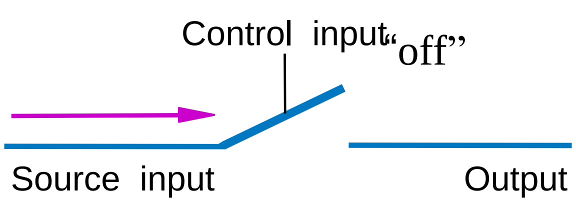
Boolean Algebra (부울 대수)

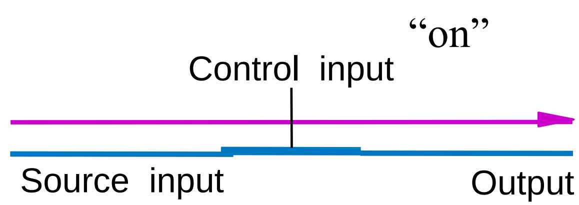


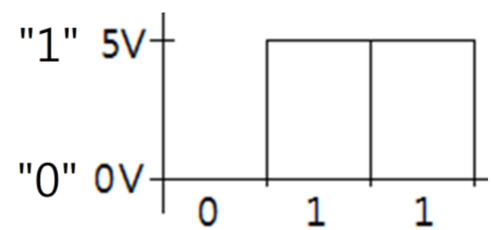
- 1854년 영국의 수학자 George Boole이 “The Laws of Thought”에서 수학적 논리 규칙으로 처음 소개

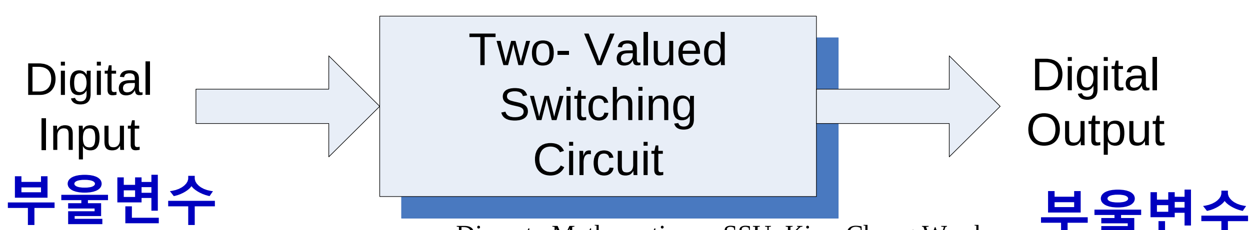
- Founder: Digital computer and digital circuit design theory
- 1938, 22 year old at MIT, wrote master’s thesis describing how Boolean algebra could be applied to switch-based circuit.

- 1938년 Shannon이 부울 대수의 기본 개념을 이용하여 회로를 설계하는데 사용할 수 있음을 증명
- 0과 1의 조합으로 연산되는 것을 부울 대수라고 함
- 컴퓨터나 디지털 전자 장치는 0, 1 두 가지 상태를 갖는 기본 장치로 구성
- 스위치나 회로는 ON과 OFF의 두 상태 중 하나인 1(T) or 0(F) 으로 표현
- 논리회로에서 Positive Logic에서는 “0”은 낮은 전압대를, “1”은 높은 전압대를 표현.









부울 변수는 단지 두 개의 서로 다른 값만을 갖는다.

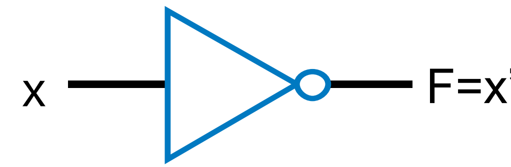
Boolean Algebra : Basic Operations

부울대수의 기본 연산: AND, OR, NOT(Complementation)

◆ NOT 연산(Boolean Complementation : 부울 보수)

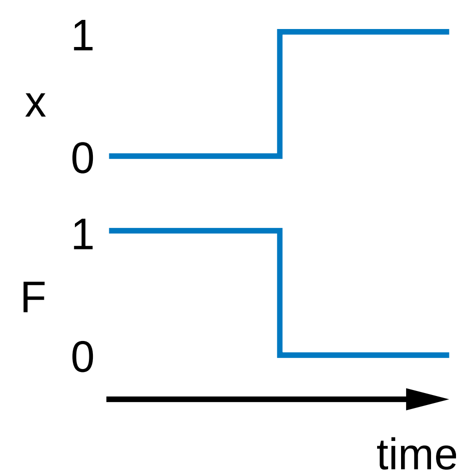
- 표기 : A' 또는 $\overline{A}$
- 2진 변수값을 반전시키는 단항 연산자.

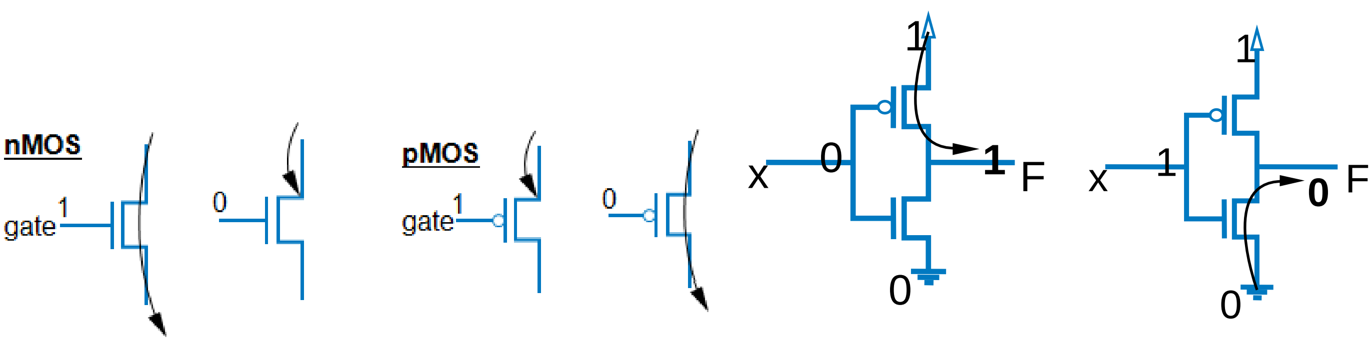
$0' = \overline{0} = 1 \qquad 1' = \overline{1} = 0$



x	F
0	1
1	0

- NOT 연산을 하면 부울변수 x의 값이 0인 경우 1, 1의 경우 0이 된다.
- 논리 게이트 : Inverter or NOT Gate
- 낮은 입력 전압은 높은 출력 전압으로 나타난다.



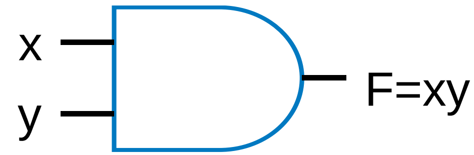


Boolean Algebra : Basic Operations

◆ AND 연산 (Boolean Multiplication: 부울곱)

- 표기 : $A \cdot B$ 또는 AB

  일반적으로, $A \cdot B$ 대신 AB 로 표기한다.

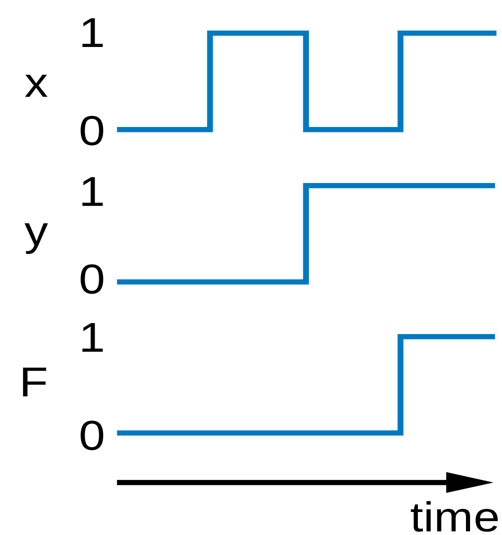


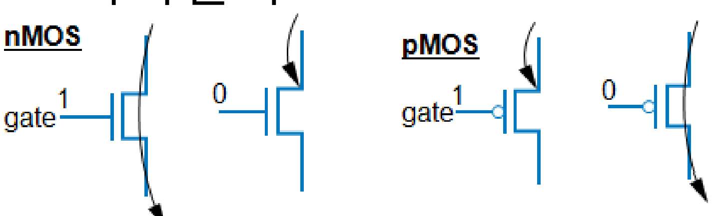
- 2진 변수의 값을 곱하는 이항 연산자

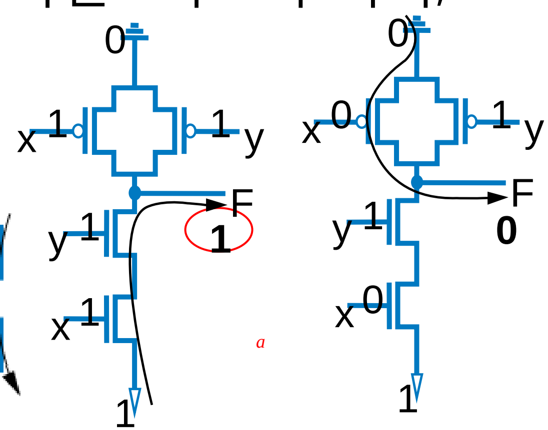
  : $0 \cdot 0 = 0 \quad 0 \cdot 1 = 0 \quad 1 \cdot 0 = 0 \quad 1 \cdot 1 = 1$

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

- AND 연산은 부울 변수 x와 y가 1일 때만 1이다.
- 논리 게이트 : AND Gate
- x와 y가 1일 때 F가 1 이 되며, 출력은 높은 전압, x 와 y 중에서 하나라도 0이면 F가 0이 되며, 출력은 낮은 전압으로 나타난다.



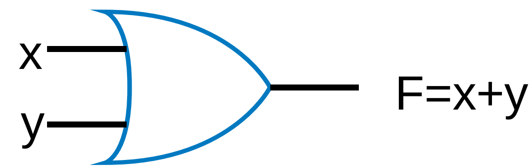




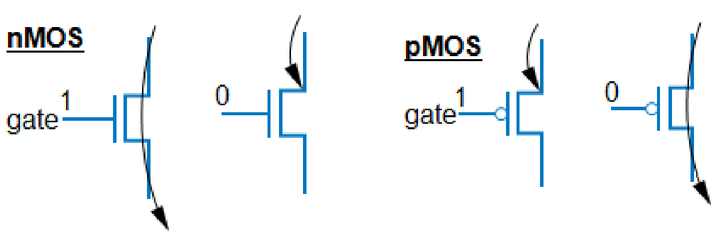
Boolean Algebra : Basic Operations

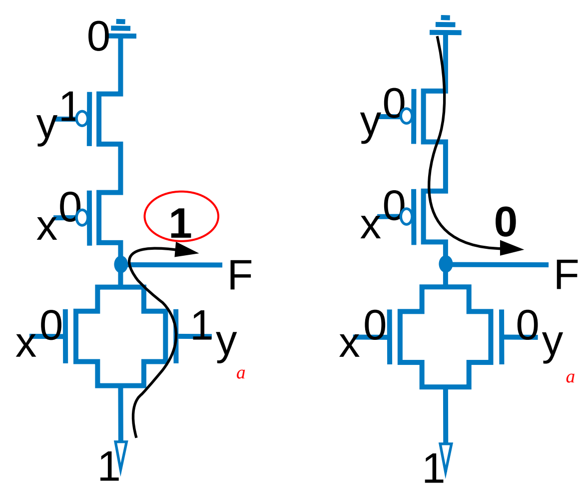
◆ OR 연산(Boolean Addition : 부울합)

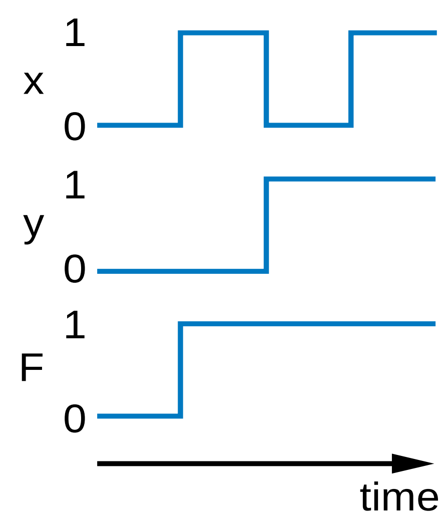
- 표기 : $A+B$
- 논리 게이트 : OR Gate
- x 또는 y가 1일 때 F=1이 되며,
  x와 y가 0일 때 F가 0이 되며,
  출력은 낮은 전압으로 나타난다.



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

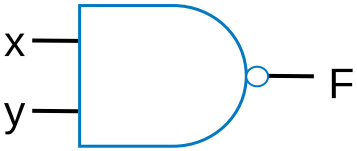






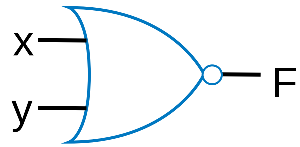
More Gates

NAND



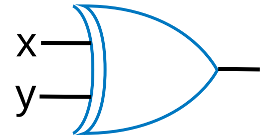
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR



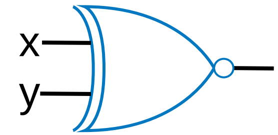
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

XOR



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

XNOR



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

- NAND : Opposite of AND (“NOT AND”), F=(ab)’
- NOR : Opposite of OR (“NOT OR”), F=(a+b)’
- XOR : “exclusive OR”, Exactly 1 input is 1,
  For 2-input XOR. variables a and b : F= a’b+ab’= $a \oplus b$
  For more inputs -- odd number of 1s three variables : F= $a \oplus b \oplus c$
- XNOR: “exclusive nor”, Opposite of XOR (“NOT XOR”) : F= a’b’+ab= $a \odot b$
  = $(a \oplus b)'$

Boolean Algebra Terminology

❖Example equation : **F(a,b,c) = a’bc + abc’ + ab + c**

❖***Variable* (변수)**

- Represents a value (0 or 1)
- Above equation has three variables: a, b, and c

❖***Literal*(문자)**

- Appearance of a variable, in true or complemented form
- Nine literals: a’, b, c, a, b, c’, a, b, and c
  Ex) $ab'c+a'b+a'bc'+b'c'$: 모두 10개의 문자로 구성

❖***Product term***

- Product of literals
- Four product terms: a’bc, abc’, ab, c

❖***Sum-of-products***

- Equation written as OR of product terms only
- sum-of-products form : F=a+b’+ac F= a’bc + abc’ + ab + c
- Following equations are not in sum-of-products form.
  F=(a+b)c F=(a’)’ +b “F = (a+b)c + d”

Logic Gates

❖**LOGIC GATES**

1. Boolean Algebra is used to model circuitry of electronic devices.
2. Each circuit is designed using the rules of Boolean Algebra.
3. The basic elements of circuits are called ***gates*** **(게이트).**
4. Each type of gate implements a Boolean operation.
5. The circuits give output that depends only on the input.
6. Circuits have no memory capabilities.

❖These circuits are called ***combinational circuits*****(조합회로)**

Combinational circuit : Output depend solely on the present combination of the circuit inputs' values.

Sequential circuit : Output depend on the present and past input values.

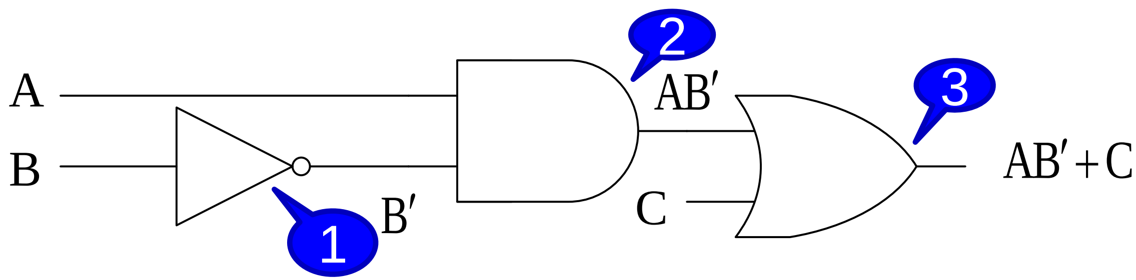
❖We will construct combinational circuits using three types of elements.   .Inverter, OR gate, AND gate

Boolean Expressions and Truth

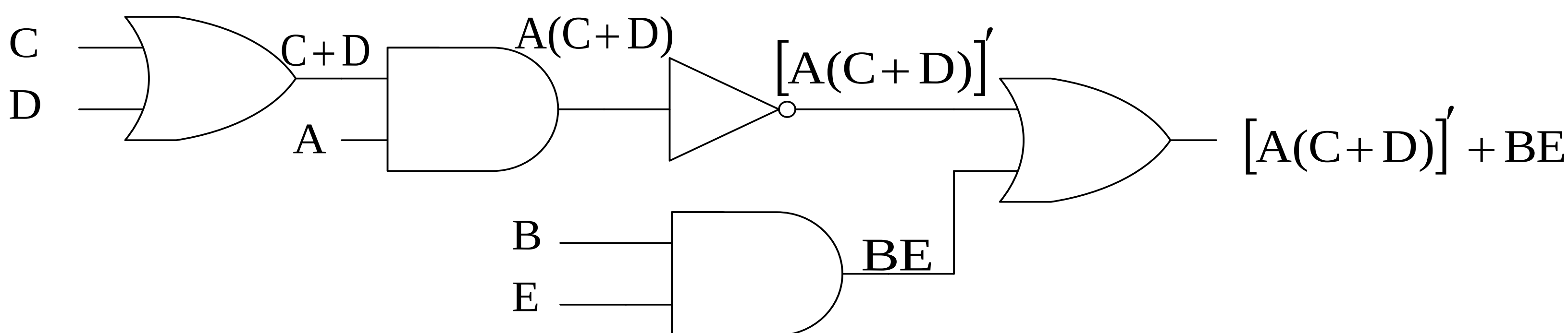
- 부울식은 하나 또는 둘 이상의 부울 변수(A, B, X, Y 등)와 상수(0또는 1) 그리고 기본 연산(AND, OR, NOT)으로 구성된 식이다.

(예 1) 가장 간단한 부울식은 단 하나의 부울 변수나 상수로만 구성된 식: B’

AB’+C식에서 ①부울 변수 B의 보수화가 먼저 실행되고, ②부울변수 A와 AND 연산을 한다. 그리고 마지막으로 ③부울변수 C와 OR연산의 순서로 진행된다.



(예 2)



A=B=C=1이고, D=E=0이면 부울식의 최종값은 $[A(C+D)]' + BE = [1(1+0)]' + 1\cdot 0$

$= [1\cdot 1]' + 0 = 0 + 0 = 0$

Basic Theorems Tables

TABLE 5 Boolean Identities.

Identity	*Name*
$\overline{\overline{x}} = x$	Law of the double complement
$x + x = x$ $x \cdot x = x$	Idempotent laws 멱등법칙
$x + 0 = x$ $x \cdot 1 = x$	Identity laws 항등법칙
$x + 1 = 1$ $x \cdot 0 = 0$	Domination laws 지배법칙
$x + y = y + x$ $xy = yx$	Commutative laws 교환법칙

Identity	*Name*
$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$	Associative laws 결합법칙
$x + yz = (x + y)(x + z)$ $x(y + z) = xy + xz$	Distributive laws 분배법칙
$\overline{(xy)} = \overline{x} + \overline{y}$ $\overline{(x + y)} = \overline{x}\,\overline{y}$	De Morgan's laws 드모르간의 법칙
$x + xy = x$ $x(x + y) = x$	Absorption laws 흡수법칙
$x + \overline{x} = 1$	Unit property 단위 성질
$x\overline{x} = 0$	Zero property 제로 성질

Commutative, Associative, and Distributive Laws

- 둘 또는 그 이상의 부울변수들이 모두 OR 연산인 경우

  변수들 중에서 어느 한 변수가 1이면 결과 값은 1.

  모든 변수들이 0이면 OR 연산의 결과는 0이다.

  $X + XY + XZ = X(1 + Y + Z) = X,$ $X = Y = Z = 0$ 일때 $X + Y + Z = 0$

- 부울대수에서의 분배법칙(제1법칙) $X(Y+Z) = XY + XZ$

- 부울대수에서의 분배법칙(제2법칙) $X + (YZ) = (X+Y)(X+Z)$

  증명) $(X+Y)(X+Z) = (X+Y)X + (X+Y)Z$

$$
\begin{aligned}
&= XX + XY + XZ + YZ \\
&= X + XZ + XY + YZ \\
&= X \cdot 1 + XZ + XY + YZ \\
&= X(1 + Z + Y) + YZ \\
&= X + YZ
\end{aligned}
$$

Simplification Theorems

◆ 부울대수에서의 간략화 법칙

$XY + XY' = X$ (증명) $XY + XY' = X(Y + Y') = X$

$(X + Y)(X + Y') = X$ (증명) $(X + Y)(X + Y') = X + X(Y + Y') + (YY')$

$= X + 0 = X$

$X + XY = X$ (증명) $X + XY = X \cdot 1 + XY = X(1 + Y) = X$

$X(X + Y) = X$ (증명) $X(X + Y) = XX + XY = X + XY = X$

$(X + Y')Y = XY$ (증명) $(X + Y')Y = XY + Y'Y = XY + 0 = XY$

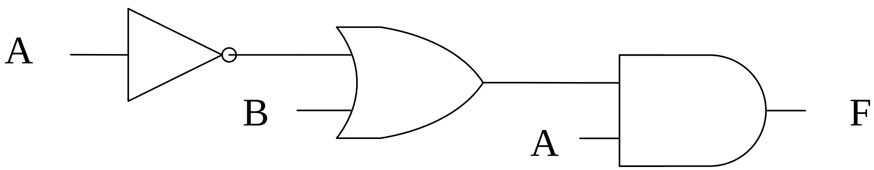
$XY' + Y = X + Y$ (증명) $XY' + Y = (X + Y)(Y' + Y) = X + Y$

Simplification Theorems

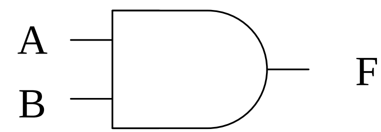
- 간략화 정리는 어떤 부울식을 더 간단한 부울식으로 대체될 수 있다는 것을 의미한다. 디지털 시스템에서 시스템을 나타내는 부울식은 논리 게이트로 구현되기 때문에 부울식을 간략화 한다는 것은 부울식을 나타내는 논리 게이트 수를 감소시켜 비용적인 이득을 볼 수 있다는 장점이 있다.

✓ 예1)

$F = A(A'+B)$



$F = A(A'+B) = AA' + AB = AB$



✓예2-4) 간략화

(1) $Z = A'BC + A' = A'BC + A' \cdot 1 = A'(BC+1) = A'$

(2) $Z = [A + B'C + D + EF][A + B'C + (D+EF)']$

$Z = [X + Y][X + Y'] = XX + X(Y+Y') + YY' = X = A + B'C$

(3) $Z = (AB+C)(B'D + C'E') + (AB+C)'$

$Z = Y'X + Y = (X+Y)(Y'+Y) = X + Y = (B'D + C'E') + (AB+C)'$

Representations of Boolean Functions

Boolean function : mapping of each possible combination of values for the function's variables(input) to either 0 or 1(output)

English 1: F outputs 1 when a is 0 and b is 0, or when a is 0 and b is 1.

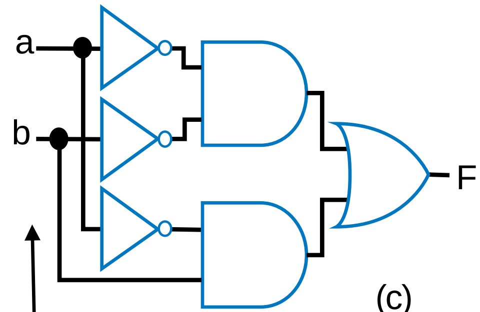
English 2: F outputs 1 when a is 0, regardless of b's value

(b)

Equation 1: F(a,b) = a'b' + a'b

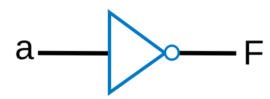
Equation 2: F(a,b) = a'

- Equation : mathematical statement
- Different equation can represent same function.
- Advantage : equation can be manipulated using properties of Boolean algebra, enabling simplification of an equation, or proving that two equations represents the same functions, etc



(c)

Circuit 1



Circuit 2

a	b	F
0	0	1
0	1	1
1	0	0
1	1	0

Truth table

(d)

- Circuit may represent an actual physical implementation of a Boolean function.
- Different circuit can represent same function.
- Advantage :circuit can enable quick and easy comprehension of a function by humans

The function F

❖A function can be represented in different ways

- Each representation has its own advantages and disadvantages,
- each is useful at different times during design
- Above shows seven representations of the same functions F(a,b),
- using four different methods: English, Equation, Circuit, and Truth Table

Truth Table Representation of Boolean Functions

❖Input variables shows all possible value combinations of those inputs.
진리표는 부울식에 있는 부울 변수들 값의 가능한 모든 조합에 대한 부울식의 값을 나타낸다.

❖Define value of F for each possible combination of input values
n-변수 식의 진리표는 2^n 개의 행으로 나타난다.

- 2-input function: 4 rows (a)
- 3-input function: 8 rows (b)
- 4-input function: 16 rows (c)
- 2^n

a	b	F
0	0	
0	1	
1	0	
1	1	

(a)

a	b	c	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

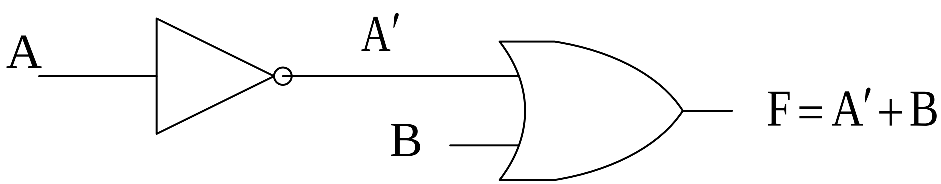
(b)

a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

(c)

❖Unlike equations and circuits, a Boolean function has only one truth table representation.

❖Ex) 디지털회로에서 진리표 작성
두변수 : AB $\longrightarrow$ 2^2=4 rows



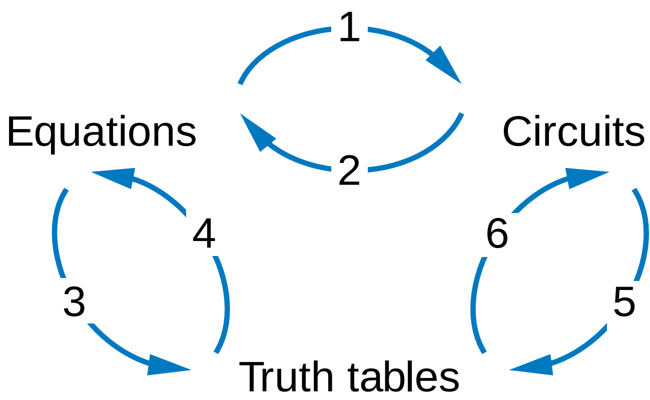
A	B	A'	$F = A' + B$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

Converting among Boolean Function Representations

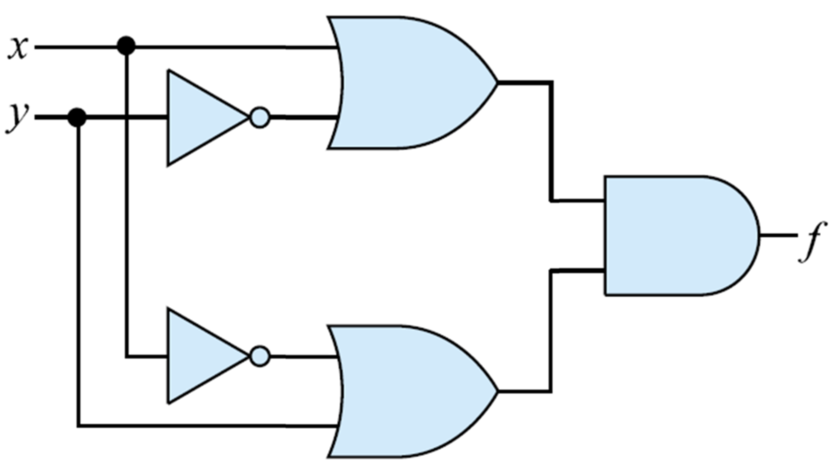
- Can convert from any representation to another

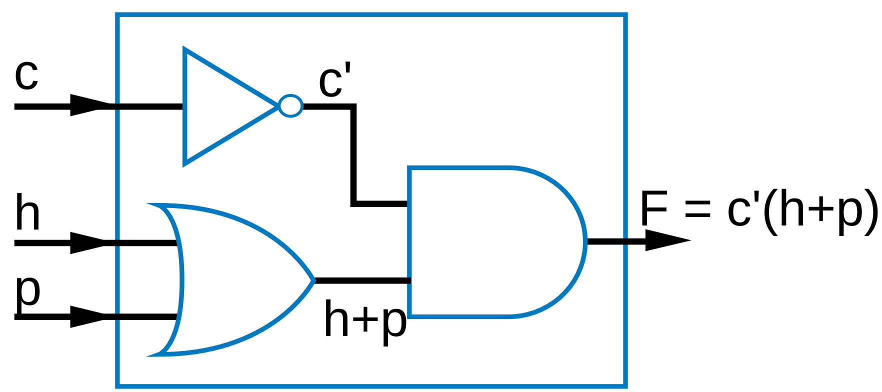
❖Common conversions

1. Equation to circuit
   - AND,OR,NOT → AND gate, OR gate, NOT gate
2. Circuit to equation
   - Start at inputs,
     write expression of each gate output
   - F(c,h,p)=c’ (h+p)

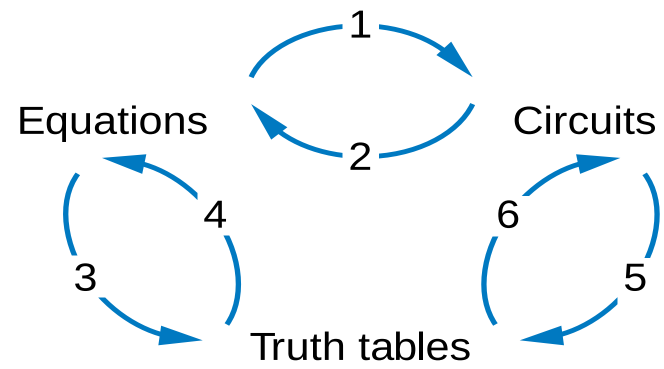


Ex) 1. $f(x,y)=(x+y')(x'+y)$ → 논리 회로 구성.   2. 논리회로 → 부울 식





Converting among Representations



❖ More common conversions

4. Truth table to equation (which we can then convert to circuit)
   - Creating a product term for each 1 in the output, then ORing all the product.

Inputs		Outputs	Term
a	b	F	F = sum of
0	0	1	a'b'
0	1	1	a'b
1	0	0	
1	1	0	

(4) F = a'b' + a'b

3. Equation to truth table
   - First creating a truth table structure appropriate for the number of input variables, then evaluating the right-hand side of the eq. for each input values.
   - Creating intermediate columns helps

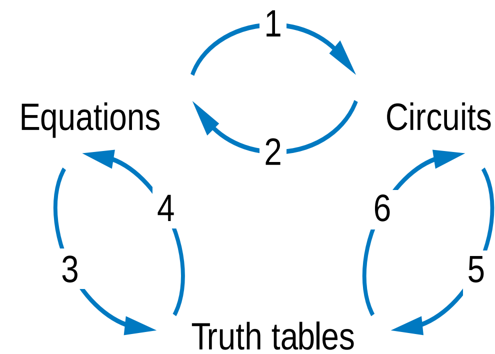
(3) Convert to truth table: F = a'b' + a'b

Inputs				Output
a	b	a' b'	a' b	F
0	0	1	0	**1**
0	1	0	1	**1**
1	0	0	0	**0**
1	1	0	0	**0**

a	b	c	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	ab'c
1	1	0	1	abc'
1	1	1	1	abc

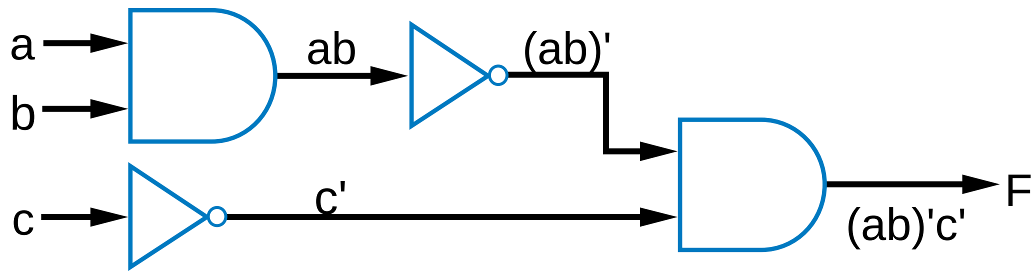
(4) F = ab'c + abc' + abc

Converting among Representations



5. Converting from Circuit to Truth Table

- First convert to circuit to equation, then equation to table
  - Starting from the gates closest to inputs, label each gate's output as an expression of the gate's inputs.
  - Finally, label the rightmost AND gate's output   F



Inputs						Outputs
a	b	c	ab	(ab)'	c'	F=(ab)'c'
0	0	0	0	1	1	1
0	0	1	0	1	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	0	1	0
1	1	1	1	0	0	0

Truth Table

$\rightarrow$ equation

F=a'b'c'+a'bc' +ab'c'

= c'(a'b'+a'b+ab')

=c'(a'(b'+b)+ab')

=c'(a'+ab')

=c'(a'+a)(a'+b')

=c'(a'+b')

=c'(ab)'

=(ab)'c'

- Distributive

a * (b + c) = ab + ac

a + (bc) = (a+b)(a+c)

- Identity

0 + a = a + 0 = a

1 * a = a * 1 = a

- Complement

a + a' = 1

a * a' = 0

- DeMorgan's Law

(a+b)'=a'b'

(ab)'=a'+b'

Standard Representation: Truth Tables

❖For any Boolean function

: many possible equations and circuits, but only one truth table

→ Truth tables represent a standard representation of a function , unique

❖How can we determine if two functions are the same?

- Solution: Convert to truth tables

  : Only ONE truth table representation of a given function - ***Standard*** representation

- Comparing truth tables works fine when a function has only 2 inputs

  If a function has many inputs n → truth table's number of rows 2^n grows very quickly. We can't realistically expect to compare 2 tables

- The number of output 1s in a truth table may be very small compared to the number of output 0s → Is there a more compact but still standard representation of a Boolean function ?

Canonical Form – Sum of Minterms Equations

❖ Truth tables too big for numerous inputs

❖ Use standard form of a Boolean equation that function outputs 1

- Known as ***canonical form***
- One canonical form for Boolean function : sum of minterms

  Minterm : product term whose literals include every variable of the function exactly once, in true or complemented form
- Converting step : any equation to sum-of-minterms canonical form
  - Manipulate the equation until it is in sum-of-products form
  - Then expand each term until all terms are minterms
  - Arrange the literals (alphabetical) and the terms in the order (truth table)

Ex: Determine if $F1 = (a+b)(a'+ac)b$ is equivalent to $F2 = a'bc'+a'bc+abc$,
by converting first equation to canonical form (second already is)

$F1 = (a+b)(a'+ac)b = aa'b+aacb+ba'b+bacb = acb+a'b$ (sum of products)

$= abc+a'b(c+c') = abc+a'bc+a'bc'$ ($a+a'=1$ $aa'=0$ 적용)

$= a'bc' + a'bc + abc$ (arrange: truth table) $\Rightarrow$ Equivalent $F1=F2$

Canonical Form – Sum of Minterms

Ex: Determine whether the functions G and H are equivalent.

$G(a,b,c,d,e) = abcd + a'bcde$ and

$H(a,b,c,d,e) = abcde + abcde' + a'bcde + a'bcde(a' + c)$

$G = abcd + a'bcde$

$G = abcd(e+e') + a'bcde$

$G = abcde + abcde' + a'bcde$

$G = a'bcde + abcde' + abcde$ (*sum of minterms form*)

expand each term until all terms are minterms

: complement property $a+a'=1$

Equivalent

$H = abcde + abcde' + a'bcde + a'bcde(a' + c)$

$H = abcde + abcde' + a'bcde + a'bcdea' + a'bcdec$

$H = abcde + abcde' + a'bcde + a'bcde + a'bcde$

$H = abcde + abcde' + a'bcde$

a

$H = a'bcde + abcde' + abcde$

Checking the equivalence using truth tables : $32(2^5)$ rows each

Using sum of minterms was more appropriate.

Canonical Form – Sum of Minterms Equations

❖Ex: Find the sum-of-products expansion for the function

$$F(x,y,z)=(x+y)z'=xz'+yz' \quad \text{: Distributive law}$$
$$=x1z'+1yz' \quad \text{: Identity law}$$
$$=x(y+y')z'+(x+x')yz' \quad \text{: Unit property}$$
$$=xyz'+xy'z'+xyz'+x'yz' \quad \text{: Distributive law}$$
$$=x'yz'+xy'z'+xyz' \quad \text{: Idempotent law}$$

x	y	z	$x+y$	$\bar{z}$	$(x+y)\bar{z}$
1	1	1	1	0	0
1	1	0	1	1	1
1	0	1	1	0	0
1	0	0	1	1	1
0	1	1	1	0	0
0	1	0	1	1	1
0	0	1	0	0	0
0	0	0	0	1	0

Maxterms

- Boolean product of Boolean sums (**maxterms)** : correspond to those combinations of values for which the function has the value 0.
- The resulting expansion is called the product-of-sums expansion.
- A maxterm is the standard sum.
  A maxterm has the value 0 for one and only combination of values of its variables.
- A minterm is the complement of a maxterm.
- To get the **maxterms**
  1. Take the complement.
  2. Obtain the **sum of minterms**.
  3. Complement both sides of the **sum of minterms** obtained in (2)
- Boolean functions expressed as a **sum of minterms** or **product of maxterms**, are said to be in **canonical form**.

	Minterms	Maxterms
$x\,y\,z$	Term	Term
0 0 0	$x'y'z'$	$x+y+z$
0 0 1	$x'y'z$	$x+y+z'$
0 1 0	$x'yz'$	$x+y'+z$
0 1 1	$x'yz$	$x+y'+z'$
1 0 0	$xy'z'$	$x'+y+z$
1 0 1	$xy'z$	$x'+y+z'$
1 1 0	xyz'	$x'+y'+z$
1 1 1	xyz	$x'+y'+z'$

Minterm and Maxterm Expansions

❖ 최소항(minterm)

- 일반적으로 n 변수의 최소항은 각 변수가 그대로 혹은 보수 형태로 한번씩만 나타나는 n 개의문자의 곱. F=A'BC+AB'C'+AB'C+ABC'

❖ 최대항(maxterm)

- 일반적으로 n 변수의 최대항은 각 변수가 그대로 혹은 보수 형태로 한번씩만 나타나는 n 개의문자의 합. F=(A+B+C)(A+B+C')(A+B'+C)

❖최소항(최대항)은 다음과 같은 간단한 형태로 표현 가능하다.

행번호	A	B	C	최소항	최대항
0	0	0	0	$A'B'C' = m_0$	$A+B+C = M_0$
1	0	0	1	$A'B'C = m_1$	$A+B+C' = M_1$
2	0	1	0	$A'BC' = m_2$	$A+B'+C = M_2$
3	0	1	1	$A'BC = m_3$	$A+B'+C' = M_3$
4	1	0	0	$AB'C' = m_4$	$A'+B+C = M_4$
5	1	0	1	$AB'C = m_5$	$A'+B+C' = M_5$
6	1	1	0	$ABC' = m_6$	$A'+B'+C = M_6$
7	1	1	1	$ABC = m_7$	$A'+B'+C' = M_7$

Minterm and Maxterm Expansions

- f 의 최소항 혹은 최대항 전개가 주어지면 f’ 에 대한 최소항 혹은 최대항 전개는 쉽게 구해질 수 있다.
- f 가 0일 때 f’ 은 1이 되며, f’ 에 대한 최소항 전개는 f 의 최소항 전개에서 제외된 최소항을 열거하면 쉽게 얻을 수있다.

$$f = A'BC + AB'C' + AB'C + ABC' + ABC$$

$$f(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7$$

$$f(A, B, C) = \sum m(3, 4, 5, 6, 7)$$

$$f'(A, B, C) = \sum m(0, 1, 2)$$

- 마찬가지로 f’ 의 최대항 전개는 f 에 없는 최대항을 포함하여 구할 수 있다.

$$f(A, B, C) = \prod M(0, 1, 2)$$

$$f'(A, B, C) = \prod M(3, 4, 5, 6, 7)$$

Multiplying Out and Factoring

◆ 논리합의 곱(Product of Sum) 형식 $X+YZ=(X+Y)(X+Z)$

논리곱의 합 부울 함수를 논리합의 곱형태로 인수화. 분배제2법칙

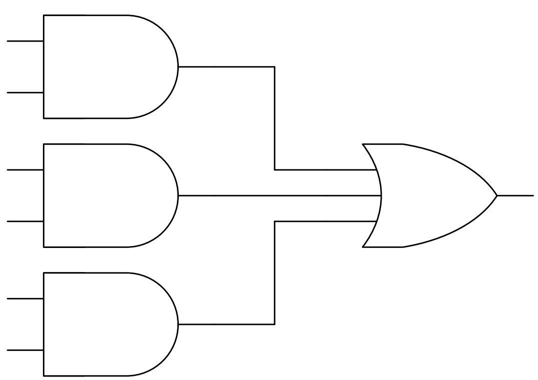
$$(1)\;\; A+B'CD=(A+B')(A+CD)=(A+B')(A+C)(A+D)$$

$$(2)\;\; AB'+C'D=(AB'+C')(AB'+D)=(A+C')(B'+C')(A+D)(B'+D)$$

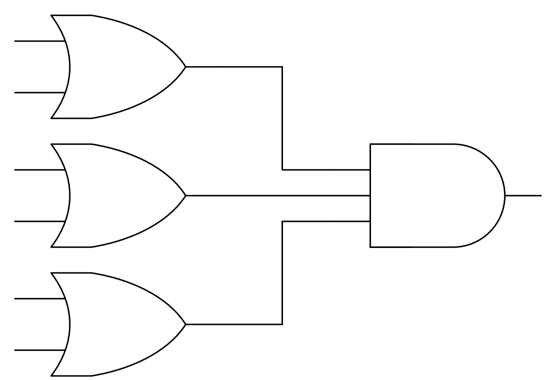
$$(3)\;\; C'D+C'E'+G'H=C'(D+E')+G'H=(C'+G'H)(D+E'+G'H)$$

$$=(C'+G')(C'+H)(D+E'+G')(D+E'+H)$$

- 논리곱의 합식을 논리 게이트로 표시할 때 하나 이상의 AND 게이트가 입력단이 되고, 이 AND 게이트의 출력들이 OR 게이트의 입력이 되는 형태를 구성한다.
- 논리합의 곱식을 논리 게이트로 표시할 때 하나 이상의 OR 게이트가 입력단이 되고, 이 OR 게이트의 출력들이 AND 게이트의 입력이 되는 형태를 구성한다.



논리곱의 합식의 논리 게이트 표현



논리합의 곱식의 논리 게이트 표현

DeMorgan's Laws

❖ DeMorgan의 정리 : $(X+Y)'=X'Y'$ $\quad$ $(XY)'=X'+Y'$

진리표에 의한 DeMorgan의 증명

X	Y	X'	Y'	$X+Y$	$(X+Y)'$	$X'Y'$	XY	$(XY)'$	$X'+Y'$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

◆ *n* 변수에 대한 DeMorgan의 정리

$$(X_1+X_2+X_3+\cdots+X_n)'=X_1{}'X_2{}'X_3{}'\cdots X_n{}' \qquad (2-23)$$

$$(X_1X_2X_3\cdots X_n)'=X_1{}'+X_2{}'+X_3{}'+\cdots+X_n{}' \qquad (2-24)$$

예)

(1) $[(A'+B)C']'=(A'+B)'+C=AB'+C$

(2) $[(AB'+C)D'+E']'=[(AB'+C)D']'E$

$$=\left[(AB'+C)'+D\right]E=\left[(AB')'C'+D\right]E=[(A'+B)C'+D]E$$

DeMorgan's Laws

❖ $(a+b)'=a'b'$

- Proof 1

  $(a+b)+(a+b)'=1$ if $(a+b)+a'b'=1$ then $(a+b)'=a'b'$

  $(a+b)+a'b'=(a+b+a')(a+b+b')=(1+b)(a+1)=1 \rightarrow (a+b)'=a'b'$

- Proof 2

  $(a+b)(a+b)'=0$ if $(a+b)(a'b')=0$ then $(a+b)'=a'b'$

  $(a+b)(a'b')=aa'b + ba'b' = 0b + 0a'=0 \rightarrow (a+b)'=a'b'$

❖ $(ab)'=a'+b'$

- Proof 1

  $(ab)+(ab)'=1$ if $(a'+b')+(ab)=1$ then $(ab)'=a'+b'$

  $(a'+b')+(ab)=(a'+b'+a)(a'+b'+b)=(1+b')(a'+1)=1 \rightarrow (ab)'=a'+b'$

- Proof 2

  $(ab)(ab)'=0$ if $(ab)(a'+b')=0$ then $(ab)'=a'+b'$

  $(ab)(a'+b')=aba' + abb' = 0b + 0a'=0 \rightarrow (ab)'=a'+b'$

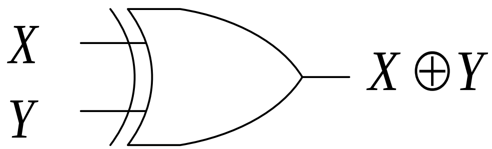
Complement Property : $a+a'=1$ $aa'=0$

Exclusive-OR and Equivalence Operations

❖Exclusive-OR의 진리표

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

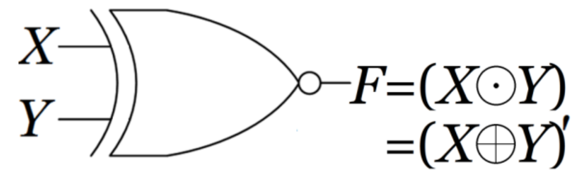
$$X \oplus Y = X'Y + XY'$$



Exclusive-NOR(XNOR) 진리표

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

$$(X \odot Y) = XY + X'Y'$$



❖Exclusive-OR에 대한 정리

$X \oplus 0 = X$ $\quad X \oplus 1 = X'$ $\quad X \oplus X = 0$ $\quad X \oplus X' = 1$ $\quad X \oplus Y = Y \oplus X$

$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$$

$$(X \oplus Y) \oplus Z = (XY' + X'Y)Z' + (XY' + X'Y)'Z = XY'Z' + X'YZ' + (XY')'(X'Y)'Z = // + (X' + Y)(X + Y')Z$$

$$= (XY' + X'Y)Z' + (X'Y' + XY)Z = X'(YZ' + Y'Z) + X(Y'Z' + YZ) = X \oplus (Y \oplus Z)$$

$$XY \oplus XZ = XY(XZ)' + (XY)'XZ = XY(X' + Z') + (X' + Y')XZ$$

$$= XYX' + XYZ' + X'XZ + Y'XZ = XYZ' + Y'XZ = X(YZ' + Y'Z) = X(Y \oplus Z)$$

$$(X \oplus Y)' = (XY' + X'Y)' = ((XY')'(X'Y)') = (X' + Y)(X + Y') = XY + X'Y' = X \oplus Y' = X' \oplus Y$$

Exclusive - NOR는 Exclusive - OR의 보수이다.

$$(X \odot Y) = (X \oplus Y)' = (X'Y + XY')' = (X + Y')(X' + Y) = XX' + XY + X'Y' + Y'Y = XY + X'Y'$$

등가 및 Exclusive - OR에 대한 유용한 식 : $(XY' + X'Y)' = XY + X'Y'$

The Consensus Theorem

◆ 합의 정리

- 합의 정리는 부울식을 간소화하는데 매우 중요한 역할을 한다 .

  $XY+X'Z+YZ$ 형태로 주어지는 경우,

  YZ는 중복되는 항으로 소거되어 $XY+X'Z$ 로 표시할 수 있다.

  소거된 항을 합의 항(Consensus term)이라고 한다.

  두 항으로 된 식에서 한쪽 항에는 한 변수가, 다른 한쪽에는 그 변수의 보수가 있을 경우, 합의 항은 변수에 보수를 취한 쪽의 남은 변수와 보수를 취하지 않은 쪽의 남은 변수를 곱한 것이다.

$$ab + a'c \quad \rightarrow \quad bc$$

$$abd + b'de' \quad \rightarrow \quad adde' = ade'$$

$$ab'd + a'bd' \quad \rightarrow \quad 0$$

The Consensus Theorem

◆ 합의 정리 : $XY + X'Z + YZ = XY + X'Z$

증명) $XY + X'Z + YZ = XY + X'Z + (X + X')YZ = XY + XYZ + X'Z + X'YZ$

$$= XY(1+Z) + X'Z(1+Y) = XY + X'Z$$

예) $a'b' + ac + bc' + b'c + ab = a'b' + ac + bc'$

예제 $A'C'D + A'BD + BCD + ABC + ACD' = A'C'D + A'BD + ABC + ACD'$

A′BD+BCD+ABC=A′BD+ABC+ABCD+A′BCD=A′BD(1+C)+ABC(1+D)

=A′BD+ABC로 BCD항을 합의 정리에 의해서 소거한 경우이다.

그러나 경우에 따라서 이 항을 소거하지 않고, 이 항과 결합하는 합의 항을 찾으면 다음과 같다. A’C’D+BCD+A’BD=A’C’D+BCD, BCD+ABC+ACD’=BCD+ACD’

$$A'C'D + A'BD + BCD + ABC + ACD' = A'C'D + BCD + ACD'$$

Algebraic Simplification of Switching Expressions

- 부울대수 법칙과 정리를 사용하여 수위칭 식을 간략화하기 위한 방법
- 수식을 간략화하는 것은 게이트를 사용하여 회로를 구현할 때 게이트 수가 줄어서 회로구성 비용을 줄이기 때문에 중요하다.

1. 항들의 조합
   - 두 항을 조합하기 위한 정리 $XY+XY'=X(Y+Y')=X$ 를 이용한다

$$abc'd' + abcd' = abd'(c'+c) = abd' \qquad X = abd', \quad Y = c$$

이 정리를 사용하여 항을 조합할 때, 조합되는 두 항은 반드시 같은 변수를 가져야 하고, 변수들 중의 한 변수는 한 항에 보수형태로 표현되어야 하고 다른 항은 보수가 아니어야 한다.

   - 또한 $X+X=X$ 이기 때문에 주어진 항은 반복할 수 있고, 둘 이상의 다른 항과 조합될 수 있다.

$$ab'c + abc + a'bc = ab'c + abc + \underline{abc} + a'bc = ac(b'+b) + bc(a+a') = ac + bc$$

$$(a+bc)(d+e') + a'(b'+c')(d+e') = YX + Y'X = (Y+Y')X = d + e'$$

$$\left[X = d + e', \quad Y = a + bc, \quad Y' = (a+bc)' = a'(bc)' = a'(b'+c')\right]$$

Algebraic Simplification of Switching Expressions

2. 항들의 소거

- 가능하다면 중복된 항을 소거하기 위하여 $X+XY=X(1+Y)=X$ 를 사용하라.

  그리고 나서 임의의 합의항을 소거하기 위해 합의항의 정리를 적용해 보라.

$$a'b + a'bc = a'b(1+c) = a'b \quad [X = a'b]$$

$$a'bc' + bcd + a'bd = a'bc' + bcd \quad [X = c, Y = bd, Z = a'b]$$

3. 문자들의 소거

- 중복된 문자를 소거하기 위하여 $X+X'Y=X+Y$ 를 사용하라.

  이 정리를 적용하기 전에 간단한 인수화를 먼저 하는 것이 필요하다.

$$A'B + A'B'C'D + ABCD' = A'(B + B'C'D') + ABCD' = A'(B + C'D') + ABCD'$$

$$= A'B + A'C'D' + ABCD' = B(A' + ACD') + A'C'D'$$

$$= B(A' + CD') + A'C'D' = A'B + BCD' + A'C'D'$$

4. 중복항 도입

중복항은 XX' 을 더하거나, $(X+X')$ 을 곱하거나,

YZ 를 $XY+X'Z$ 에 더하거나, X에 XY를 더하여 중복항을 도입할 수 있다.

$$WX + XY + X'Z' + WY'Z' = WX + XY + X'Z' + WY'Z' + WZ'$$

$$= WX + XY + X'Z' + WZ' = WX + XY + X'Z'$$

Combinational Logic Design Using a Truth Table

•하나의 출력을 갖는 조합논리 스위칭회로를 설계하는 세 단계는 다음과 같다.

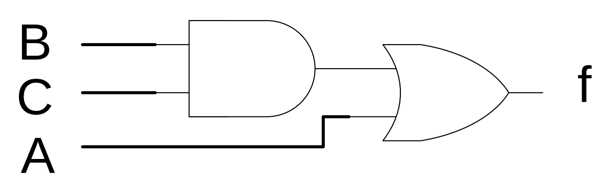
1. 회로에 필요한 기능을 나타내는 스위칭 함수를 구한다.
2. 함수에 대하여 간략화 된 대수식을 구한다.
3. 활용 가능한 논리소자를 사용하여 간략화 된 함수를 실현한다.

예) 3입력 / 1출력 스위칭 회로

1. 입력 A, B, C는 2진수 N의 각각 첫 번째, 두 번째, 세 번째 bit
2. 출력 f는 N ≥ 011_2이면 1, N < 011_2이면 0

A	B	C	f	f'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$
\begin{aligned}
f &= A'BC + AB'C' + AB'C + ABC' + ABC \\
&= A'BC + AB'(C'+C) + AB(C'+C) \\
&= A'BC + AB' + AB \\
&= A'BC + A(B'+B) \\
&= A'BC + A \\
&= (A'+A)(BC+A) \\
&= A + BC
\end{aligned}
$$



Examples of Truth Table Construction

예) 두개의 1비트 이진수 A와 B를 더하여 2비트 결과를 얻는 2진수 덧셈기를 설계.

덧셈기의 두 입력은 부울변수 A와 B로, 출력 역시 2비트로 부울 변수 X와 Y로 한다.

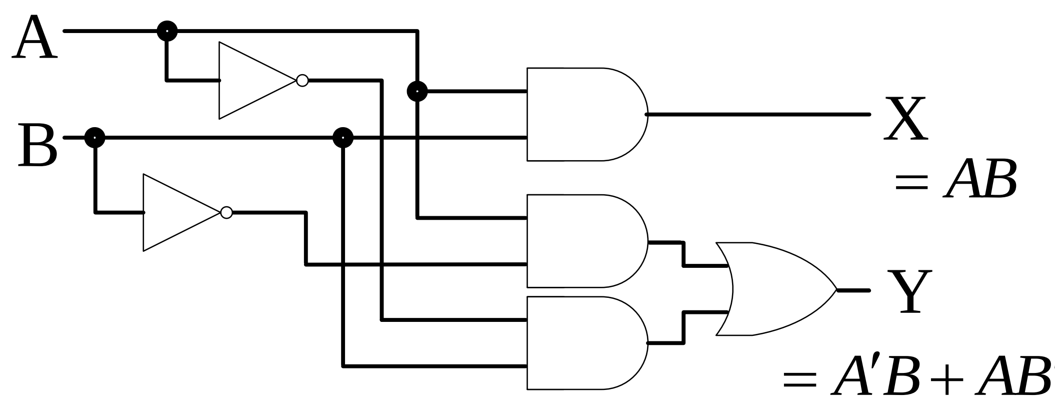
A	B	합	
0	0	00	$(0+0=0)$
0	1	01	$(0+1=1)$
1	0	01	$(1+0=1)$
1	1	10	$(1+1=2)$

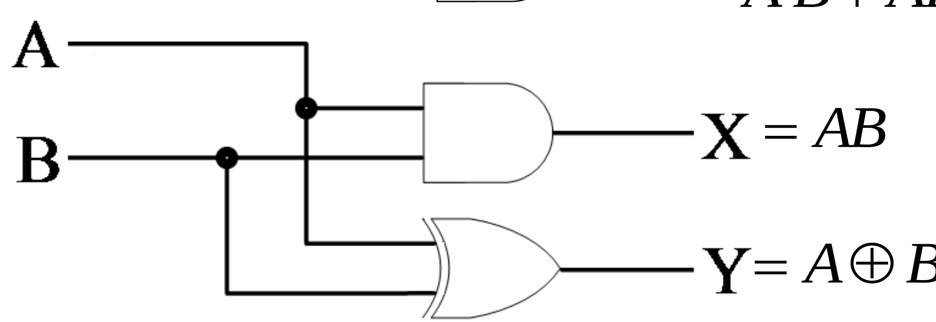
A	B	X	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$X = AB$$

$$Y = A'B + AB'$$

$$= A \oplus B$$





Introduction : Optimizations and Tradeoffs

❖We now know how to build digital circuits

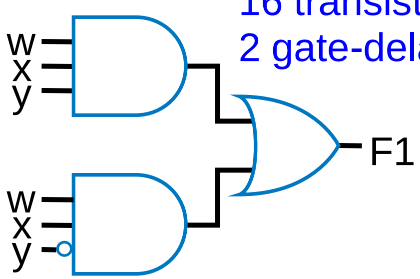
- How can we build ***better*** circuits?

❖Let's consider two important design criteria

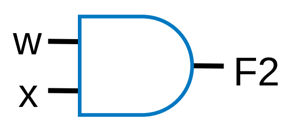
- ***Delay*** – the time from inputs changing to new correct stable output
- ***Size*** – the number of transistors
- For quick estimation, assume
  - Every gate has delay of "1 gate-delay"
  - Every gate *input* requires 2 transistors
  - Ignore inverters

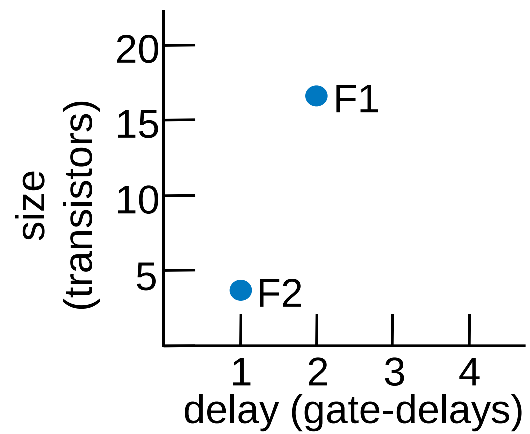
Transforming F1 to F2 represents an ***optimization***
: Better in all criteria of interest

16 transistors
2 gate-delays



4 transistors
1 gate-delay





(c)

F1 = wxy + wxy'　　　　F2 = wx

(a)　　　　(b)

= wx(y+y')　=　wx

a

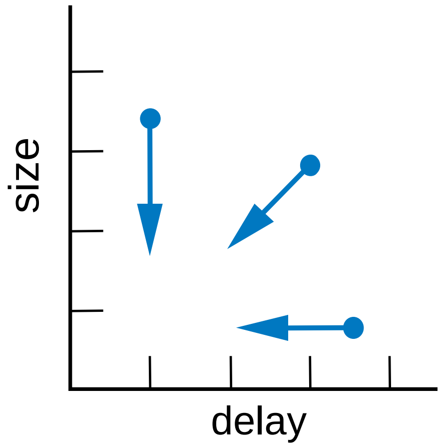
Introduction : Optimizations and Tradeoffs

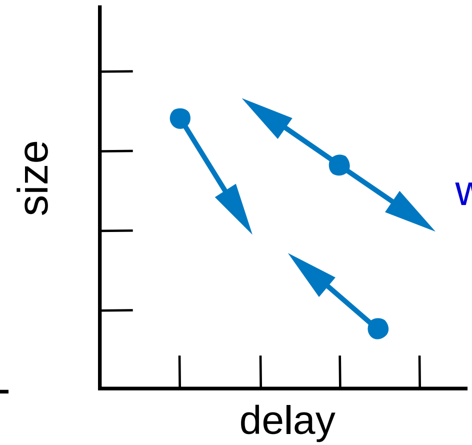
❖Tradeoff : improves one criteria at the expense of another criteria

- Improves some, but worsens other, criteria of interest

Optimizations

All criteria of interest are improved (or at least kept the same)





Tradeoffs

Some criteria of interest are improved, while others are worsened

❖We obviously prefer optimizations, but often must accept tradeoffs

❖Some criteria commonly of interest to digital system designers include.

- Performance : execution time for a computation on the system
- Size : the number of transistors, or silicon area, of a digital system
- Power : the energy consumed by the system per second, directly relating to both the heat generated by the system and to the battery energy consumed by computations.
- etc.

Combinational Logic Optimization and Tradeoffs

- Two-level size optimization using algebraic methods
  - Goal: minimize the number of transistors of Two-level circuit (ORed AND gates)
    . two-level logic size optimization
    . distinguish such optimization
    : performance, power, other optimization
    - Though transistors getting cheaper (Moore's Law), still cost something
- Define problem algebraically
  - Sum-of-products yields two levels
    - F = abc + abc' is sum-of-products; G = w(xy + z) is not.
  - Transform sum-of-products equation to have *fewest literals and terms*
    - Each literal and term translates to a gate input, each of which translates to about 2 transistors.
    - For simplicity, ignore inverters

Example

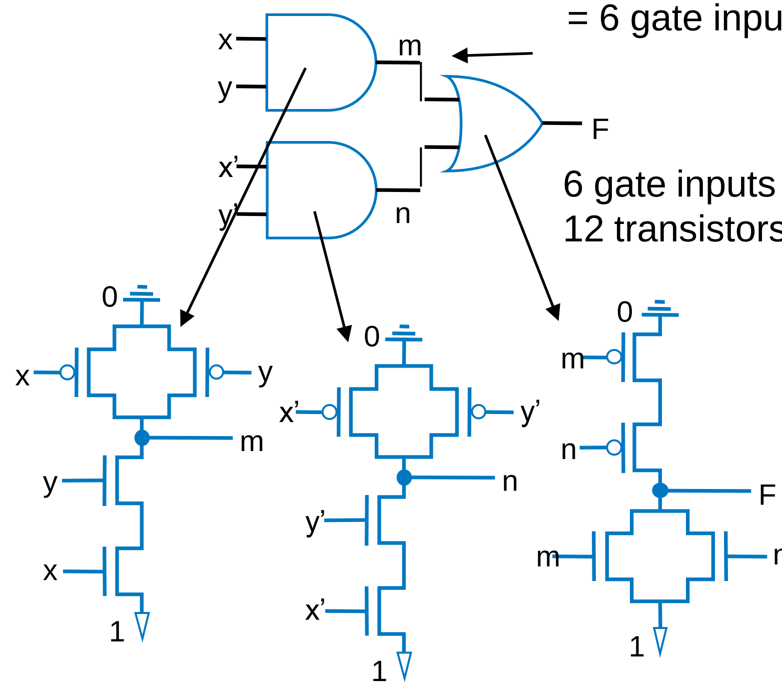
F = xyz + xyz' + x'y'z' + x'y'z

F = xy(z + z') + x'y'(z + z')

F = xy*1 + x'y'*1

F = xy + x'y'

4 literals + 2 terms = 6 gate inputs



6 gate inputs = 12 transistors

Note: Assuming 4-transistor 2-input AND/OR circuits; in reality, only NAND/NOR use only 4 transistors.

Algebraic Two-Level Size Optimization

- Previous example showed common algebraic minimization method
  - (Multiply out to sum-of-products, then...)
  - Apply following as much as possible
    - $ab + ab' = a(b + b') = a*1 = a$
    - "*Combining terms to eliminate a variable*"
      - ✓(Formally called the "*Uniting theorem*")
  - Duplicating a term sometimes helps
    - Doesn't change function
      - ✓$c + d = c + d + d = c + d + d + d + d \ldots$
  - Sometimes after combining terms, can combine resulting terms

Ex 6.1

$F = xyz + xyz' + x'y'z' + x'y'z$

$F = xy(z + z') + x'y'(z + z')$

$F = xy*1 + x'y'*1$

$F = xy + x'y'$

Ex 6.2

$F = x'y'z' + x'y'z + x'yz$

$F = x'y'z' + x'y'z + x'y'z + x'yz$

$F = x'y'(z+z') + x'z(y'+y)$

$F = x'y' + x'z$

Ex 6.3

$G = xy'z' + xy'z + xyz + xyz'$

$G = xy'(z'+z) + xy(z+z')$

$G = xy' + xy$ *(now do again)*

$G = x(y'+y)$

$G = x$

- How did we "see" the opportunities to combine terms to eliminate a variable?
  There is a visual method to help us see opportunities to combine terms to eliminate a variable,
  next section! ⇒ Karnaugh Maps

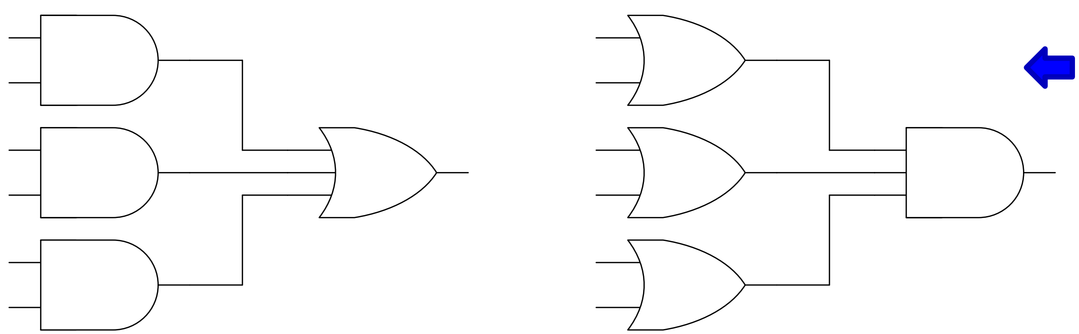
Karnaugh Map (카르노맵) Minimum Forms of Switching Functions

- 부울 함수는 여러 가지 대수학적인 정리들을 이용하여 간략화 될 수 있다.
- 그러나 대수학적인 정리들을 이용한 간략화 방법은

  1. 체계적인 방법을 적용하기가 어렵다.

  2. 완전한 최소식을 얻지 못할 수도 있다.
- AND나 OR 게이트를 사용하여 함수를 회로로 구성하는 경우, 그 구현 비용은 게이트의 개수와 사용된 입력의 개수에 의존한다.
- 카르노맵 방식은 AND 와 OR 게이트로 이루어진 2단 회로를 최소비용으로 구현할 수 있도록 해준다.
- 그러므로 최소비용으로 2단 회로를 구성하기 위해서는 반드시 최소 논리곱의 합식이나 최소 논리합의 곱식을 찾아 내야 한다.

논리곱의 합식의 논리 게이트 표현



논리합의 곱식의 논리 게이트 표현

Minimum Forms of Switching Functions

◆ 최소 논리곱의 합 (minimum sum of product)

- 다음 조건을 만족하는 논리 곱의 합 식으로 정의됨.

  (a) 최소 수의 항들을 가진다.

  (b) 각 항은 각각 최소 개수의 문자를 가진다.

- 또는 다음 조건을 만족하는 2단 게이트 회로에 해당됨.

  (a) 최소 개수의 게이트 수를 가진다.

  (b) 최소 개수의 게이트 입력을 가진다.

- 함수에 대한 최소항 전개와 달리,

  최소 논리곱의 합식은 반드시 하나인 것은 아니다.

  즉 주어진 함수는 같은 항의 수와 문자의 수를 가지는 두 가지 다른 최소식을 가질 수 있다.

Minimum Forms of Switching Functions

- 최소항 식이 주어졌을 때, 최소 논리곱의 합식은 다음 절차를 통해 구한다.
  1. $XY' + XY = X$ 정리를 이용하여 가능한 많은 문자들을 소거한다.
  2. 합의 정리 등을 이용하여 여분의 중복항을 소거한다.
- 위 절차의 결과는 항을 결합하고 소거하는 순서에 의존하므로 얻어진 결과식이 반드시 최소식은 아니다.

예제) $F(a, b, c) = \sum m(0, 1, 2, 5, 6, 7)$

$$F = a'b'c' + a'b'c + a'bc' + ab'c + abc' + abc$$

$$= a'b' \quad + \quad bc' \quad + \quad b'c \quad + \quad ab$$

$$F = a'b'c' + a'b'c + a'bc' + ab'c + abc' + abc$$

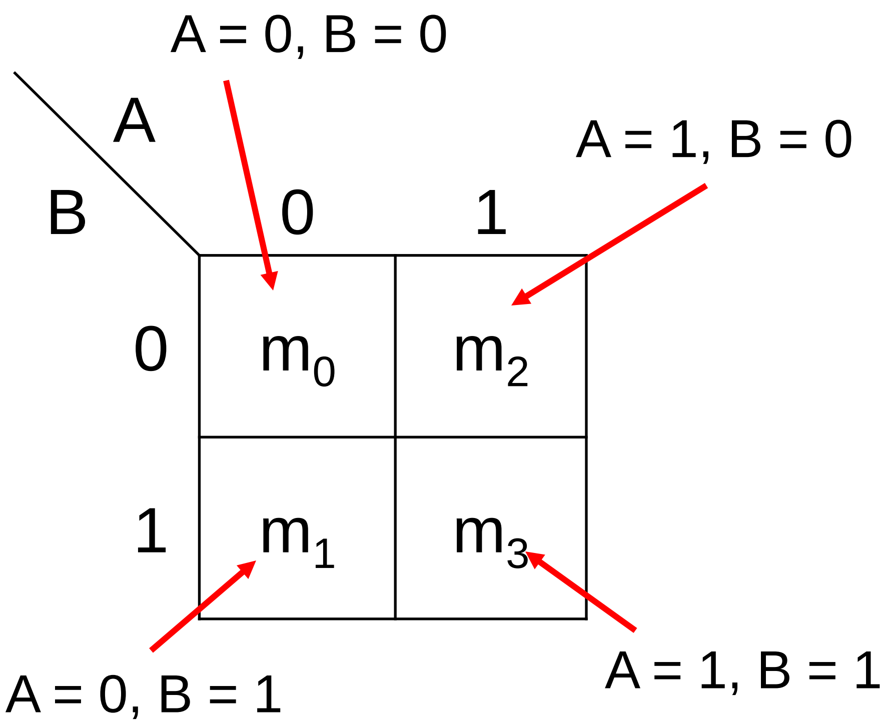
$$= a'b' \quad + \quad bc' \quad + \quad ac$$

Two-and Three-variable Karnaugh Maps

◆ 2변수 카노맵

- 진리표와 마찬가지로, 카르노맵 역시 독립적인 변수 값들의 모든 가능한 조합을 각각의 사각형에 표현 가능하다.
- 2변수 함수 F에 대한 진리표와 이에 해당하는 카르노맵은 다음과 같다.

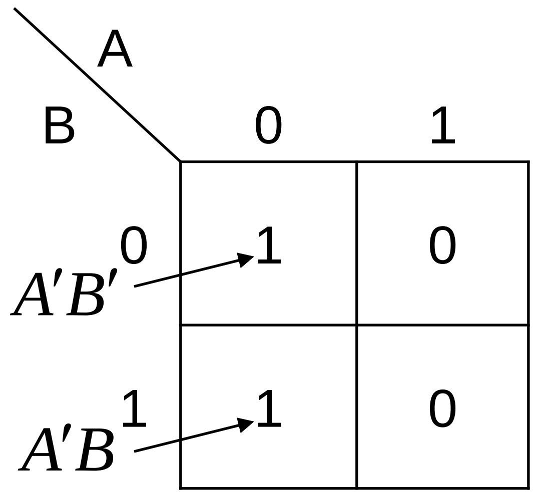
A	B	F
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3



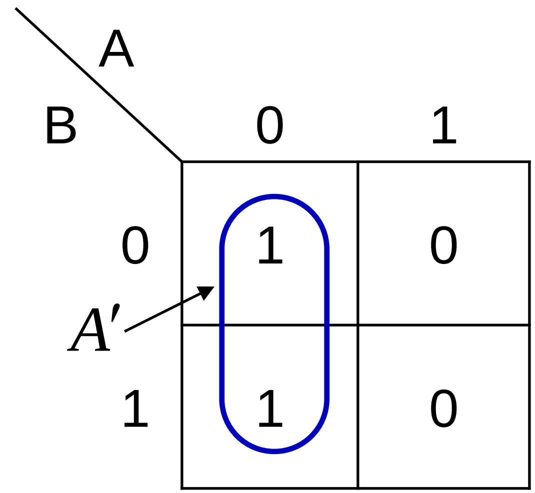
Two-and Three-variable Karnaugh Maps

- 어떤 함수 F에 대한 진리표와 카르노맵은 아래와 같다.
- 카르노맵에서 각 1은 F의 최소항에 해당한다.
- 카르노맵에서 인접한 칸의 최소항들은 하나로 결합시킬 수 있다.

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



$$F = A'B' + A'B$$



$$F = A'B' + A'B$$
$$= A'$$

Two-and Three-variable Karnaugh Maps

◆ 3변수 카르노맵

- 3변수 함수 F에 대한 진리표와 이에 해당하는 카르노맵은 다음과 같다.

  1. 3변수 카르노맵에서는 하나의 변수 값(A)을 맵의 위에 기입하고
  2. 다른 두 변수의 순서쌍(B, C)을 맵의 왼편에 00, 01, 11, 10순으로 기입한다.
  3. 각 조합에 대한 F값을 맵에 채워서 완성한다.

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

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

Two-and Three-variable Karnaugh Maps

- 다음은 3변수 카르노맵에서 최소항의 인접관계를 나타낸 그림이다.
- 물리적으로 인접하지 않아도 한 변수의 값만 다르면 인접한 것으로 규정한다.
- 각 인접항들은 $XY' + XY = X$를 이용하여 결합할 수 있다.

BC \ A	0	1
00	000	100
01	001	101
11	011	111
10	010	110

BC \ A	0	1
00	000	100
01	001	101
11	011	111
10	010	110

BC \ A	0	1
00	000	100
01	001	101
11	011	111
10	010	110

Two-and Three-variable Karnaugh Maps

- 최소항 전개식에 대해 카르노맵에 표시하는 방법
  - 최소항 전개식 : 최소항에 대응하는 칸에 1, 나머지 칸에 0

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

$$F(A,B,C) = A'B'C + AB'C + A'BC$$
$$= \sum m(1,3,5)$$

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

$$F(A,B,C)$$
$$= A'B'C' + ABC + A'BC' + ABC'$$
$$= \sum m(0,2,6,7)$$

Two-and Three-variable Karnaugh Maps

- 곱항을 카르노맵에 표시하는 방법의 예

$$F1 = A'BC + ABC + A'BC' + ABC'$$

$$F2 = A'BC' + ABC'$$

$$F3 = AB'C' + ABC'$$

$F1$

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

B = x1x

$F2$

BC \ A	0	1
00		
01		
11		
10	1	1

BC′ = x10

$F3$

BC \ A	0	1
00		1
01		
11		
10		1

AC′ = 1x0

Two-and Three-variable Karnaugh Maps

- 함수의 대수식을 카르노맵에 표시하는 방법의 예

$$F(A,B,C) = A'B'C' + A'B'C + A'BC + A'BC' + AB'C + ABC'$$

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

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

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

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

A’ $\Rightarrow$ A = 0

B’C $\Rightarrow$ B = 0 , C = 1

BC’ $\Rightarrow$ B = 1, C = 0

Two-and Three-variable Karnaugh Maps

- 카르노맵을 이용한 함수의 간략화 식 유도하는 방법
  1. 간략화 할 함수를 카르노맵에 작성한다.
  2. 인접항들을 결합한다(루프로 묶는다).
  3. 결합한 인접항들을 최소 논리곱의 합 식으로 표현한다.
- 보수 함수 경우 카르노맵의 0을 1로, 1을 0으로 대체하여 위 과정을 수행.

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

$$F = AB'C + A'B'C + A'BC$$
$$= \sum m(1,3,5)$$

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

A’C

B’C

$$F = A'C + B'C$$

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

C’

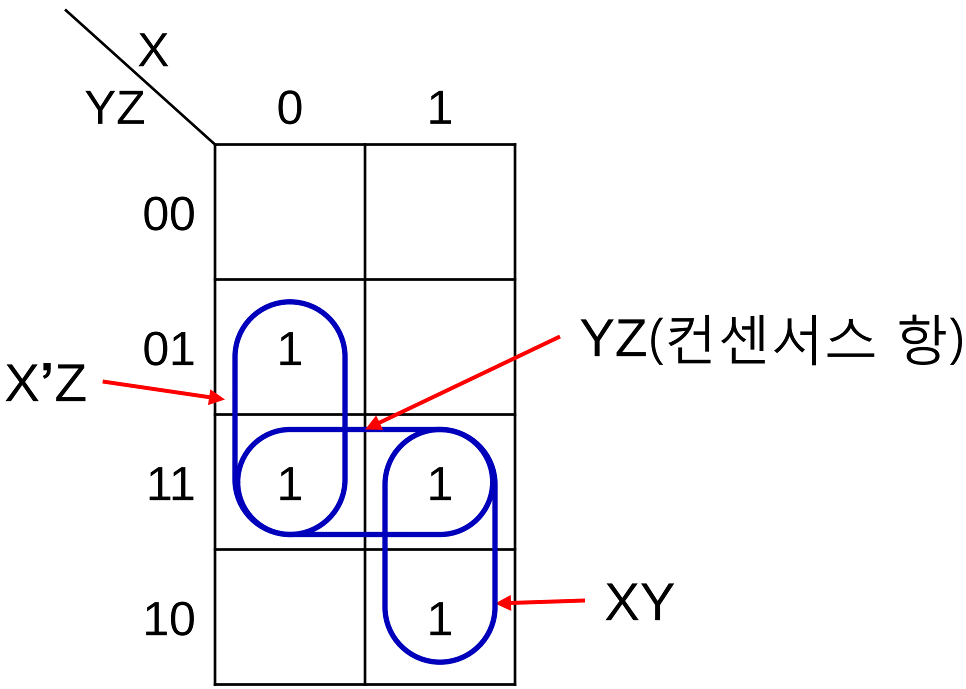
AB

$$F = AB'C' + A'B'C' + ABC$$
$$+ A'BC' + ABC' = C' + AB$$

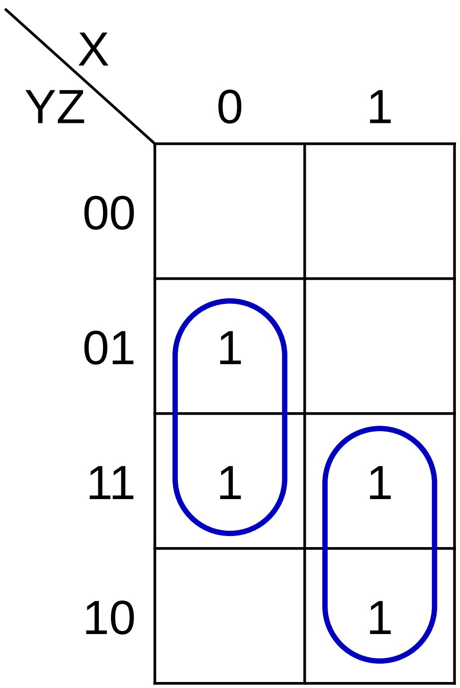
Two-and Three-variable Karnaugh Maps

- 카르노맵을 이용한 부울대수의 기본 정리 증명

$$XY + X'Z + YZ = XY + X'Z$$ (컨센서스 이론)



XY + X'Z + YZ



XY + X'Z

Two-and Three-variable Karnaugh Maps

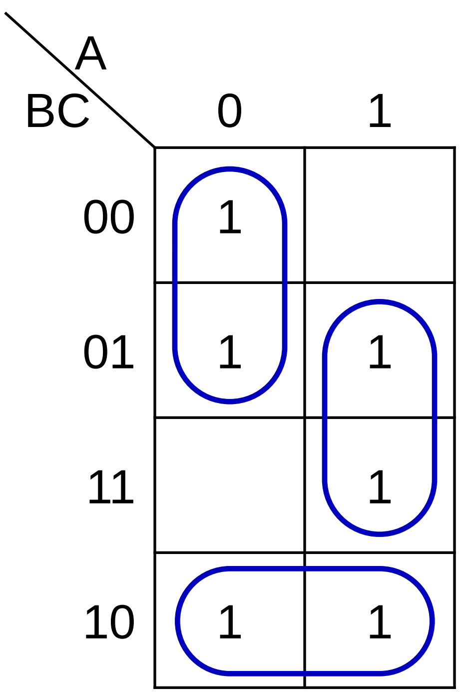
- 함수가 둘 이상의 최소 논리곱의 합 식을 가진다면,

  카르노맵을 통해서 모든 식을 간략화 할 수 있다.

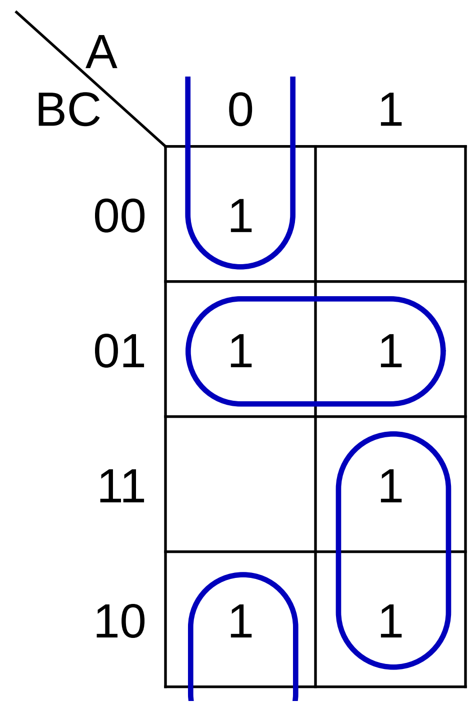
$$F=A'B'C' + A'B'C + A'BC' + AB'C + ABC + ABC'$$

$$=A'B+BC'+AC$$

$$=A'C'+B'C+AB$$



F = A’B + BC’ + AC



F = A’C’ + B’C + AB

Four-variable Karnaugh Maps

◆ 4변수 카르노맵

- 3변수 카르노맵과 마찬가지로 두 변수의 순서쌍 (AB, CD)을 각각 00, 01, 11, 10순으로 기입한다.

A	B	C	D	F
0	0	0	0	m_0
0	0	0	1	m_1
0	0	1	0	m_2
0	0	1	1	m_3
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1	1	1	0	m_{14}
1	1	1	1	m_{15}

CD \ AB	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}

Four-variable Karnaugh Maps

- 다음은 4변수 카르노맵의 예.

(1) $F=A'B'C'D'+A'BC'D'+ABC'D'+AB'C'D'+A'BC'D+A'BCD$

$+ABCD+AB'CD+A'B'CD'+A'BCD'+ABCD'+AB'CD'$

(2) $F=A'BC'D'+ABC'D'+A'B'C'D+A'BC'D+ABC'D+A'B'CD+AB'CD'$

$$F(A,B,C,D) = A'B + AC + D'$$

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

A'B

D'

AC

$$F(A,B,C,D) = BC' + A'B'D + AB'CD'$$

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

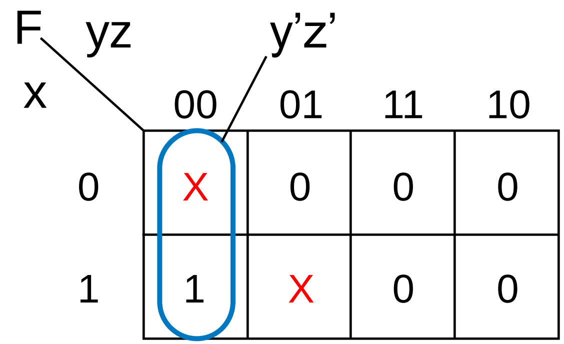
BC'

A'B'D

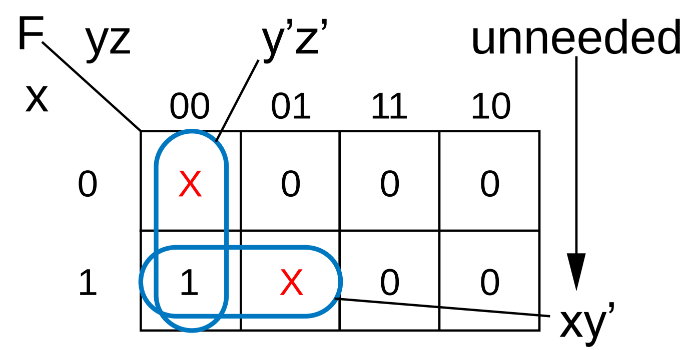
AB'CD'

Don’t Care Input Combinations

- ❖ 함수 F의 특정 입력값이 출력값에 영향을 끼치지 않는 경우, 이때의 특정 입력조합을 무관항(don’t care term)이라 한다.
- ❖ What if we know that particular input combinations can never occur?
  - e.g., Minimize F = xy’z’, given that x’y’z’ (xyz=000) can *never* be true, and that xy’z (xyz=101) can *never* be true
  - So it doesn’t matter what F outputs when x’y’z’ or xy’z is true, because those cases *will never occur*
  - Thus, make F be 1 or 0 for those cases *in a way that best minimizes the equation*
- ❖ On K-map
  - Draw **X**s for don’t care combinations
    - Include X in circle ONLY if minimizes equation
    - Don’t include other Xs



Good use of don’t cares



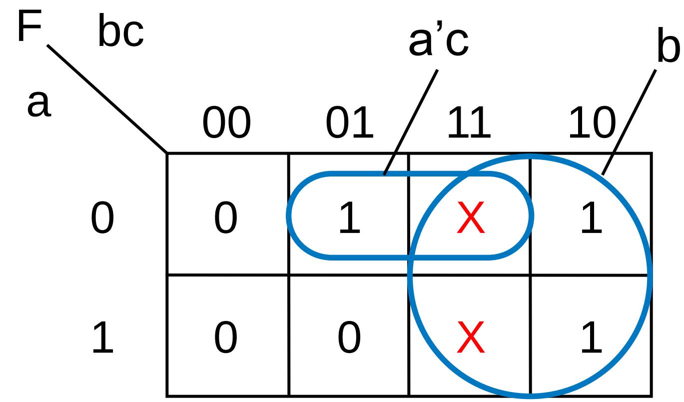
Unnecessary use of don’t cares; results in extra term

$$F = xy'z' = y'z' \neq xy'$$

Optimization Example using Don't Cares

❖ Minimize:

- F = a'bc' + abc' + a'b'c
- Given don't cares: a'bc, abc
  don't care mean that bc can never be 11
- F=a'c+b : with two don't care Xs
  number of transistors: (2+0+2)$*$2=8
- F=a'b'c+bc' : without don't cares
  number of transistors: (3+2+2)$*$2=14



F = a'c + b

❖ Note: Introduce don't cares with caution

- Must be *sure* that we really don't care what the function outputs for that input combination
- If we do care, even the slightest,
  then it's probably safer to set the output to 0

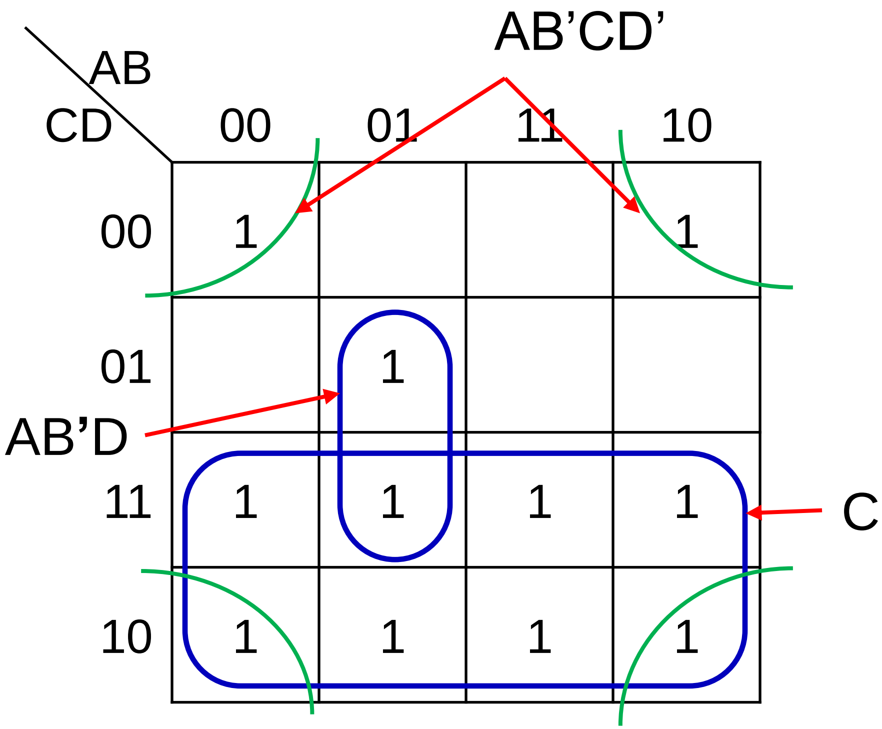
Four-variable Karnaugh Maps

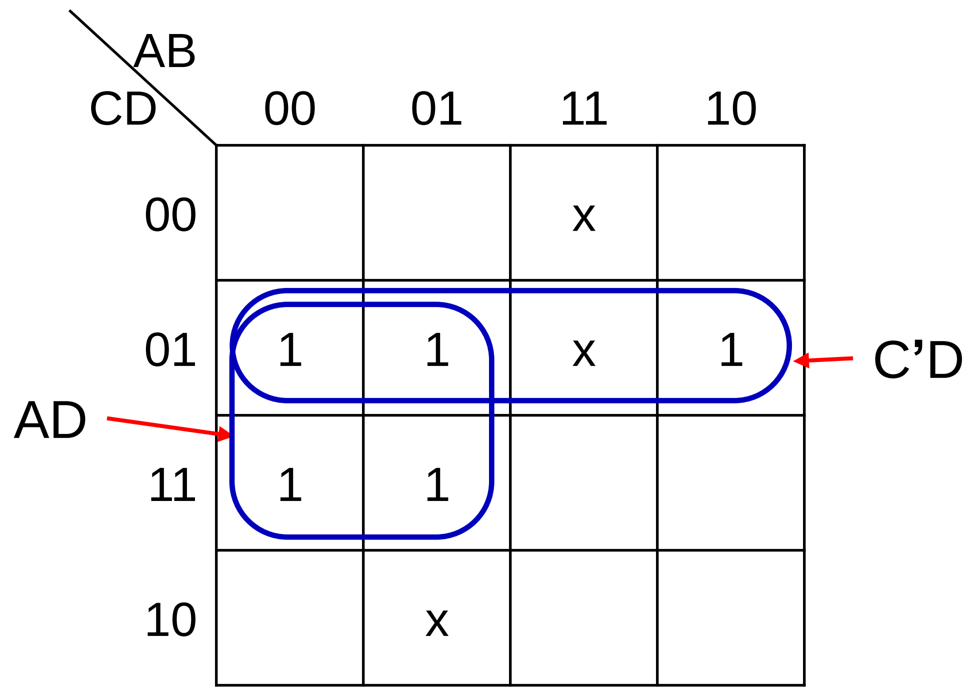
- 다음은 4변수 카르노맵을 이용한 함수 간략화의 예이다.

(좌) $F_2(A, B, C, D) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$

(우) 무관항을 포함한 4변수 카르노맵 예. 무관항은 필요할 경우에만 1로 사용.

$$F(A, B, C, D) = \sum m(1, 3, 5, 7, 9) + \sum d(6, 12, 13)$$



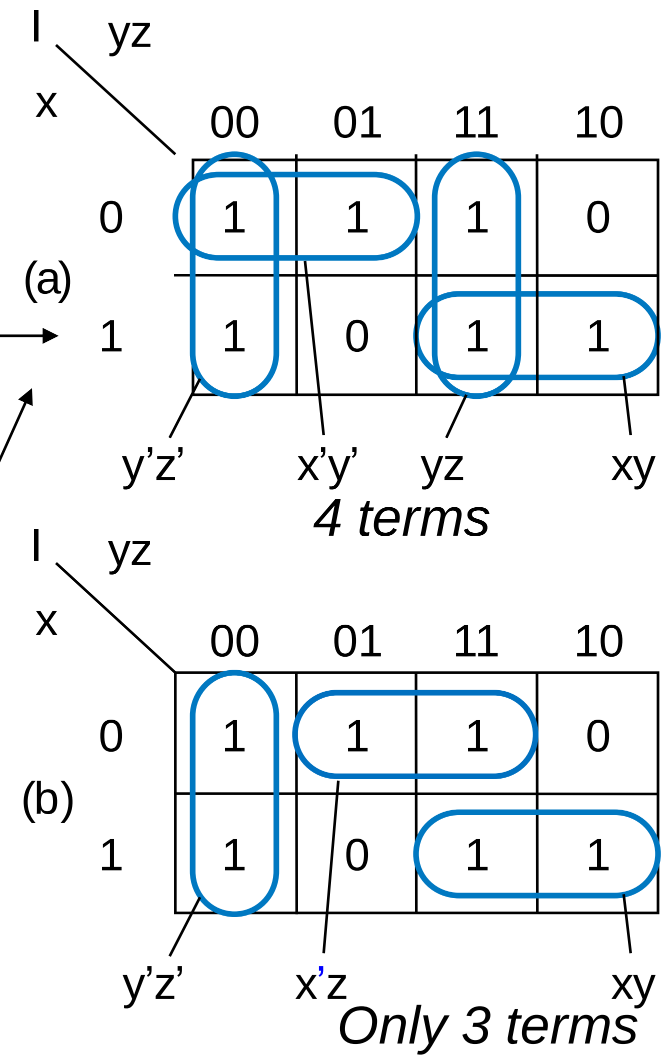


$F_2(A, B, C, D) = C + B'D' + A'BD$

$F(A, B, C, D) = AD + C'D$

Automating Two-Level Logic Size Optimization

- ❖ Minimizing by hand
  - Is hard for functions with 5 or more variables
  - May not yield minimum cover depending on order we choose
  - Is error prone
- ❖ Minimization thus typically done by automated tools
  - ***Exact algorithm*** : finds optimal solution
  - ***Heuristic*** : finds good solution, but not necessarily optimal



Determination of Minimum Expressions Using Essential Prime Implicants

❖ 관련항(implicant)

- 카르노맵에서 1이 하나 있거나 루프로 묶여진 1들을 관련항이라 한다.

❖주항(prime implicant)

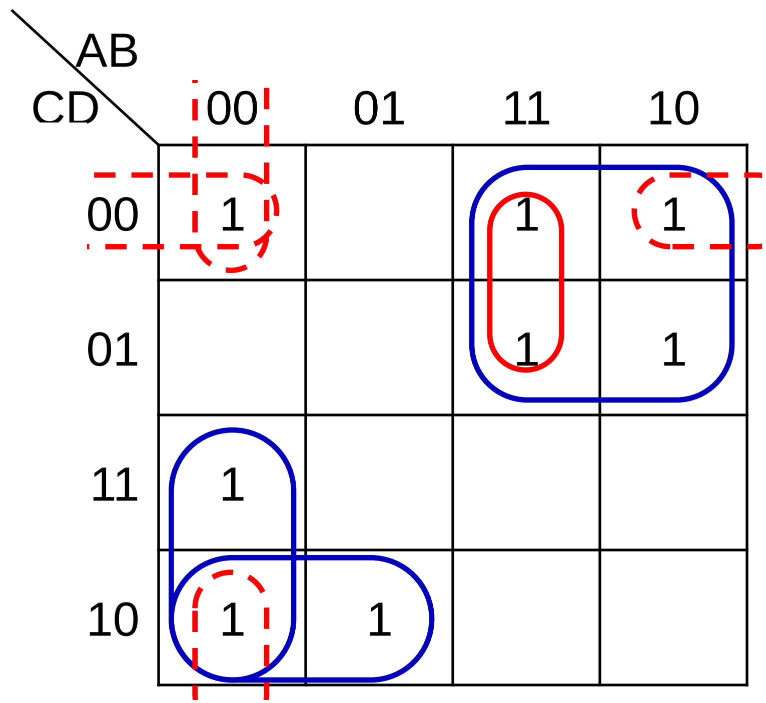
- 관련항 중에서 더 이상 다른 항과 결합될 수 없는 관련항을 주항이라 한다.

✓ A’B’C’D’는 주항이 아니다.

✓ A’B’C, A’CD’는 주항이다.

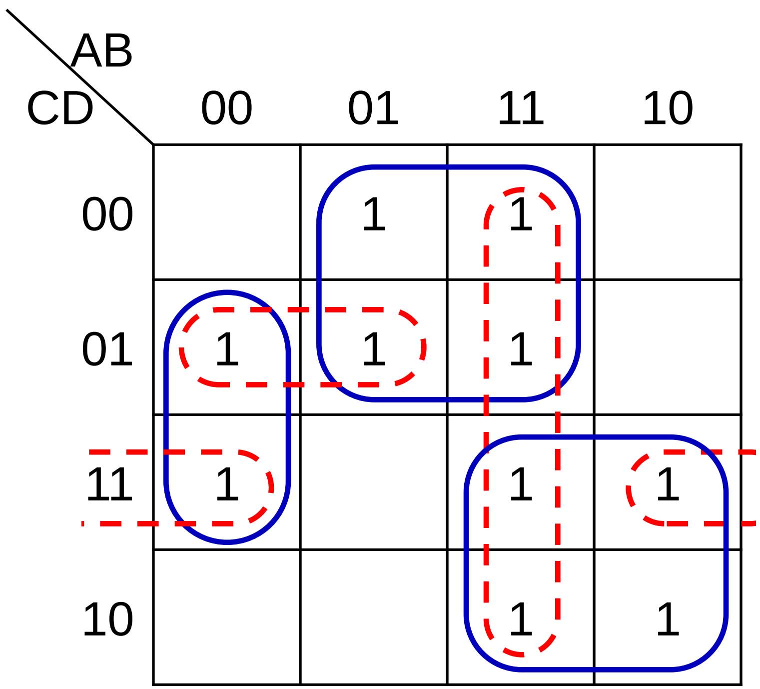
✓ ABC’는 주항이 아니다.

✓ AC’는 주항이다.



Determination of Minimum Expressions Using Essential Prime Implicants

- 어떤 함수의 최소 논리곱의 합 식은 몇 개의 주항으로 구성된다 (반드시 그런 것은 아니다).
- 카르노맵 상에서 최소 논리곱의 합 식을 찾기 위해서, 모든 1을 포함할 수 있는 최소 개수의 주항을 찾아야 한다.
- 무관항은 1로 취급하나 무관항으로 구성된 최소식은 인정하지 않는다.
- 모든 주항이 최소 논리곱의 합 식에 포함되는 것은 아니다.



모든 주항
: A’B’D, BC’, AC, A’C’D, AB, B’CD

최소식 F = A’B’D + BC’ + AC

Determination of Minimum Expressions Using Essential Prime Implicants

◆ 필수주항(essential prime implicant)

- 하나의 최소항이 단 하나의 주항에만 포함되면
  이 주항을 필수주항이라 한다.
- 필수주항은 최소 논리곱의 합 식에 포함시켜야 한다.

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

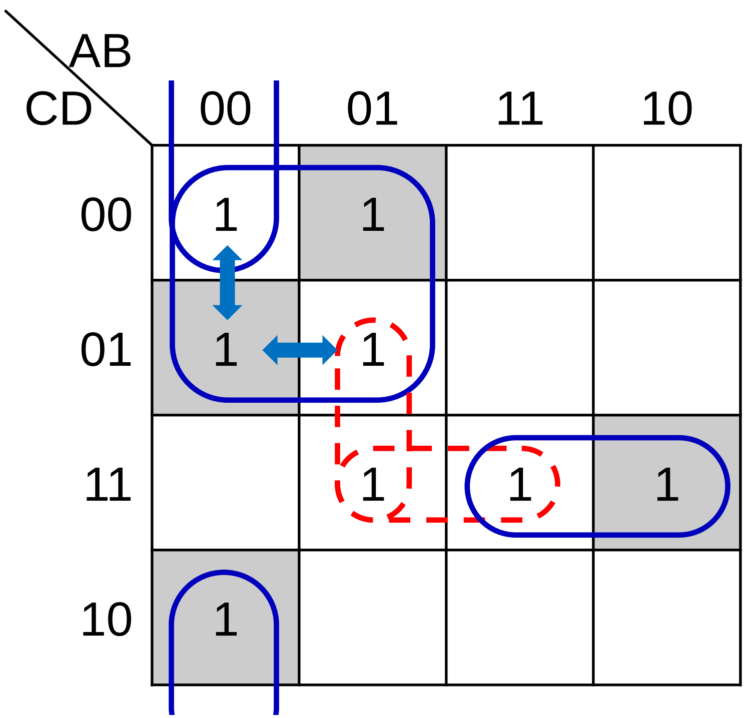
모든 주항
: BD, B’C, AC, CD

A’BC’D, A’B’CD’, ABCD’는 각각
BD, B’C, AC에만 포함되므로
필수주항이다.

따라서,
최소식 F = BD + B’C + AC

Determination of Minimum Expressions Using Essential Prime Implicants

- 카르노맵에서 최소 논리곱의 합 식을 찾기 위해
  먼저 모든 필수주항을 루프로 묶어야 한다.
- 아직 포함되지 않은 1이 단 하나의 주항에 포함되면 그 주항은 필수주항
- 주어진 최소항과 인접한 모든 1이 단 하나의 항에 포함되면
  그 항은 필수주항(인접한 무관항도 1로 취급)



A′B′C′D와 인접한 A′B′C′D′, A′BC′D는 모두 A′C′에 포함되므로 A′C′는 필수주항

마찬가지로 A′B′CD′에 대해서는 A’B’D’, AB’CD에 대해서는 ACD가 필수주항

따라서 F의 최소 논리곱의 합 식은
A’C’ + A’B’D’ + ACD + A’BD(or BCD)

Determination of Minimum Expressions Using Essential Prime Implicants

- 카르노맵에서 최소 논리곱의 합 식을 찾기 위한 과정은 다음과 같다.
  1. 아직 루프로 묶여진 항에 포함되지 않은 최소항(하나의 1)을 선택
  2. 이 최소항에 인접하는 모든 1과 x를 찾는다.(n-변수 맵에서는 n개)
  3. 하나의 단일 항이 이 최소항과 인접한 모든 1과 x를 포함할 때, 이 단일 항은 필수주항이므로 이 단일 항을 선택
  4. 모든 필수주항이 선택될 때까지 과정 1, 2, 3 반복
  5. 나머지 1을 포함하는 주항의 최소 집합을 찾음.

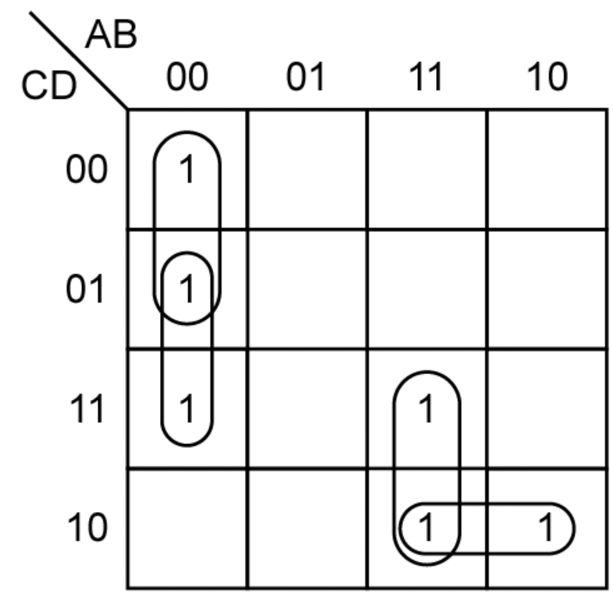
Other Uses of Karnaugh Maps

- 식의 인수화를 쉽게 할 수 있다.

첫 행의 두 항은 A′B′를 공통으로 가지고 있다.
두 항은 A′B′(C′+D)로 표현 할 수 있다.

나머지 두 항은 AC를 공통으로 가지고 있다.
두 항은 AC(B+D′)로 표현 할 수 있다.

함수 F 이들 항의 합으로 표현 할 수 있다.



F = A'B'(C' + D) + AC(B + D')