Advanced Logic Design
Lecture 2: Combinational Logic Circuits Refresher

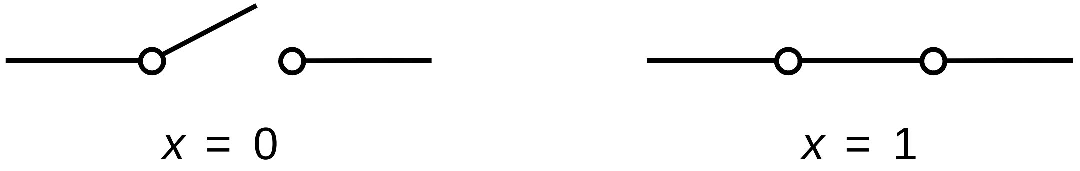
Mingoo Seok
Columbia University

BV: Sec. 2.1-2.8, 2.11-2.22

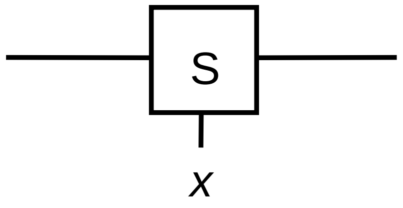
Logic Circuits

- Logic circuits: in which the signals are constrained to have only some number of discrete values
- Binary logic: only two values, 0 and 1
  - Other logic circuits exist: e.g., ternary (-1,0,+1)
  - But binary logic has an advantage in robust and scalable VLSI hardware implementation

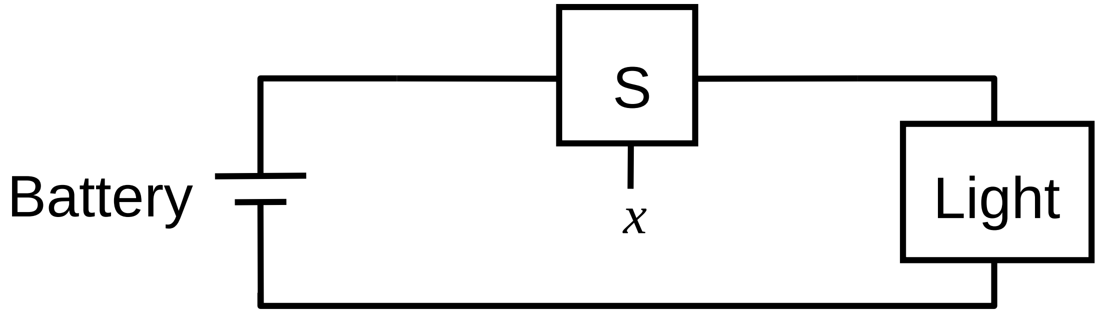
Binary Switch



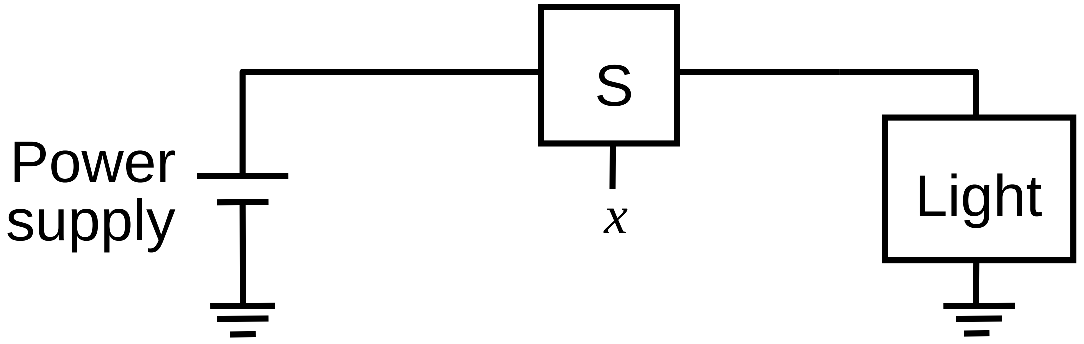
(a) Two states of a switch



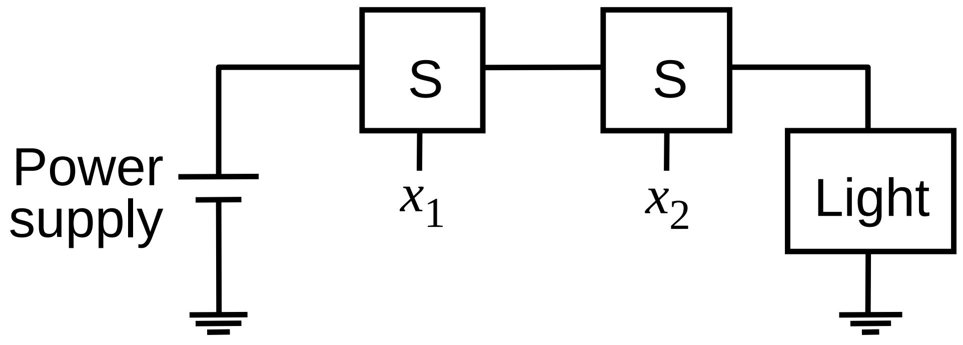
(b) Symbol for a switch



(a) Simple connection to a battery

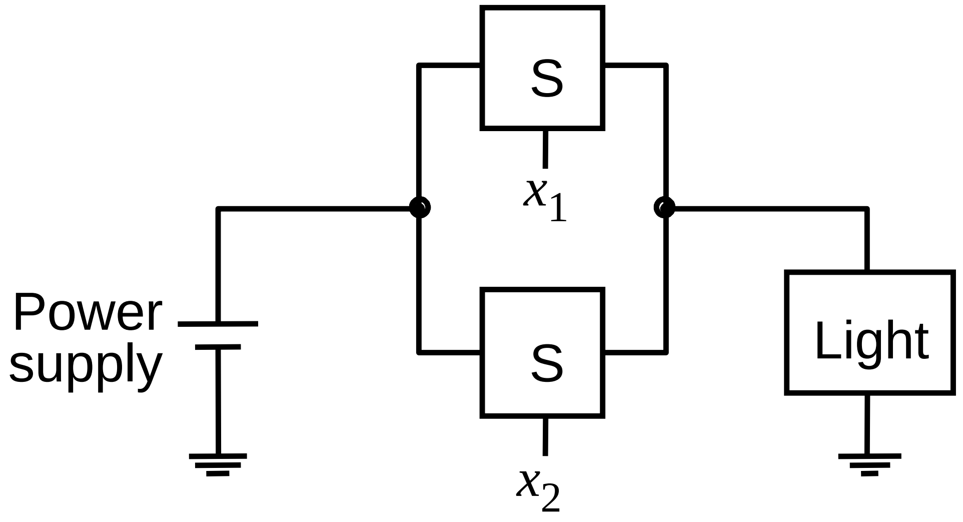


(b) Using a ground connection as the return path



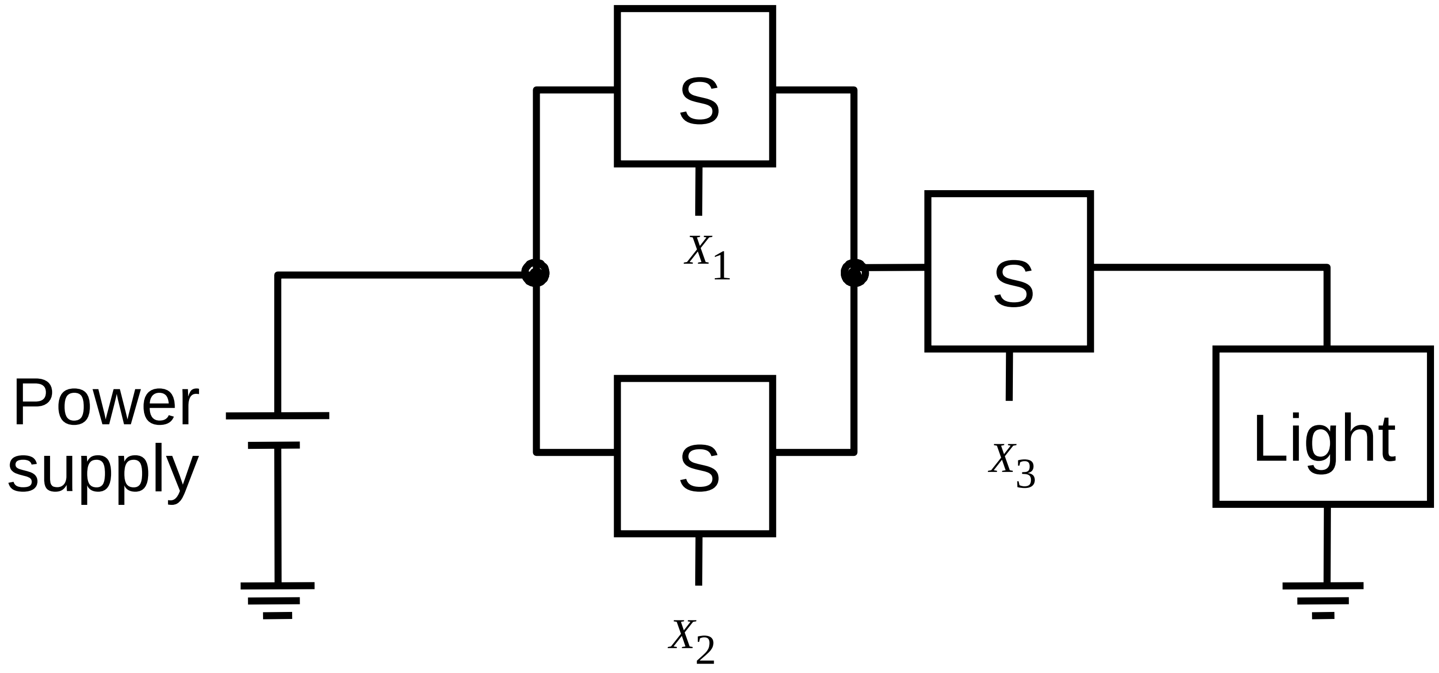
$$L(x_1, x_2) = x_1 \cdot x_2$$
$$\text{where} \quad L = 1 \text{ if } x_1 = 1 \text{ and } x_2 = 1,$$
$$L = 0 \text{ otherwise.}$$

(a) The logical AND function (series connection)



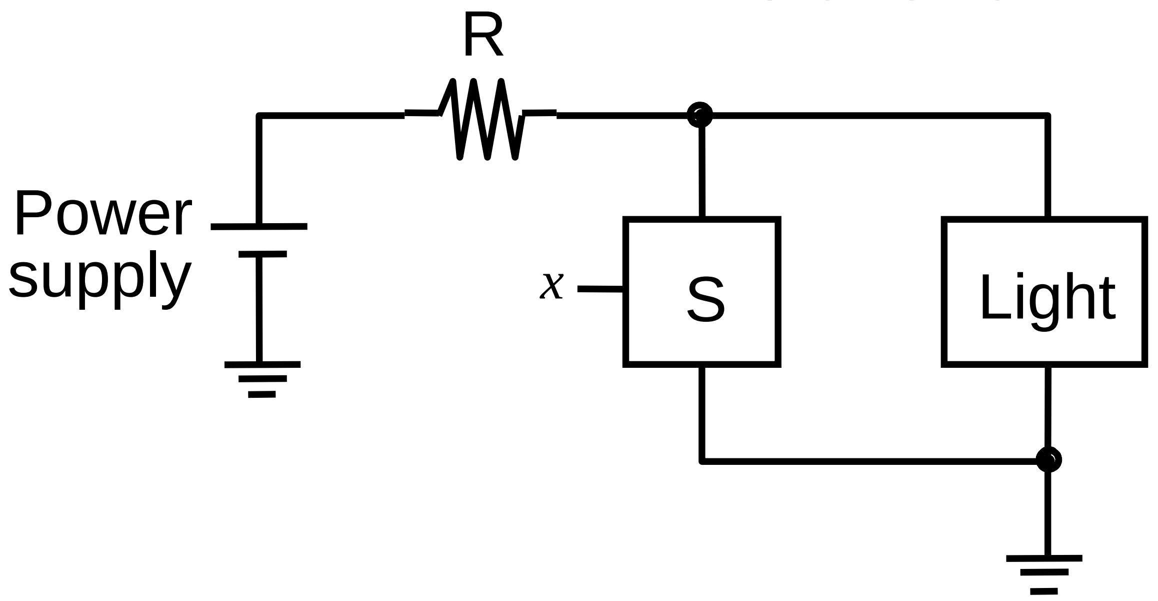
$$L(x_1, x_2) = x_1 + x_2$$
$$\text{where} \quad L = 1 \text{ if } x_1 = 1 \text{ or } x_2 = 1 \text{ or if } x_1 = x_2 = 1,$$
$$L = 0 \text{ if } x_1 = x_2 = 0.$$

(b) The logical OR function (parallel connection)



$$L(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$$

Inversion



$$L(x) = \overline{x}$$
$$\text{where} \quad L = 1 \text{ if } x = 0,$$
$$L = 0 \text{ if } x = 1$$

$$\overline{x} \;=\; x' \;=\; !x \;=\; \sim x \;=\; \text{NOT } x$$

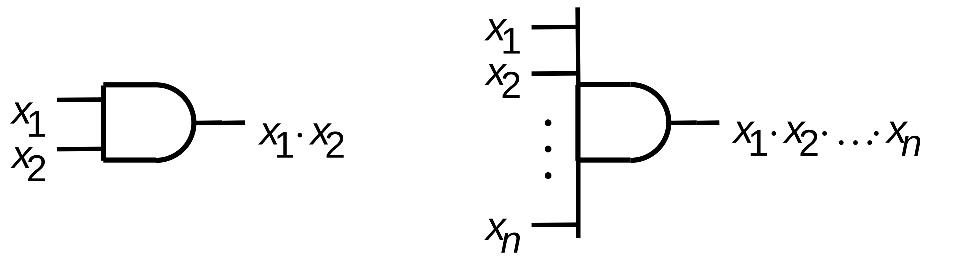
- The switch will short-circuit the light and prevent the current from flowing through it
- Inverse = complement = NOT

Truth Table

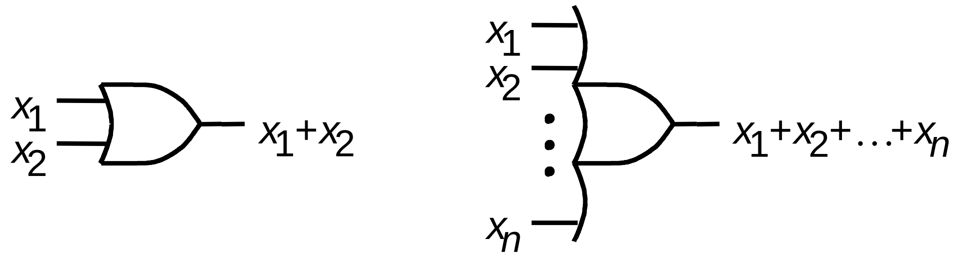
x_1	x_2	$x_1 \cdot x_2$	$x_1 + x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1
		AND	OR

- Logic function defined in the form of a table

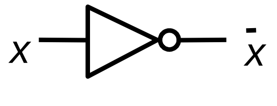
Logic Gates and Networks



(a) AND gates

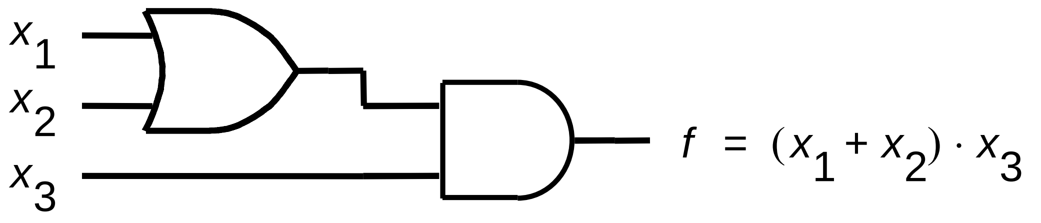


(b) OR gates

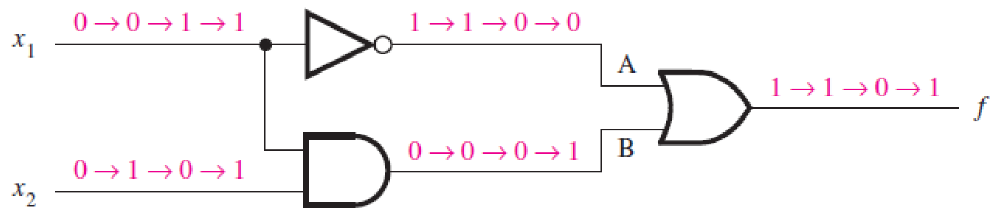


(c) NOT gate

Logic Analysis & Synthesis



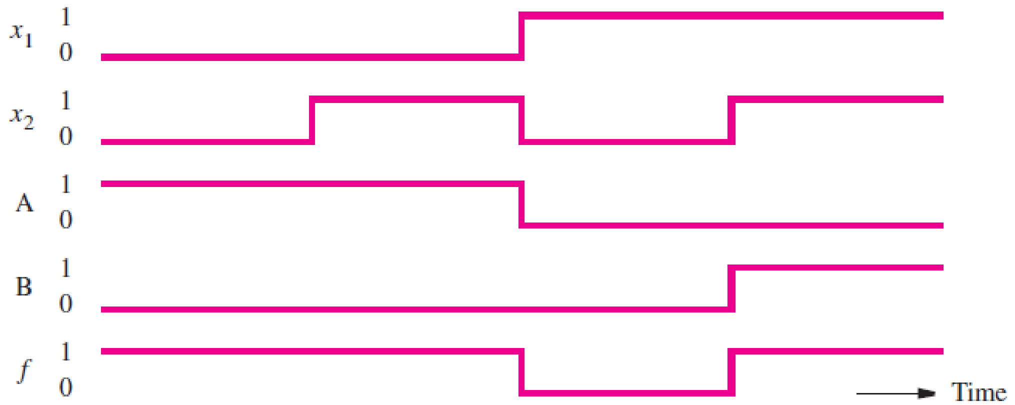
- A digital system designer needs:
  - Determine the function performed by a logic network → ***Logic analysis***
  - Designing a new (logic) network that implements a desired function behavior → ***Logic synthesis***



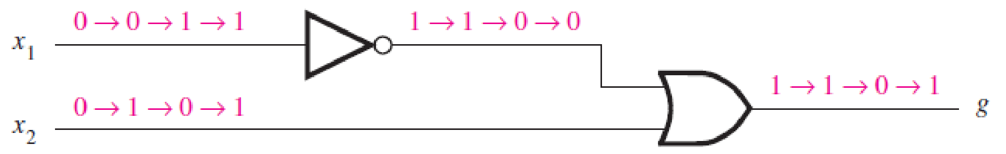
(a) Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$

x_1	x_2	$f(x_1, x_2)$	A	B
0	0	1	1	0
0	1	1	1	0
1	0	0	0	0
1	1	1	0	1

(b) Truth table



(c) Timing diagram



(d) Network that implements $g = \bar{x}_1 + x_2$

Functionally equivalent networks

→$f(x_1,x_2)$ and $g(x_1,x_2)$ are functionally equivalent

→It makes sense to use the simpler one, which is less costly to implement

Boolean Algebra

- In 1849 George Boole published a scheme for the algebraic description of processes involved in logical thoughts and reasoning
- In the late 1930s Claude Shannon showed that Boolean algebra provides an effective means of describing circuits built with switches

Axioms (Assumptions)

$1a.\quad 0 \cdot 0 = 0$

$1b.\quad 1 + 1 = 1$

$2a.\quad 1 \cdot 1 = 1$

$2b.\quad 0 + 0 = 0$

$3a.\quad 0 \cdot 1 = 1 \cdot 0 = 0$

$3b.\quad 1 + 0 = 0 + 1 = 1$

$4a.$ If $x = 0$, then $\overline{x} = 1$

$4b.$ If $x = 1$, then $\overline{x} = 0$

Single-Variable Theorem

5*a*.	$x \cdot 0 = 0$
5*b*.	$x + 1 = 1$
6*a*.	$x \cdot 1 = x$
6*b*.	$x + 0 = x$
7*a*.	$x \cdot x = x$
7*b*.	$x + x = x$
8*a*.	$x \cdot \overline{x} = 0$
8*b*.	$x + \overline{x} = 1$
9.	$\overline{\overline{x}} = x$

Properties

10*a*.	$x \cdot y = y \cdot x$	*Commutative*
10*b*.	$x + y = y + x$	
11*a*.	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	*Associative*
11*b*.	$x + (y + z) = (x + y) + z$	
12*a*.	$x \cdot (y + z) = x \cdot y + x \cdot z$	*Distributive*
12*b*.	$x + y \cdot z = (x + y) \cdot (x + z)$	
13*a*.	$x + x \cdot y = x$	*Absorption*
13*b*.	$x \cdot (x + y) = x$	
14*a*.	$x \cdot y + x \cdot \overline{y} = x$	*Combining*
14*b*.	$(x + y) \cdot (x + \overline{y}) = x$	
15*a*.	$\overline{x \cdot y} = \overline{x} + \overline{y}$	*DeMorgan's theorem*
15*b*.	$\overline{x + y} = \overline{x} \cdot \overline{y}$	
16*a*.	$x + \overline{x} \cdot y = x + y$	
16*b*.	$x \cdot (\overline{x} + y) = x \cdot y$	
17*a*.	$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$	*Consensus*
17*b*.	$(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z)$	

More Comments

- Duality
  - Dual of any true statement in Boolean algebra is also a true statement
  - Implying that at least two different ways exist to express every logic function with Boolean algebra; often one expression leads to a simpler physical implementation than the other
- Huntington’s basic postulates
  - Theorems 5 and 8 and Properties 10 and 12
  - All the other identities can be derived from these postulates

More Comments

- OR
  - Logical sum
  - +
- AND
  - Logical product
  - ·
- Operation order
  - NOT → AND → OR
  - To be clear, use parentheses for a complex logic function

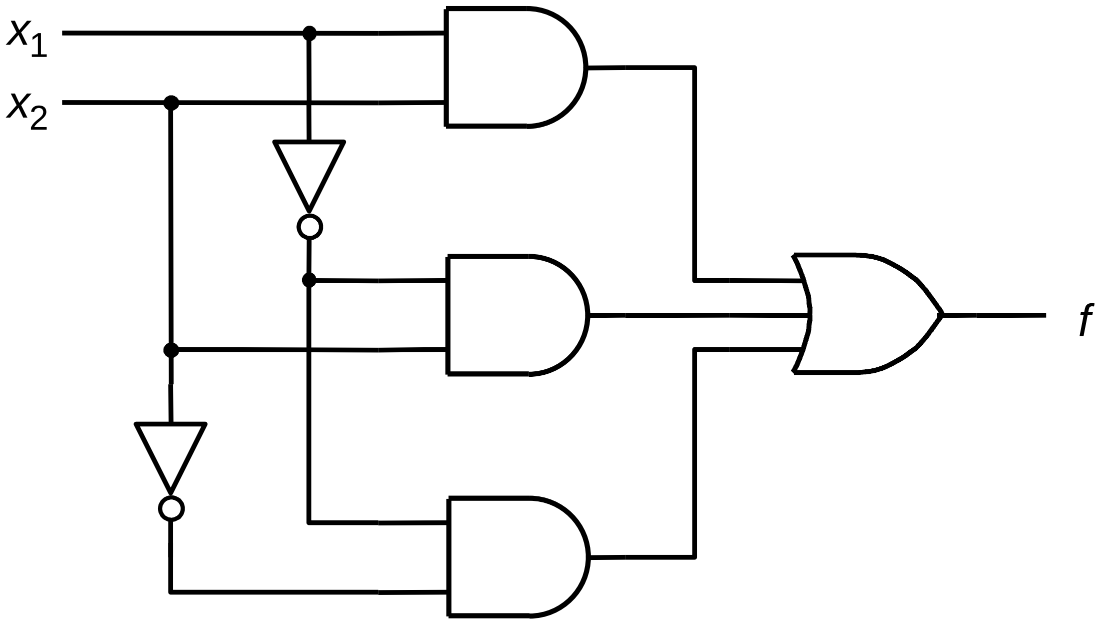
Logic Analysis & <u>Synthesis</u>

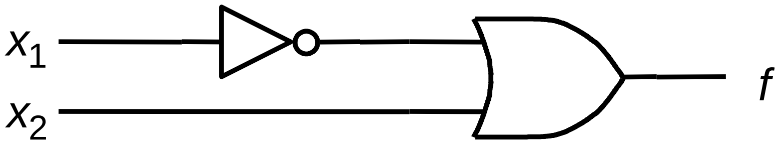
x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

$$f(x_1, x_2) = x_1x_2 + \overline{x}_1\overline{x}_2 + \overline{x}_1x_2$$

- A possible procedure for designing a logic circuit that implements this truth table is to create a product term that has a value of 1 for each valuation for which the output function *f* has to be 1. Then we can take a logical sum of these product terms to realize *f*.

Logic Synthesis





- Not necessarily the optimal in terms of implementation cost (i.e., many logic gates)
- Further simplification using Boolean algebra's theorems and properties

Sum of Products

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

- Minterms: a product term in which each of the n variables appears once
- Sum of minterms → sum of products
- Maxterms: the complements of minterms

Sum of Products

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3$$

$$\begin{aligned} f &= (\bar{x}_1 + x_1)\bar{x}_2x_3 + x_1(\bar{x}_2 + x_2)\bar{x}_3 \\ &= 1 \cdot \bar{x}_2x_3 + x_1 \cdot 1 \cdot \bar{x}_3 \\ &= \bar{x}_2x_3 + x_1\bar{x}_3 \end{aligned}$$

$$f(x_1, x_2, x_3) = \sum(m_1, m_4, m_5, m_6)$$

Product of Sums

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

- Maxterm: the dual of minterm
- Product of sums: the dual of SoP

Maxterms; Product of Sums

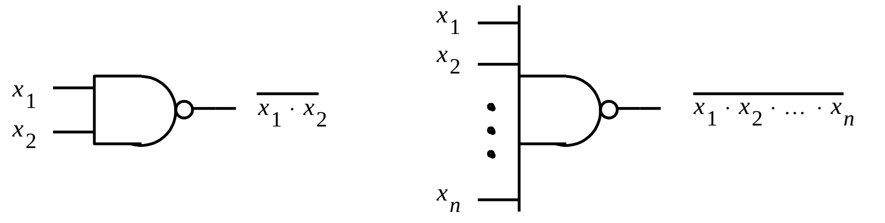
Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

$$\begin{aligned}\overline{f}(x_1, x_2, x_3) &= m_0 + m_2 + m_3 + m_7 \\ &= \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1 x_2\overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 x_2 x_3\end{aligned}$$

$$\begin{aligned}f &= \overline{m_0 + m_2 + m_3 + m_7} \\ &= \overline{m}_0 \cdot \overline{m}_2 \cdot \overline{m}_3 \cdot \overline{m}_7 \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_7 \\ &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)\end{aligned}$$

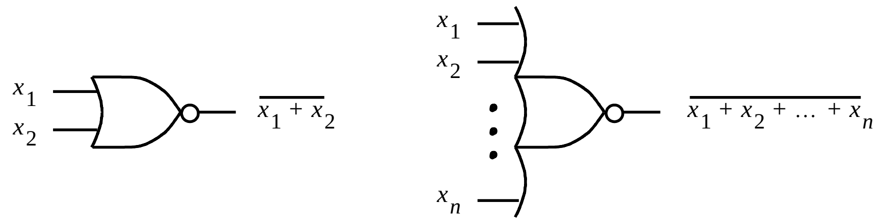
$$f(x_1, x_2, x_3) = \Pi(M_0, M_2, M_3, M_7)$$

NAND and NOR Networks



x_1
x_2
$\overline{x_1 \cdot x_2}$
x_1
x_2
x_n
$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$

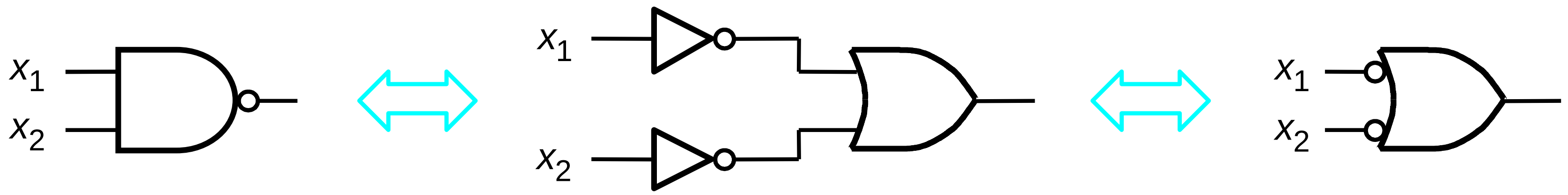
(a) NAND gates



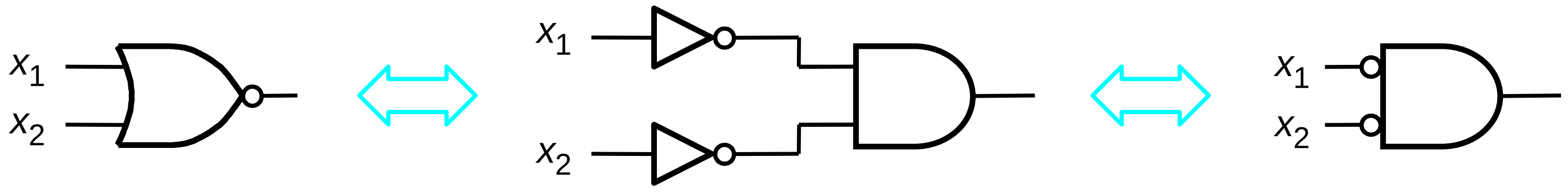
(b) NOR gates

- Considering CMOS implementation technology, NAND and NOR attractive as they are implemented with simpler electronic circuits than AND and OR

DeMorgan's Theorem

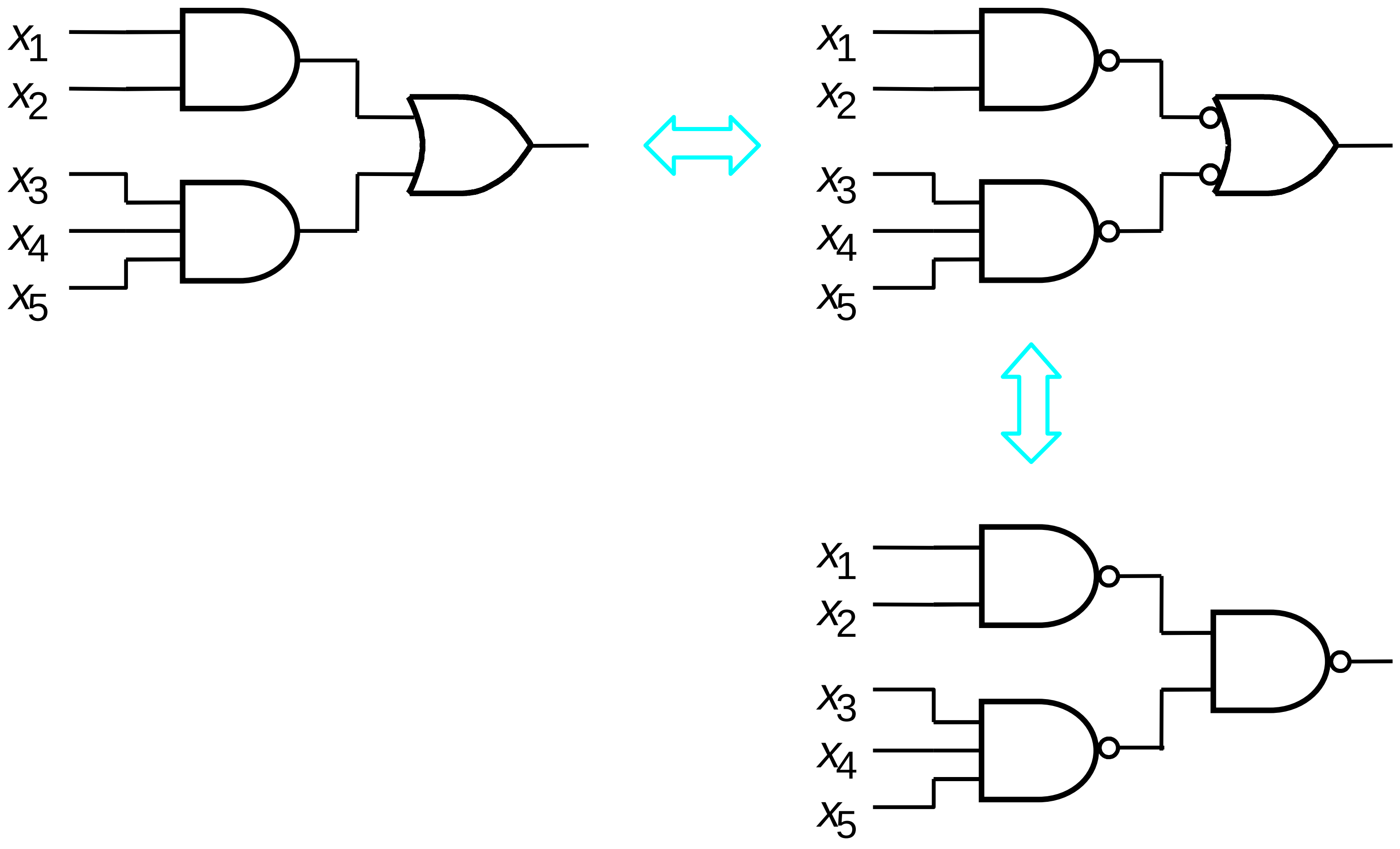


(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

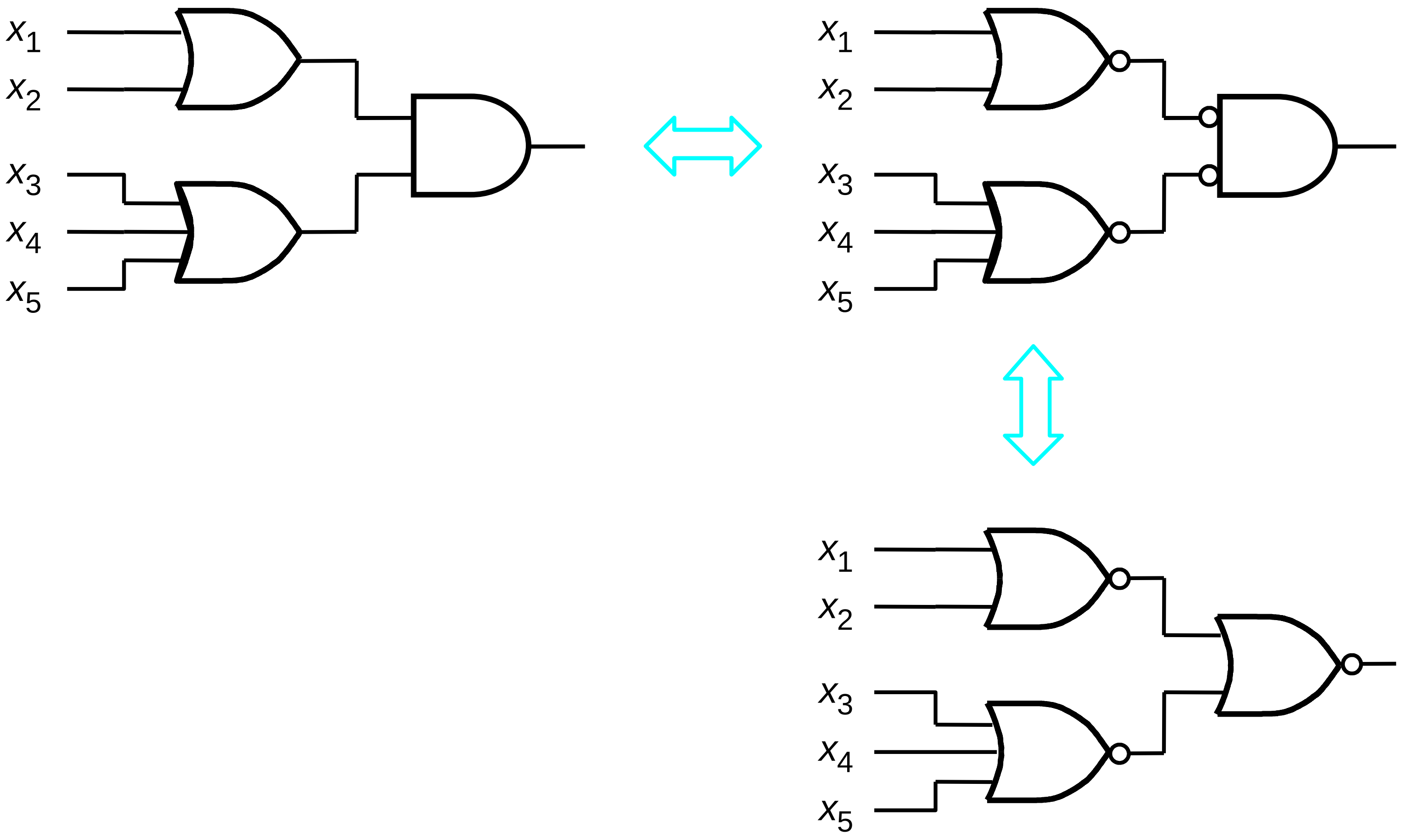


(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

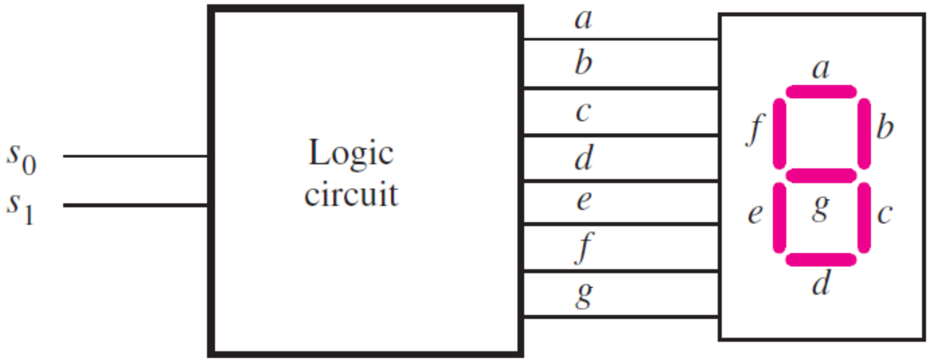
NAND for a SoP Network



NOR for a PoS Network



Number Display Logic Function Example



(a) Logic circuit and 7-segment display

	s_1	s_0	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	0
1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	1	1	0	1

(b) Truth table

$$a = d = e = \bar{s}_0$$

$$b = 1$$

$$c = \bar{s}_1$$

$$f = \bar{s}_1\bar{s}_0$$

$$g = s_1\bar{s}_0$$

Karnaugh Map: A Method to Minimize Logic Circuits

x_1	x_2	x_3	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

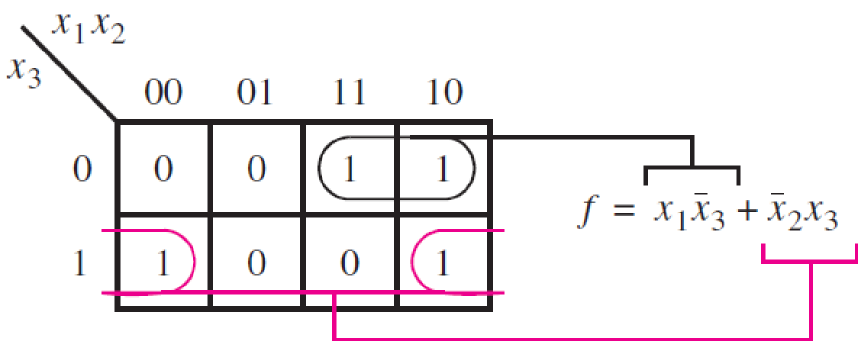
(a) Truth table

x_3 \ $x_1 x_2$	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

(b) Karnaugh map

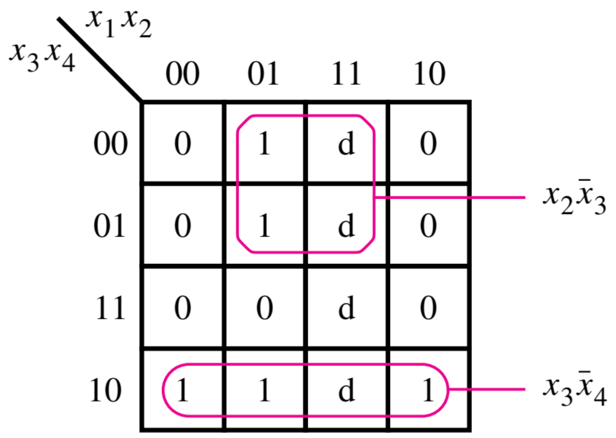
Karnaugh Map: A Method to Minimize Logic Circuits

Row number	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



$$\begin{array}{c|cccc} x_3 \backslash x_1x_2 & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{array}$$

Incompletely Specified Function



$$f(x_1, \ldots, x_4) = \sum m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$$

- Certain input conditions can never occur
- We can set the outputs of such input conditions for the given logic function f to *don't care'*s (DCs)
- A logic function definition w/ don't care's

Strategy for General Logic Minimization

Literal: a variable (x) or its complement (x')

Product: an "AND" of literals (e.g. xy'z, a'bcd')

Cube: a product (another equivalent name)

Minterm: a product including a literal for every input of the function

Example: If a function has 3 inputs, A/B/C, then A'BC' is a minterm, but A'C is not.

A minterm is also an input vector or combination (i.e. corresponds to a single row in the truth table)

ON-set minterm: minterm where the function is 1

OFF-set minterm: minterm where the function is 0

DC-set minterm: minterm where the function is DC (-)

Implicant: a cube/product which contains no OFF-set minterm (i.e. 0 value)

Prime Implicant (PI, prime): a maximal implicant (i.e. it is contained in no larger implicant); it cannot be combined into another implicant that has fewer literals.

Essential Prime Implicant (essential): a prime which contains at least one ON-set minterm (i.e. 1 value) which is not contained by any other prime

Sum-of-products (SOP, disjunctive normal form):

a sum of products ("AND-OR" 2-level circuit)

Cover: a set of primes (SOP) containing all the ON-set minterms (1 points) of a function

Complete Sum: a cover containing all possible prime implicants of the function

Source: S. Nowick

2-Level Logic Minimization Problem

The 2-Level Logic Minimization Problem: given a Boolean function *f*

(i) Find a minimum-cost set of prime implicants which "covers" (i.e. contains) all ON-set minterms -- (… and possibly some DC-set minterms)

Or, equivalently:

(ii) Find a minimum-cost cover F of function *f*

Cost is defined as:

e.g., the number of gates + the total number of inputs to all gates in the circuits. What are the costs of the following functions?

$$f = x_1\bar{x}_2 + x_3\bar{x}_4$$

$$g = \left(\overline{x_1\bar{x}_2 + x_3}\right)(\bar{x}_4 + x_5)$$

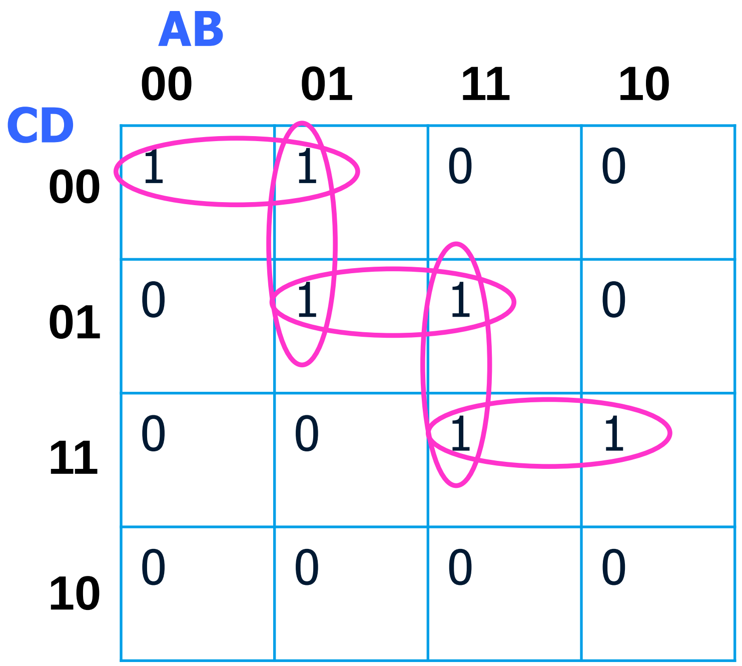
2-Level Logic Minimization: Example

AB

CD	00	01	11	10
00	1	1	0	0
01	0	1	1	0
11	0	0	1	1
10	0	0	0	0

2-Level Logic Minimization: Example

Solution #1: All Primes = 5 Products (AND gates)

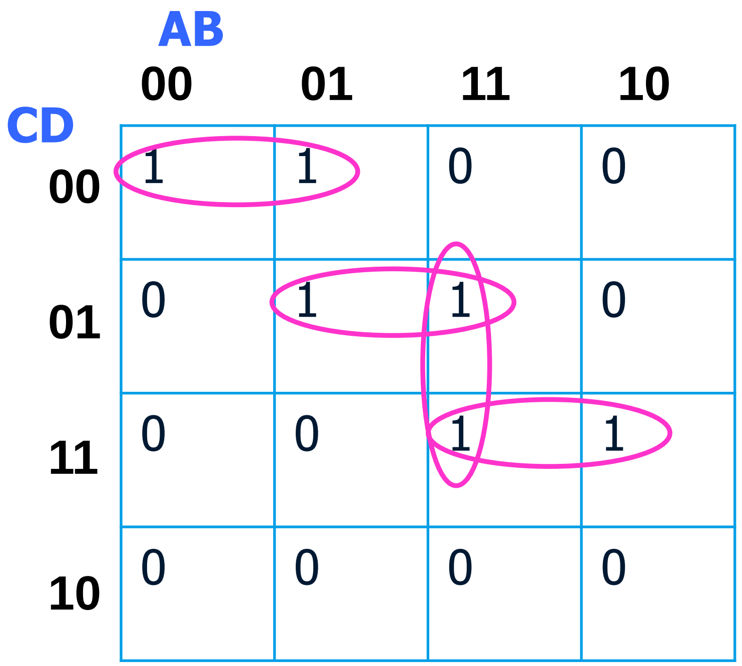


CD \ AB	00	01	11	10
00	1	1	0	0
01	0	1	1	0
11	0	0	1	1
10	0	0	0	0

"Complete Sum" = cover containing all prime implicants

2-Level Logic Minimization: Example

Solution #2: Subset of Primes = 4 Products (AND gates)

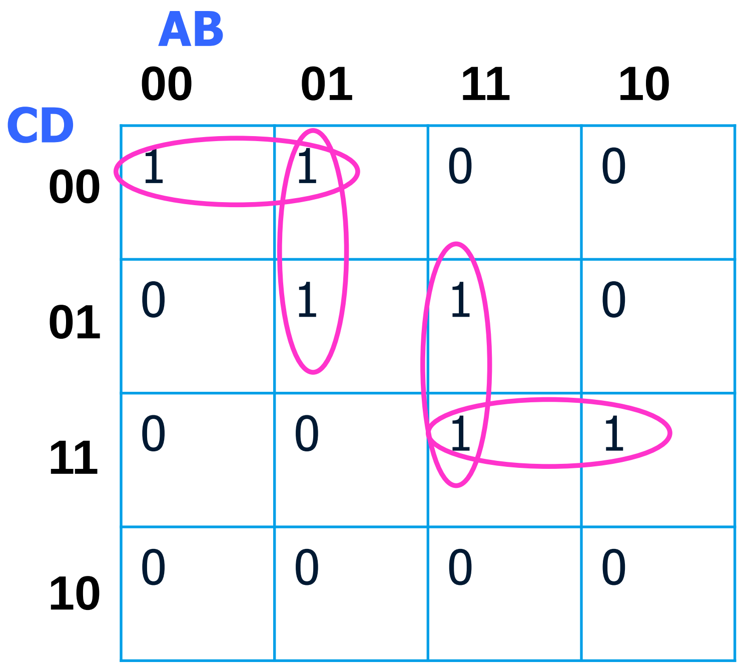


Locally sub-optimal solution

"Redundant Cover" = can remove a product and still have legal cover

2-Level Logic Minimization: Example

Solution #3: Subset of Primes = 4 Products (AND gates)



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

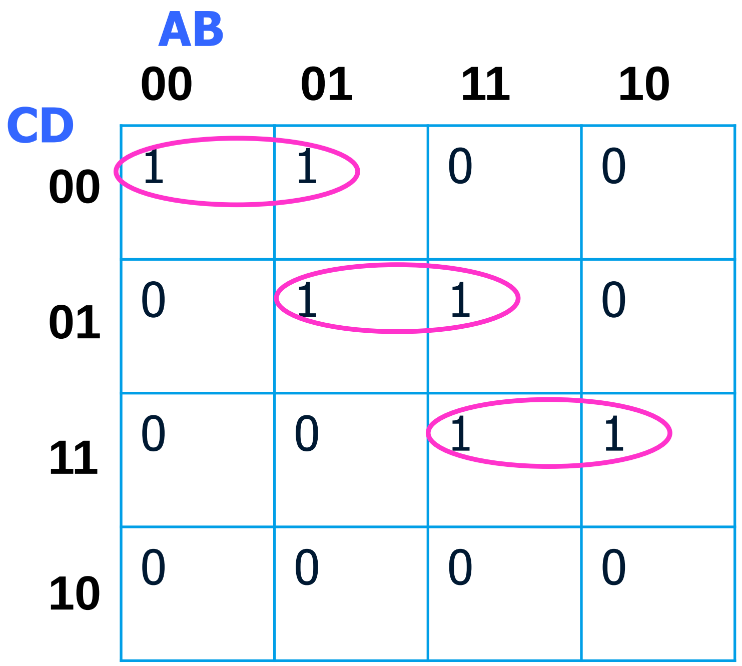
Locally optimal solution

"Irredundant Cover" (but still globally sub-optimal!)
= cannot remove any product and still have legal cover

Source: S. Nowick

2-Level Logic Minimization: Example

Solution #4: Subset of Primes = 3 Products (AND gates)



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

Globally optimal solution

OPTIMAL SOLUTION (also irredundant)

Quine-McCluskey Method

- A systematic method for logic minimization
- Steps
  - Generate all prime implicants for the given function *f*
  - Find the set of essential prime implicants
  - If the set of essential prime implicants covers all valuations for which *f=1*, then this set is the desired cover of *f*. Otherwise, determine the nonessential prime implicants that should be added to form a complete minimum-cost cover
- The last step follows heuristic rules and cost comparisons
- https://en.wikipedia.org/wiki/Quine%E2%80%93McCluskey_algorithm

Quine-McCluskey Method: Examples

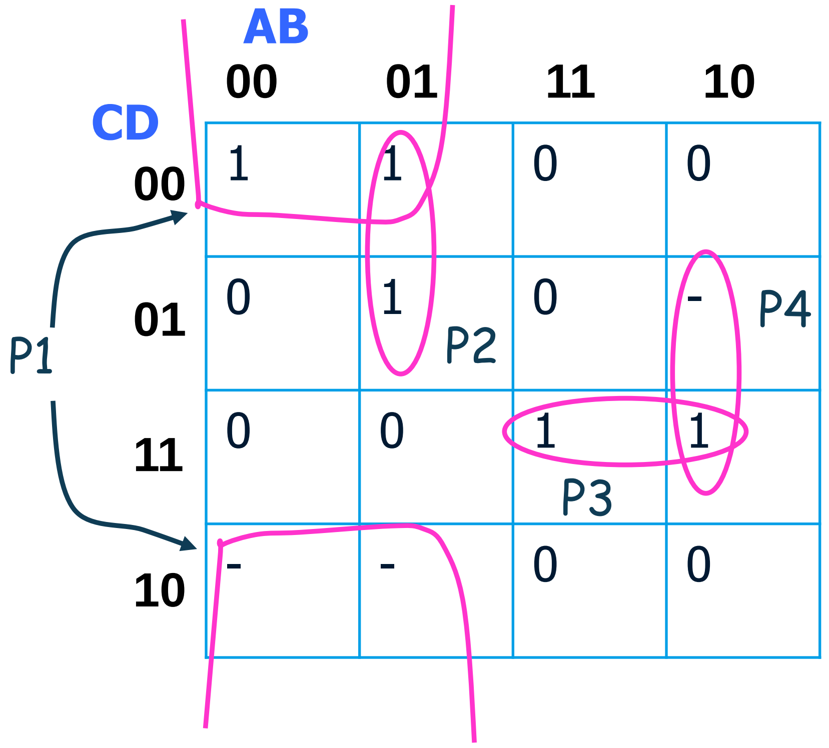
Example #1: f(A,B,C,D) = m(0,4,5,11,15) + d(2,6,9)

[m = ON-set minterms, d = DC-set minterms]

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

Quine-McCluskey Method: Examples

Example #1 (cont.)

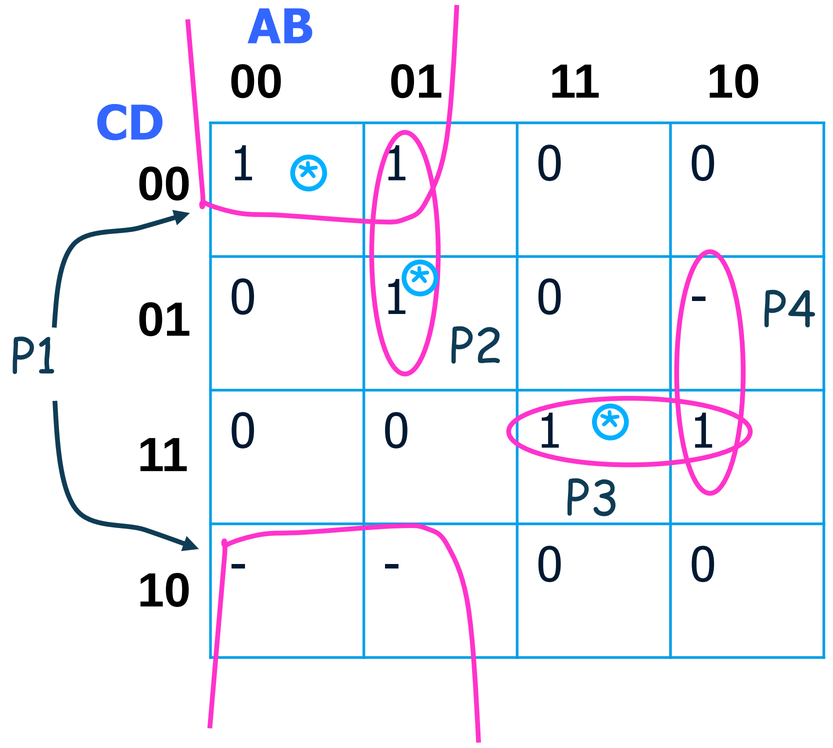


CD \ AB	00	01	11	10
00	1	1	0	0
01	0	1	0	-
11	0	0	1	1
10	-	-	0	0

Generate all prime implicants

Quine-McCluskey Method: Examples

Example #1 (cont.)



⊛ = distinguished minterm

Prime Implicant Table

prime implicants

ON-set minterms

	P1	P2	P3	P4
⊛ 0	X			
4	X	X		
⊛ 5		X		
11			X	X
⊛ 15			X	

○ = essential prime

Approach: remove & save essentials {p1, p2, p3}, and delete intersecting rows … *empty table: nothing left to cover*.

Source: S. Nowick

Quine-McCluskey Method: Examples

Example #2: f(A,B,C) = m(0,1,2,6) + d(5)

[m = ON-set minterms, d = DC-set minterms]

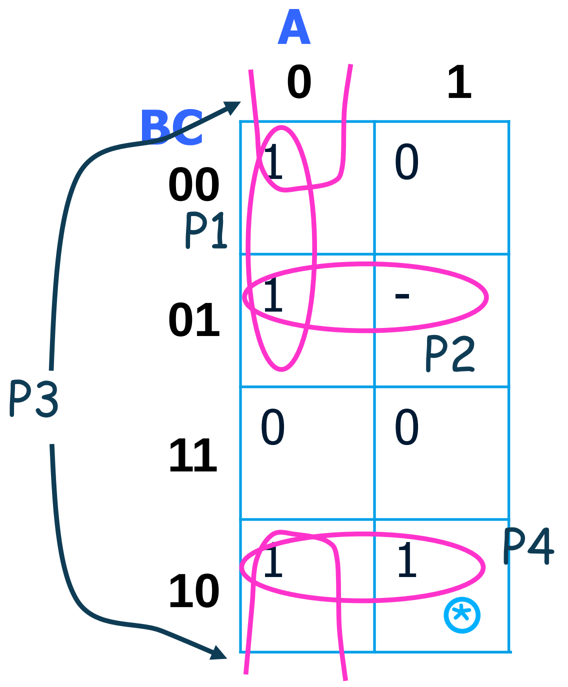
BC \ A	0	1
00	1	0
01	1	-
11	0	0
10	1	1

More complex example: illustrates
"table reduction step" using column dominance

Quine-McCluskey Method: Examples

<u>Example #2</u>: f(A,B,C) = m(0,1,2,6) + d(5)

[m = ON-set minterms, d = DC-set minterms]



⊛ = distinguished minterm

Prime Implicant Table

prime implicants

ON-set minterms	P1	P2	P3	P4
0	X		X	
1	X			X
2		X	X	
⊛ 6		X		

◯ = essential prime

<u>Initial PI Table</u>

Source: S. Nowick

Quine-McCluskey Method: Examples

Example #2: f(A,B,C) = m(0,1,2,6) + d(5)

[m = ON-set minterms, d = DC-set minterms]

		prime implicants			
		P1	◯P2	P3	P4
ON-set minterms	0	X		X	
	1	X			X
	2		X	X	
	⊛ 6		X		

◯ = essential prime

Initial PI Table

		prime implicants		
		P1	P3	P4
ON-set minterms	0	X	X	
	1	X		X

Reduced PI Table (a)

Approach: remove & save essential p2, and delete intersecting rows.

Quine-McCluskey Method: Examples

Example #2: $f(A,B,C) = m(0,1,2,6) + d(5)$

[m = ON-set minterms, d = DC-set minterms]

prime implicants

ON-set minterms	P1	P3	P4
0	X	X	
1	X		X

Reduced PI Table (a)

prime implicants

	P1
0	X
1	X

Reduced PI Table (b)

"Column Dominance":
- column p1 'column-dominates' column p3
- column p1 'column-dominates' column p4

…*delete dominated columns {p3,p4}*

Quine-McCluskey Method: Examples

Example #2: f(A,B,C) = m(0,1,2,6) + d(5)

[m = ON-set minterms, d = DC-set minterms]

prime implicants

	P1
0	X
1	X

◎ = secondary essential prime

"Secondary Essential Primes":
- column p1 *has now become 'essential'*

Approach: remove & save secondary essential p1, and delete intersecting rows.
… *empty table: nothing left to cover*.

Reduced PI Table (b)

Final solution: {p1,p2}