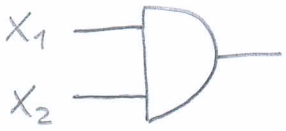
# Logic circuit elements (gates) and logic circuits

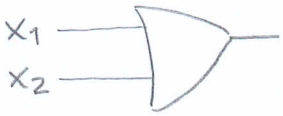
-Some of the most commonly used logic circuit elements are given as follows:

## AND gate




$$z = x_1 \cdot x_2$$

## OR gate

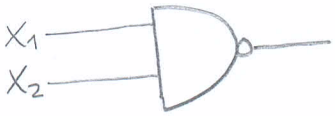



$$z = x_1 + x_2$$

## NOT gate (COMPLEMENT, INVERTER)

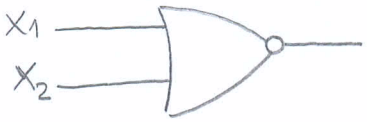

$$z = x'$$

## NAND gate (NOT AND)



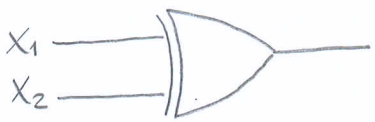

$$z = (x_1 \cdot x_2)'$$

## NOR gate (NOT OR)



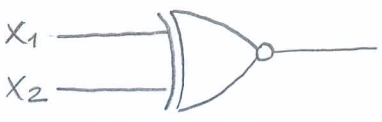

$$z = (x_1 + x_2)'$$

## XOR gate (Exclusive OR)




$$z = x_1 \oplus x_2$$

## XNOR gate (Exclusive NOR)




$$z = (x_1 \oplus x_2)'$$

## Buffer gate


**Definition 1.** The number of inputs of a gate is referred to as "fan-in"; and the number of standard gates that a gate can drive is referred to as "fan-out".

## Positive and negative logic

- The inputs that take 0 and 1, in general, correspond to the low and high levels of a physical quantity
  - such as voltage, current, pressure, etc.

**Definition 2.** If the high level and low level are represented by 1 and 0, respectively then it is said to be positive logic, otherwise it is said to be negative logic

- Consider a 2-input, 1-output gate with input-output relationship as

| $X_1$ | $X_2$ | Z |
|---|---|---|
| L | L | L |
| L | H | L |
| H | L | L |
| H | H | H |

- if positive logic is used, i.e. L ≡ 0, H ≡ 1, then

| $X_1$ | $X_2$ | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

⇒ an AND gate

-if negative logic is used, i.e. $L \equiv 1$, $H \equiv 0$ then

| $x_1$ | $x_2$ | $z$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

$\Rightarrow$ an OR gate

Note that ;

-throughout a logic circuit, either positive logic or negative logic is used

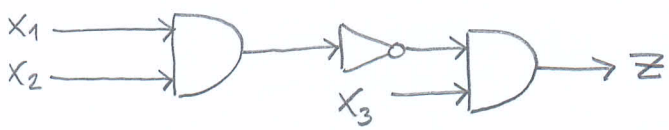
**Definition 3.** A circuit which consists of finite number of gates and satisfies the following two conditions

(i) Only a single output is connected to a certain input

(ii) The circuit is connected.

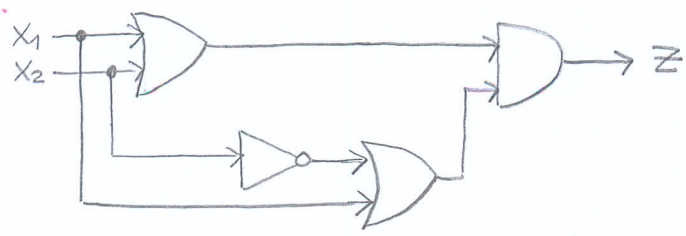
is said to be logic circuit.

**Example.**




**Definition 4.** If the longest path between input and output in a feedforward logic circuit contain n gates, then the circuit is said to be an n-level circuit.

**Example.**




A 2-level logic circuit

# Minimal complete set

- A logic circuit that consists of a single gate
  - ↳ implements a Boolean function
- The objective is to implement a Boolean function through a logic circuit
  - ↳ by employing gates
- Let us consider a set A which is composed of several types of gates

e.g. $A = \{ \text{AND}, \text{COMPLEMENT} \}$

**Definition 5.** If every Boolean function can be implemented by using the gate elements from set A, then set A is said to form a <u>complete set.</u>

Moreover;

- If every Boolean function can NOT be implemented once any gate element from set A is discarded, then set A is said to be a <u>minimal complete set.</u>

Example.

- the set $A = \{ \text{AND}, \text{OR}, \text{COMPLEMENT} \}$ forms a complete set

However;

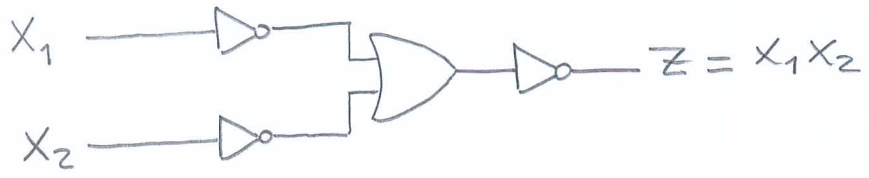
- set A is NOT a minimal complete set
  - ↳ justified by the following theorem

Theorem 9. The set

$$A_1 = \{\text{OR}, \text{COMPLEMENT}\}$$

constitutes a minimal complete set.

Proof. It follows that




AND gate

- any Boolean function can be implemented using the elements of set $A_1$

However;

- any Boolean function can NOT be implemented with $A_2 = \{\text{OR}\}$, or $A_3 = \{\text{COMPLEMENT}\}$

Hence;

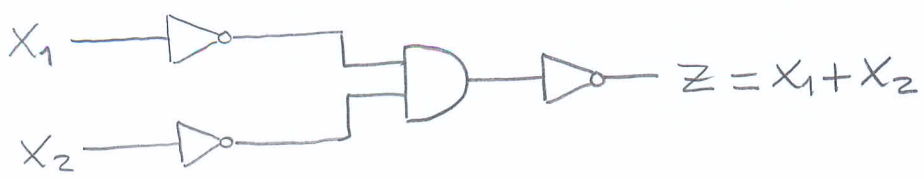
- the elements of set $A_1$ form a minimal complete set

Theorem 10. The set

$$A_2 = \{\text{AND}, \text{COMPLEMENT}\}$$

constitutes a minimal complete set.

Proof. In a similar manner, we can obtain




OR gate

- and any Boolean function can NOT be implemented

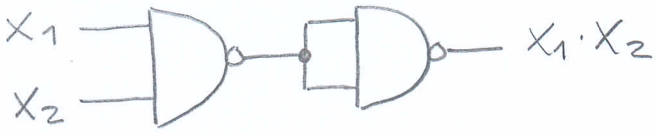
with $A_3 = \{AND\}$

-Thus, this completes the proof.

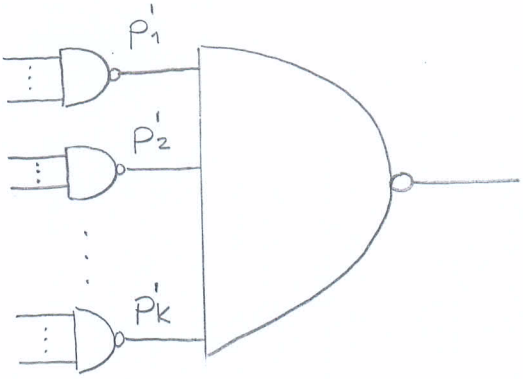
**Theorem 11.** The set $A = \{NAND\}$ constitutes a minimal complete set. Any Boolean function in the form of sum of products can be implemented with a 2-level NAND logic circuit.

**Proof.** Note that we have


COMPLEMENT




AND gate




$$z = [p_1' \cdot p_2' \cdots p_k']'$$
$$= p_1 + p_2 + \cdots + p_k = f$$

-hence A forms a minimal complete set and the logic circuit implementation for any Boolean function f

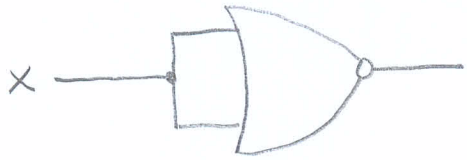
↳ emerge to be a 2-level NAND circuit as f can be represented by a sum of products expression

**Theorem 12.** The set $A = \{NOR\}$ constitutes a minimal complete set. Any Boolean function in the form of product of sums

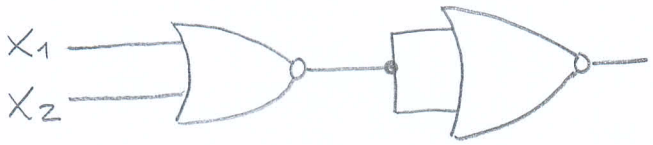
$$f = S_1 \cdot S_2 \cdots S_k$$

can be implemented with a 2-level NOR logic circuit.

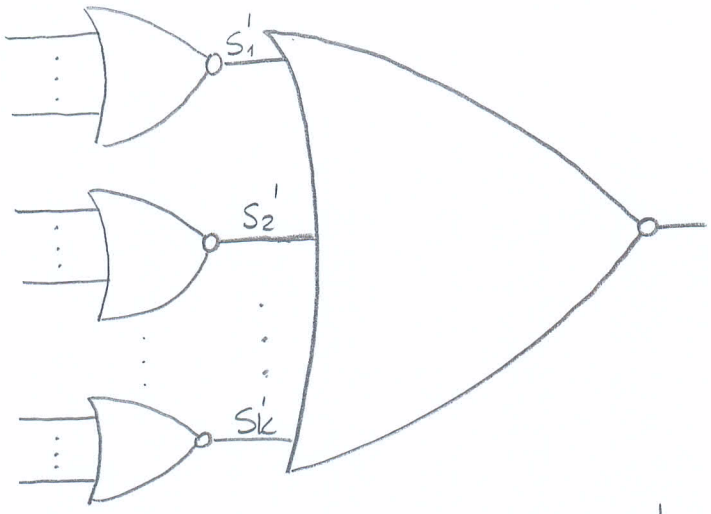
**proof.** Note that we have




COMPLEMENT




OR gate




$$z = [S_1' + S_2' + \cdots + S_k']'$$
$$= S_1 \cdot S_2 \cdots S_k = f$$

- hence A forms a minimal complete set and the logic circuit implementation for any Boolean function, f

  ↳ emerge to be a 2-level NOR circuit as f can be given by a product of sums expression

# Optimization of Boolean functions

- The combinational logic circuit design is, in general, described as

  ↳ an implementation of m Boolean functions of n variables, i.e. $z_i = f_i(x_1, \ldots, x_n), i = 1, 2, \ldots, m$ with a logic circuit

- There exist many logic circuits that can implement these Boolean functions

However;

- it is quite difficult to find out the optimal circuit

## what is an optimal logic circuit ?

- we consider the 2-level AND-OR or 2-level OR-AND combinational logic circuit, then
- an optimal circuit is a circuit that possesses a total of minimum logic gates and inputs

## Sum of products form and prime implicants

- In a Boolean algebra with $S = \{0, 1\}$, the 1st canonical expansion given in Theorem 5 is a

  ↳ a sum of products form

- for example, let $n = 3$ and let the sum of minterms expansion for a Boolean function be

$$f(x_1, x_2, x_3) = m_1 + m_5 + m_7 = x_1' x_2' x_3 + x_1 x_2' x_3 + x_1 x_2 x_3$$

- note that we have a simpler sum of products form

$$f(x_1, x_2, x_3) = (x_1' + x_1)x_2'x_3 + x_1x_2x_3$$
$$= (x_2' + x_1x_2)x_3$$
$$= (x_2' + x_1)(x_2' + x_2)x_3$$
$$= x_2'x_3 + x_1x_3$$

which is more convenient for a 2-level AND-OR or 2-level OR-AND implementation

## prime implicants

- let $f: S^n \to S$ be a Boolean function and let $g_i: S^n \to S$ be a Boolean function with a single product term of $k$ variables, $g_i = x_{i1}^* \cdots x_{ik}^*$, such that $k \leq n$, if

(i) $\exists\ x_{i1}, \ldots, x_{ik} \in S$ such that $g_i = 1 \Rightarrow f = 1$

(ii) the condition (i) collapses when any of $x_{ij}$, $j = 1, \ldots, k$ is removed from $g_i$

then $g_i$ is referred to be a prime implicant (PI) of $f$.

Namely,

$$PI_i \equiv x_{i1}^* x_{i2}^* \cdots x_{ik}^*, \quad k \leq n$$

- we can find all of the PI's of a Boolean fn.
  - ↳ which can be expressed as a sum of these $PI_i$'s
- we can also collect a set of minimum number of PI's
  - ↳ to set another expression for the Boolean fn.

such that

(4) the number of inputs in the AND-OR circuit is also MINIMUM

Therefore ;

- this reduced Boolean expression is so called as the minimized sum of products form of the Boolean function (SOP)

(5) and the corresponding circuit is called as an optimal AND-OR logic circuit

Example. Let us consider

$$f(x_1, x_2, x_3) = m_0 + m_3 + m_4 + m_7$$

- which can be given as the sum of its PI's

$$f(x_1, x_2, x_3) = x_2' x_3' + x_1 x_3 + x_2 x_3 + x_1 x_2'$$

- using Concensus theorem, a minimized expression can be given as

$$f(x_1, x_2, x_3) = x_2 x_3 + x_2' x_1 + x_1 x_3 + x_2' x_3'$$

$$= x_2 x_3 + x_2' x_1 + x_2' x_3'$$

OR alternatively,

$$f(x_1, x_2, x_3) = x_2' x_3' + x_1 x_3 + x_2' x_1 + x_2 x_3$$

$$= x_2' x_3' + x_1 x_3 + x_2 x_3$$

# Minimized product of sums form

- Let $f: S^n \to S$ be a Boolean function and let $g_i: S^n \to S$ be a Boolean function expressed as a single sum term with $k$ variables, $x_{i1}^* + \dots + x_{ik}^*$ such that $k \leq n$, if

  (i) $\exists\ x_{i1}, \dots, x_{ik} \in S$ such that $g_i = 0 \Rightarrow f = 0$

  (ii) the condition (i) collapses when any of $x_{ij}$ $j = 1, \dots, k$ is removed from $g$

then $g_i$ is referred to a prime implicant (PI) of $f$

- collecting all such PI's and establishing the product of these PI's

  ↳ give an expression for $f$

- if we can find out a minimum number of such PI's, then

  ↳ we can obtain another expression for $f$ such that the corresponding OR-AND logic circuit has minimum number of inputs

- this expression is so called as a minimized product of sums form (POS), and

  ↳ the corresponding circuit is said to be the optimal OR-AND logic circuit

# Minimization with Tabulation Method

- Reduction with tabulation method can be done with SOP and POS forms

- the method is composed of two basic steps:

1. Finding out the prime implicants
2. Reducing these prime implicants

## Finding the prime implicants

- we use the following property to get PI's

$$A x_j + A x_j' = A \underbrace{(x_j + x_j')}_{1} = A$$

where A refers to a term of (n-1) variables

- if we repeatedly apply the above rule, we get the PI's of the Boolean fn. f such that

$$f = \sum_{i=1}^{k} PI_i = PI_1 + PI_2 + \dots + PI_k$$

- note that f is expressed in SOP form in such a manner that

↳ we can NO LONGER apply the above simplification rule

- now we can choose a minimum number of PI's

↳ to get a minimized expression for f

# Properties of a reducable pair

1. The binary numbers corresponding to the terms of a reducable pair differ by 1 with respect to the number of 1's.

e.g. Consider $x_1 x_2 x_3' x_4'$ and $x_1 x_2 x_3 x_4'$

$\downarrow$ $\downarrow$

1100 1110

the number of 1's differ by

2. Let the reducable terms be $m_i$ and $m_j$ such that $m_i > m_j$, then

$$m_i - m_j = 2^k \quad , \quad k = 0, \cdots, n-1$$

Note that;

- these conditions are necessary but NOT sufficient

e.g. Indeed, for $n=5$, consider

$m_{17} = x_1 x_2' x_3' x_4' x_5$ 10001 2 1's

$m_{13} = x_1' x_2 x_3 x_4' x_5$ 01101 3 1's

differing by 1

$$\begin{array}{r} 10001 \\ -\ 01101 \\ \hline 00100 \ (=2^2) \end{array}$$

- but $m_{17}$, $m_{13}$ can NOT be reduced.

# A procedure for finding PI's

Step 1. List the binary numbers corresponding to the product terms of the function f given in SOP form by tabulating in accordance with the total number of 1's involved in each binary number with an increasing order.

Step 2. Compare every term of a group with each term of the subsequent group and reduce if possible. Place a check (✓) sign nearby each term which is subject to reduction. Any term which has NO check (✓) sign nearby is said to be a PRIME IMPLICANT.

- Repeat Step 1 and Step 2 to set List 2, 3, etc. starting with the reducable pairs that can be obtained from the former list.
- Stop until a new list can NOT be obtained anymore.

↳ implying that all the PI's are obtained

Example.

- Consider the following Boolean function

$f(x_1, x_2, x_3, x_4) = \Sigma m(2,4,6,8,9,10,12,13,15)$

## List #1

| $m_i$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | |
|---|---|---|---|---|---|
| 2 | 0 | 0 | 1 | 0 | ✓ |
| 4 | 0 | 1 | 0 | 0 | ✓ |
| 8 | 1 | 0 | 0 | 0 | ✓ |
| 6 | 0 | 1 | 1 | 0 | ✓ |
| 9 | 1 | 0 | 0 | 1 | ✓ |
| 10 | 1 | 0 | 1 | 0 | ✓ |
| 12 | 1 | 1 | 0 | 0 | ✓ |
| 13 | 1 | 1 | 0 | 1 | ✓ |
| 15 | 1 | 1 | 1 | 1 | ✓ |

- reducable pairs:

(2,6), (2,10)
(4,6), (4,12)
(8,9), (8,10), (8,12)
(9,13)
(12,13)
(13,15)

## List #2

| $m_i$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | |
|---|---|---|---|---|---|
| 2,6 | 0 | – | 1 | 0 | PI2 |
| 2,10 | – | 0 | 1 | 0 | PI3 |
| 4,6 | 0 | 1 | – | 0 | PI4 |
| 4,12 | – | 1 | 0 | 0 | PI5 |
| 8,9 | 1 | 0 | 0 | – | ✓ |
| 8,10 | 1 | 0 | – | 0 | PI6 |
| 8,12 | 1 | – | 0 | 0 | ✓ |
| 9,13 | 1 | – | 0 | 1 | ✓ |
| 12,13 | 1 | 1 | 0 | – | ✓ |
| 13,15 | 1 | 1 | – | 1 | PI7 |

- reducable pairs:

((8,9), (12,13))
((8,12), (9,13))

List # 3

| $m_i$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | |
|---|---|---|---|---|---|
| (8,9,12,13) | 1 | – | 0 | – | PI1 |
| (8,12,9,13) | 1 | – | 0 | – | |

Therefore ;

$$f(x_1, x_2, x_3, x_4) = \sum_{i=1}^{7} PI_i$$

$$= x_1 x_3' + x_1' x_3 x_4' + x_2' x_3 x_4' + x_1' x_2 x_4' + x_2 x_3' x_4' + x_1 x_2' x_4' + x_1 x_2 x_4$$

## Reducing Prime Implicants

Step 3. We construct a chart satisfying:

- each row corresponds to a PI
- each column corresponds to a minterm
- the rows are organized such that a PI involving fewer number of variables will be on top of the other
- place X if the PI involves the corresponding minterm in the chart

-let us form the chart for our example as follows

| $PI_i/m_j$ | 2 | 4 | 6 | 8 ✓ | 9 ✓ | 10 | 12 ✓ | 13 ✓ | 15 ✓ |
|---|---|---|---|---|---|---|---|---|---|
| * PI1 | | | | X | X | | X | X | |
| PI2 | X | | X | | | | | | |
| PI3 | X | | | | | X | | | |
| PI4 | | X | X | | | | | | |
| PI5 | | X | | | | | X | | |
| PI6 | | | | X | | X | | | |
| * PI7 | | | | | | | | X | X |

Step 4. We choose a minimum number of PI's from the above chart such that their sum yields an optimal expression for f as follows:

- if a column involves a single X, take it as an essential PI, place a * nearby
- then encircle all essential PI's and place a check (✓) sign nearby the minterms covered by the essential PI's

Note that;

- the essential prime implicants (EPI) will inevitably be included into the minimal expression

Reduced chart

- We construct the reduced chart by removing EPI's

and the minterms covered by EPI's.

| $PI_i / m_j$ | 2 | 4 | 6 | 10 |
|---|---|---|---|---|
| PI2 | X | | X | |
| PI3 | X | | | X |
| PI4 | | X | X | |
| PI5 | | X | | |
| PI6 | | | | X |

"Reduced Chart"

Objective

- our goal is to collect a minimum number of PI's such that

(1) all minterms are covered entirely

- we describe how to achieve that via the following definition

Definition 6. Consider the $i^{th}$ and $j^{th}$ rows (columns) of the reduced chart. If there exist an X at the $i^{th}$ row corresponding to the column (row) of the $j^{th}$ row (column), then it is said that the $i^{th}$ row (column) covers the $j^{th}$ row (column).

Thus;

- the rows covered by other rows entirely can be discarded from the chart

(1) to obtain a more reduced chart

- we can eliminate 5th and 6th rows as they are covered by 4th and 3rd rows, respectively

| $PI_i / m_j$ | 2 ✓ | 4 ✓ | 6 ✓ | 10 ✓ |
|---|---|---|---|---|
| PI2 | X | | X | |
| * PI3 | X | | | X |
| * PI4 | | X | X | |

Step 5. We apply the rule specified in Step 4 to find out the PI's that will be included in the minimal expression.

Note that;

- $m_4$, and $m_{10}$ are covered only by PI4, and PI3, respectively

↳ so PI3 and PI4 are essential prime implicants

- and if we remove PI3, PI4 and the corresponding minterms covered by PI3, PI4

↳ yielding a result that all minterms are covered

As a result ;

- we can give the minimal expression as

$$f(x_1, x_2, x_3, x_4) = \sum PI\,(1,3,4,7)$$

$$= x_1 x_3' + x_2' x_3 x_4' + x_1' x_2 x_4'$$

$$+ x_1 x_2 x_4$$

## Cyclic PI chart

- if a PI chart does NOT have any EPI, and
- if there exist no overlapping rows and columns, then the PI chart is

↳ said to be a "cyclic PI chart"

## Branching method

- we use the branching method for handling cyclic PI chart
- we randomly pick any of the PI's (preferrably the one with less overlapping) and treat it as an essential PI

↳ and apply step 4 to obtain a reduced PI chart

## Don't care conditions

- In a combinational circuit, if some of the input combinations are NEVER applied
  - ↳ then the values of the Boolean function for such points can be selected as 0 or 1

example. Consider a Boolean function

$$f(x_1, x_2, x_3) = \sum m(0,1,5) + d(2,6)$$

where $m_2$, and $m_6$ correspond to don't care conditions

## Minimization with don't care conditions

- we include the don't care minterms into the function to be minimized, and
  - ↳ the PI's are specified
- but while organizing the PI charts, we do NOT take the don't care minterms into the columns
- if there exist a PI composed of don't care minterms only, then discard this PI too