Hardware Design I Chap. 2
Basis of logical circuit, logical expression, and logical function

Computing Architecture Lab.
Hajime Shimada
E-mail: shimada@is.naist.jp

Outline

- Combinational logical circuit
  - Logic gate (logic element)
  - Definition of combinational logical circuit
  - How to create output signal?
- Logical function
  - Definition of logical function
  - Relationship between logical circuit
- Logical expression
  - Definition of logical expression
  - Minterm and maxterm
  - Axiomatic systems
  - Amount of logical expression

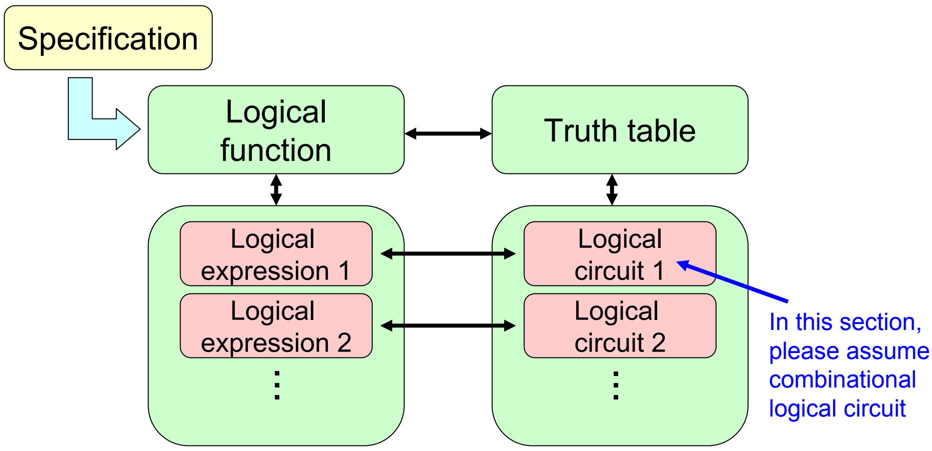
Review: outlined flow of LSI design

- Define specification
- Definition in hardware description language
  - Architectural design

    ⇩ Logic synthesis
- Circuit with basic logic gates
  - Logical design ← This chapter treats this area
    - •Logical function
    - •Logical expression

    ⇩ Place and route
- Mask pattern
  - Physical design
- Manufacturing

Relationship between technical terms



- If we minimize logical expression, we can implement minimized logical circuit

Detailed talk of logical design

- Specification of sequential machine -> Chap. 6

  ⇓

- Specification of logical function -> later Chap. 2

  ⇓

- Logical expression -> later Chap. 2

  ⇓
  - •Simplify of two level logic -> Chap. 3 and 7
  - •Simplify of multi level logic -> Chap. 7 and 8

- Simplified logical expression

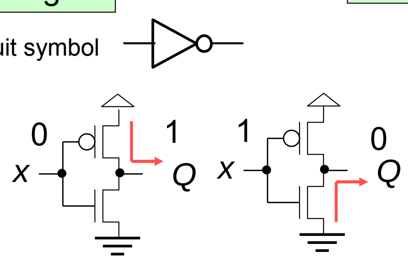
  = Basic logic gates

Logic gate (logic element)

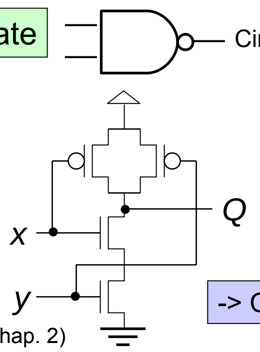
- The electric circuit witch outputs result of logical operation
  - e.g. NOT, NAND
  - Both inputs and outputs can only take 0 or 1

NOT gate

Circuit symbol



NAND gate



Circuit symbol

-> Chap. 1

NOT, AND, and OR on Boolean algebra

- Logical circuit operates on Boolean algebra
- Here's basic logic from Boolean algebra

NOT

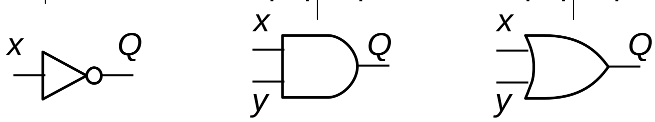
x	Q
0	1
1	0

AND

x	y	Q
0	0	0
0	1	0
1	0	0
1	1	1

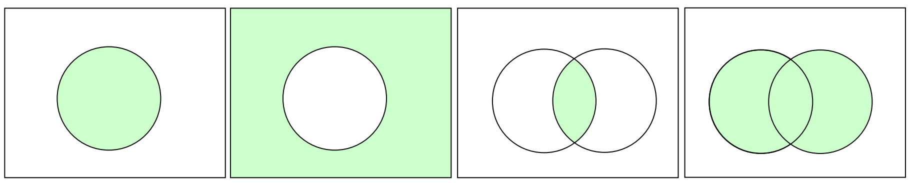
OR

x	y	Q
0	0	0
0	1	1
1	0	1
1	1	1



NOT, AND, and OR on Venn diagram

- In some case, imaging Venn diagram helps understanding
  - NOT: left area
  - AND: shared area
  - OR: sum of area



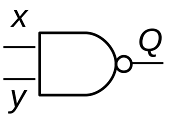
A $\quad \overline{A} \quad$ A·B $\quad$ A+B

NAND and NOR on Boolean algebra

- Physical implementation is easy -> Chap. 1
  - Usually, AND and OR are implemented by combining NOT and NAND/NOR

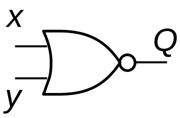
NAND

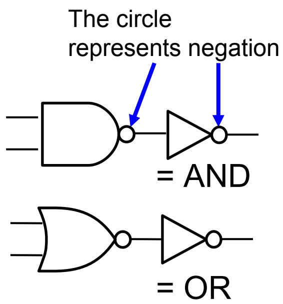
x	y	Q
0	0	1
0	1	1
1	0	1
1	1	0



NOR

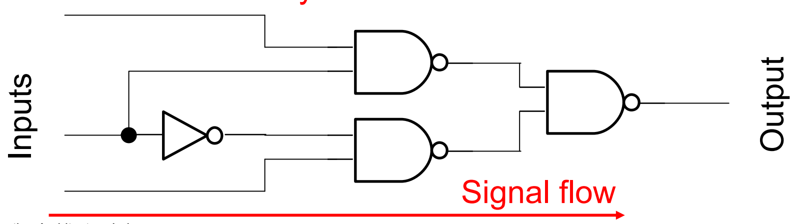
x	y	Q
0	0	1
0	1	0
1	0	0
1	1	0





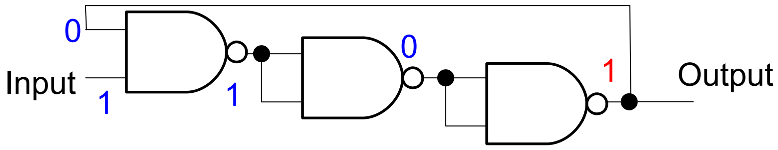
Combinational logical circuit

- The signal flow must be contra flow
- The output of the gate will be defined from input side
- The output is defined with current input
  - No loop in it
  - It is also called "acyclic circuit"

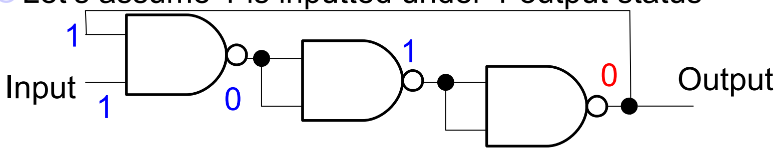


Let's assume looped logic circuit (1/2)

- It sometimes gives unstable output
  - Let's assume 1 is inputted under 0 output status



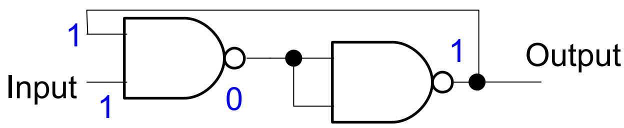
  - Let's assume 1 is inputted under 1 output status



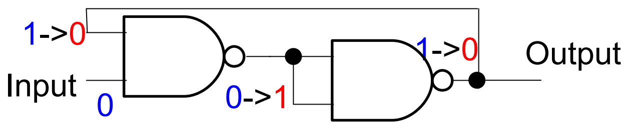
The output switches 0/1 forever!!! ->oscillator

Let's assume looped logic circuit (2/2)

- We rarely achieve stable circuit with looped combinational circuit
  - Let's assume 1 is inputted under 1 output status
    - It continues to output 1



  - Once input falls to 0, the output changes to 0 forever

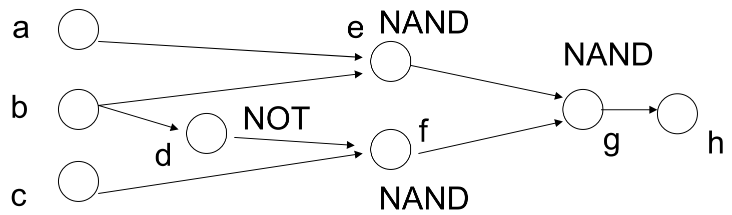


Usually, they are rare and utilization is limited...

How to crate loop?
-> Sequential circuit
(Chap. 6)

Definition of combinational logic with directed graph

- Set of vertices: V={a, b, c, d, e, f, g, h}
- Set of edges: E⊆(V×V)
  E={(a,e), (b,e), (b,d), (c,f), (d,f), (e,g), (f,g), (g,h)}
- Label of vertex: NOT, NAND, and so on



If you felt "what is directed graph?" ...

- Please relearn "graph theory"
  - The sets of vertices and edges
  - e.g. network connection graph, schematic diagram, ...
  - Specific graph: tree, directed graph, weighted graph, ...
- It is widely used in informatics world
  - Syntax tree (compiler)
  - Markov chain (voice recognition)
  - Perceptron (neural network)

About technical terms of set theory

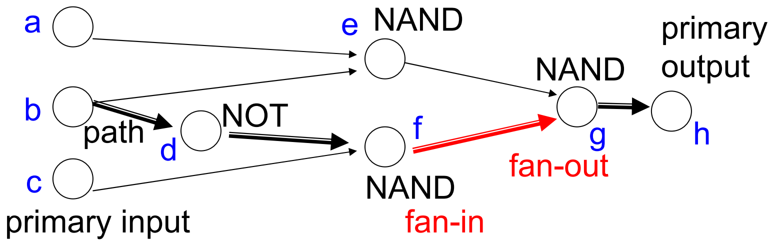
- Set
  - Gathered set of elements
  - e.g. {0, 1}, {a, b, ..., z}, ...
- Cartesian product
  - A set of ordered pairs of elements
  - Notation: A × B (A,B: set)
  - e.g. {0, 1} × {a, b} = {(0,a), (0,b), (1,a), (1,b)}
  - Other notation: V^2, $\{0, 1\}^2$

The syntax of combinational logic from graph theory

- Directed Acyclic Graph (DAG): (V, E)
  - V: set of vertices
  - E: set of edges, subset of (V × V)
    - (V × V) denotes set of Cartesian product
- Allocate logic gate (e.g. NAND) label to vertices
  - Allocate 1 label to 1 vertex

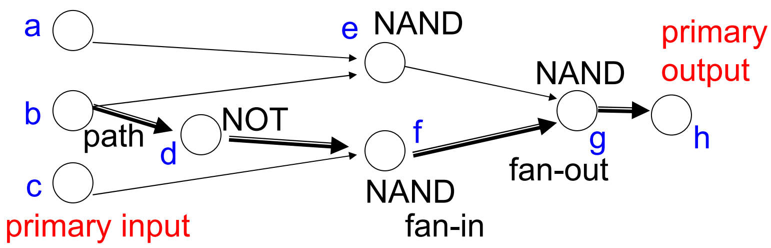
Terms of combinational logic (1/3)

- Fan-in: a input side of edge
  - e.g. v_1 is the fan-in of edge (v_1, v_2)
  - Viewpoint from the v_2 side
- Fan-out: a output side of edge (v_1, v_2)
  - e.g. v_2 is the fan-out of edge (v_1, v_2)



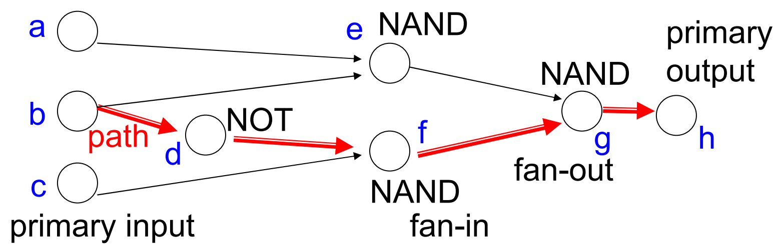
Terms of combinational logic (2/3)

- Primary input: a vertex which does not have fan-in
- Primary output: a vertex which does not have fan-out



Terms of combinational logic (3/3)

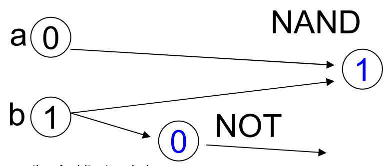
- Path: a set of edges from primary input to primary output
  - e.g. $(v_1, v_2)\ (v_2, v_3)\ \ldots\ (v_{n-1}, v_n)$
  - v_1 is transitive fan-in
  - v_n is transitive fan-out



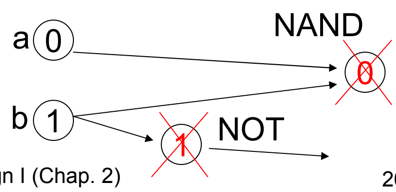
Value allocation to logic circuit

- Value allocation
  - Allocate 0/1 value to (output of) each vertex
  - Adequate allocation: satisfies the truth of gate
- The allocation will be defined if all of primary input has defined
- It is also called logic simulation

Adequate allocation



Not adequate allocation

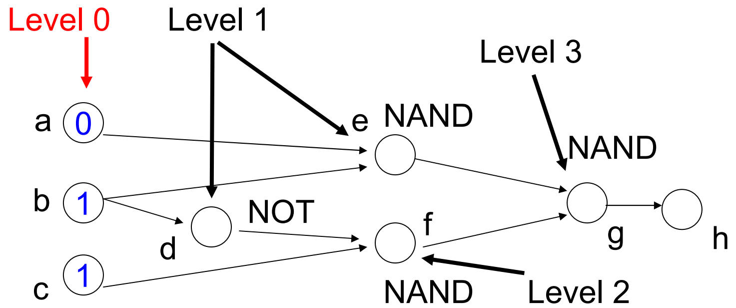


The algorithm of value allocation

1. Define the value of primary inputs
   - Primary inputs are called level 0 vertices
2. Define the value of level 1 vertices
   - Level 1 vertices: all inputs of them are primary input
   - All inputs value are already defined in 1.
3. Define the value of level 2 vertices
   - Level 2 vertices: all inputs of them are less than level 1 (level 0 or 1)
4. Define level n vertices until the all of the vertices have defined
   - Level n vertices: all inputs of them are less than level n-1

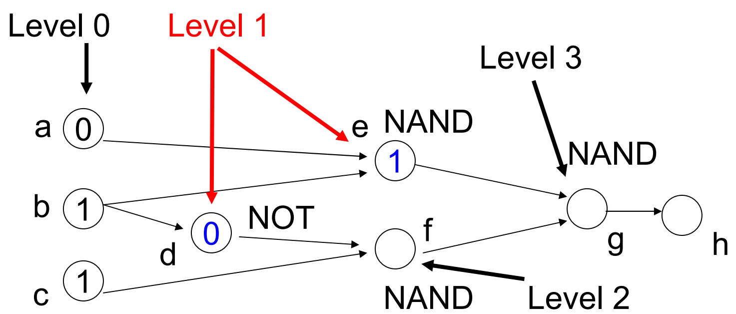
Example of value allocation (1/4)

- Allocate value to primary inputs (level 0 vertices)
  - We can allocate them without constraint
  - Usually, they are given



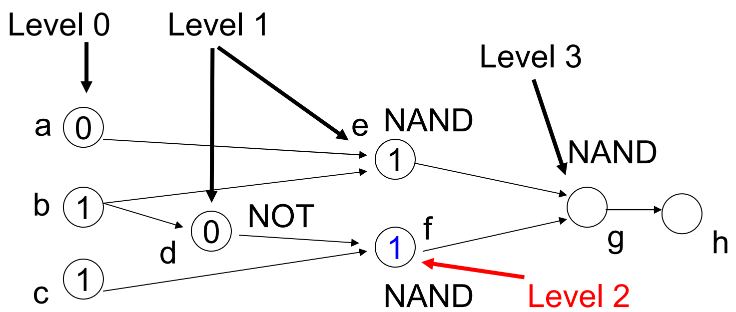
Example of value allocation (2/4)

- Allocate values to level 1 vertices
  - Which are only connected to primary inputs



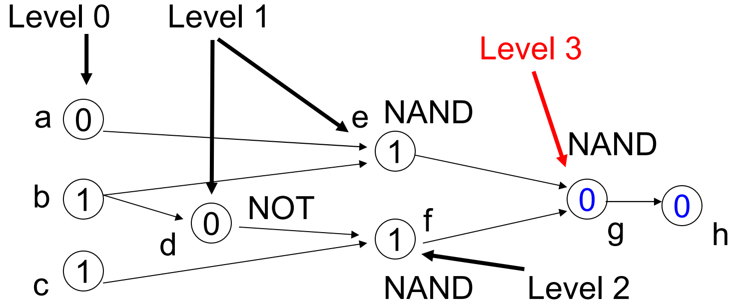
Example of value allocation (3/4)

- Allocate values to level 2 vertices
  - Which are only connected to less than level 1 vertices
  - See the vertices which values have already allocated



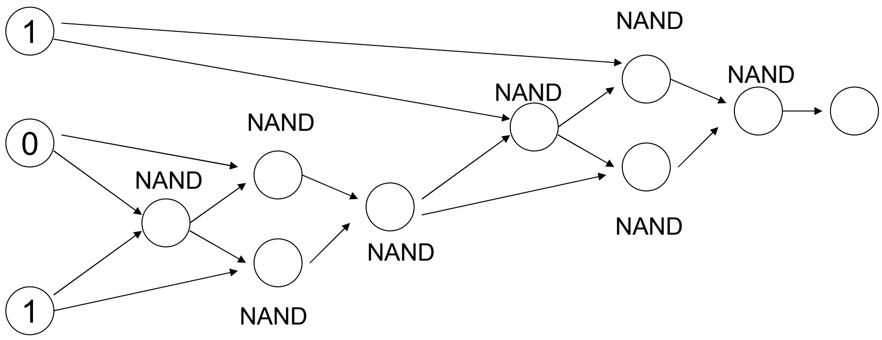
Example of value allocation (4/4)

- Allocate value to level 3 vertices
  - Which are only connected to less than level 2 vertices
  - The allocation of primary outputs are the same to the prior vertices

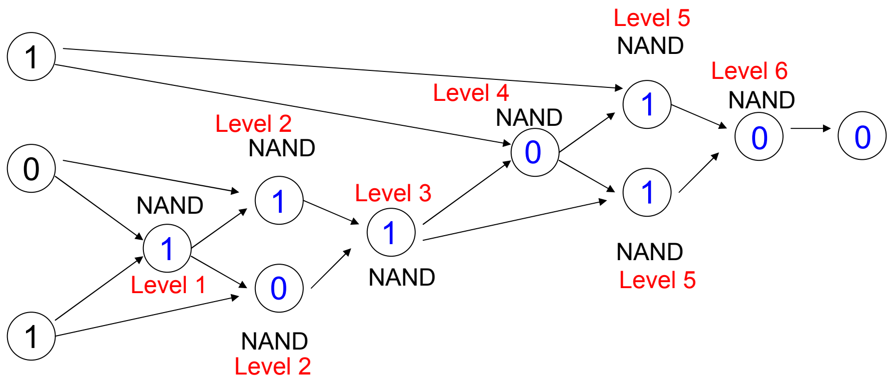


Short exercise

- Allocate values to left vertices
  - If you left time, add level notations to the vertices



The answer of short exercise



Outline

- Combinational logical circuit
  - Logic gate (logic element)
  - Definition of combinational logical circuit
  - How to create output signal?
- Logical function
  - Definition of logical function
  - Relationship between logical circuit
- Logical expression
  - Definition of logical expression
  - Minterm and maxterm
  - Axiomatic systems
  - Amount of logical expression

Definition of logical function from mathematical viewpoint

- Representation of the relationship between input value and output value
- The definition of *n*-value logical function:

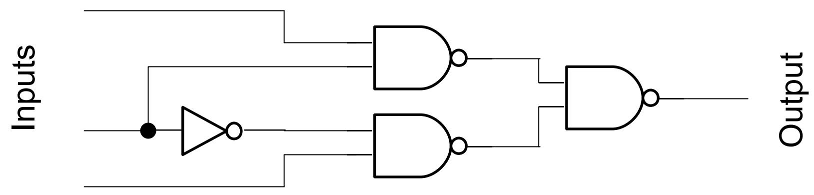
  Projection from $\{0, 1\}^n$ to $\{0, 1\}$
  - Subset $f \subseteq \{0, 1\}^n \times \{0, 1\}$ which does not include both $(X, 0) \in f$ and $(X, 1) \in f$ in arbitrary X
  - We denote it $y = f(X)$ if $(X, y) \in f$
  - $\{0, 1\}^n$ is called domain
  - $\{0, 1\}$ is called codomain

Example of definition of 3-value logical function (notated by logical circuit)

- It outputs 0 if we input (0, 0, 0) into it
- It outputs 1 if we input (0, 0, 1) into it

  ⋮
- It outputs 1 if we input (1, 1, 1) into it

This is logical function!



Examples of definition of representative logical function

- The function of NOT $\subseteq \{0,1\} \times \{0,1\}$
  - $\{(0, 1), (1, 0)\}$
- The function of AND $\subseteq \{0,1\}^2 \times \{0,1\}$
  - $\{((0, 0), 0), ((0, 1), 0), ((1, 0), 0), ((1, 1), 1)\}$
- The function of AND $\subseteq \{0,1\}^2 \times \{0,1\}$
  - $\{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), (\underbrace{(1, 1)}_{\text{Input}}, \underbrace{1}_{\text{Output}})\}$

Hot to denote them in usual?

- Usually, we do not use mathematical definition
- We usually use following notations
  - Logical circuit
  - Truth table
  - Logical expression

Truth table

- One of the representation style of logical function
- Aligning output values for all possible inputs
- The size of *n* values logical function is 2^n

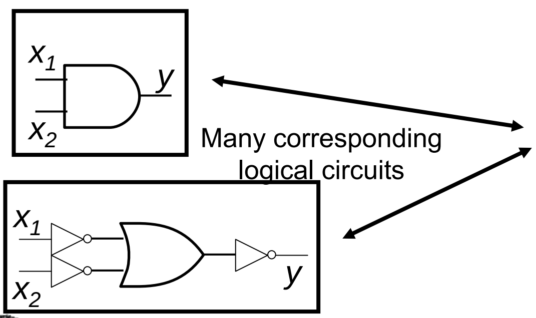
x_1	x_2	$f(x_1,x_2)$	$g(x_1,x_2)$	$h(x_1,x_2)$
0	0	0	0	$h(0, 0)$
0	1	0	1	$h(0, 1)$
1	0	0	1	$h(1, 0)$
1	1	1	0	$h(1, 1)$

If truth tables of two functions are identical, the functions are identical

Logical function ↔ One for one relationship ↔ Truth table

Relationship between logical function and logical circuit

- Logical function represents the relationship of input value and output value in combinational logical circuit



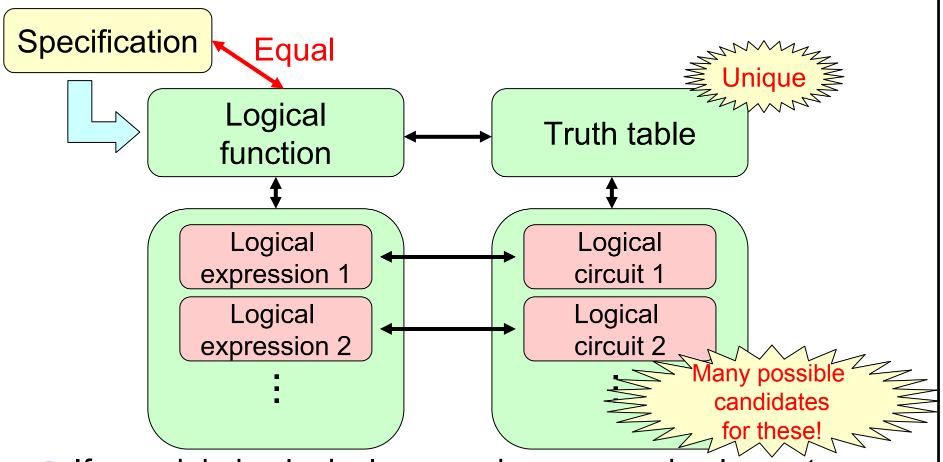
Logical function y

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

Relationship of input/output

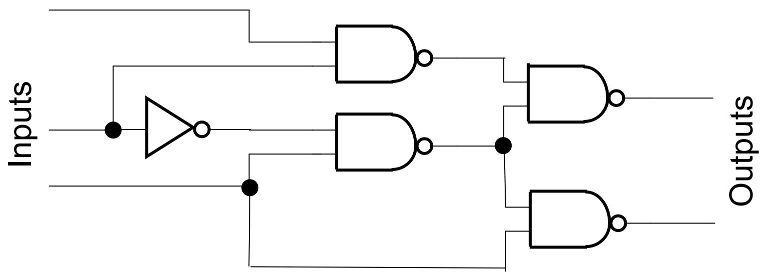
Relationship between technical terms



- If we minimize logical expression, we can implement minimized logical circuit

Multiple output logical function

- In many case, digital system has multiple outputs
- Usually, we decompose it to multiple single output function for simplicity



Truth table of multiple output logical function

- Multiple output function (m outputs):
  Projection from $\{0, 1\}^n$ to $\{0, 1\}^m$
  - List of m projections from $\{0, 1\}^n$ to $\{0, 1\}$

x_1	x_2	$f_0(x_1,x_2)$	$f_1(x_1,x_2)$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

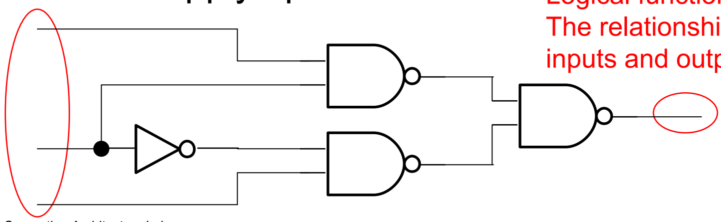
Operation between logical functions

- We can extend operation on logical value to logical function
  - $(f \cdot g)\ (x_1, x_2, ..., x_n) = f(x_1, ..., x_n) \cdot g(x_1,..., x_n)$
  - $(f + g)\ (x_1, x_2, ..., x_n) = f(x_1, ..., x_n) + g(x_1,..., x_n)$
  - $(f')\ (x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n)'$
- Detail is taught in following logical expression section

Summary of logical function

- It is a function from $\{0, 1\}^n$ to $\{0, 1\}$
  - $\{0, 1\}^n \times \{0,1\}$ with some constraint
- It is represented uniquely with truth table
  - List of relationship between all inputs and outputs
  - But it requires 2^n size of memory
- We can apply operation on it

Logical function:
The relationship between inputs and outputs

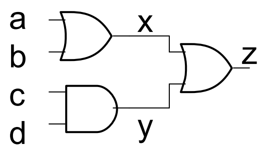


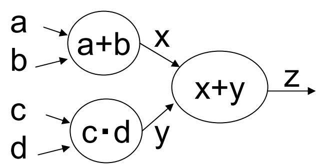
Outline

- Combinational logical circuit
  - Logic gate (logic element)
  - Definition of combinational logical circuit
  - How to create output signal?
- Logical function
  - Definition of logical function
  - Relationship between logical circuit
- Logical expression
  - Definition of logical expression
  - Minterm and maxterm
  - Axiomatic systems
  - Amount of logical expression

Logical expression

- One of the expression of logical function
  - Represent it with arrangement of variable which denotes logical function
  - e.g. $x + y \cdot z + x \cdot y' \cdot z'$
- Efficient than truth table
- But there's no uniqueness
- $x = a+b$; $y = c \cdot d$; $z = x+y$ -> $z = (a+b) + (c \cdot d)$





The definition of logical expression

1. Logical variables are logical expression
   - e.g. $x, y, z, x_1, x_2, a, b, \ldots$
2. If E_1 and E_2 are logical expression, $(E_1 \cdot E_2)$, (E_1+E_2), (E_1') are logical expression
   - e.g. $(x \cdot y)$, $(x+y)$, $(x+(y \cdot z))$, $(x+(y'))$

- Generated in recursively
- We can omit brackets by adding order to operations
  - Order: ', ・, and +

The expression of logical function with logical expression (1/2)

- Pay attention to the logical function which has only one "1" output in truth table
  - Called minterm
  - Minterm can be represented by AND and NOT

Minterm

x y	x'·y'	x'·y	x·y'	x·y
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

The expression of logical function with logical expression (2/2)

- The logical function which has multiple "1" output is represented by OR of minterms
- The arbitrary function can be represented with AND, OR, and NOT of logical variable

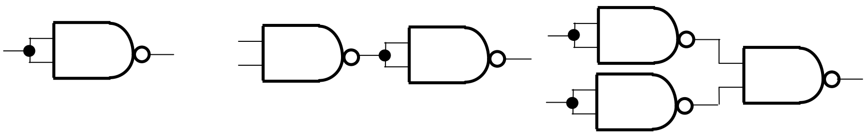
Minterm

x y	x'·y'	x'·y	x·y'	x·y	f(x,y) = x'·y + x·y'
0 0	1	0	0	0	0
0 1	0	1	0	0	1
1 0	0	0	1	0	1
1 1	0	0	0	1	0

Notation only 2-input NAND or NOR

- We can represent NOT, AND, and OR with NAND gates by following wire connection
  - Called “NAND has functional completeness”
- Similar representation can be done with only NOR gates

NOT with NAND   AND with NAND   OR with NAND



See De Morgan’s law in later

Sum of products

- Definition
  - Literal: Logical value or the negation of logical value
    - a: positive literal
    - a': negative literal
  1. Create term with AND of literals
  2. Create logical expression with OR of 1.
- e.g. abc + a'b'c + ac, ac + bc + ad'e
- Other names: AND-OR type, two level logic
- The sum of minterms has special name

  ->Disjunctive Normal Form (DNF)

Disjunctive Normal Form (DNF)

- Sum of minterms without same minterm
  - Arbitrary logical function can be expressed with DNF

a	b		f	g
0	0	a'b'	0	1
0	1	a'b	1	0
1	0	ab'	1	0
1	1	ab	0	1

f = a'b + ab'
g = a'b' + ab

a	b	c		h	s	t
0	0	0	a'b'c'	0	0	1
0	0	1	a'b'c	1	1	1
0	1	0	a'bc'	0	0	0
0	1	1	a'bc	1	1	0
1	0	0	ab'c'	0	0	0
1	0	1	ab'c	1	1	0
1	1	0	abc'	1	0	1
1	1	1	abc	0	1	1

h = a'b'c + a'bc + ab'c + abc'
s = a'b'c + a'bc + ab'c + abc
t = a'b'c' + a'b'c + abc' + abc

Product of sums

- Definition
  1. Create term with OR of literals
  2. Create logical expression with AND of 1.
- e.g. (a+b'+c) (a'+b+c)(d+e')
- There's a counterpart notation of DNF

  ->Conjunctive Normal Form (CNF)
  - Sum of maxterms
  - Maxterm: the logical function which has only one "0" output in truth table

Maxterm

- Pay attention to the logical function which has only one "0" output in truth table
  - Called maxterm
  - Maxterm can be represented by OR and NOT

Maxterm (columns x+y, x+y', x'+y, x'+y')

x y	x+y	x+y'	x'+y	x'+y'	f(x,y) = (x+y)(x'+y')
0 0	0	1	1	1	0
0 1	1	0	1	1	1
1 0	1	1	0	1	1
1 1	1	1	1	0	0

Conjunctive Normal Form (CNF)

- Sum of maxterms without same maxterm
  - Arbitrary logical function can be expressed with CNF

a b		f	g
0 0	a'b'	0	1
0 1	a'b	1	0
1 0	ab'	1	0
1 1	ab	0	1

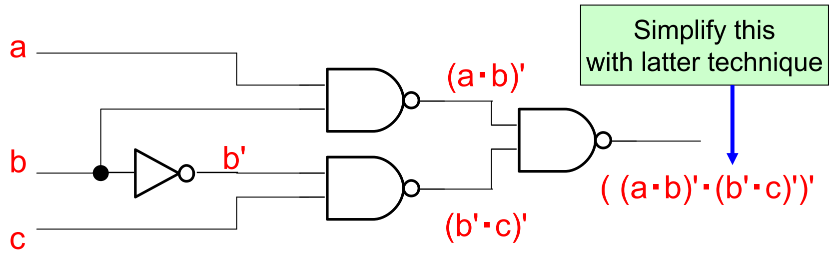
f = (a'+b')(a+b)
g = (a'+b)(a+b')

a b c		h	s	t
0 0 0	a'b'c'	0	0	1
0 0 1	a'b'c	1	1	1
0 1 0	a'bc'	0	0	0
0 1 1	a'bc	1	1	0
1 0 0	ab'c'	0	0	0
1 0 1	ab'c	1	1	0
1 1 0	abc'	1	0	1
1 1 1	abc	0	1	1

h = (a+b+c)(a+b'+c)(a'+b+c)(a'+b'+c')
s = (a+b+c)(a+b'+c)(a'+b+c)(a'+b'+c)
t = (a+b'+c)(a+b'+c')(a'+b+c)(a'+b+c')

Symbol simulation

- A method to obtain logical expression from logical circuit
- Propagate symbol from inputs
  - Operate expression from lower level
    ->Similar to value allocation



Simplify with operation on Boolean algebra

- The logical expression given from symbol simulation has complexity
  - e.g. ((a･b)'･(b'･c)')'
- How to simplify them?

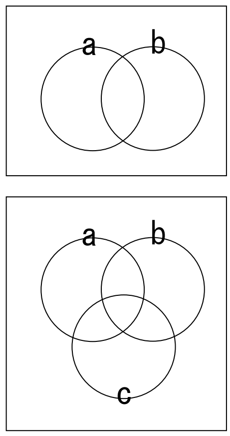
⇩

- Simplify with operation on Boolean algebra
  - General operation rule
  - De Morgan’s law
  - Shannon's expansion

Axiomatic systems related simplification on Boolean algebra

- General operation rules
  - Idempotent: $a+a = a$
  - Commutativity: $a+b = b+a$
  - Associatively: $(a+b)+c = a+(b+c)$
  - Absorption: $a+(a \cdot b) = a$
  - Distributive: $(a+b) \cdot c = a \cdot c+b \cdot c$
  - Involution: $(a')' = a$
  - Complements: $a+a' = 1$
  - Identity: $a \cdot 1 = a$
  - Domination: $a \cdot 0 = 0$
  - De Morgan's law: $(a+b)' = a' \cdot b'$

Venn diagram



Axiomatic systems related simplification on Boolean algebra

- Duality
  - The rule that exchanged “+ and ･” and “0 and 1” will be approved (Dual rule)
  - e.g. $a+a = a \longleftrightarrow a \cdot a = a$
  - e.g. $a+a' = 1 \longleftrightarrow a \cdot a' = 0$
- We can insert arbitrary logical expressions into a, b, and c in prior equations

Review: 2-input logical operation

- AND, OR, NAND, and NOR: described before
- XOR: output 1 if the inputs are not equal
- XNOR: output 1 if the inputs are equal

	AND	OR	NAND	NOR	XOR	XNOR
x y	x·y	x+y	(x·y)'	(x+y)'	$x \oplus y$	$(x \oplus y)'$
0 0	0	0	1	1	0	1
0 1	0	1	1	0	1	0
1 0	0	1	1	0	1	0
1 1	1	1	0	0	0	1

De Morgan's law

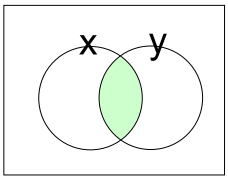
- (x·y)' = x'+y'
- (x+y)' = x'·y'
- We can insert arbitrary logical expressions into x and y

Equal Equal

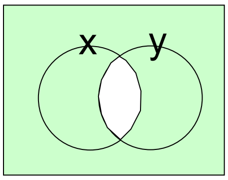
x y	(x·y)'	x'+y'	(x+y)'	x'·y'
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

De Morgan's law on Venn diagram

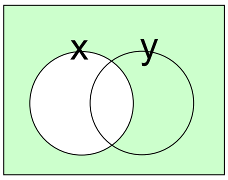
- Here's $(x \cdot y)' = x' + y'$ on Venn diagram



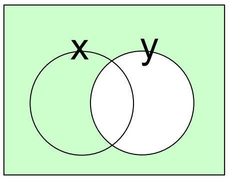
$x \cdot y$



$(x \cdot y)'$



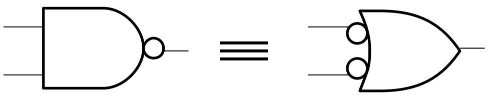
x'



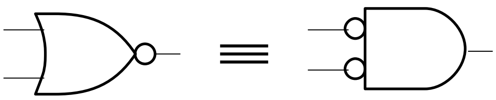
y'

De Morgan's law on circuit level

- NAND and NOR becomes AND and OR with negated inputs
- $(x \cdot y)' = x' + y'$

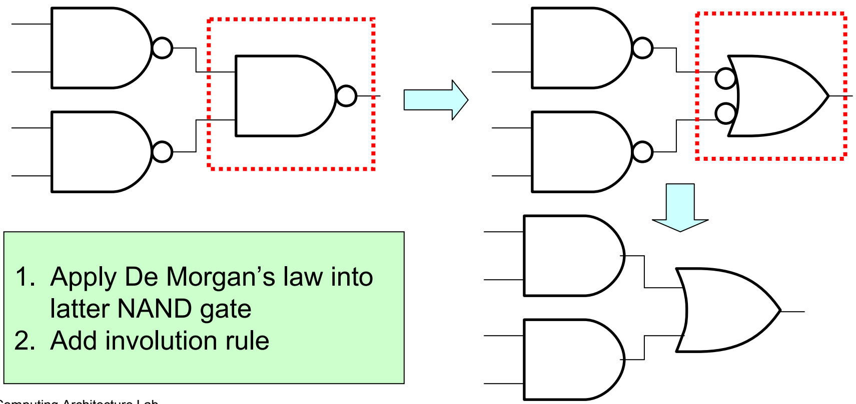


- $(x + y)' = x' \cdot y'$



A practical use of De Morgan's law on circuit level

- NAND-NAND two level logic circuit
  = AND-OR two level logic circuit



1. Apply De Morgan's law into latter NAND gate
2. Add involution rule

Generalized De Morgan's law

$$F'(x_1, x_2, \cdots, x_n) = G(x_1, x_2, \cdots, x_n)$$

Under
- $Xi \longleftrightarrow Xi'$
- $+ \longleftrightarrow \cdot$

- Widely used when you want to negate arbitrary logical function *f*

  e.g. (a'b'+a'b+ab')' = (a+b)(a+b')(a'+b)
  = aaa'+aab+ab'a'+ab'b+baa'+bab+bb'a'+bb'b
  = ab + ab = ab
  e.g. ((a･b)'･(b'･c)')' = (a･b) + (b'･c) = a･b + b'･c

How to create CNF?

1. Gain DNF of negated function
   - Sum of “0” term in truth table
2. Negate function obtained in 1.

a	b	c		h	s	t
0	0	0	a’b’c’	0	0	1
0	0	1	a’b’c	1	1	1
0	1	0	a’bc’	0	0	0
0	1	1	a’bc	1	1	0
1	0	0	ab’c’	0	0	0
1	0	1	ab’c	1	1	0
1	1	0	abc’	1	0	1
1	1	1	abc	0	1	1

h’ = a’b’c’ + a’bc’ + ab’c’ + abc

h’’ = (a’b’c’ + a’bc’ + ab’c’ + abc)’

De Morgan’s law

h = (a+b+c)(a+b’+c)
(a’+b+c)(a’+b’+c’)

Short exercise

- Show CNF of following logical function

a	b	c	d	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Answer

- Show CNF of following logical function

f' = a'b'cd' + a'bcd + abcd'
f = f'' = (a'b'cd' + a'bcd + abcd')'
= (a+b+c'+d)(a+b'+c'+d')(a'+b'+c'+d)

a	b	c	d	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

How to translate logical expression to sum of products or product of sums

h = a'(b'c + bc) + b'c' → (Negate) h' = (a'(b'c + bc) + b'c')'

↓ Expand

h = a'b'c + a'bc + b'c'
= a'c + b'c'
Sum of products

Note that the expansion route is not unique

↓ Expand

h' = ab + ac + bc'

↓ Negate

h'' = h = (ab + ac + bc')'

↓ De Morgan's law

h = (a'+b')(a'+c')(b'+c)
Product of Sums

Shannon's expansion

- A technique also used for translating logical expression to sum of products notation

$$f(x_1, x_2, \cdots, x_n) = x_1' \cdot f(0, x_2, \cdots, x_n) + x_1 \cdot f(1, x_2, \cdots, x_n)$$

e.g. (a'b'+a'b+ab')'

$$= a'(\underbrace{(1\cdot b'+1\cdot b+0\cdot b')'}_{\text{Substitute } a=0})+a(\underbrace{(0\cdot b+0\cdot b+1\cdot b')'}_{\text{Substitute } a=1})$$

$$= a'(\underbrace{(b'+b)}_{=1}')+a((b')')$$

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

Short exercise

- Expand following function by Shannon's expansion and translate it to sum of products

  f = {(a･b)'･(b'･c)'}'

Answer

- Expand following function by Shannon's expansion and translate it to sum of products

  $f = \{(a \cdot b)' \cdot (b' \cdot c)'\}'$

$$
\begin{aligned}
f &= a' \cdot \{\underbrace{(0 \cdot b)'}_{=1} \cdot (b' \cdot c)'\}' + a \cdot \{\underbrace{(1 \cdot b)'}_{=b'} \cdot (b' \cdot c)'\}' \\
&= a' \cdot \{(b' \cdot c)'\}' + a \cdot \{b' \cdot (b' \cdot c)'\}' \\
&= b' \cdot [a' \cdot \underbrace{\{(1 \cdot c)'\}'}_{=c} + a \cdot \underbrace{\{1 \cdot (1 \cdot c)'\}'}_{=c}] + b \cdot [a' \cdot \underbrace{\{(0 \cdot c)'\}'}_{=0} + a \cdot \underbrace{\{0 \cdot (0 \cdot c)'\}'}_{=1}] \\
&= b' \cdot (a' \cdot c + a \cdot c) + b \cdot a \\
&= \underbrace{(a' + a)}_{=1} \cdot b' \cdot c + a \cdot b = a \cdot b + b' \cdot c
\end{aligned}
$$

Equivalence of logical function

- There are equivalent logical expression in each logical function
  - In logical circuits design, there's possibility that it includes same circuits (= same logical expression)

    -> Redundant! (consume unnecessary silicon resources)
- How to check equivalence of them?
  - Checking on truth table is one method
    - The size of truth table is 2^n on n-value
  - Cogitated algorithm or data structure are required

-> Later Chap. 2

Quantity of logical function

- The logical function can be represented uniquely with truth table
- But there are 2^{2^n} of logical functions in n-value logical function

x	y	Q
0	0	?
0	1	?
1	0	?
1	1	?

There are 2^4 possible outputs

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

Examples of 2-input logical function

- There's possible functions which are not named
- But usually, there's no use

x y	AND $x \cdot y$	XOR $x \oplus y$	(= x)	(= 0)	(= y)	(= 1)
0 0	0	0	1	0	0	1
0 1	0	1	0	0	1	1
1 0	0	1	1	0	0	1
1 1	1	0	0	0	1	1

Quantity of logical function

- It increases dramatically in proportion to the number of values
  - $2^8 = 256$ in 3-value function
  - $2^{16} = 65536$ in 4-value function
  - $2^{32} = 4294967296$ in 5-value function
  - 2^{64} ($\fallingdotseq 1.8 \times 10^{19}$) in 6-value function
    - Too hard to check all of them even if we use computer!
- Let's consider how to reduce number of logical functions

Symmetry logical function

The logical function is symmetry on x_i and x_j if outputs do not change under permutation of x_i and x_j

Example of symmetry: $f(x_1,x_2) = x_1 + x_2 \quad (= x_2 + x_1)$

Example of not symmetry: $f(x_1,x_2) = x'_1 + x_2 \quad (\neq x'_2 + x_1)$

- Quantity of logical function becomes 2^{n+1} if the function has perfect symmetry
  - The outputs do not change under permutation of all variables
  - e.g. $x'_1 \cdot x_2 \cdot x_3 + x_1 \cdot x'_2 \cdot x_3 + x_1 \cdot x_2 \cdot x'_3$