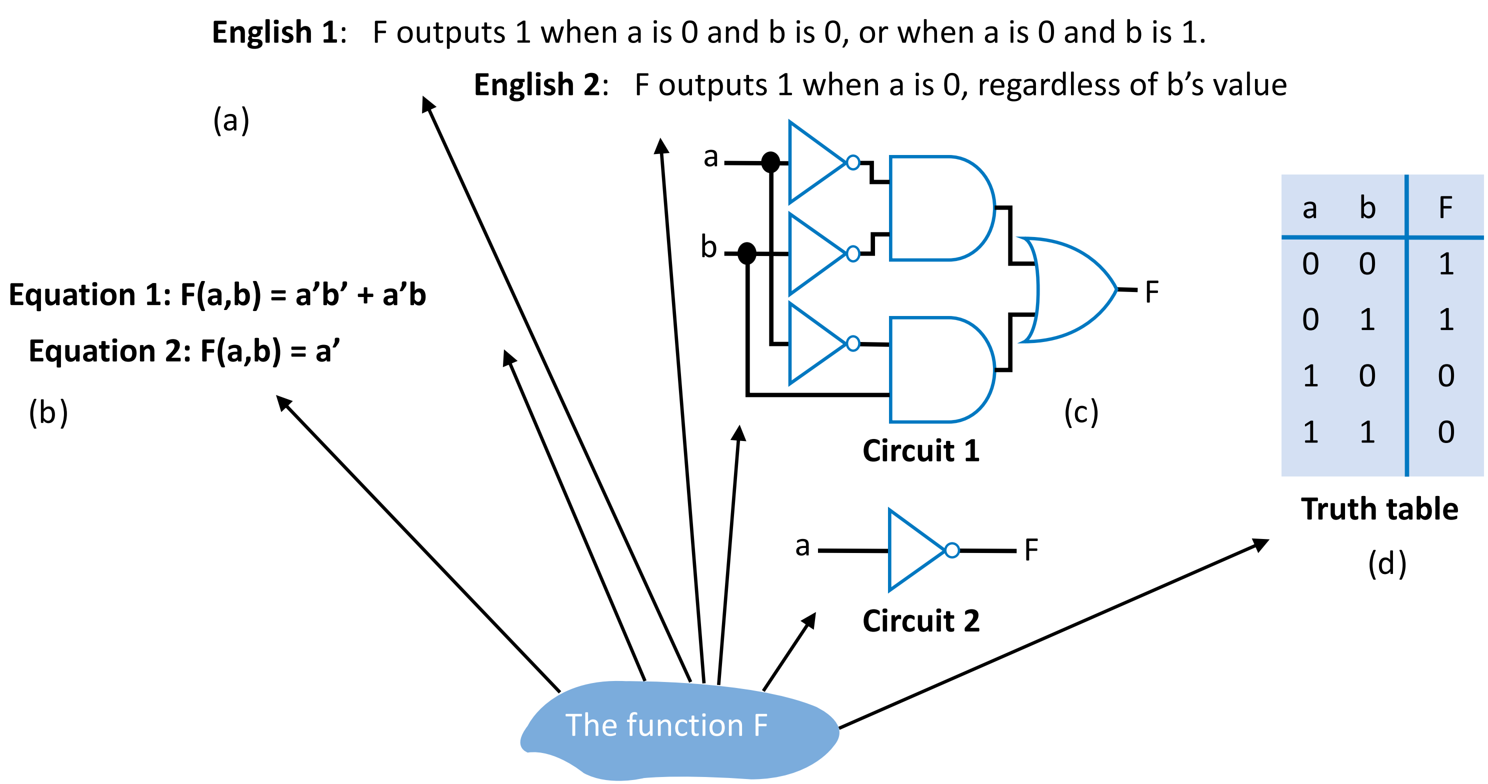
Representations of Boolean Functions



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

- A function can be represented in different ways
  - 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

- Define value of F for each possible combination of input values
  - 2-input function: 4 rows
  - 3-input function: 8 rows
  - 4-input function: 16 rows
- Q: Use truth table to define function F(a,b,c) that is 1 when abc is 5 or greater in binary

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)

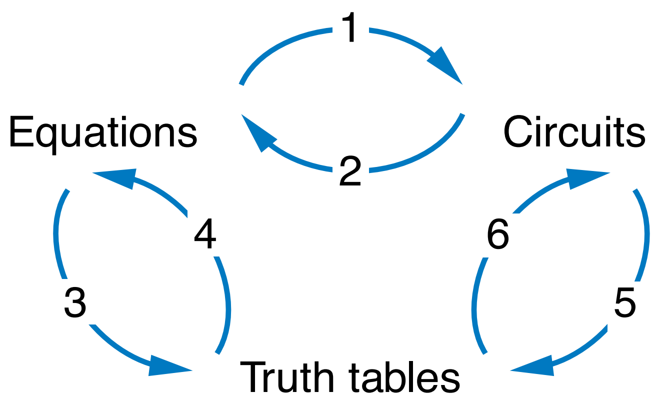
a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

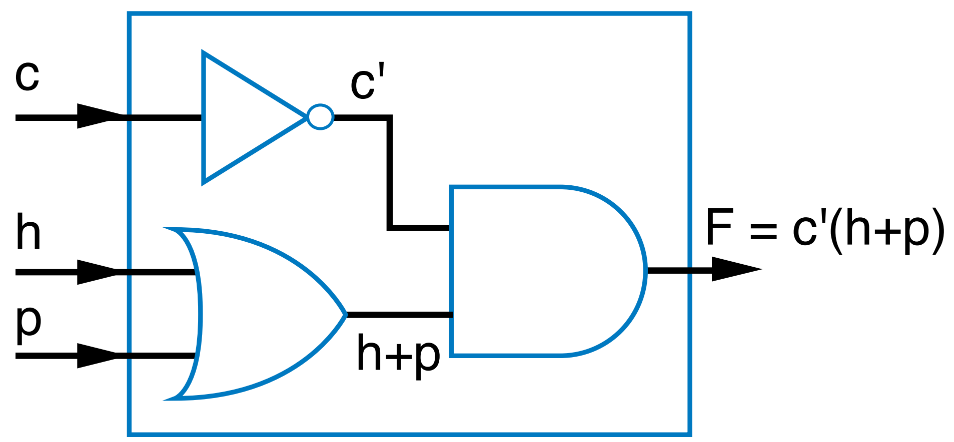
(c)

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**
1	1	0	**1**
1	1	1	**1**

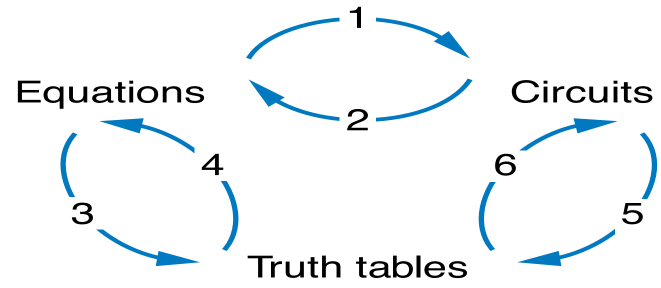
Converting among Representations

- Can convert from any representation to another
- Common conversions
  - Equation to circuit (we did this earlier)
  - Circuit to equation
    - Start at inputs, write expression of each gate output





Converting among Representations



- More common conversions
  - Truth table to equation (which we can then convert to circuit)
    - Easy–just OR each input term that should output 1
  - Equation to truth table
    - Easy—just evaluate equation for each input combination (row)
    - Creating intermediate columns helps

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	

F = a’b’ + a’b

Q: Convert to equation

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

F = ab’c + abc’ + abc

Q: 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**

Example: Converting from Truth Table to Equation

- Parity bit: Extra bit added to data, intended to enable detection of error (a bit changed unintentionally)
  - e.g., errors can occur on wires due to electrical interference
- Even parity: Set parity bit so total number of 1s (data + parity) is even
  - e.g., if data is 001, parity bit is 1 → 0011 has even number of 1s
- Want equation, but easiest to start from truth table for this example

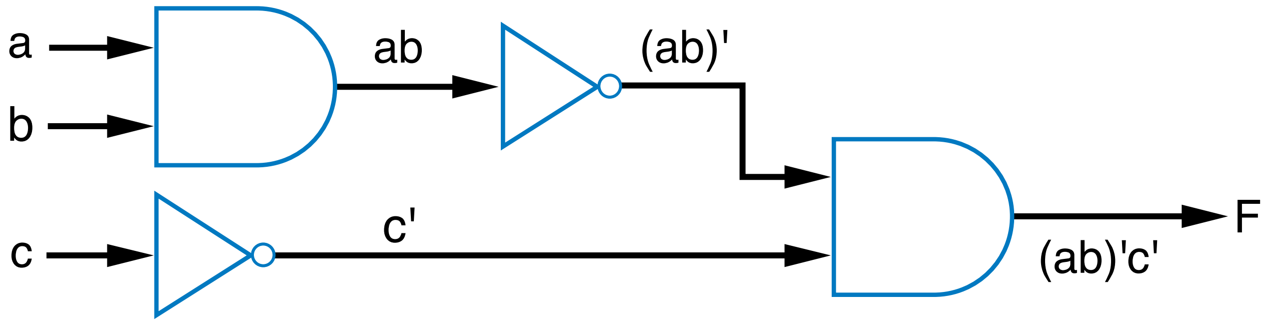
a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Convert to eqn.

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

Example: Converting from Circuit to Truth Table

- First convert to circuit to equation, then equation to table



Inputs						Outputs
a	b	c	ab	(ab)'	c'	F
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

Standard Representation: Truth Table

- How can we determine if two functions are the same?
  - Recall automatic door example
    - Same as f = hc' + h'pc'?
    - Used algebraic methods
    - But if we failed, does that prove *not* equal? No.
- Solution: Convert to truth tables
  - Only ONE truth table representation of a given function
    - ***Standard*** representation—for given function, only one version in standard form exists

f = c'hp + c'hp' + c'h'

f = c'h(p + p') + c'h'p

f = c'h(1) + c'h'p

f = c'h + c'h'p

(what if we stopped here?)

f = hc' + h'pc'

Q: Determine if F=ab+a' is same function as F=a'b'+a'b+ab, by converting each to truth table first

F = ab + a'

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

F = a'b' + a'b + ab

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

Same

Truth Table Canonical Form

- Q: Determine via truth tables whether ab+a' and (a+b)' are equivalent

$F = ab + a'$

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

$F = (a+b)'$

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

Not equivalent

a

Canonical Form – Sum of Minterms

- Truth tables too big for numerous inputs
- Use standard form of equation instead
  - Known as ***canonical form***
  - Regular algebra: group terms of polynomial by power
    - $ax^2 + bx + c$ $(3x^2 + 4x + 2x^2 + 3 + 1 \text{ --> } 5x^2 + 4x + 4)$
  - Boolean algebra: create sum of minterms
    - ***Minterm***: product term with every function literal appearing exactly once, in true or complemented form
    - Just multiply-out equation until sum of product terms
    - Then expand each term until all terms are minterms

Q: Determine if F(a,b)=ab+a’ is equivalent to F(a,b)=a’b’+a’b+ab, by converting first equation to canonical form (second already is)

F = ab+a’ (already sum of products)
a F = ab + a’(b+b’) (expanding term)
F = ab + a’b + a’b’ (Equivalent – same three terms as other equation)

Canonical Form – Sum of Minterms

- Q: Determine whether the functions G(a,b,c,d,e) = abcd + a'bcde and H(a,b,c,d,e) = abcde + abcde' + a'bcde + a'bcde(a' + c) are equivalent.

G = abcd + a'bcde

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

G = abcde + abcde' + a'bcde

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

Equivalent

a

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

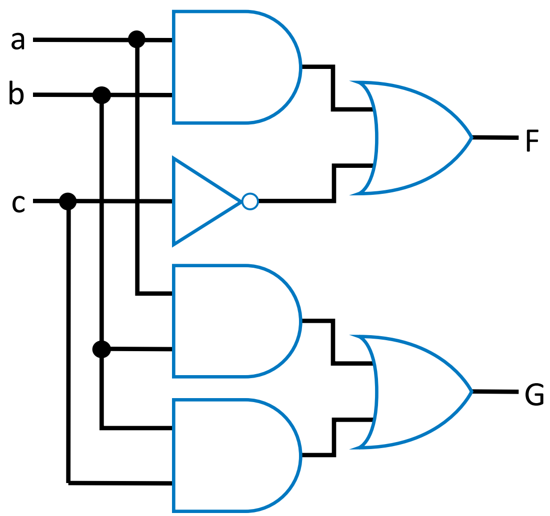
H = a'bcde + abcde' + abcde

Compact Sum of Minterms Representation

- List each minterm as a number
- Number determined from the binary representation of its variables' values
  - a'bcde corresponds to 01111, or 15
  - abcde' corresponds to 11110, or 30
  - abcde corresponds to 11111, or 31
- Thus, H = a'bcde + abcde' + abcde can be written as:
  - $H = \sum m(15,30,31)$
  - "H is the sum of minterms 15, 30, and 31"

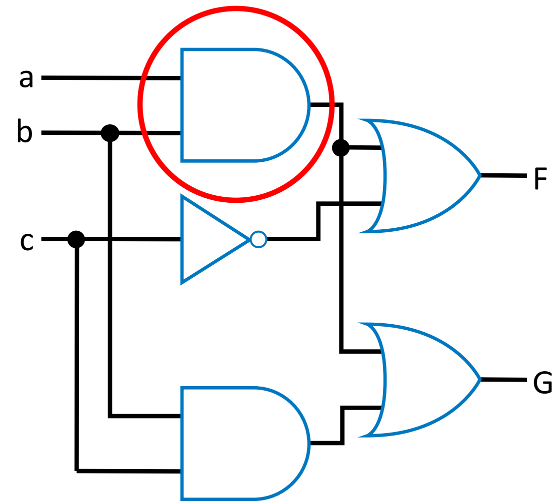
Multiple-Output Circuits

- Many circuits have more than one output
- Can give each a separate circuit, or can share gates
- Ex: F = <u>ab</u> + c', G = <u>ab</u> + bc



(a)

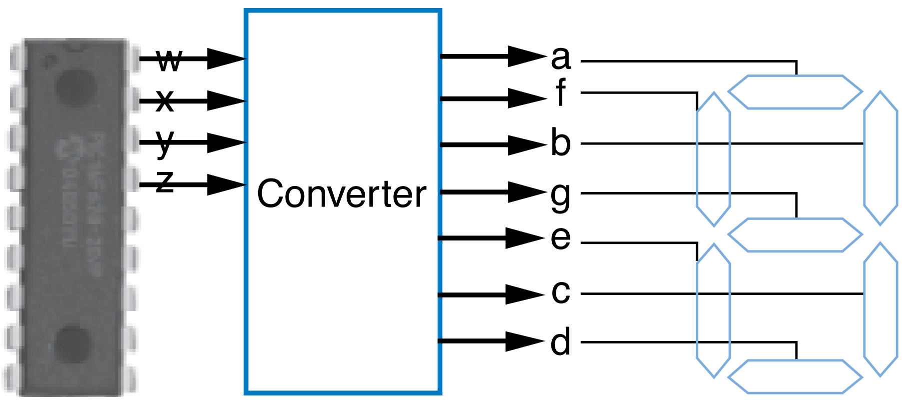
Option 1: Separate circuits

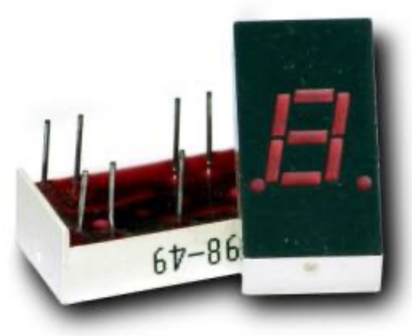


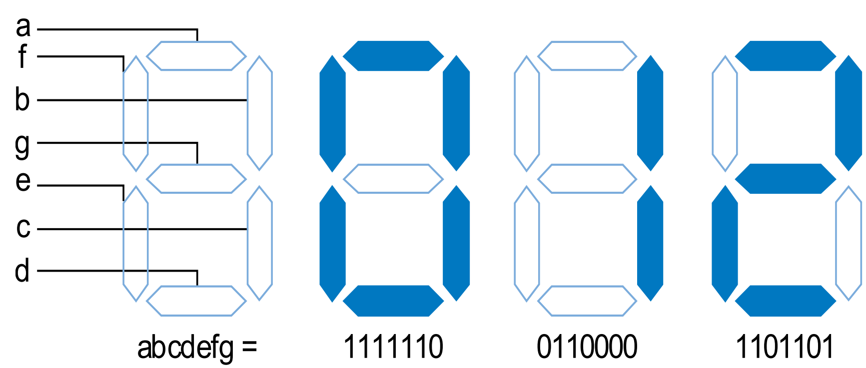
(b)

Option 2: Shared gates

Multiple-Output Example:
BCD to 7-Segment Converter





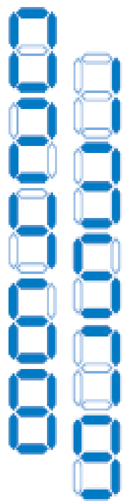


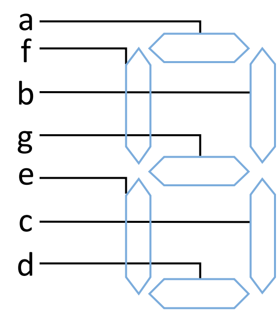
(a) (b)

Multiple-Output Example:
BCD to 7-Segment Converter

TABLE 2-4 4-bit binary number to seven-segment display truth table

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0





$a = w'x'y'z' + w'x'yz' + w'x'yz + w'xy'z + w'xyz' + w'xyz + wx'y'z' + wx'y'z$

$b = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'xyz + wx'y'z' + wx'y'z$

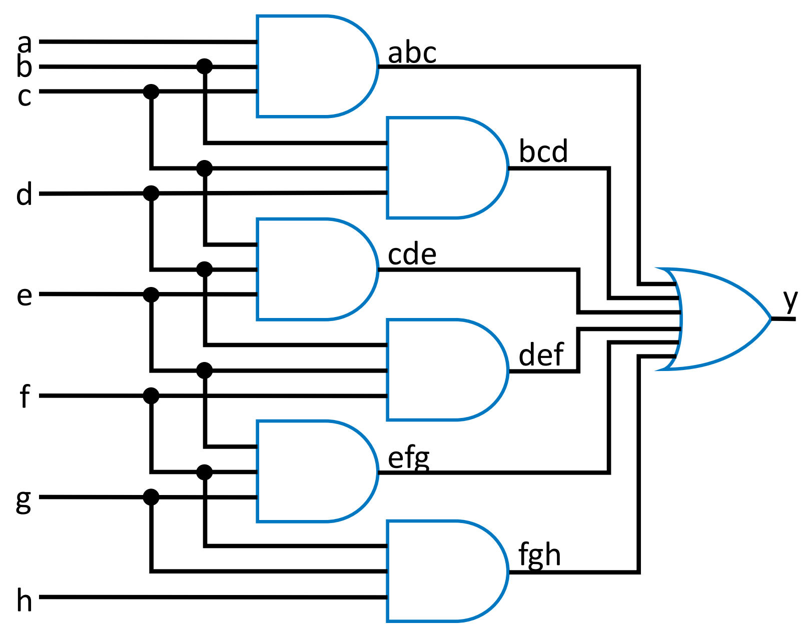
•••

Combinational Logic Design Process

Step		Description
Step 1: Capture behavior	**Capture** the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of each output of the combinational logic.
Step 2: Convert to circuit	2A: **Create** equations	This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
	2B: **Implement** as a gate-based circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

Example: Three 1s Pattern Detector

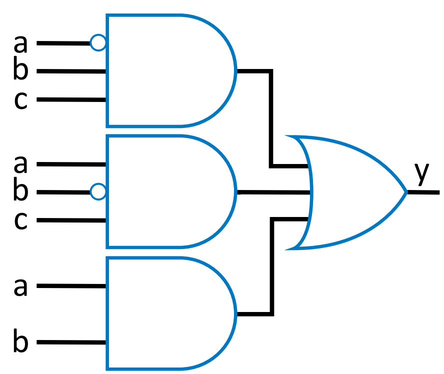
- Problem: Detect three consecutive 1s in 8-bit input: abcdefgh
    - 00**111**01 → 1
    - 10101011 → 0
    - **111**10000 → 1
  - **Step 1: Capture** the function
    - Truth table or equation?
      - Truth table too big: 2^8=256 rows
      - Equation: create terms for each possible case of three consecutive 1s
    - **y = abc + bcd + cde + def + efg + fgh**
  - **Step 2a: Create** equation -- already done
  - **Step 2b: Implement** as a gate-based circuit

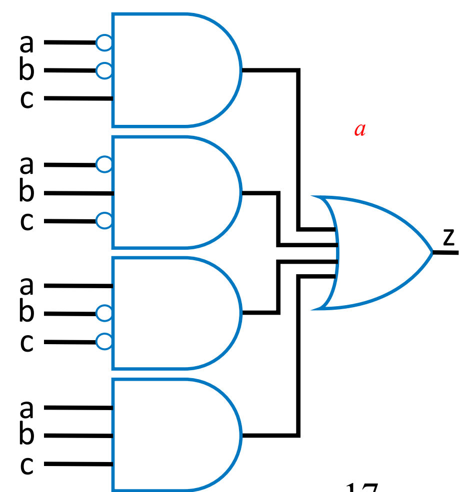


Example: Number of 1s Counter

- Problem: Output in binary on two outputs yz the # of 1s on three inputs
    - 010 → 01
    - 101 → 10
    - 000 → 00
  - **Step 1: Capture** the function
    - Truth table or equation?
      - Truth table is straightforward
  - **Step 2a: Create** equations
    - y = a’bc + ab’c + abc’ + abc
    - z = a’b’c + a’bc’ + ab’c’ + abc
    - Optional: Let's simplify y:
      - y = a'bc + ab'c + ab(c' + c) = a'bc + ab'c + ab
  - **Step 2b: Implement** as a gate-based circuit

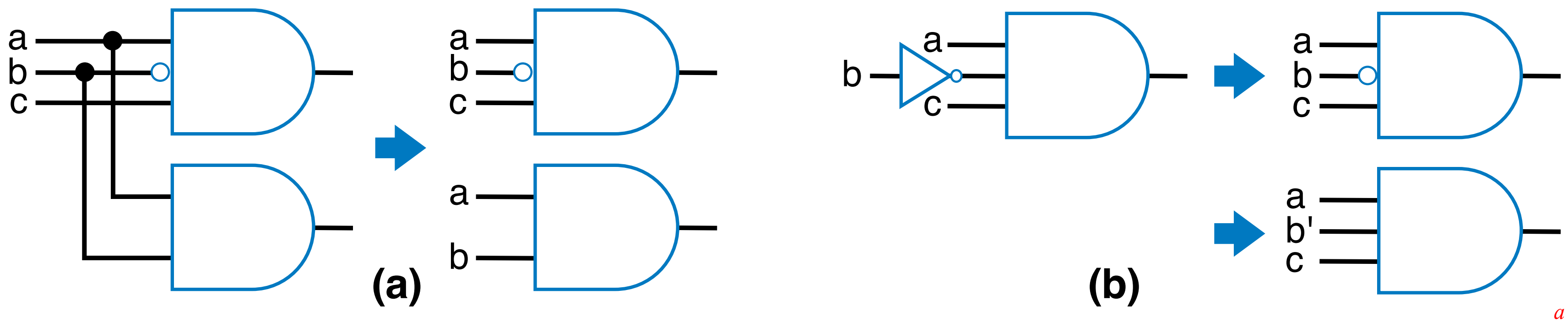
Inputs			(# of 1s)	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1





Simplifying Notations

- Used in previous circuit



List inputs multiple times
→ Less wiring in drawing

Draw inversion bubble rather than inverter. Or list input as complemented.