Digital circuit design: combinational logic and optimization

Contents

- Truth table
- Canonical form – sum of minterms
- Combinational logic design with examples
- K-map
- More gates and their properties
- Multiplexer and decoder

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

Same

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

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

a

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)
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

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

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

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

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

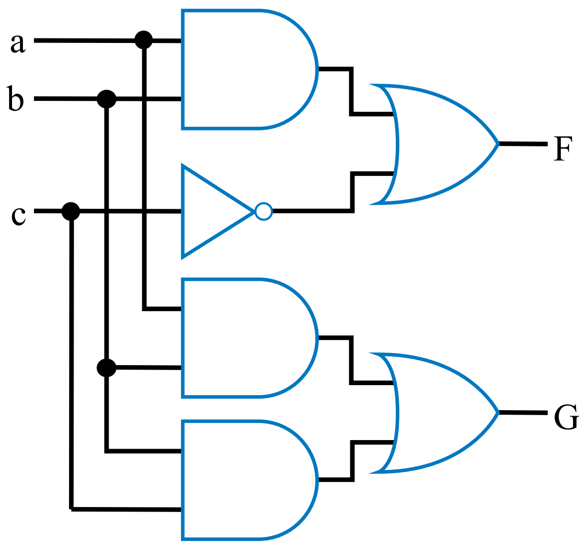
$H = a'bcde + abcde' + abcde$

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"

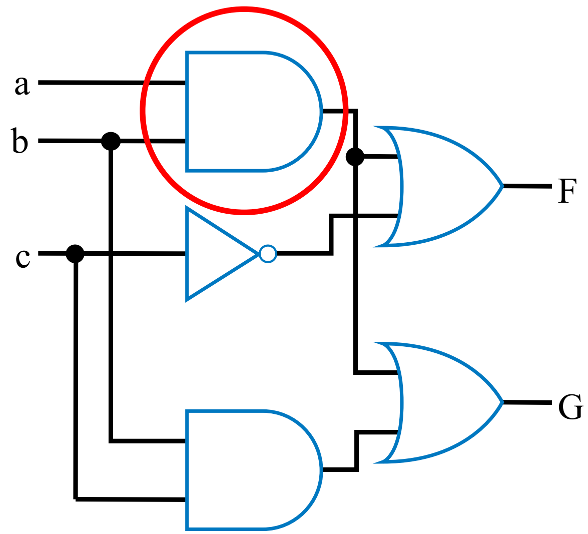
Multiple-Output Circuits

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



(a)

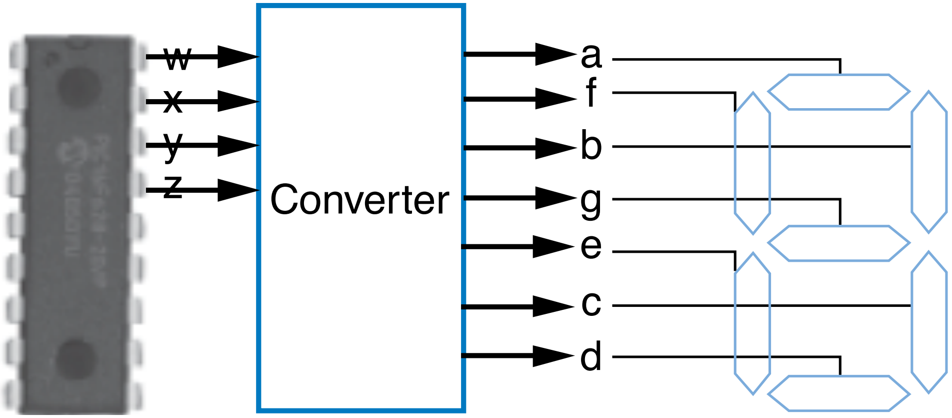
Option 1: Separate circuits

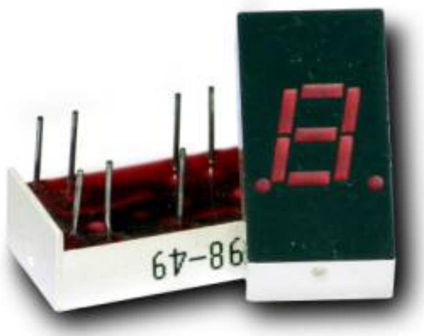


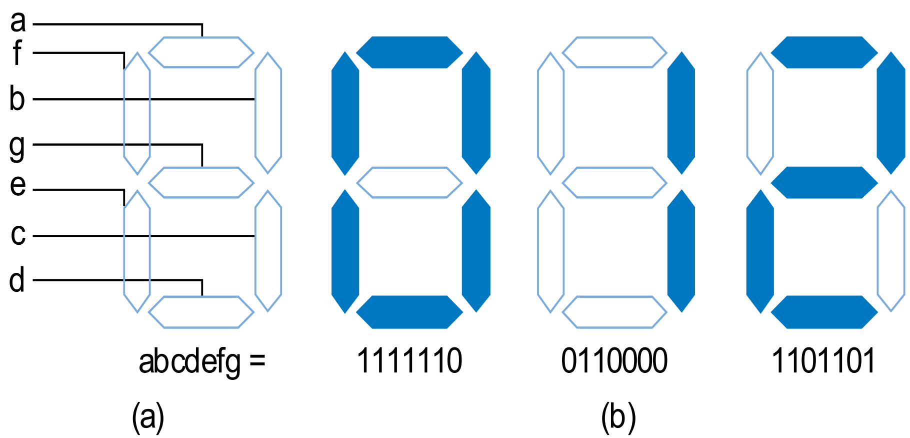
(b)

Option 2: Shared gates

Multiple-Output Example:
BCD to 7-Segment Converter



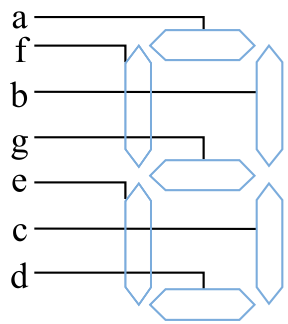




Multiple-Output Example:
BCD to 7-Segment Converter

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

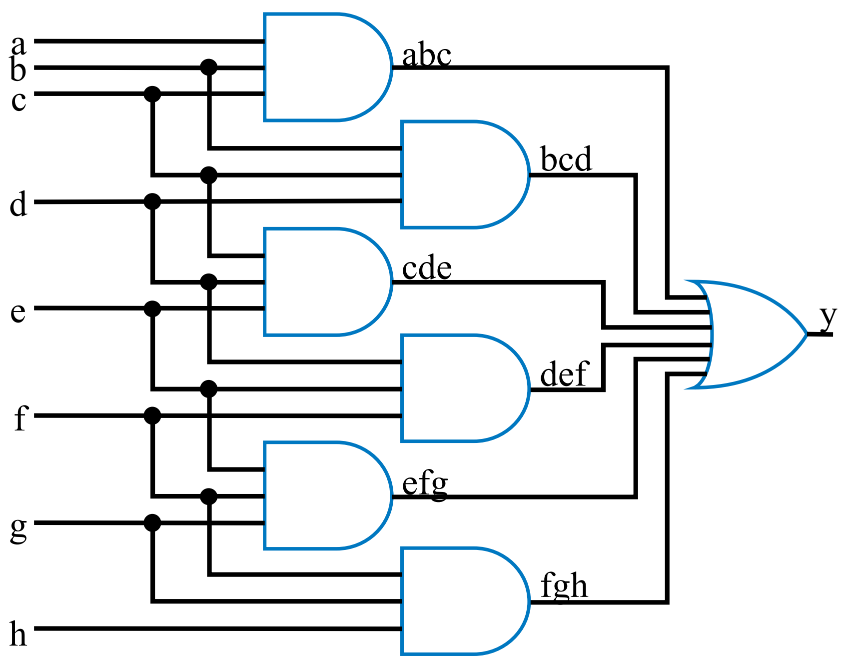
Step		Description
Step 1: Capture behavior	**Capture** the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of each output of the combinational logic.
Step 2: Convert to circuit	2A: **Create** equations	This substep is 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
    - 000**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

a

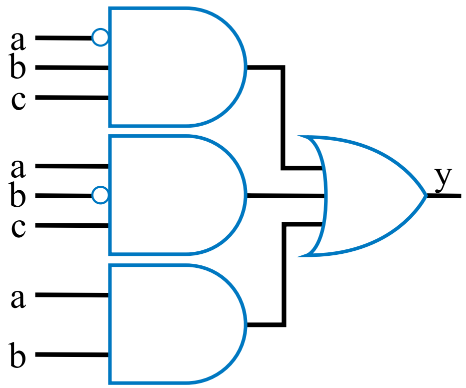
a

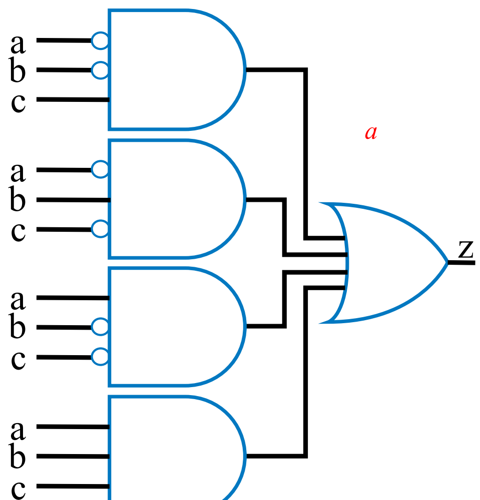


Example: Number of 1s Counter

- 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

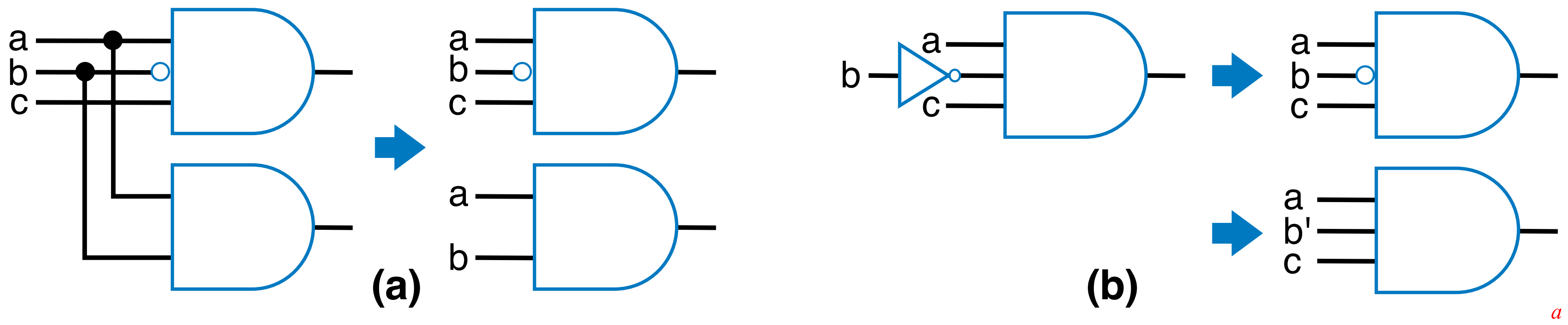
Inputs			(# of 1s)	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1





Simplifying Notations

- Used in previous circuit

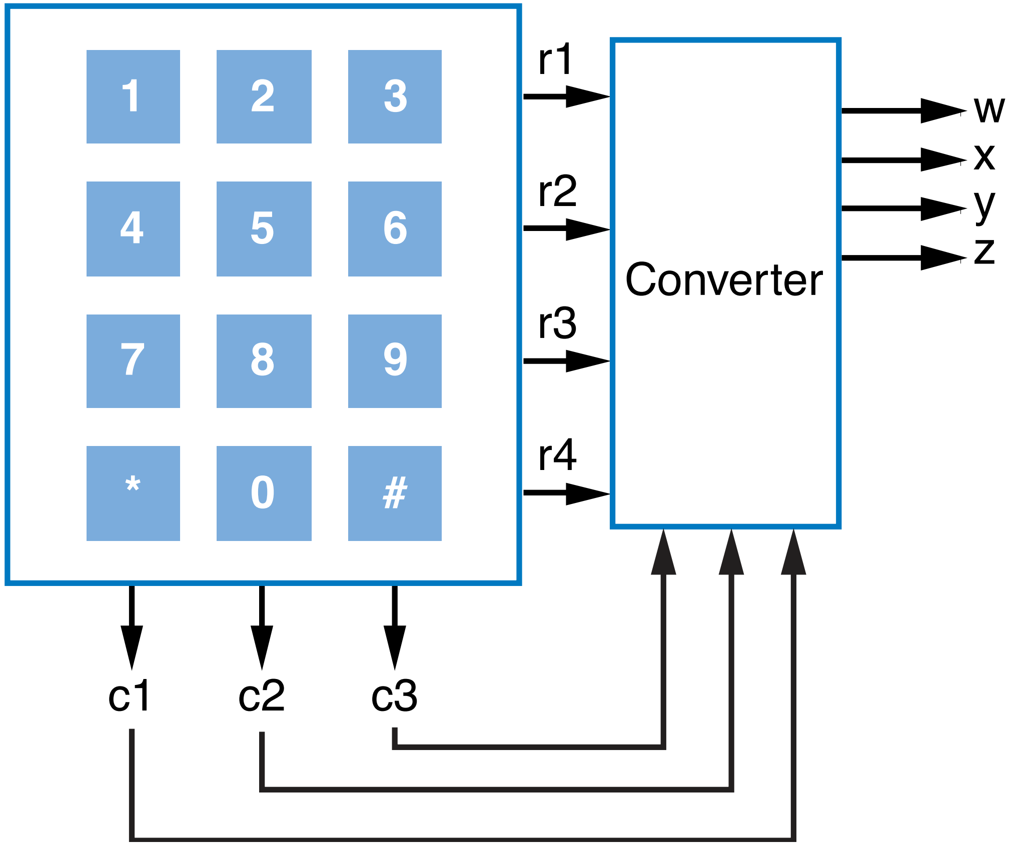


List inputs multiple times
→ Less wiring in drawing

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

Example: Keypad Converter

- Keypad has 7 outputs
  - One per row
  - One per column
- Key press sets one row and one column output to 1
  - Press "5" → r2=1, c2=1
- Goal: Convert keypad outputs into 4-bit binary number
  - 0-9 → 0000 to 1001
  - * → 1010, # → 1011
  - nothing pressed: 1111



Example: Keypad Converter

- Step 1: Capture behavior
  - Truth table too big (2^7 rows); equations not clear either
  - Informal table can help

TABLE 2.7 Informal table for the 12-button keypad to 4-bit code converter.

Button	Signals		4-bit code outputs			
			w	x	y	z
1	r1	c1	0	0	0	1
2	r1	c2	0	0	1	0
3	r1	c3	0	0	1	1
4	r2	c1	0	1	0	0
5	r2	c2	0	1	0	1
6	r2	c3	0	1	1	0
7	r3	c1	0	1	1	1
8	r3	c2	1	0	0	0
9	r3	c3	1	0	0	1
*	r4	c1	1	0	1	0
0	r4	c2	0	0	0	0
#	r4	c3	1	0	1	1
(none)			1	1	1	1

Step 2b: Implement as circuit *(note sharable gates)* …

w = r3c2 + r3c3 + r4c1 + r4c3 + r1'r2'r3'r4'c1'c2'c3'

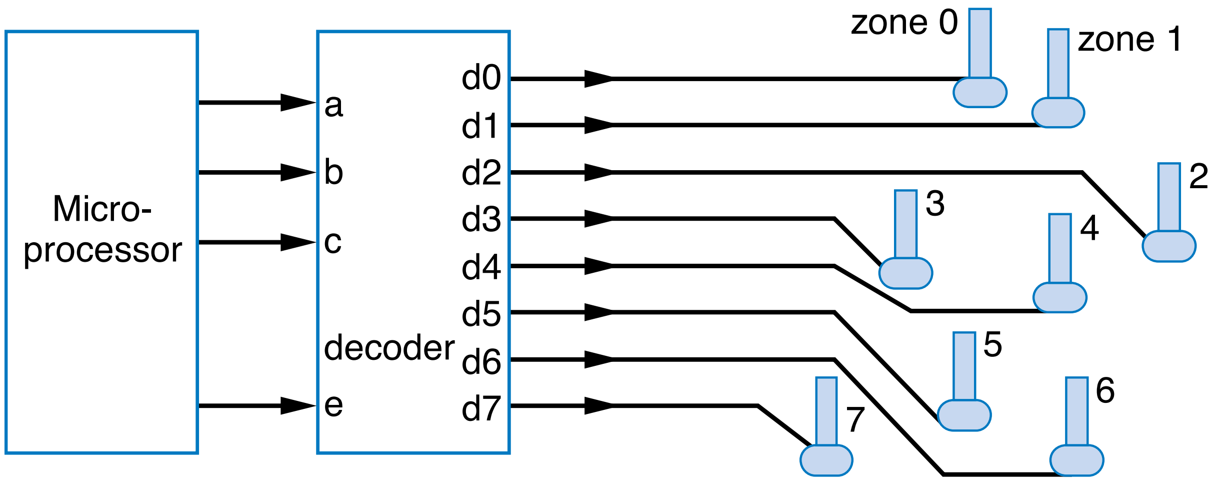
x = r2c1 + r2c2 + r2c3 + r3c1 + r1'r2'r3'r4'c1'c2'c3'

y = r1c2 + r1c3 + r2c3 + r3c1 + r4c1 + r4c3 + r1'r2'r3'r4'c1'c2'c3'

z = r1c1 + r1c3 + r2c2 + r3c1 + r3c3 + r4c3 + r1'r2'r3'r4'c1'c2'c3'

Example: Sprinkler Controller

- Microprocessor outputs which zone to water (e.g., cba=110 means zone 6) and enables watering (e=1)
- Decoder should set appropriate valve to 1



Step 1: Capture behavior

d0 = a'b'c'e

a

d1 = a'b'ce

d2 = a'bc'e

d3 = a'bce

d4 = ab'c'e

d5 = ab'ce

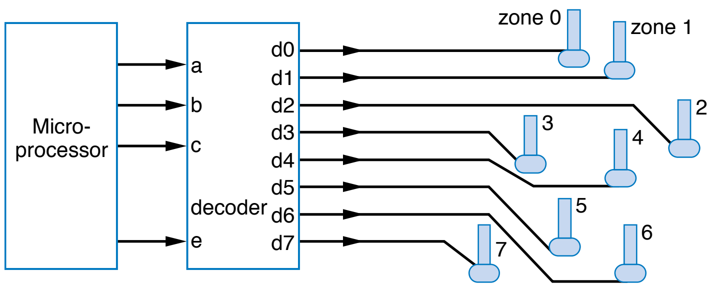
d6 = abc'e

d7 = abce

Equations seem like a natural fit

Example: Sprinkler Controller

- Step 2b: Implement as circuit



$$d0 = a'b'c'e$$

$$d1 = a'b'ce$$

$$d2 = a'bc'e$$

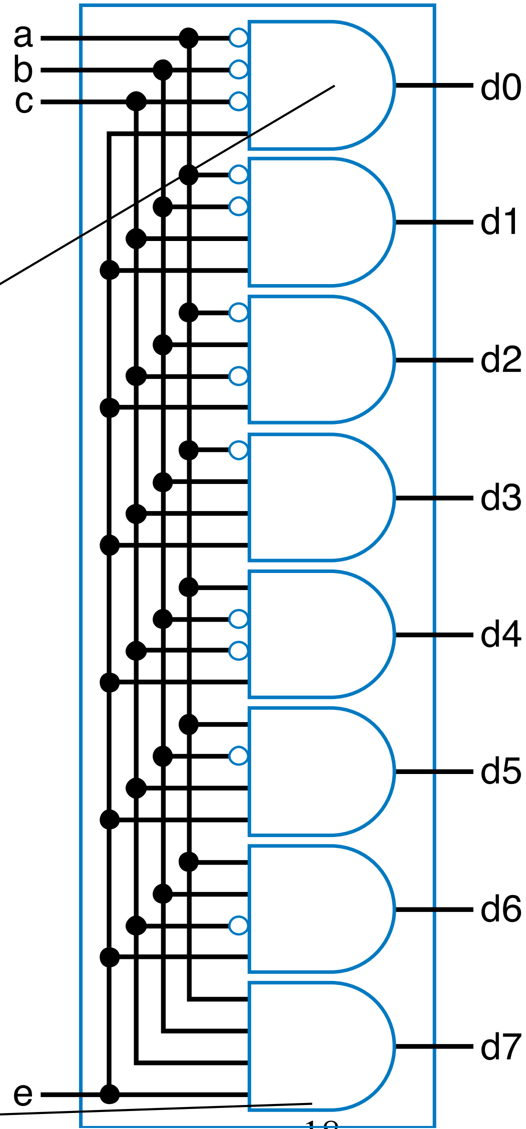
$$d3 = a'bce$$

$$d4 = ab'c'e$$

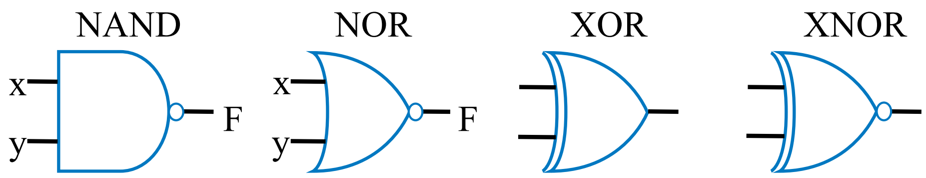
$$d5 = ab'ce$$

$$d6 = abc'e$$

$$d7 = abce$$



More Gates



NAND

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

NOR

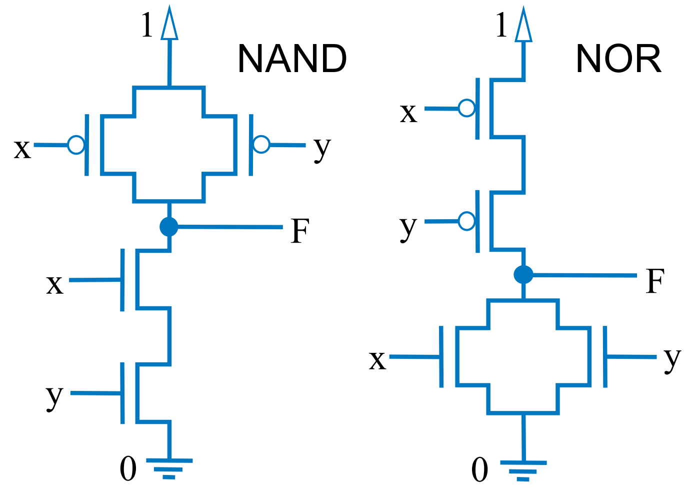
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

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

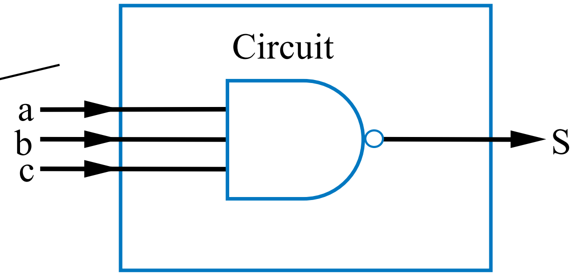


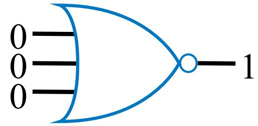
- NAND: Opposite of AND (“NOT AND”)
- NOR: Opposite of OR (“NOT OR”)
- XOR: Exactly 1 input is 1, for 2-input XOR. (For more inputs -- odd number of 1s)
- XNOR: Opposite of XOR (“NOT XOR”)

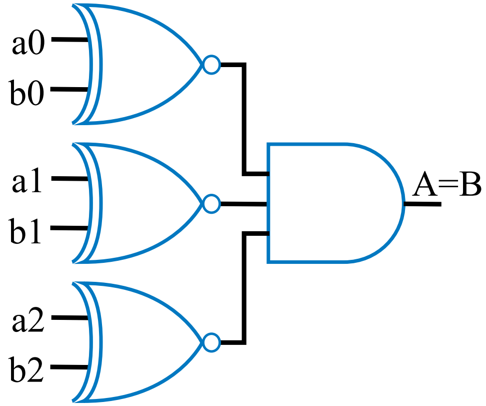
- NAND same as AND with power & ground switched
  - nMOS conducts 0s well, but not 1s (reasons beyond our scope) – so NAND is more efficient
- Likewise, NOR same as OR with power/ground switched
- NAND/NOR more common
- AND in CMOS: NAND with NOT
- OR in CMOS: NOR with NOT

More Gates: Example Uses

- Aircraft lavatory sign example
  - S = (abc)'
- Detecting all 0s
  - Use NOR
- Detecting equality
  - Use XNOR
- Detecting odd # of 1s
  - Use XOR
  - Useful for generating "parity" bit common for detecting errors

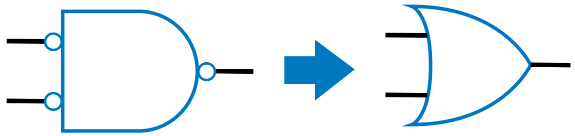






Completeness of NAND

- Any Boolean function can be implemented *using just NAND gates.* Why?
  - Need AND, OR, and NOT
  - NOT: 1-input NAND (or 2-input NAND with inputs tied together)
  - AND: NAND followed by NOT
  - OR: NAND preceded by NOTs
  - Thus, NAND is a universal gate
    - Can implement any circuit using just NAND gates



- Likewise for NOR

Number of Possible Boolean Functions

- How many possible functions of 2 variables?
  - 2^2 rows in truth table, 2 choices for each
  - $2^{(2^2)} = 2^4 = 16$ possible functions
- N variables
  - 2^N rows
  - $2^{(2^N)}$ possible functions

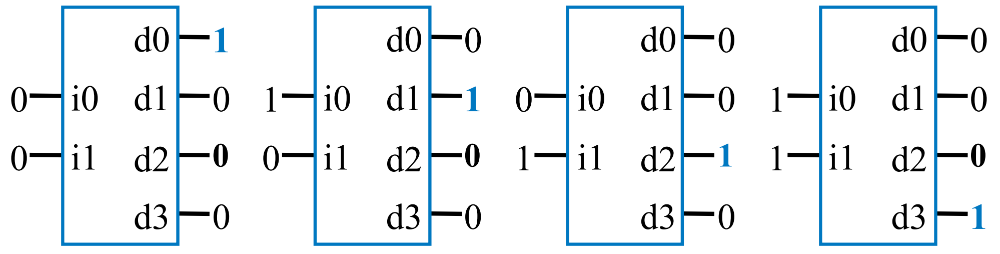
a	b	F	
0	0	0 or 1	2 choices
0	1	0 or 1	2 choices
1	0	0 or 1	2 choices
1	1	0 or 1	2 choices

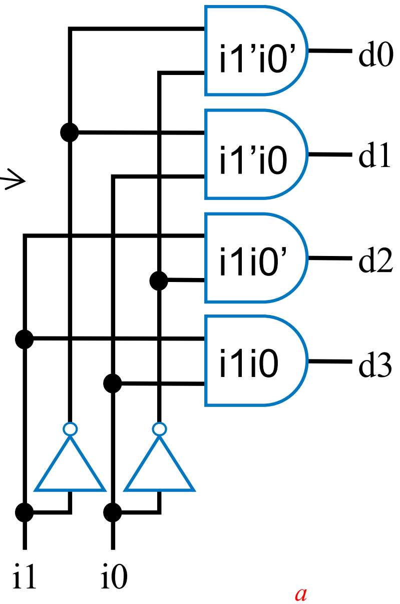
$2^4 = 16$
possible functions

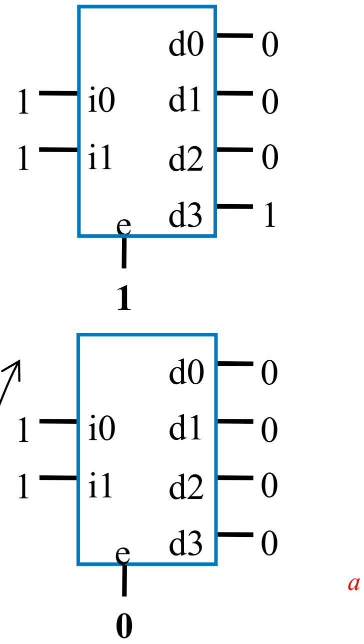
a	b	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
0	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**1**	**1**	**0**	**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**
		0	a AND b		a		b	a XOR b	a OR b	a NOR b	a XNOR b	b'		a'		a NAND b	1

Decoders and Muxes

- **Decoder**: Popular combinational logic building block, in addition to logic gates
  - Converts input binary number to one high output
- 2-input decoder: four possible input binary numbers
  - So has four outputs, one for each possible input binary number
- Internal design
  - AND gate for each output to detect input combination
- Decoder with enable e
  - Outputs all 0 if e=0
  - Regular behavior if e=1
- n-input decoder: 2^n outputs







Decoder Example

- New Year's Eve Countdown Display
  - Microprocessor counts from 59 down to 0 in binary on 6-bit output
  - Want illuminate one of 60 lights for each binary number
  - Use 6x64 decoder
    - 4 outputs unused