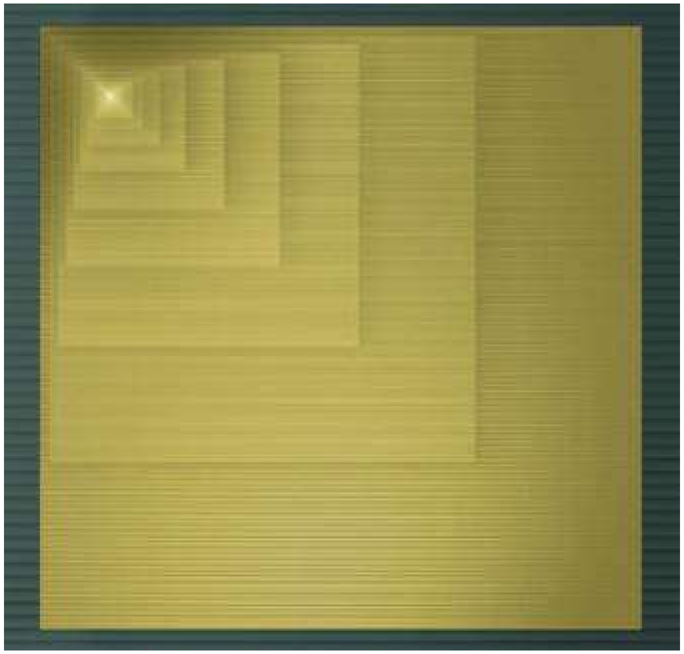


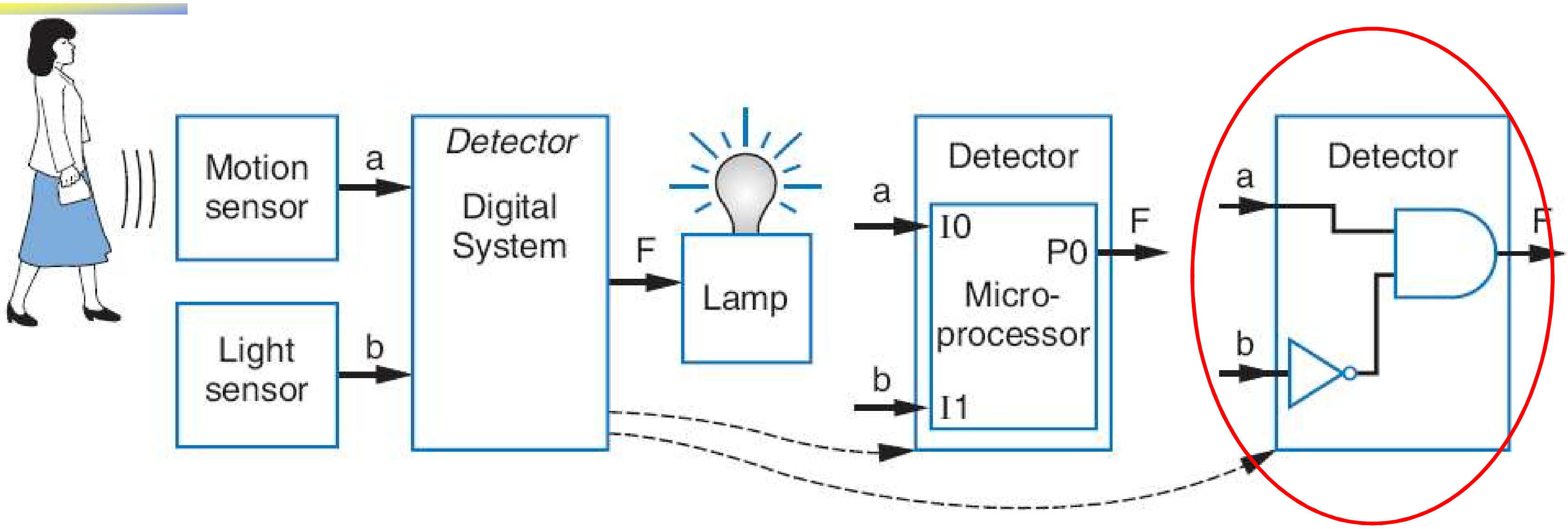
Digital Design

Chapter 2:

Combinational Logic Design

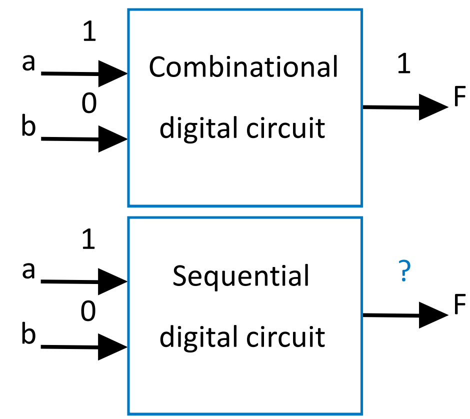
Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.

Introduction



Digital circuit

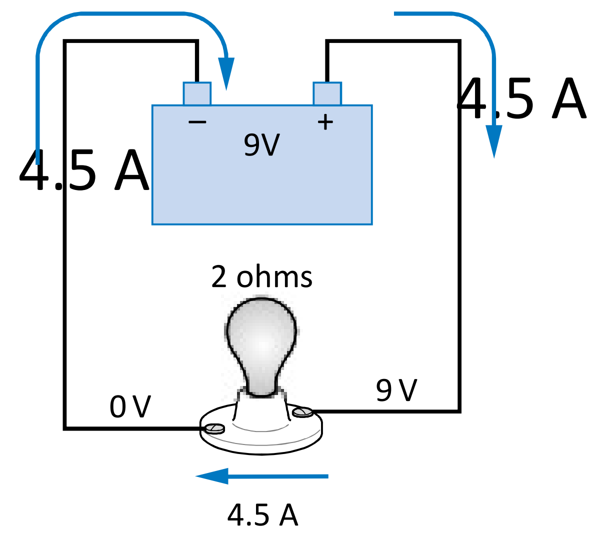
- Let's learn to design digital circuits
- We'll start with a simple form of circuit:
  - *Combinational circuit*
    - A digital circuit whose outputs depend solely on the *present* ***combination*** *of the circuit inputs' values*



Note: Slides with animation are denoted with a small red "a" near the animated items

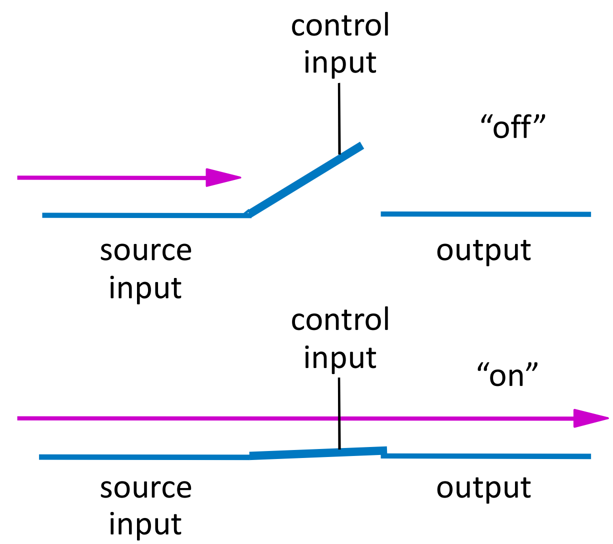
Switches

- Electronic switches are the basis of binary digital circuits
  - Electrical terminology
    - ***Voltage***: Difference in electric potential between two points
    - ***Current***: Flow of charged particles
    - ***Resistance***: Tendency of wire to resist current flow
    - V = I * R (Ohm's Law)

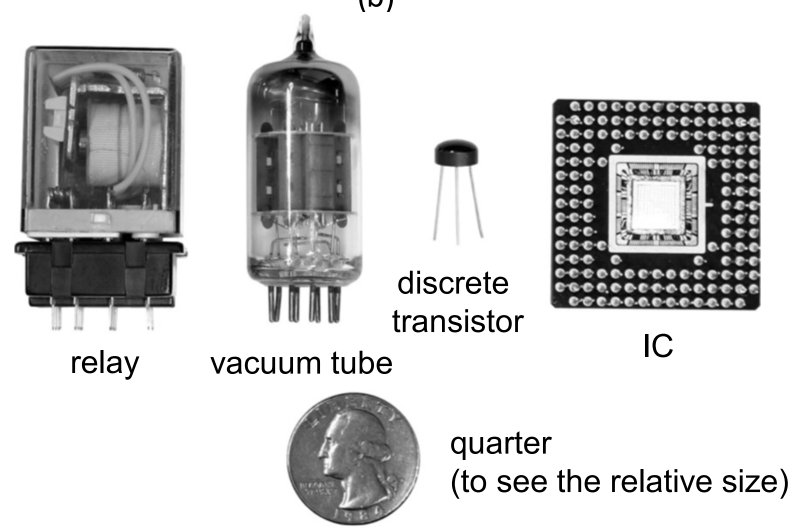


Switches

- A switch has three parts
  - Source input, and output
    - Current wants to flow from source input to output
  - Control input
    - Voltage that controls whether that current can flow

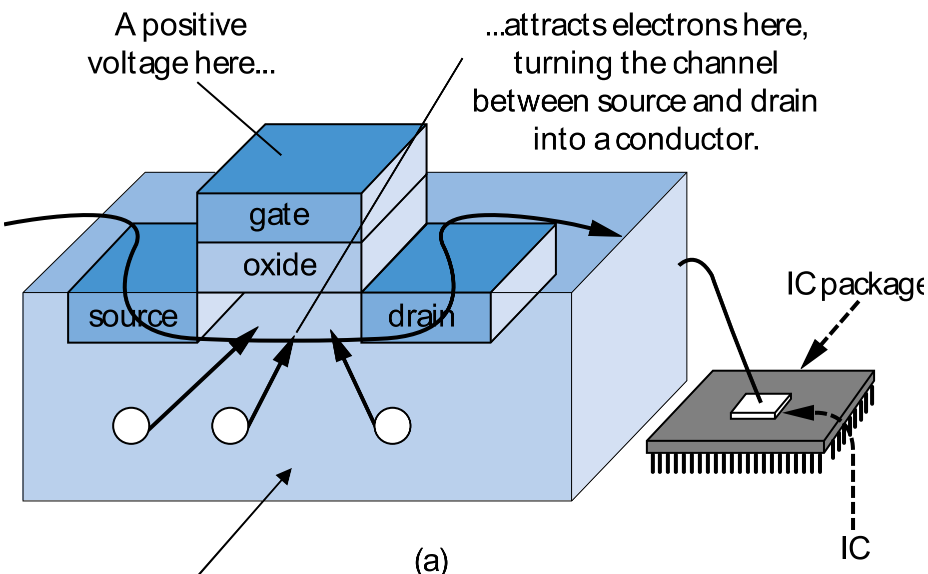


(b)

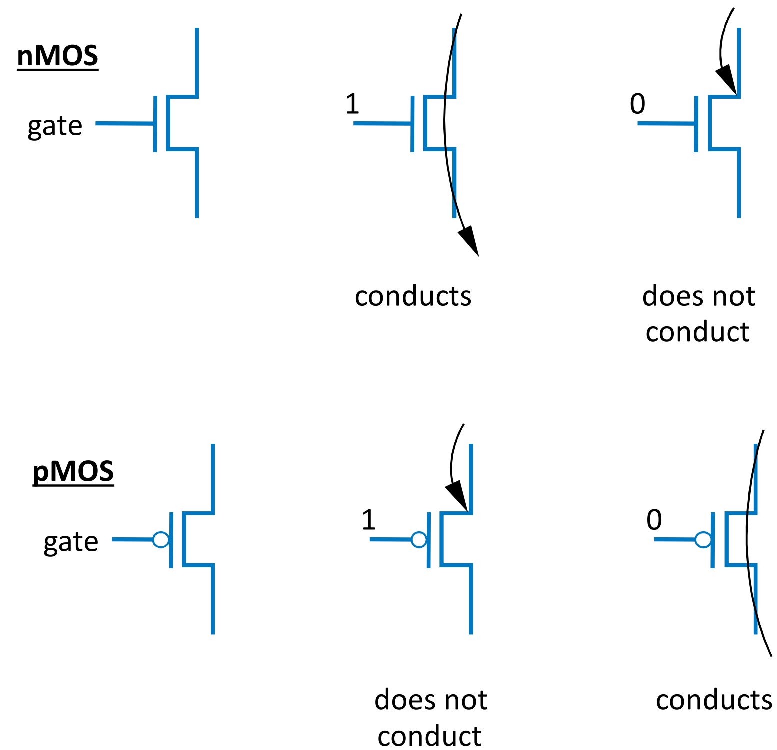


The CMOS Transistor

- CMOS transistor
  - Basic switch in modern ICs



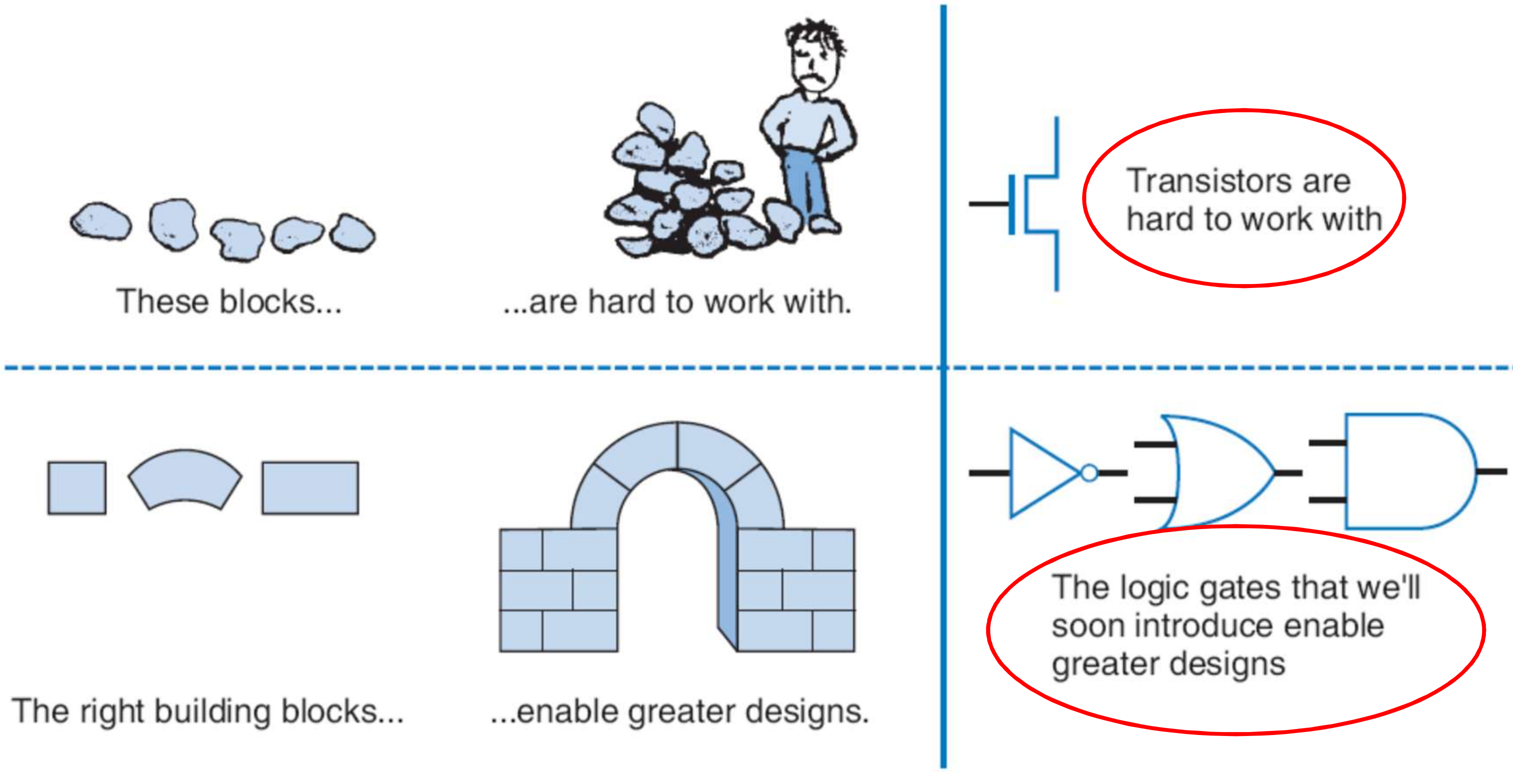
Silicon -- not quite a conductor or insulator:
Semiconductor



Boolean Logic Gates

Building Blocks for Digital Circuits

(Because Switches are Hard to Work With)



- "Logic gates" are better digital circuit building blocks than switches (transistors)
  - Why?...

Boolean Algebra and its Relation to Digital Circuits

- ***Boolean Algebra***
  - Variables represent 0 or 1 only
  - Operators return 0 or 1 only
  - Basic operators
    - AND: *a AND b* returns 1 only when both a=1 and b=1
    - OR: *a OR b* returns 1 if either (or both) a=1 or b=1
    - NOT: *NOT a* returns the opposite of a (1 if a=0, 0 if a=1)

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0

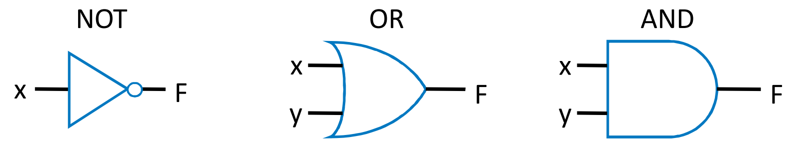
Converting to Boolean Equations

- Convert the following English statements to a Boolean equation
  - Q1. a is 1 and b is 1.
    - Answer: F = a AND b
  - Q2. either of a or b is 1.
    - Answer: F = a OR b
  - Q3. both a and b are not 0.
    - Answer:
      - (a) Option 1: F = NOT(a) AND NOT(b)
      - (b) Option 2: F = a OR b

Relating Boolean Algebra to Digital Design

	NOT	OR	AND

Symbol



Truth table

NOT

x	F
0	1
1	0

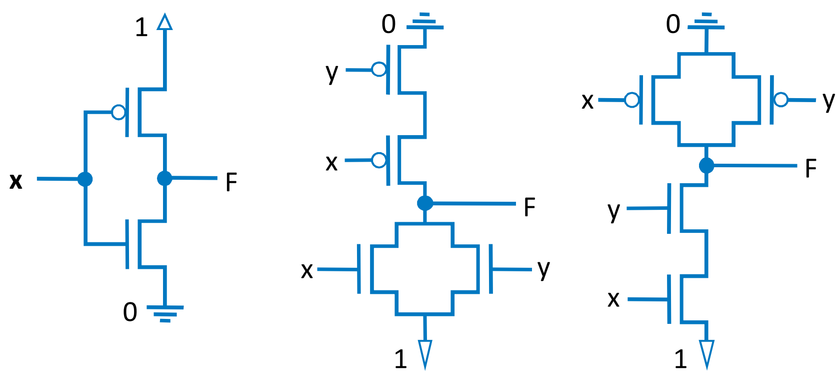
OR

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

AND

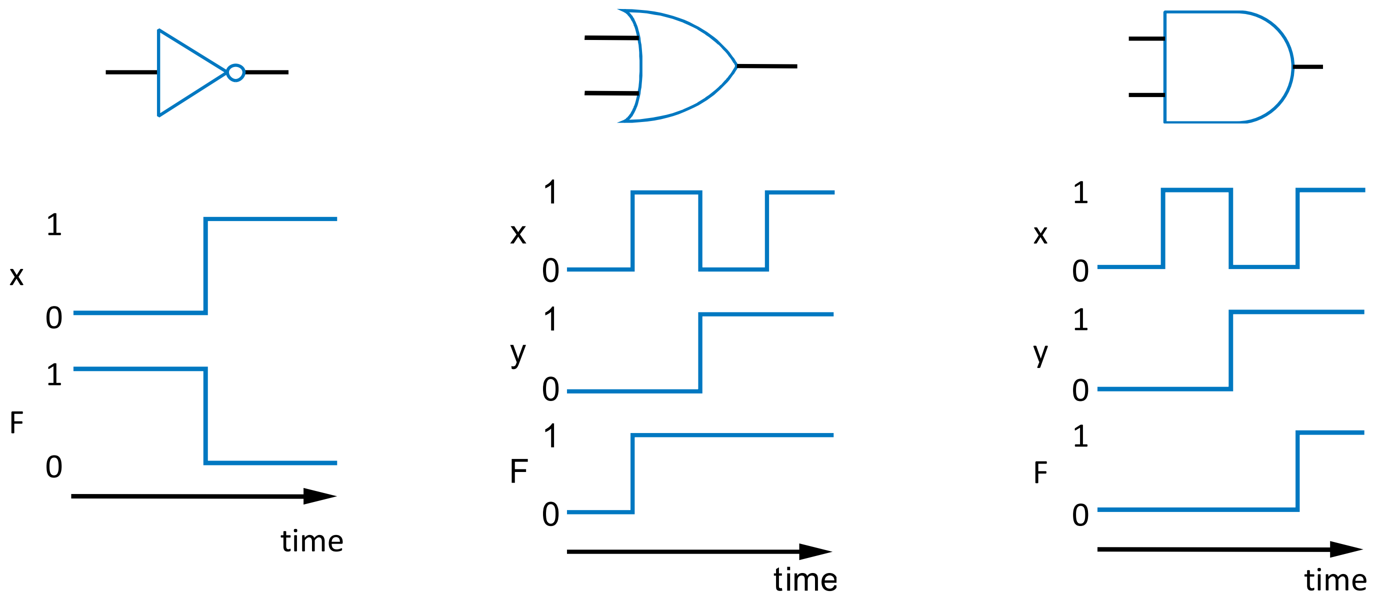
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

Transistor circuit



- Implement Boolean operators using transistors
  - Call those implementations ***logic gates***.

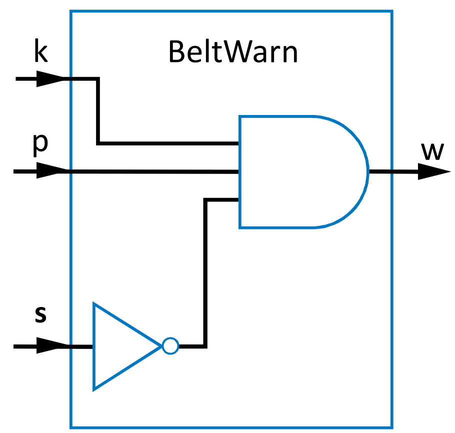
NOT/OR/AND Logic Gate Timing Diagrams



Example: Seat Belt Warning Light System

- Design circuit for warning light
- Sensors
  - s=1: seat belt fastened
  - k=1: key inserted
  - p=1: person in seat
- Capture Boolean equation
  - person in seat, and seat belt not fastened, and key inserted
- Convert equation to circuit

w = p AND NOT(s) AND k



Boolean Algebra Terminology

- Example equation: **F(a,b,c) = a’bc + abc’ + ab + c**
- ***Variable***
  - Represents a value (0 or 1)
  - Three variables: a, b, and c
- ***Literal***
  - Appearance of a variable, in true or complemented form
  - Nine literals: a’, b, c, a, b, c’, a, b, and c
- ***Product term***
  - Product of literals
  - Four product terms: a’bc, abc’, ab, c
- ***Sum-of-products***
  - Equation written as OR of product terms only
  - Above equation is in sum-of-products form. “F = (a+b)c + d” is not.

Boolean Algebra Properties

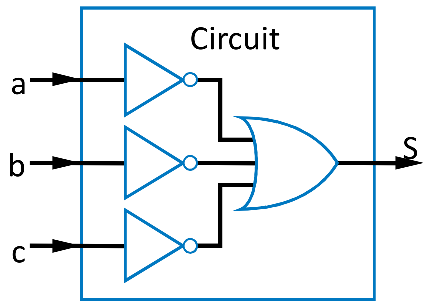
- Commutative
  - a + b = b + a
  - a * b = b * a
- Distributive
  - a * (b + c) = a * b + a * c
  - a + (b * c) = (a + b) * (a + c)
    - (this one is tricky!)
- Associative
  - (a + b) + c = a + (b + c)
  - (a * b) * c = a * (b * c)
- Identity
  - 0 + a = a + 0 = a
  - 1 * a = a * 1 = a
- Complement
  - a + a’ = 1
  - a * a’ = 0
- To prove, just evaluate all possibilities

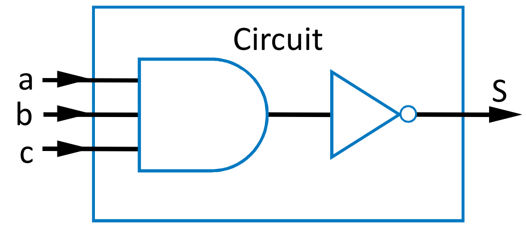
Example uses of the properties

- Show abc + abc’ = ab.
  - Use first distributive property
    - abc + abc’ = ab(c+c’).
  - Complement property
    - Replace c+c’ by 1: ab(c+c’) = ab(1).
  - Identity property
    - ab(1) = ab*1 = ab.

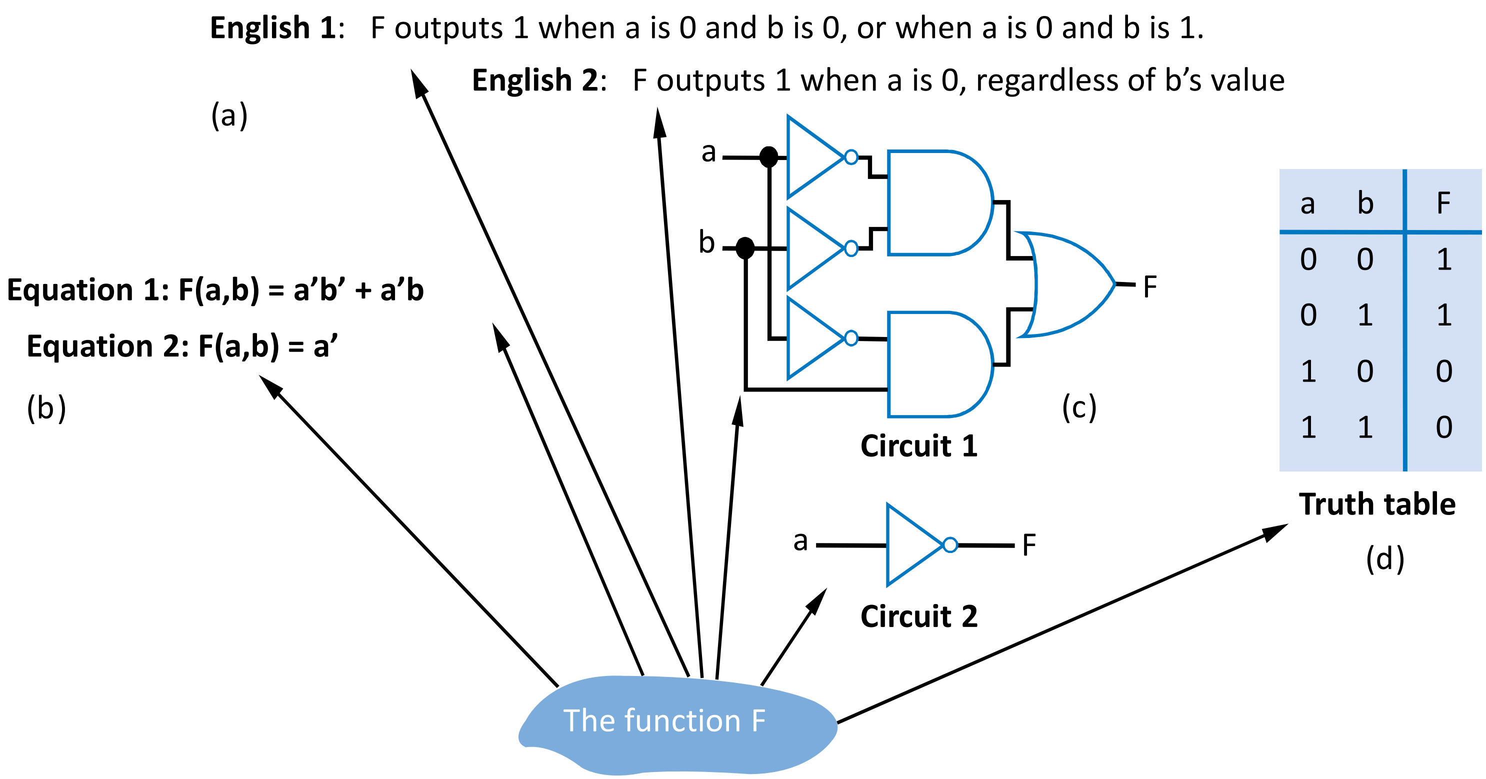
Boolean Algebra: Additional Properties

- Null elements
  - a + 1 = 1
  - a * 0 = 0
- Idempotent Law
  - a + a = a
  - a * a = a
- Involution Law
  - (a')' = a
- DeMorgan's Law
  - (a + b)' = a'b'
  - (ab)' = a' + b'
  - Very useful!
- To prove, just evaluate all possibilities





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**

Standard Representation: Truth Table

- How can we determine if two functions are the same?
    - 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

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

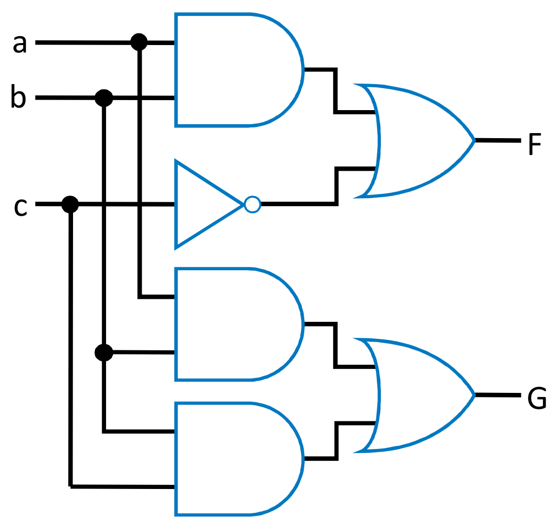
Canonical Form -- Sum of Minterms

- Truth tables too big for numerous inputs
- Use standard form of equation instead
  - Known as ***canonical form***
  - 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

a

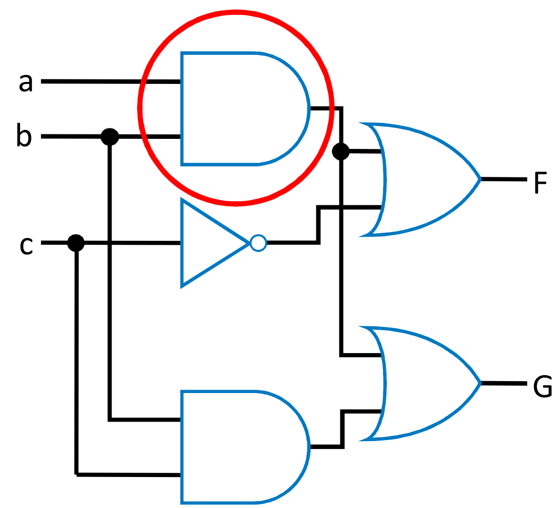
Multiple-Output Circuits

- 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

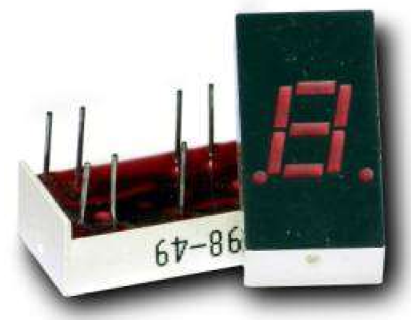
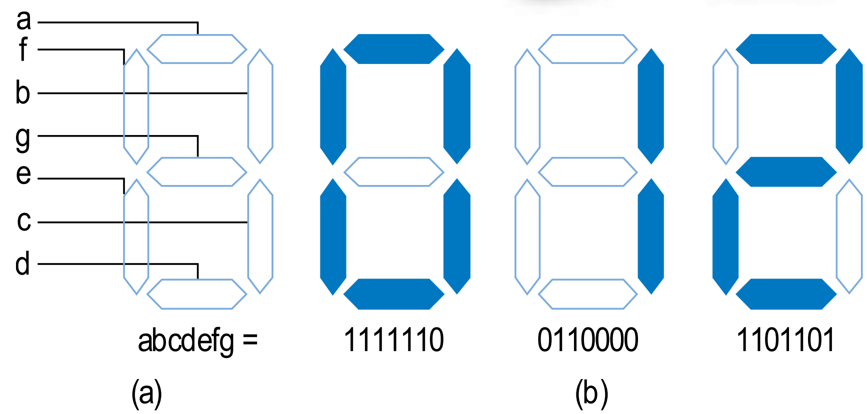


TABLE 2-4 4-bit binary number to seven-segment display truth table

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



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

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

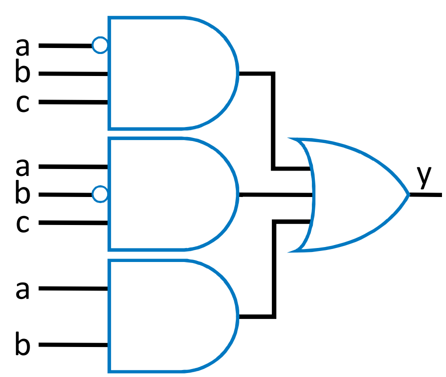
Combinational Logic Design Process

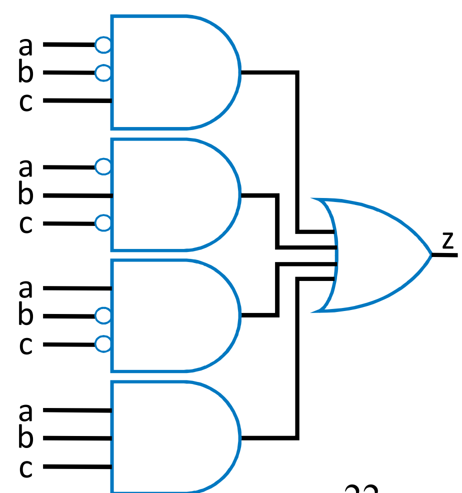
	Step	Description
Step 1	**Capture** the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of the combinational logic.
Step 2	**Convert** to equations	This step is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
Step 3	**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: Number of 1s Count

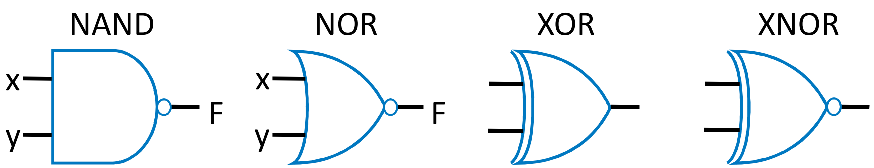
- Problem: Output in binary on two outputs yz the number of 1s on three inputs
    - 010 → 01   101 → 10   000 → 00
  - **Step 1: Capture** the function
    - Truth table or equation?
      - Truth table is straightforward
  - **Step 2: Convert** to equation
    - y = a'bc + ab'c + abc' + abc
    - z = a'b'c + a'bc' + ab'c' + abc
  - **Step 3: 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





More Gates



NAND

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

NOR

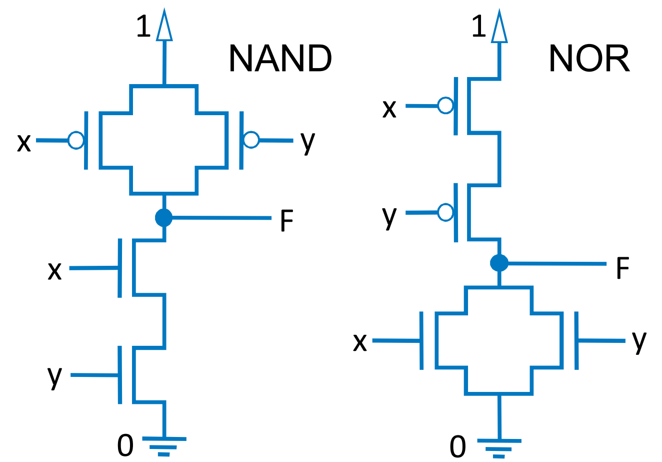
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

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

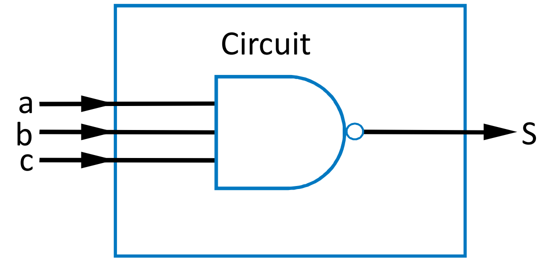


- NAND: Opposite of AND (“NOT AND”)
- NOR: Opposite of OR (“NOT OR”)
- XOR: Exactly 1 input is 1, for 2-input XOR. (For more inputs -- odd number of 1s)
- XNOR: Opposite of XOR (“NOT XOR”)

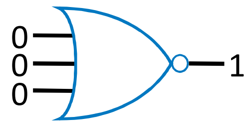
- NAND same as AND with power & ground switched
  - Why? nMOS conducts 0s well, but not 1s (reasons beyond our scope) -- so NAND more efficient
- Likewise, NOR same as OR with power/ground switched
- AND in CMOS: NAND with NOT
- OR in CMOS: NOR with NOT
- So NAND/NOR more common

More Gates: Example Uses

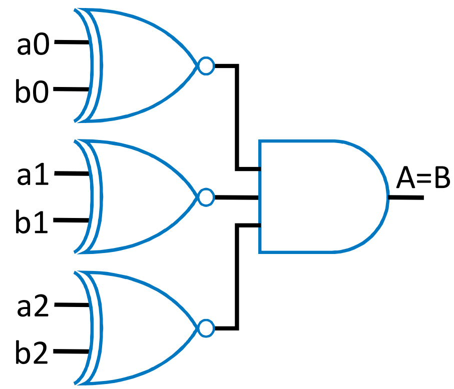
- Aircraft lavatory sign example
  - S = (abc)’



- Detecting all 0s
  - Use NOR



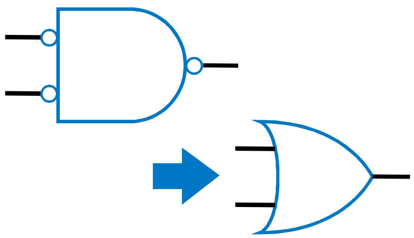
- Detecting equality
  - Use XNOR



- Detecting odd # of 1s
  - Use XOR
  - Useful for generating “parity” bit common for detecting errors

Completeness of NAND

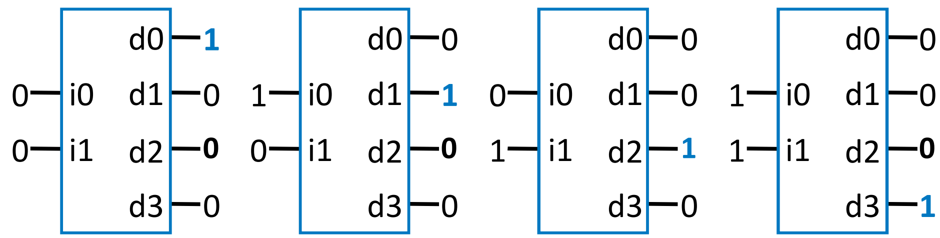
- Any Boolean function can be implemented *using just NAND gates.* Why?
  - Need AND, OR, and NOT
  - NOT: 1-input NAND (or 2-input NAND with inputs tied together)
  - AND: NAND followed by NOT
  - OR: NAND preceded by NOTs

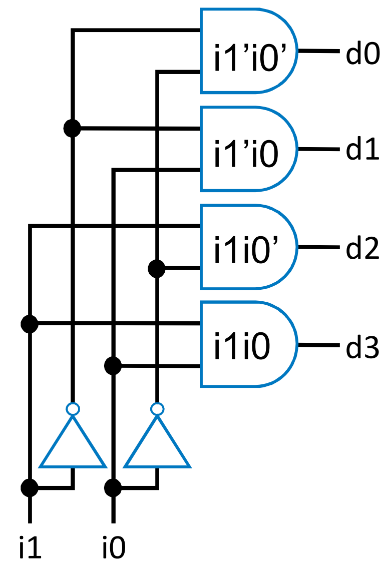


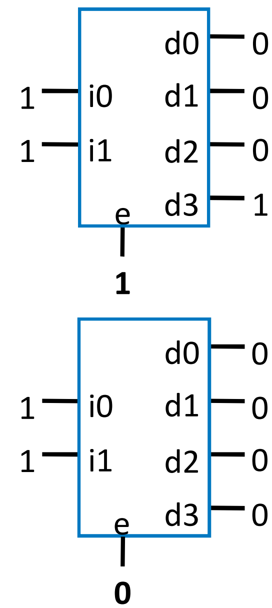
- Likewise for NOR

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

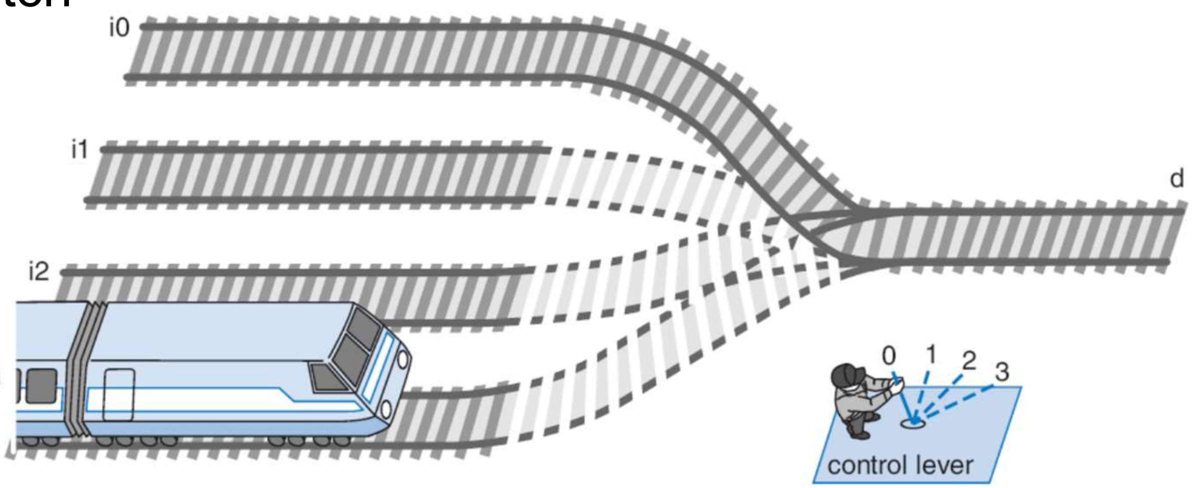




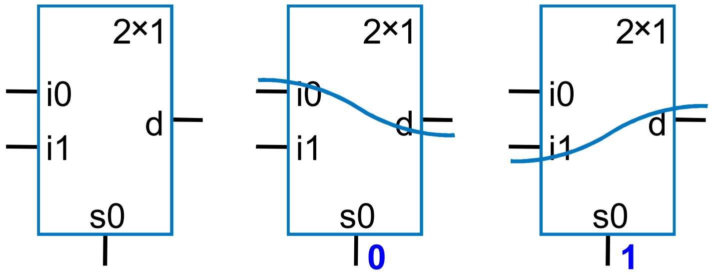


Multiplexor (Mux)

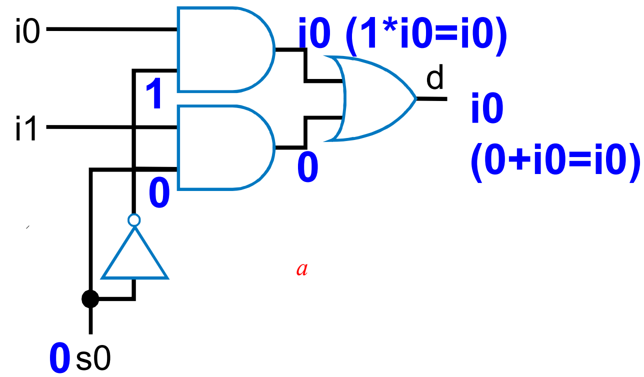
- Mux: Another popular combinational building block
  - Routes one of its N data inputs to its one output, based on binary value of select inputs
    - 4 input mux → needs 2 select inputs to indicate which input to route through
    - 8 input mux → 3 select inputs
    - N inputs → $\log_2(N)$ selects
  - Like a railyard switch

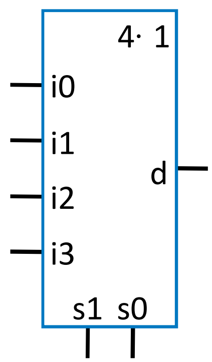


Mux Internal Design

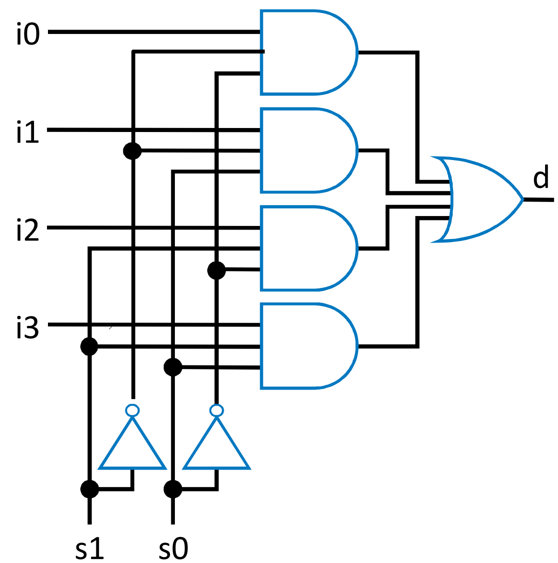


2x1 mux

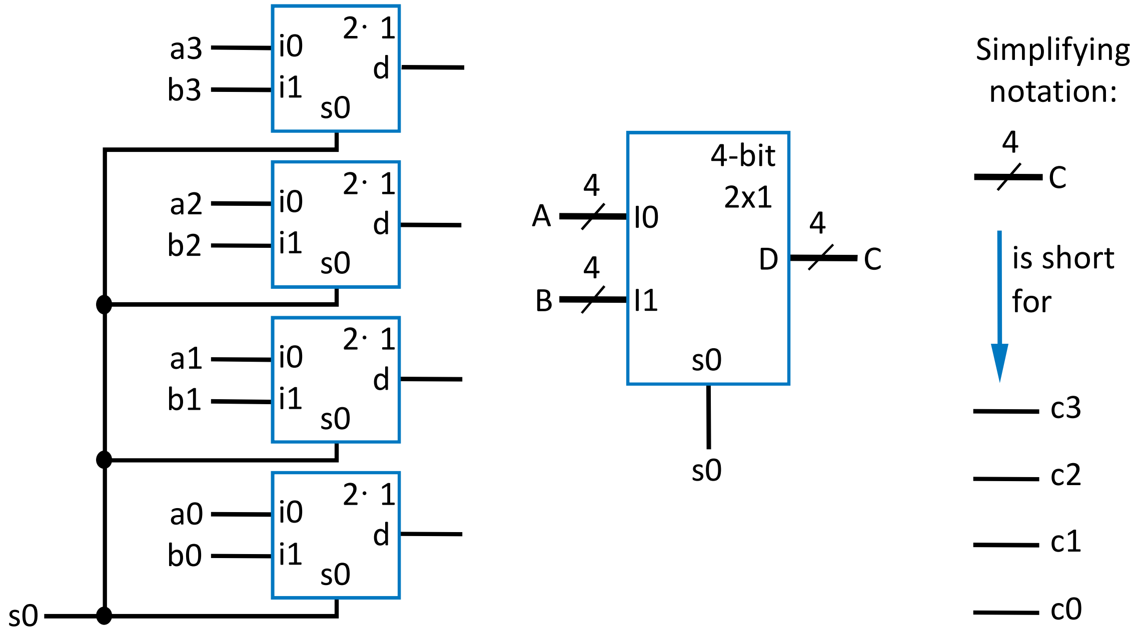




4x1 mux

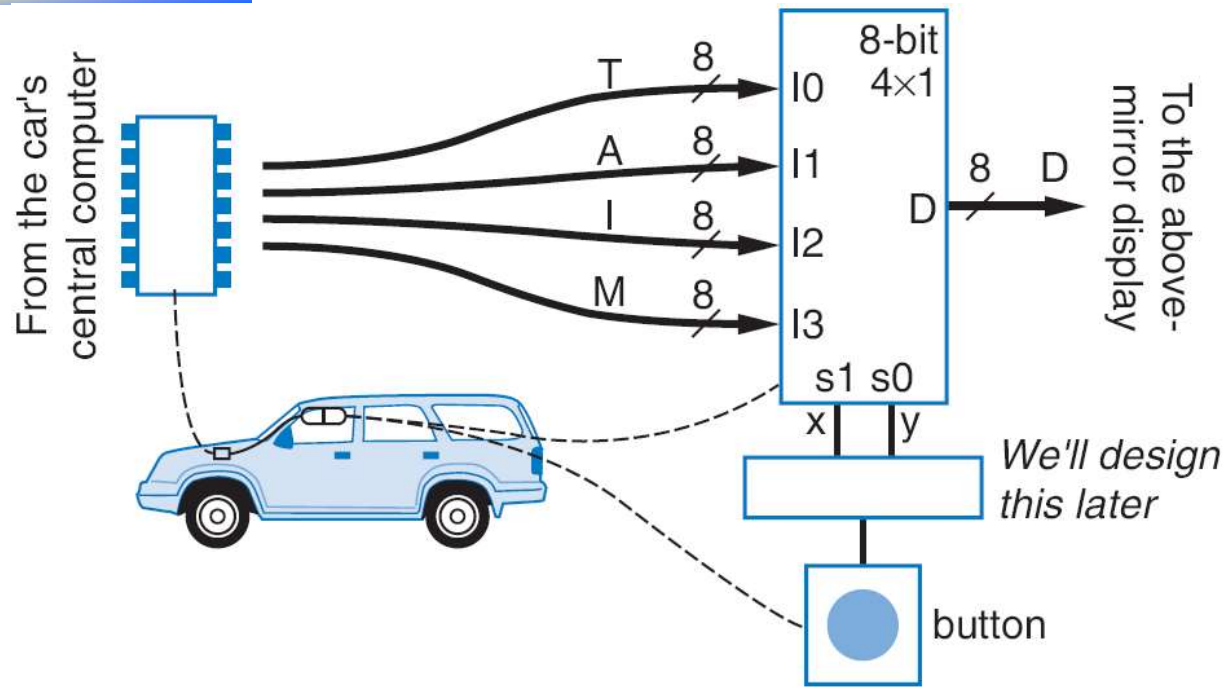


Muxes Commonly Together -- N-bit Mux



- Ex: Two 4-bit inputs, A (a3 a2 a1 a0), and B (b3 b2 b1 b0)
  - 4-bit 2x1 mux (just four 2x1 muxes sharing a select line) can select between A or B

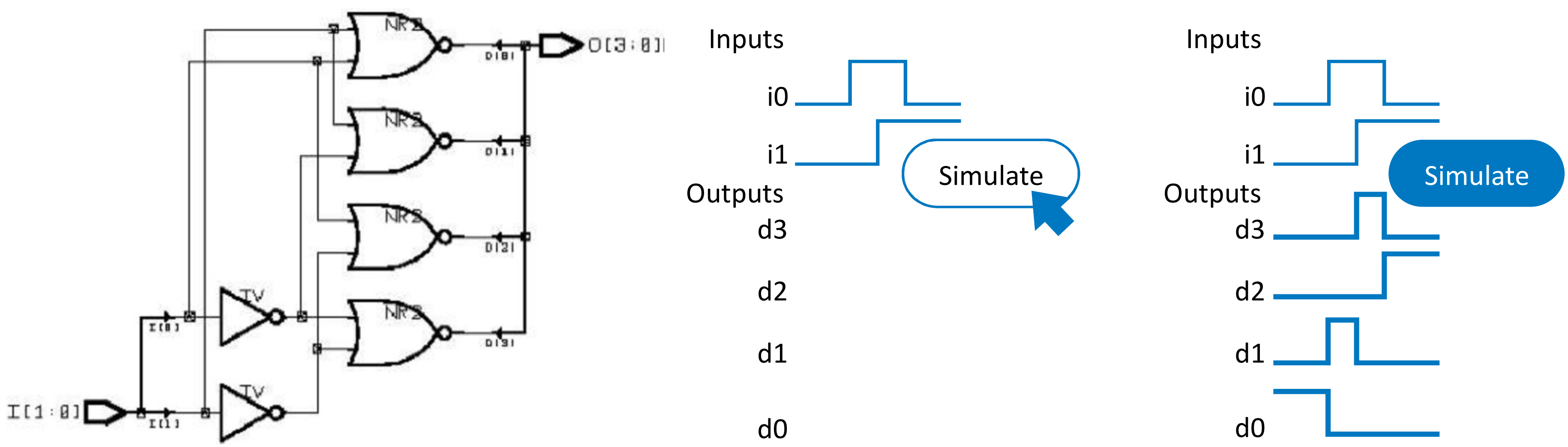
N-bit Mux Example



- Four possible display items
  - Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) -- each is 8-bits wide
  - Choose which to display using two inputs x and y
  - Use 8-bit 4x1 mux

Additional Considerations

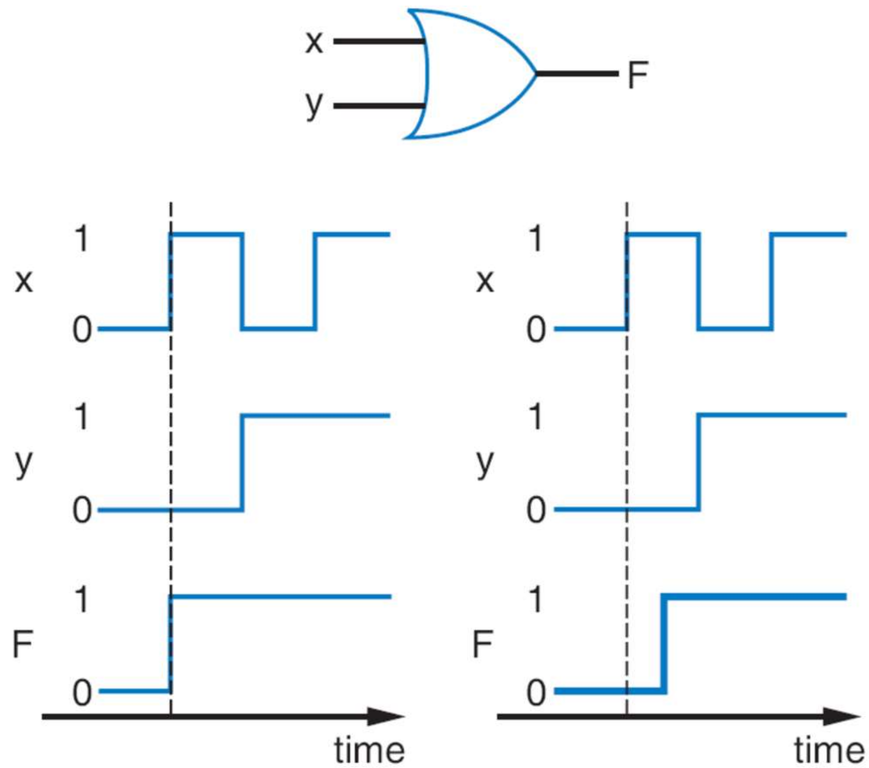
Schematic Capture and Simulation



- **Schematic capture**
  - Computer tool for user to capture logic circuit graphically
- **Simulator**
  - Computer tool to show what circuit outputs would be for given inputs
    - Outputs commonly displayed as **waveform**

Additional Considerations

Non-Ideal Gate Behavior -- Delay



- Real gates have some delay
  - Outputs don't change immediately after inputs change

Chapter Summary

- Combinational circuits
  - Circuit whose outputs are function of present inputs
    - No “state”
- Switches: Basic component in digital circuits
- Boolean logic gates: AND, OR, NOT -- Better building block than switches
  - Enables use of Boolean algebra to design circuits
- Boolean algebra: uses true/false variables/operators
- Representations of Boolean functions: Can translate among
- Combinational design process: Translate from equation (or table) to circuit through well-defined steps
- More gates: NAND, NOR, XOR, XNOR also useful
- Muxes and decoders: Additional useful combinational building blocks