CSE140: Components and Design Techniques for Digital Systems

Introduction

Prof. Tajana Simunic Rosing

Welcome to CSE 140!

- **Instructor: Tajana Simunic Rosing**
  - Email: tajana@ucsd.edu; please put CSE140 in the subject line
  - Office Hours: T 3:30-4:30pm, Th 12:45-1:45pm; CSE 2118
- **Instructor's Assistant: Sheila Manalo**
  - Email: shmanalo@ucsd.edu
  - Phone: (858) 534-8873
- **Discussion session:** F 4:00-4:50am, CENTR 119
- **TAs**: (office hrs and emails to be updated at course website shortly)

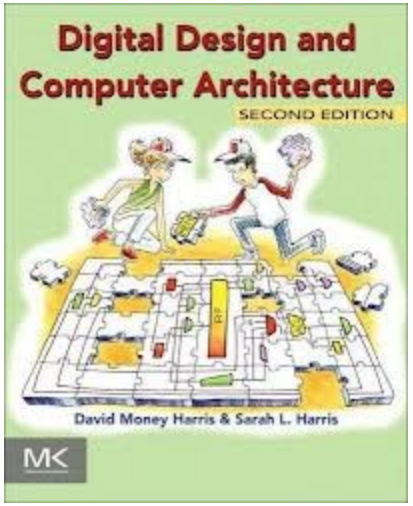
  Lu, Jingwei jlu@cs.ucsd.edu; Th 10-11am, Sunday 7-8pm
  Mast, Ryan Andrew rmast@ucsd.edu; Wed 4-5pm, B250
  Nath, Rajib Kumar rknath@ucsd.edu; Tu 11am-12pm
  Supanekar, Ketan Pranav ksupanek@eng.ucsd.edu ; Mon 7-8pm
- **Class Website:**
  - http://www.cse.ucsd.edu/classes/sp13/cse140-a/
- **Grades:** http://ted.ucsd.edu

Course Description

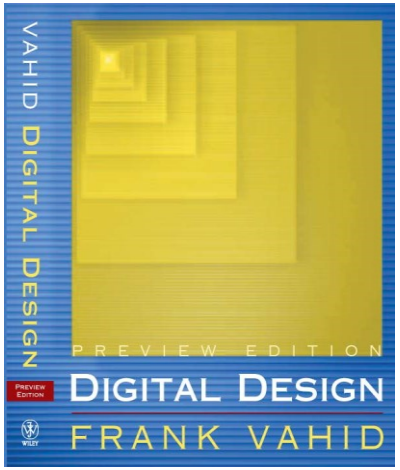
- Prerequisites:
  - CSE 20 or Math 15A, and CSE 30.
  - CSE 140L **must** be taken concurrently
- Objective:
  - Introduce digital components and system design concepts
- Grading
  - Homeworks (~7): 10%
    - HW picked up at beginning of the class, ZERO pts if late
  - Three exams: #1 – 25%; #2 – 30%; #3 – 35%
    - No makeup exams; exceptions only for:
      - documented illness (signed doctor's statement), death in the family
    - Third exam will occur at the final time, but will be the same length as the other midterms, so you will have 1hr 20min to complete it
- Regrade requests:
  - turn in a written request at the end of the class where your work (HW or exam) is returned

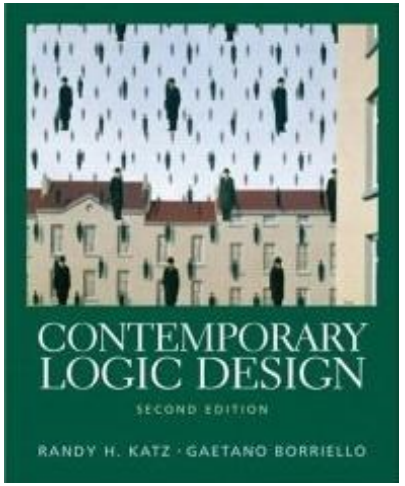
Textbook and Recommended Readings

- **Required textbook:**
  - **Digital Design & Computer Architecture**, **2nd Edition** by David & Sarah Harris



- Recommended textbook:
  - Digital Design by F. Vahid, & Contemporary Logic Design by R. Katz & G. Borriello

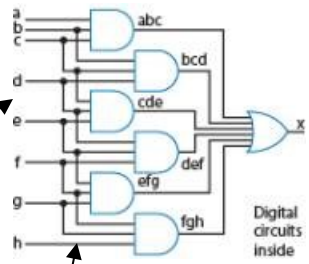


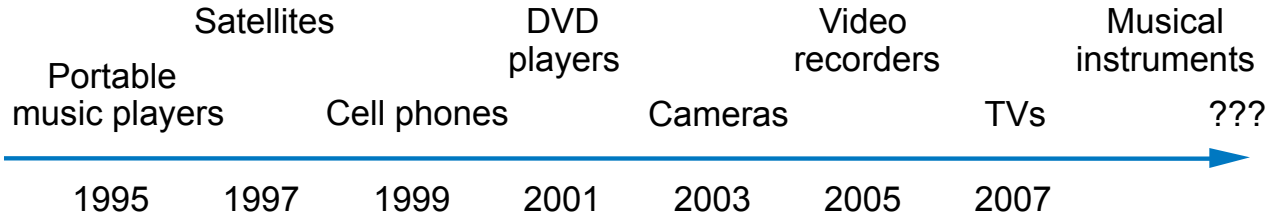


- Lecture slides are derived from the slides designed for all three books

Why Study Digital Design?

- Look “under the hood” of computers
  - Become a better programmer when aware of hardware resource issues
- Everyday devices becoming digital
  - Enables:
    - Better devices: Better sound recorders, cameras, cars, cell phones, medical devices,...
    - New devices: Video games, PDAs, ...
  - Known as “embedded systems”
    - Thousands of new devices every year
    - Designers needed: Potential career



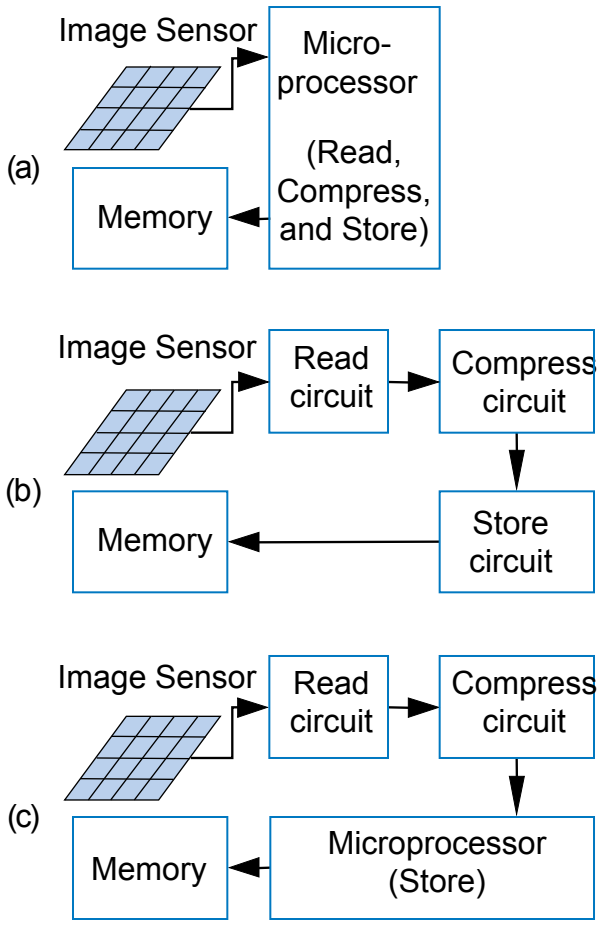


When Microprocessors Aren't Good Enough

- With microprocessors so easy to work with, cheap, and available, why design a digital circuit?
  - Microprocessor may be too slow
  - Or too big, power hungry, or costly

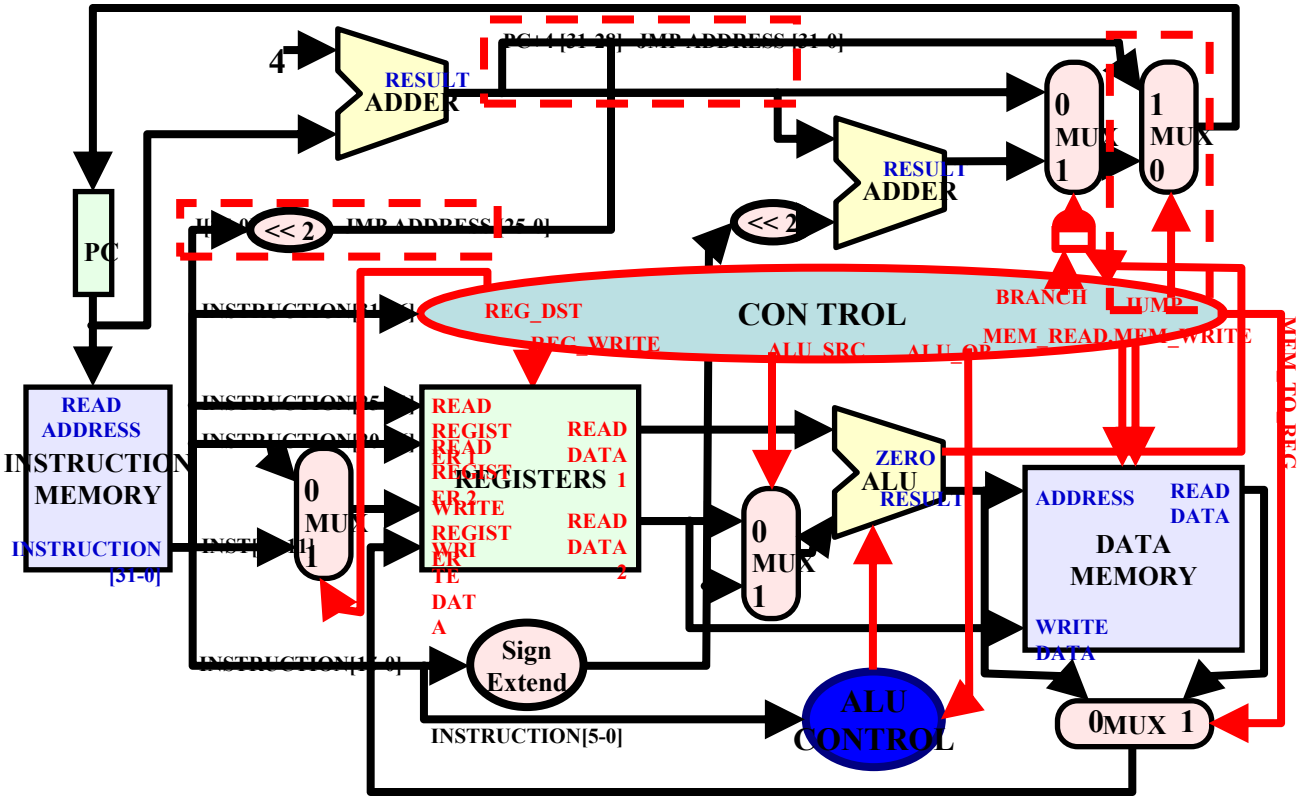
Sample digital camera task execution times (in seconds) on a microprocessor versus a digital circuit:

Task	Microprocessor	Custom Digital Circuit
Read	5	0.1
Compress	8	0.5
Store	1	0.8



The big picture

- We start with Boolean algebra Y = A and B
- We end with a hardware design of a simple CPU



- What's next? CSE141 – more complex CPU architecture

Outline

- Number representations
  - Analog vs. Digital
  - Digital representations:
    - Binary, Hexadecimal, Octal
  - Binary addition, subtraction, multiplication, division
- Boolean algebra
  - Properties
  - How Boolean algebra can be used to design logic circuits
- Switches, MOS transistors, Logic gates
  - What is a switch
  - How a transistor operates
  - Building logic gates out of transistors
  - Building larger functions from logic gates

➢ Textbook chapter 1

CSE140: Components and Design Techniques for Digital Systems

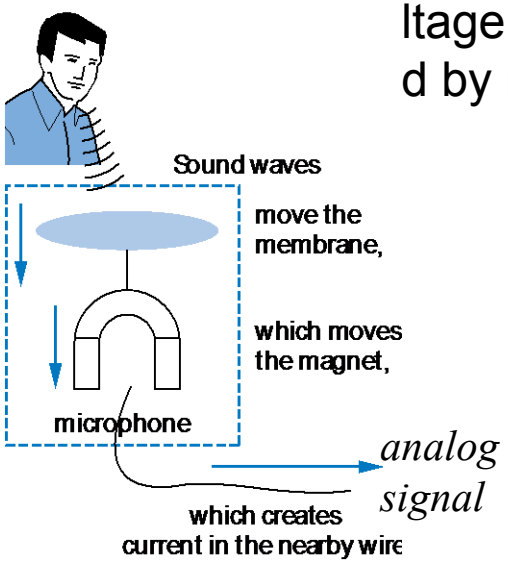
Number representations & Binary arithmetic

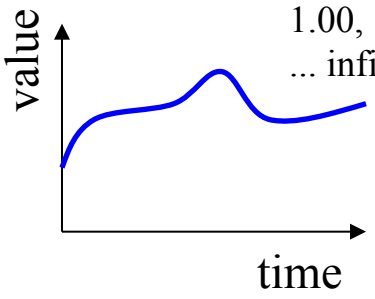
Tajana Simunic Rosing

What Does "Digital" Mean?

- Analog signal
  - Infinite possible values

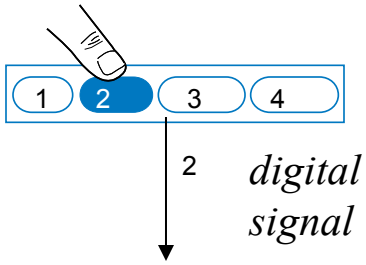
ltage on a wire
d by microphone

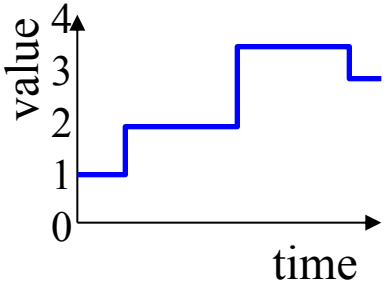




Possible values:
1.00, 1.01, 2.0000009,
... infinite possibilities

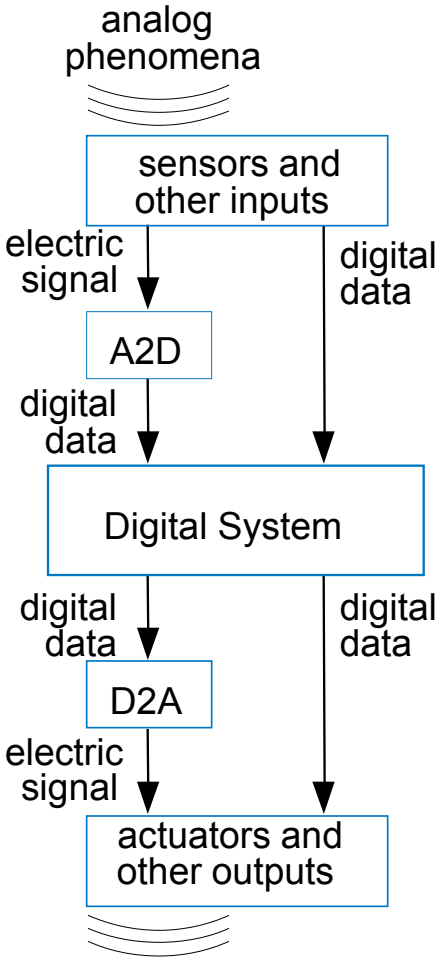
- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad



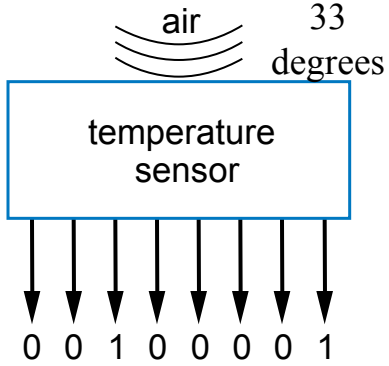


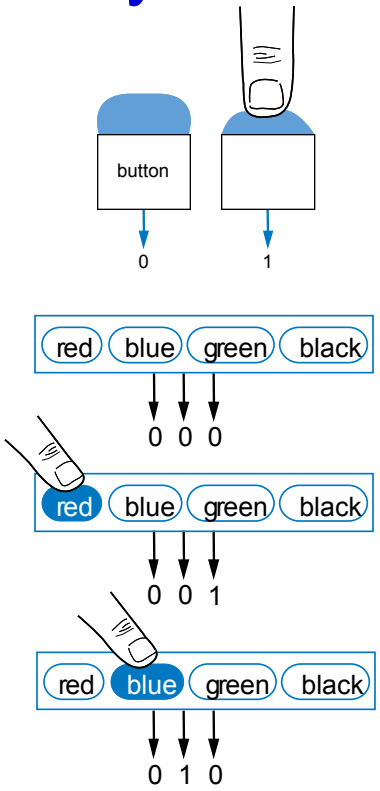
Possible values:
0, 1, 2, 3, or 4.
That's it.

How Do We Encode Data into Binary?



- Some inputs are inherently binary
  - Button: not pressed (0), pressed (1)
- Some inputs are inherently digital
  - Just need encoding into binary
  - e.g., multi-button input: encode red=001, blue=010, ...
- Other inputs are analog
  - Need analog-to-digital conversion





Binary digIT = BIT

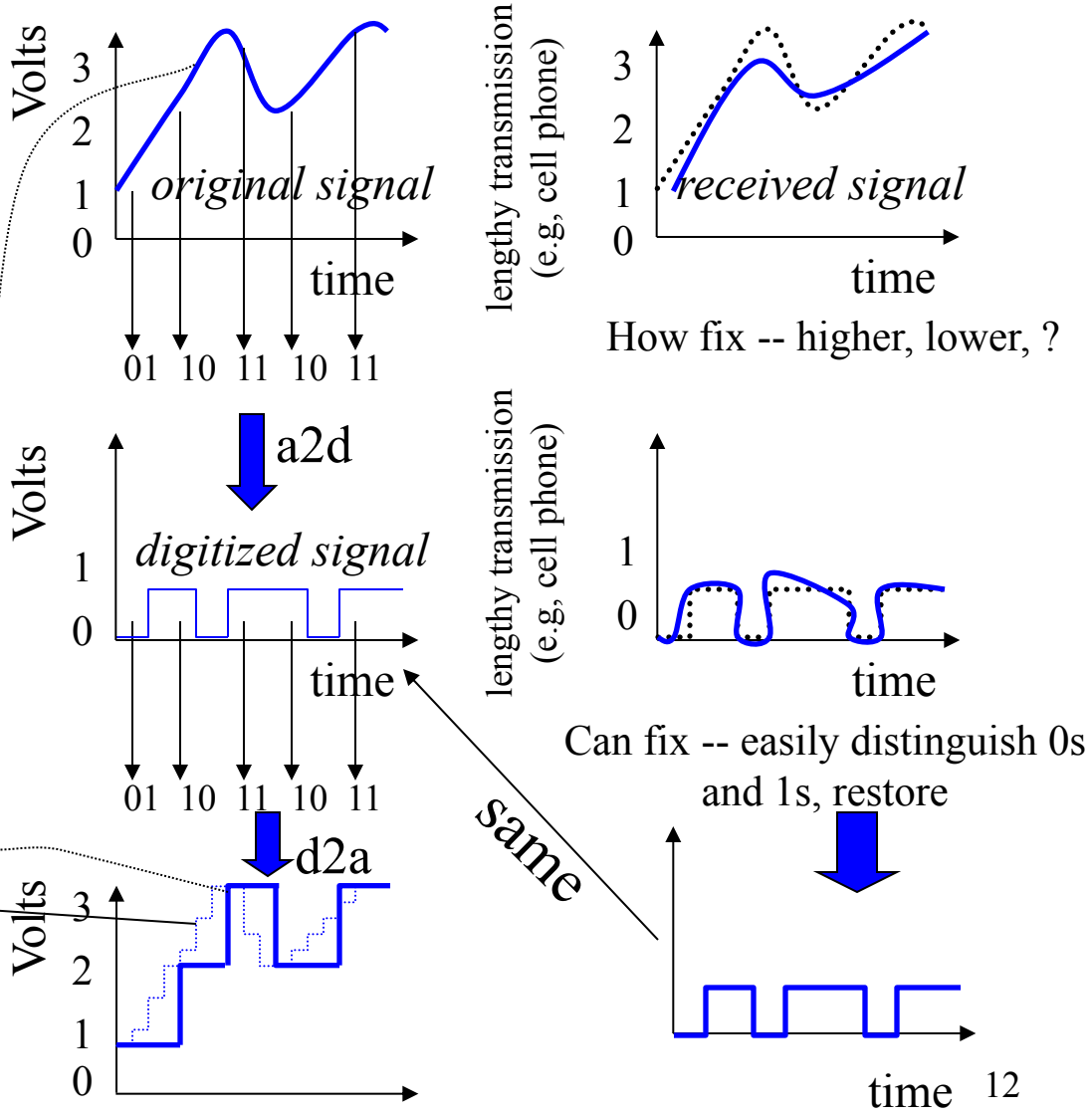
Has 2 values: 0 & 1

A/D conversion & digitization benefits

- Analog signal (e.g., audio) may lose quality
  - Voltage levels not saved/copied/transmitted perfectly
- Digitized version enables near-perfect save/cpy/trn.
  - "Sample" voltage at particular rate, save sample using bit encoding
  - Voltage levels still not kept perfectly
  - But we can distinguish 0s from 1s

Let bit encoding be:
1 V: "01"
2 V: "10"
3 V: "11"

Digitized signal not perfect re-creation, but higher sampling rate and more bits per encoding brings closer.



Encoding Text: ASCII, Unicode

- ASCII: 7- (or 8-) bit encoding of each letter, number, or symbol
- Unicode: Increasingly popular 16-bit bit encoding
  - Encodes characters from various world languages

Symbol	Encoding
R	1010010
S	1010011
T	1010100
L	1001100
N	1001110
E	1000101
0	0110000
.	0101110
<tab>	0001001

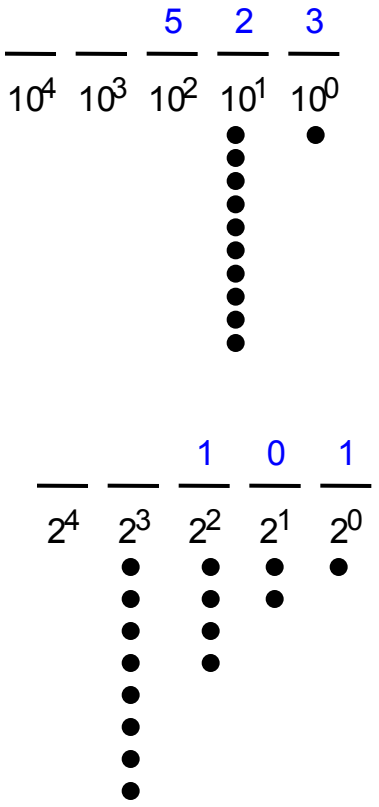
Symbol	Encoding
r	1110010
s	1110011
t	1110100
l	1101100
n	1101110
e	1100101
9	0111001
!	0100001
<space>	0100000

What does this ASCII bit sequence represent?

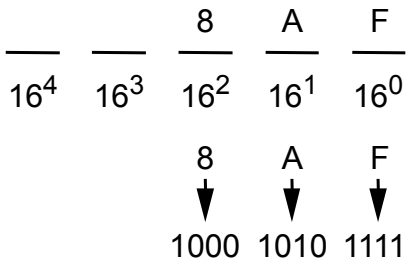
1010010 1000101 1010011 1010100

Encoding Numbers

- Each position represents a quantity; symbol in position means how many of that quantity
  - Base ten (***decimal***)
    - Ten symbols: 0, 1, 2, ..., 8, and 9
    - More than 9 -- next position
      - So each position power of 10
    - Nothing special about base 10 -- used because we have 10 fingers
  - Base two (***binary***)
    - Two symbols: 0 and 1
    - More than 1 -- next position
      - So each position power of 2



Bases Sixteen & Eight



hex	binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

hex	binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

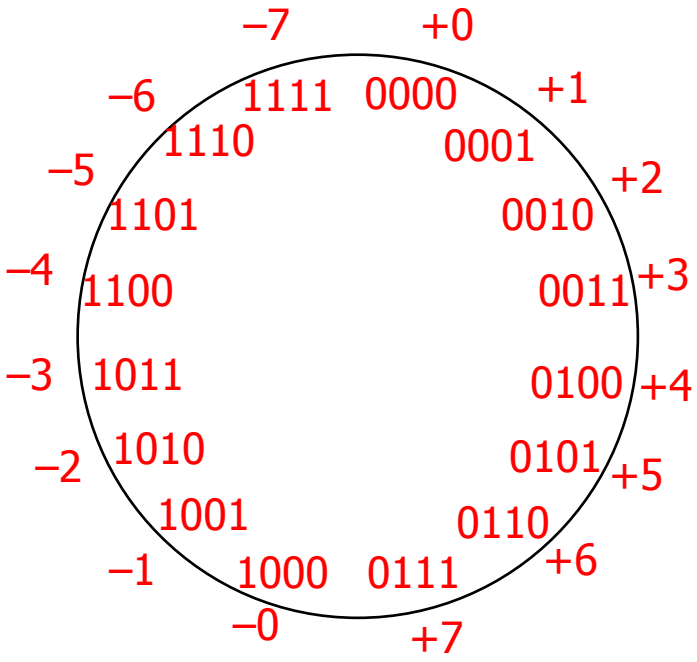
- Base sixteen
  - nice because each position represents four base two positions
  - Used as compact means to write binary numbers
  - Basic digits: 0-9, A-F
  - Known as ***hexadecimal***, or just ***hex***
- Base eight
  - Used in some digital designs
  - Each position represents three base two positions
  - Basic digits: 0-7

Write 11110000 in hex

Write 11110000 in octal

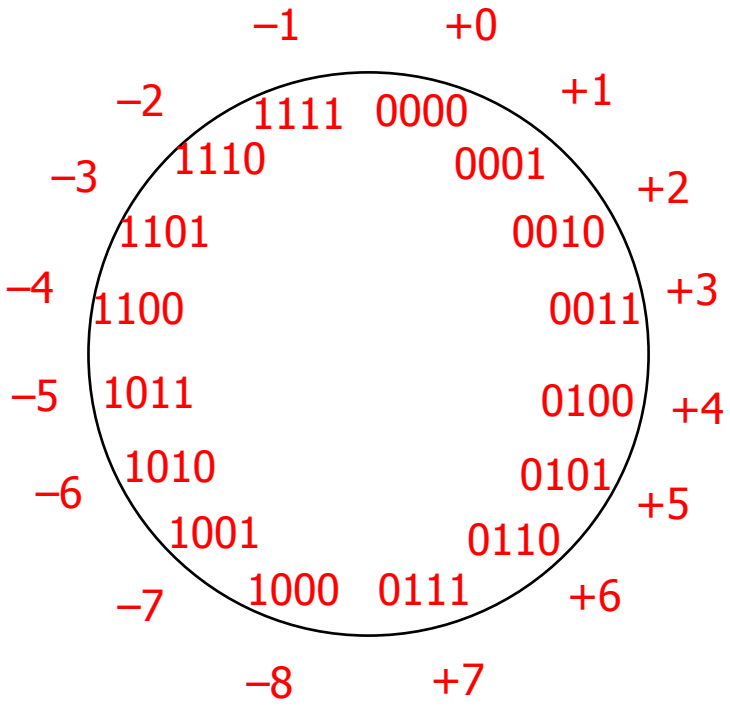
Sign and magnitude

- One bit dedicate to sign (positive or negative)
  - sign: 0 = positive (or zero), 1 = negative
- Rest represent the absolute value or magnitude
  - three low order bits: 0 (000) thru 7 (111)
- Range for n bits
  - +/– 2n–1 –1 (two representations for 0)
- Cumbersome addition/subtraction
  - must compare magnitudes
    to determine the sign of the result

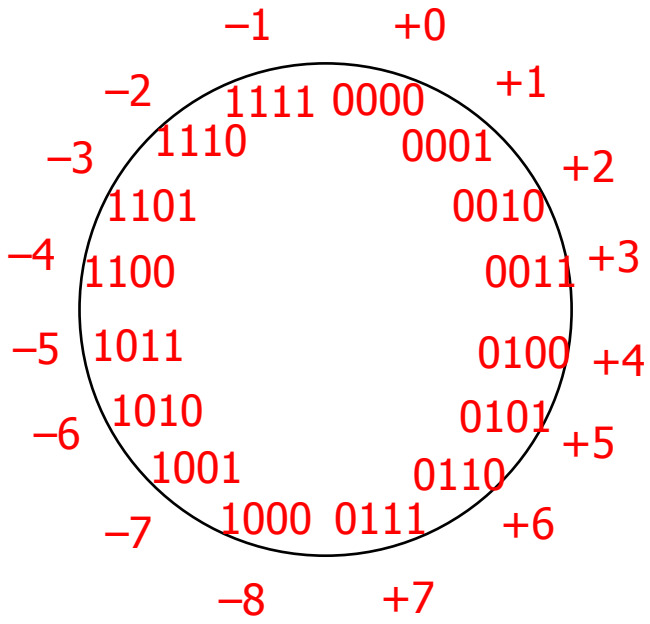


2s complement

- If N is a positive number, then the negative of N (its 2s complement or N*) is bit-wise complement plus 1
  - 7* is -7 : 0111 -> 1000 + 1 = 1001 (-7)
  - -7* is 7: 1001 -> 0110 + 1 = 0111 (7)

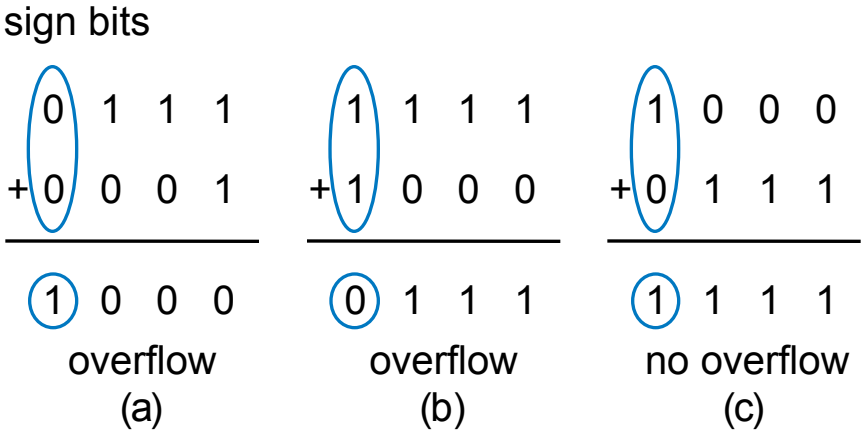


2s complement addition and subtraction



Detecting Overflow: Method 1

- Assuming 4-bit two's complement numbers, one can detect overflow by detecting when the two numbers' sign bits are the same but are different from the result's sign bit
  - If the two numbers' sign bits are different, overflow is impossible
    - Adding a positive and negative can't exceed the largest magnitude positive or negative
- Simple circuit
  - overflow = a3'b3's3 + a3b3s3'



If the numbers' sign bits have the same value, which differs from the result's sign bit, overflow has occurred.

Detecting Overflow: Method 2

- Even simpler method: Detect difference between carry-in to sign bit and carry-out from sign bit
- Yields a simpler circuit: overflow = c3 xor c4 = c3 c4' + c3' c4

1 1 1	0 0 0	0 0 0
0 1 1 1	1 1 1 1	1 0 0 0
+ 0 0 0 1	+ 1 0 0 0	+ 0 1 1 1
0 1 0 0 0	1 0 1 1 1	0 1 1 1 1
overflow	overflow	no overflow
(a)	(b)	(c)

If the carry into the sign bit column differs from the carry out of that column, overflow has occurred.

Multiplication of positive binary numbers

- Generalized representation of multiplication by hand

```
                    a3    a2    a1    a0
                  x b3    b2    b1    b0
-------------------------------------------
                    b0a3  b0a2  b0a1  b0a0    (pp1)
              b1a3  b1a2  b1a1  b1a0    0     (pp2)
        b2a3  b2a2  b2a1  b2a0    0     0     (pp3)
+ b3a3  b3a2  b3a1  b3a0    0     0     0     (pp4)
-------------------------------------------
p7  p6    p5    p4    p3    p2    p1    p0
```

Division of positive binary numbers

- Repeated subtraction
  - Set quotient to 0
  - Repeat while dividend >= divisor
    - Subtract divisor from dividend
    - Add 1 to quotient
  - When dividend < divisor:
    - Reminder = dividend
    - Quotient is correct

Example:

- Dividend: 101; Divisor: 10

Dividend		Quotient	
101	-	0	+
10		1	
11	-	1	+
10		1	
1		10	

Summary of number representation

- Conversion between basis
  - Decimal
  - Binary
  - Octal
  - Hex
- Addition & subtraction in binary
  - Overflow detection
- Multiplication
  - Partial products
    - For demo see: http://courses.cs.vt.edu/~cs1104/BuildingBlocks/multiply.010.html
- Division
  - Repeated subtraction
    - For demo see: http://courses.cs.vt.edu/~cs1104/BuildingBlocks/Binary.Divide.html

CSE140: Components and Design Techniques for Digital Systems

Boolean algebra

Tajana Simunic Rosing

Boolean algebra

- B = {0, 1}
- Variables represent 0 or 1 only
- Operators return 0 or 1 only
- Basic operators
  - • is logical AND: *a AND b* returns 1 only when both a=1 and b=1
  - + is logical OR: *a OR b* returns 1 if either (or both) a=1 or b=1
  - ’ is logical NOT: *NOT a* returns the opposite of a (1 if a=0, 0 if a=1)

AND

a • b

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

OR

a+b

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

NOT

a’

a	NOT
0	1
1	0

BUF

a

a	BUF
0	
1	

- Derived operators:

NAND

a	b	NAND
0	0	
0	1	
1	0	
1	1	

NOR

a	b	NOR
0	0	
0	1	
1	0	
1	1	

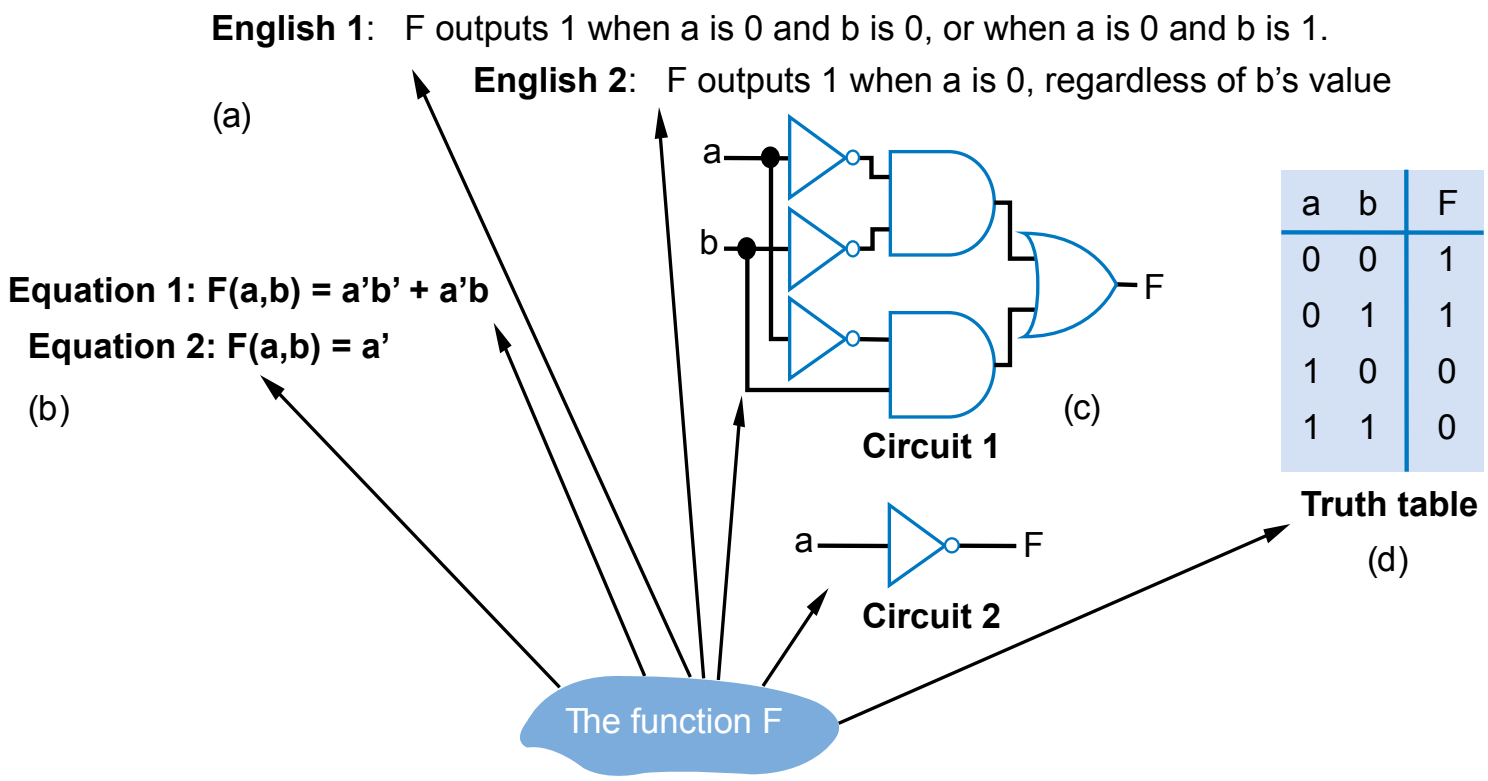
XOR

a	b	XOR
0	0	
0	1	
1	0	
1	1	

XNOR

a	b	XOR
0	0	
0	1	
1	0	
1	1	

Representations of Boolean Functions



Examples: Converting to Boolean Functions

- Convert the following English statements to a function
  - Q1. answer is 1 if a is 1 and b is 1.
    - Answer: F =
  - Q2. answer is 1 if either of a or b is 1.
    - Answer: F =
  - Q3. answer is 1 if both a and b are not 0.
    - Answer: F=
  - Q4. answer is 1 if a is 1 and b is 0.
    - Answer: F =

Example: Convert equation to logic gates

- More than one way to map expressions to gates
  e.g., $Z = A' \bullet B' \bullet (C + D) = (A' \bullet (B' \bullet (C + D)))$

Boolean Duality

- Derived by replacing • by +, + by •, 0 by 1, and 1 by 0 & leaving variables unchanged

$$X + Y + \ldots \Leftrightarrow X \bullet Y \bullet \ldots$$

- Generalized duality:

$$f(X_1, X_2, \ldots, X_n, 0, 1, +, \bullet) \Leftrightarrow f(X_1, X_2, \ldots, X_n, 1, 0, \bullet, +)$$

- Any theorem that can be proven is also proven for its dual! Note: this is NOT deMorgan's Law

Boolean Axioms & Theorems

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

Boolean theorems of multiple variables

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + B \bullet D = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

Proving theorems

- Using the axioms of Boolean algebra (or a truth table):
  - e.g., prove the theorem: $X \cdot Y + X \cdot Y' = X$

distributivity	$X \cdot Y + X \cdot Y'$	=	$X \cdot (Y + Y')$
complementarity	$X \cdot (Y + Y')$	=	$X \cdot (1)$
identity	$X \cdot (1)$	=	X ✓

  - e.g., prove the theorem: $X + X \cdot Y = X$

identity	$X + X \cdot Y$	=	$X \cdot 1 + X \cdot Y$
distributivity	$X \cdot 1 + X \cdot Y$	=	$X \cdot (1 + Y)$
identity	$X \cdot (1 + Y)$	=	$X \cdot (1)$
identity	$X \cdot (1)$	=	X ✓

Proving theorems example

- Prove the following using the laws of Boolean algebra:
  - $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$

	$(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z)$
identity	$(X \bullet Y) + (1) \bullet (Y \bullet Z) + (X' \bullet Z)$
complementarity	$(X \bullet Y) + (X' + X) \bullet (Y \bullet Z) + (X' \bullet Z)$
distributivity	$(X \bullet Y) + (X' \bullet Y \bullet Z) + (X \bullet Y \bullet Z) + (X' \bullet Z)$
commutativity	$(X \bullet Y) + (X \bullet Y \bullet Z) + (X' \bullet Y \bullet Z) + (X' \bullet Z)$
factoring	$(X \bullet Y) \bullet (1 + Z) + (X' \bullet Z) \bullet (1 + Y)$
null	$(X \bullet Y) \bullet (1) + (X' \bullet Z) \bullet (1)$
identity	$(X \bullet Y) + (X' \bullet Z)$ ✓

Proving theorems (perfect induction)

- Using perfect induction (complete truth table):
  - e.g., de Morgan's:

$(X + Y)' = X' \bullet Y'$

NOR is equivalent to AND with inputs complemented

X	Y	X′	Y′	(X + Y)′	X′ • Y′
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

$(X \bullet Y)' = X' + Y'$

NAND is equivalent to OR with inputs complemented

X	Y	X′	Y′	(X • Y)′	X′ + Y′
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

Completeness of NAND

- Any logic function can be implemented *using just NAND gates.* Why?
  - Boolean algebra: need AND, OR and NOT

Implement using only NAND

- F = X'Y + Z

Completeness of NOR

- Any logic function can be implemented *using just NOR gates.* Boolean algebra needs AND, OR and NOT

Implement using only NOR

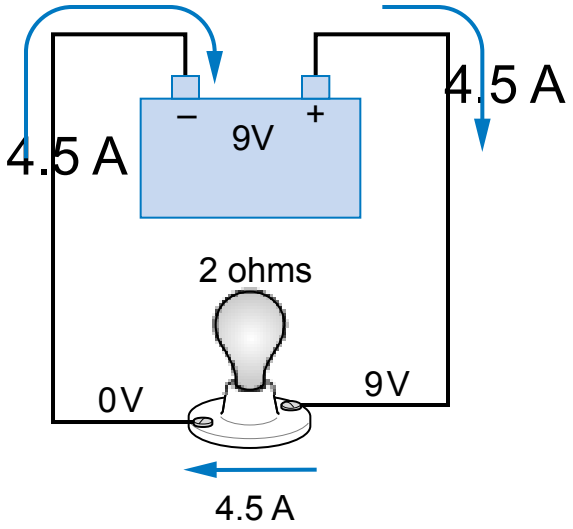
- F = X'Y + Z

Combinational circuit building blocks: Transistors, gates and timing

Tajana Simunic Rosing

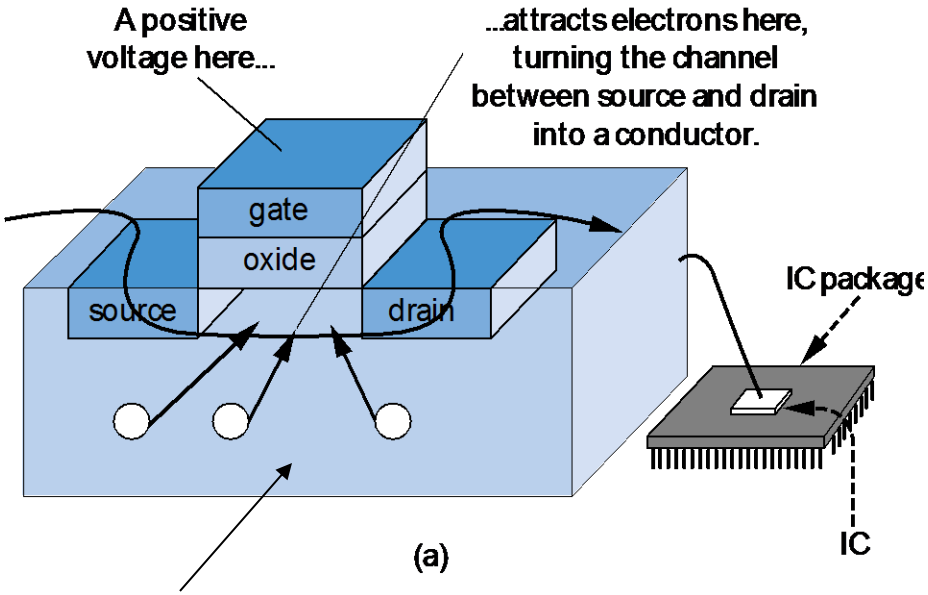
Switches

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

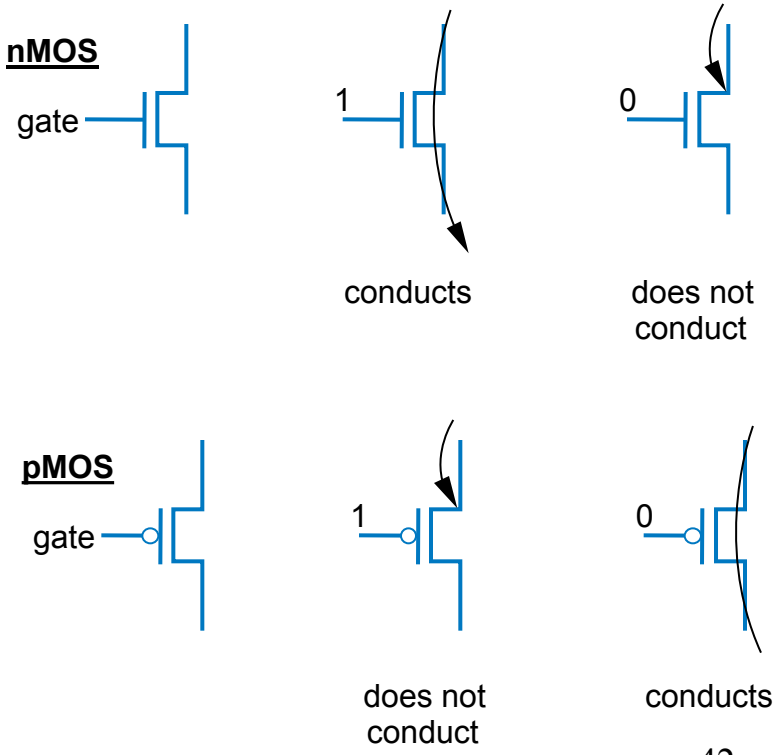


The CMOS Switches

- CMOS circuit
  - Consists of N and PMOS transistors
  - Both N and PMOS are similar to basic switches
  - Rp ~ 2 Rn => PMOS in series is much slower than NMOS

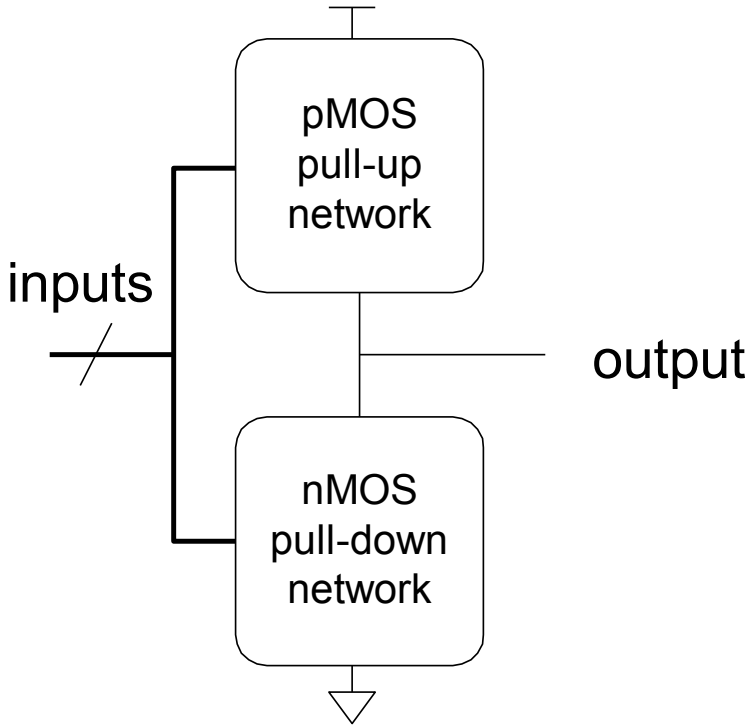


Silicon -- not quite a conductor or insulator:
Semiconductor



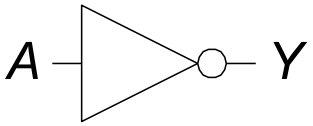
Transistor Circuit Design

- **nMOS:** pass 0's well, so connect source to GND
- **pMOS:** pass 1's well, so connect source to V_{DD}



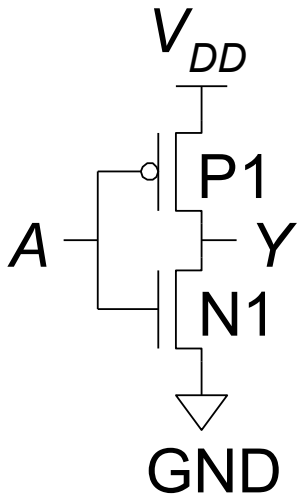
CMOS Gates: NOT Gate

NOT



$Y = \overline{A}$

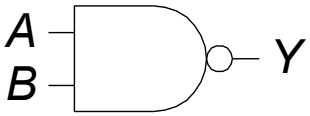
A	Y
0	1
1	0



A	P1	N1	*Y*
0			
1			

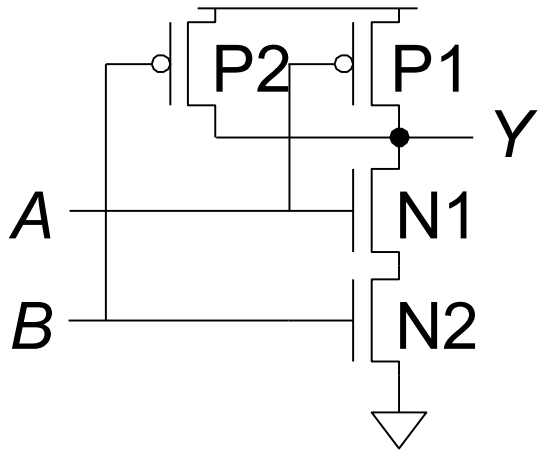
CMOS Gates: NAND Gate

NAND



$Y = \overline{AB}$

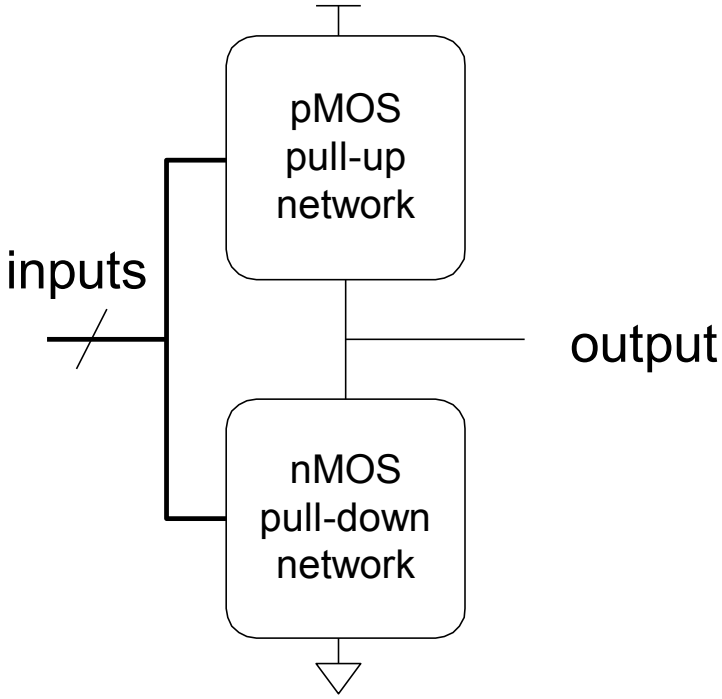
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



A	*B*	P1	P2	N1	N2	*Y*
0	0					
0	1					
1	0					
1	1					

Three input NOR gate

CMOS gate structure:

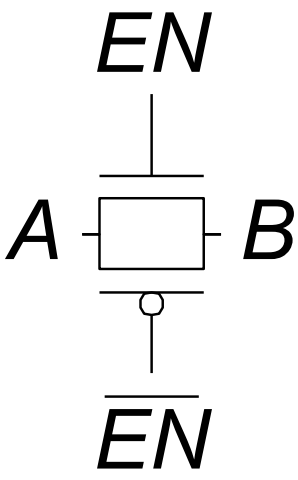


Three-input NOR

Building a two-input AND gate

Transmission Gates

- nMOS pass 1's poorly
- pMOS pass 0's poorly
- Transmission gate is a better switch
  - passes both 0 and 1 well
- When *EN* = 1, the switch is ON:
  - *EN* = 0 and *A* is connected to *B*
- When *EN* = 0, the switch is OFF:
  - *A* is not connected to *B*



How to make CMOS gates

- Reducing Logic Functions
  - fewest operations ⇒ fewest txs
  - minimized function to eliminate txs
  - Example: x y + x z + x v = x (y + z + v)

    5 operations: 3 AND, 2 OR — # txs =

    3 operations: 1 AND, 2 OR — # txs =

- Suggested approach to implement a CMOS logic function
  - create nMOS network
    - invert output
    - reduce function, use DeMorgan to eliminate NANDs and NORs
    - implement using **series for AND** and **parallel for OR**
  - create pMOS network
    - complement each operation in nMOS network

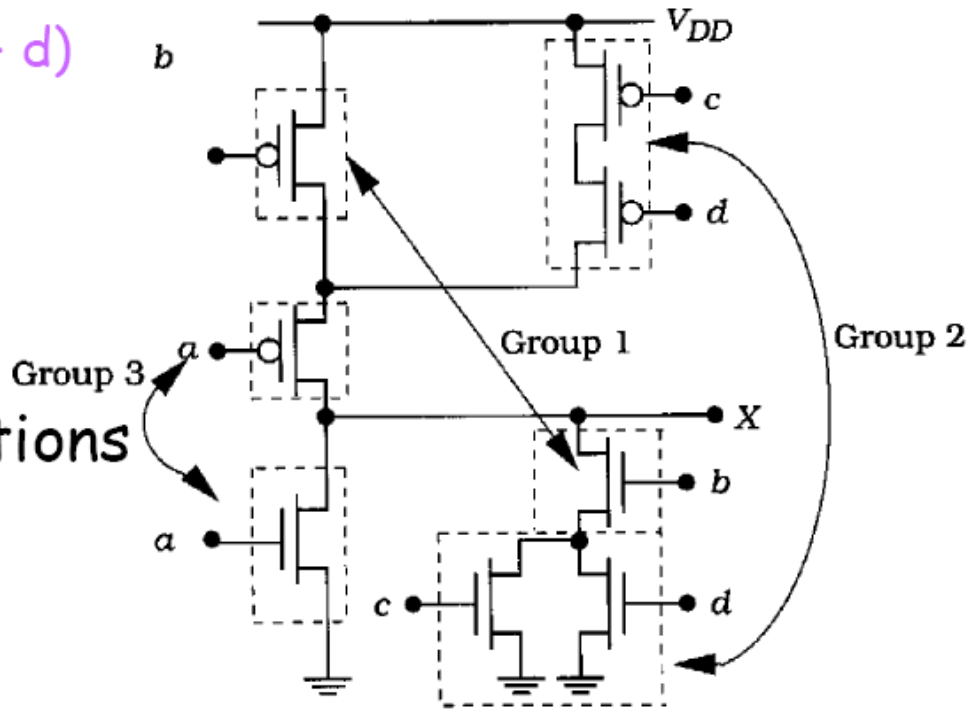
CMOS Example

- Construct the function below in CMOS

  $F = \overline{a + b \cdot (c + d)}$; remember AND operations occur before OR

- Step 1, invert output and find nMOS
  - nMOS; implement $a + b \cdot (c + d)$
  - Group 1: c & d in parallel
  - Group 2: b in series with G1
  - Group 3: a parallel to G2

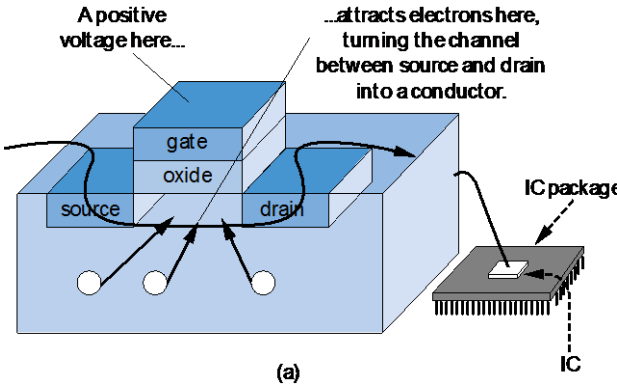
- Step 2, complement operations
  - pMOS
  - Group 1: c & d in series
  - Group 2: b parallel to G1
  - Group 3: a in series with G2



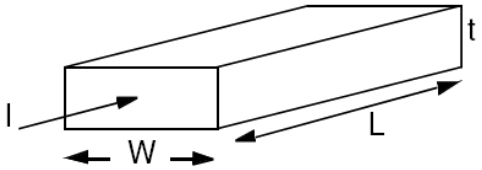
A CMOS design example

- Implement F and F' using CMOS: F=A*(B+C)

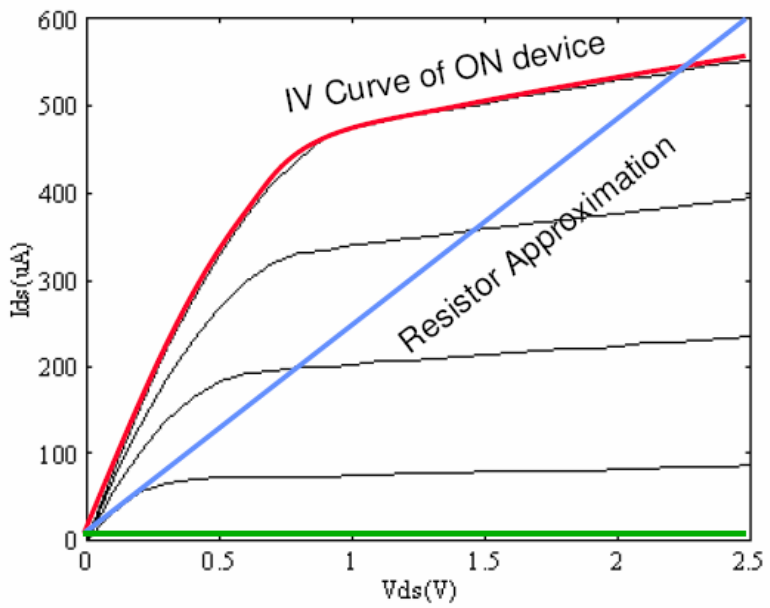
CMOS delay: resistance

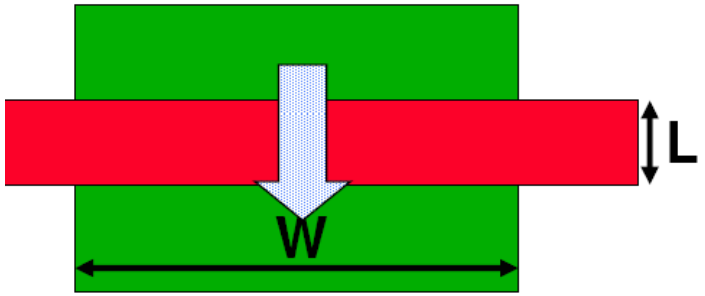


- Resistivity
  - Function of:
    - resistivity r, thickness t : defined by technology
    - Width W, length L: defined by designer
  - Approximate ON transistor with a resistor
    - R = r' L/W
    - L is usually minimum; change only W



$$R = \frac{\rho L}{tW} = \frac{\rho}{t}\frac{L}{W}$$

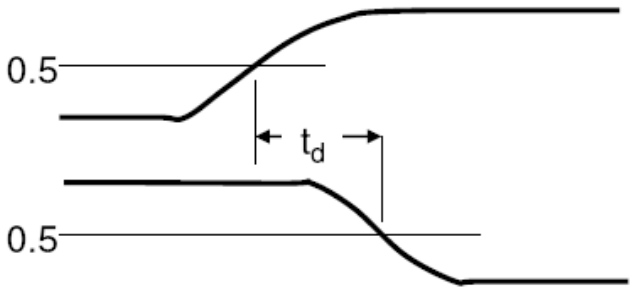


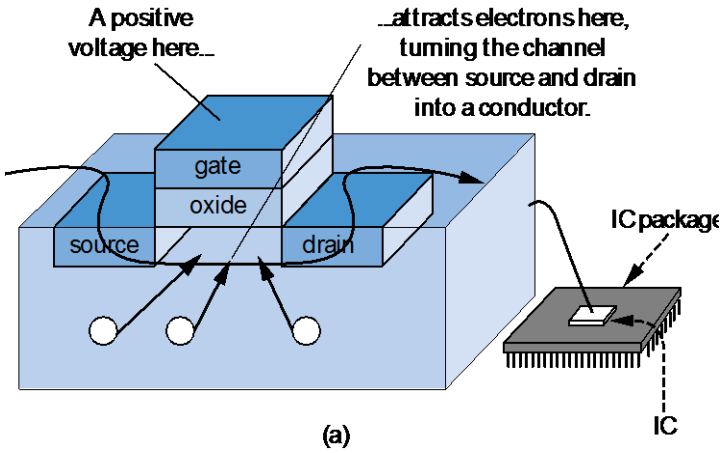


Source: Prof. Subhashish Mitra

CMOS delay: capacitance & timing estimates

- Capacitor
  - Stores charge Q = C V (capacitance C; voltage V)
  - Current: dQ/dt = C dV/dt
- Timing estimate
  - D t = C dV/ i = C dV / (V/R_{trans}) = R_{trans}C dV/V
- Delay: time to go from 50% to 50% of waveform

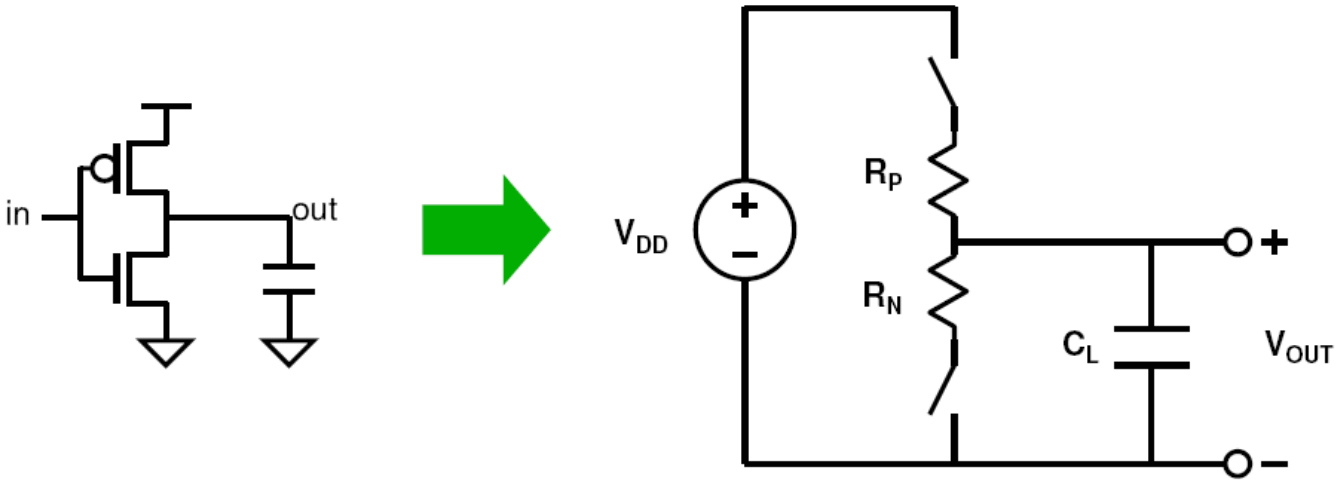




Source: Prof. Subhashish Mitra

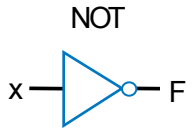
Charge/discharge in CMOS

- Calculate on resistance
- Calculate capacitance of the gates circuit is driving
- Get RC delay & use it as an estimate of circuit delay
  - $V_{out} = V_{dd} (1- e^{-t/RpC})$
- Rp ~ 2Rn

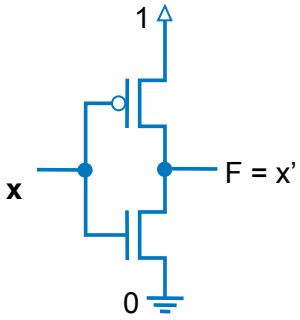


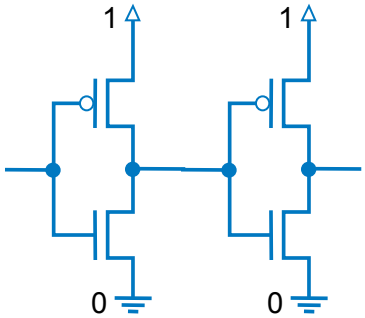
Source: Prof. Subhashish Mitra

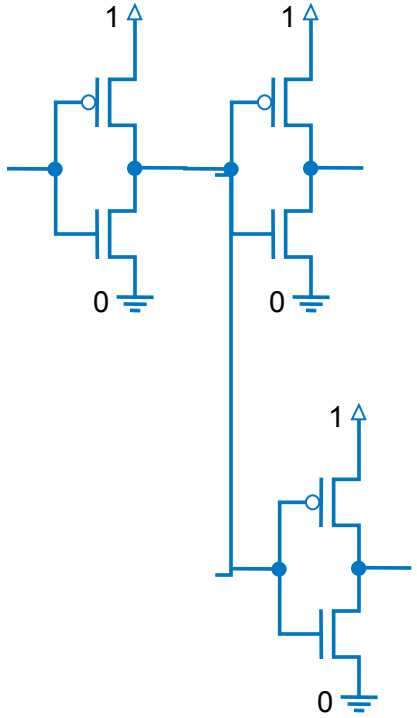
Timing analysis: Inverter



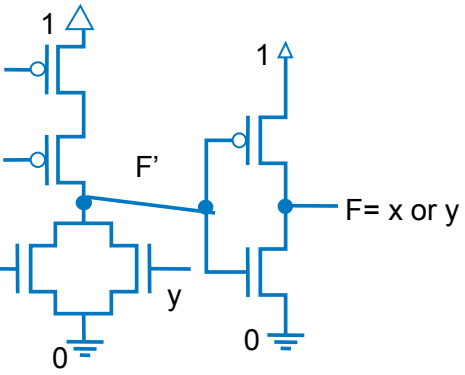
x	F
0	1
1	0



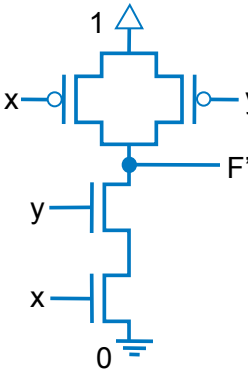




Timing analysis in gates



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



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

Power consumption in CMOS

- **Power = Energy consumed per unit time**
  - **Dynamic** power consumption
  - **Static** power consumption
- **Dynamic power consumption:**
  - **Power to charge transistor gate capacitances**
  - Energy required to charge a capacitance, C, to V_{DD} is CV_{DD}^2
  - Circuit running at frequency f: transistors switch (from 1 to 0 or vice versa) at that frequency
  - Capacitor is charged $f/2$ times per second (discharging from 1 to 0 is free)

$$P_{dynamic} = \tfrac{1}{2} C V_{DD}^2 f$$

- ***Static power consumption***
  - Power consumed when no gates are switching
  - Caused by the *leakage supply current*, I_{DD} :

$$P_{static} = I_{DD} V_{DD}$$

Power estimate example

- Estimate the power consumption of a tablet PC
  - V_{DD} = 1.2 V
  - C = 20 nF
  - f = 1 GHz
  - I_{DD} = 20 mA

$$P = \tfrac{1}{2}CV_{DD}^2 f + I_{DD}V_{DD}$$

$$= \tfrac{1}{2}(20\text{ nF})(1.2\text{ V})^2(1\text{ GHz}) + (20\text{ mA})(1.2\text{ V})$$

$$\mathbf{= 14.4\text{ W}}$$

Summary

- What we covered thus far:
  - Number representations
  - Boolean algebra
  - Switches, Logic gates
  - How to build logic gates from CMOS transistors
  - Timing and power estimates
- What is next:
  - Combinatorial logic:
    - Minimization
    - Implementations