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 ksuppenek@eng.ucsd.edu; Mon 7-8pm

- **Class Website:**
  - [http://www.cse.ucsd.edu/classes/sp13/cse140-a/](http://www.cse.ucsd.edu/classes/sp13/cse140-a/)

- **Grades:** [http://ted.ucsd.edu](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:

• 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

<table>
<thead>
<tr>
<th>Year</th>
<th>Satellites</th>
<th>DVDs</th>
<th>Video recorders</th>
<th>Musical instruments</th>
</tr>
</thead>
</table>
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:

<table>
<thead>
<tr>
<th>Task</th>
<th>Microprocessor</th>
<th>Custom Digital Circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>5</td>
<td>0.1</td>
</tr>
<tr>
<td>Compress</td>
<td>8</td>
<td>0.5</td>
</tr>
<tr>
<td>Store</td>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>
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
  - Example: voltage on a wire created by microphone

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

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

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
  - Increasingly popular 16-bit bit encoding
- **Unicode**: Increasingly popular 16-bit bit encoding
  - Encodes characters from various world languages

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1010010</td>
</tr>
<tr>
<td>S</td>
<td>1010011</td>
</tr>
<tr>
<td>T</td>
<td>1010100</td>
</tr>
<tr>
<td>L</td>
<td>1001100</td>
</tr>
<tr>
<td>N</td>
<td>1001110</td>
</tr>
<tr>
<td>E</td>
<td>1000101</td>
</tr>
<tr>
<td>0</td>
<td>0110000</td>
</tr>
<tr>
<td>.</td>
<td>0101110</td>
</tr>
<tr>
<td>&lt;tab&gt;</td>
<td>0001001</td>
</tr>
<tr>
<td>r</td>
<td>1110010</td>
</tr>
<tr>
<td>s</td>
<td>1110011</td>
</tr>
<tr>
<td>t</td>
<td>1110100</td>
</tr>
<tr>
<td>l</td>
<td>1101100</td>
</tr>
<tr>
<td>n</td>
<td>1101110</td>
</tr>
<tr>
<td>e</td>
<td>1100101</td>
</tr>
<tr>
<td>9</td>
<td>0111001</td>
</tr>
<tr>
<td>!</td>
<td>0100001</td>
</tr>
<tr>
<td>&lt;space&gt;</td>
<td>0100000</td>
</tr>
</tbody>
</table>

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

- **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
  - +/- $2^{n-1}$ – 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 $\rightarrow$ 1000 + 1 = 1001 (-7)
  - $-7^*$ is 7: 1001 $\rightarrow$ 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 = $c_3 \oplus c_4 = c_3 \cdot c_4' + c_3' \cdot c_4$

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

\[
\begin{array}{cccc}
\text{a} & \text{a} & \text{a} & \text{a} \\
\times & \text{b} & \text{b} & \text{b} & \text{b} \\
\hline
& \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & (\text{pp1}) \\
& \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & (\text{pp2}) \\
& \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & (\text{pp3}) \\
+ & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & (\text{pp4}) \\
\hline
\text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} & \text{p} \\
\end{array}
\]
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

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Quotient</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
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
  - \( \cdot \) 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 \( a=1 \))

- Derived operators:
  - NAND
  - NOR
  - XOR
  - XNOR
Representations of Boolean Functions

English 1: F outputs 1 when a is 0 and b is 0, or when a is 0 and b is 1.

English 2: F outputs 1 when a is 0, regardless of b’s value.

Equation 1: \( F(a,b) = a'b' + a'b \)
Equation 2: \( F(a,b) = a' \)

Truth table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot (C + D)))$
Boolean Duality

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

\[ X + Y + ... \iff X \cdot Y \cdot ... \]

- Generalized duality:

\[ f (X_1,X_2,...,X_n,0,1,+,\cdot) \iff f(X_1,X_2,...,X_n,1,0,\cdot,+) \]

- Any theorem that can be proven is also proven for its dual! Note: this is NOT deMorgan’s Law
# Boolean Axioms & Theorems

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Dual</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A1'</td>
<td>Binary field</td>
</tr>
<tr>
<td>$B = 0$ if $B \neq 1$</td>
<td>$B = 1$ if $B \neq 0$</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>A2'</td>
<td>NOT</td>
</tr>
<tr>
<td>$\overline{0} = 1$</td>
<td>$T = 0$</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>A3'</td>
<td>AND/OR</td>
</tr>
<tr>
<td>$0 \cdot 0 = 0$</td>
<td>$1 + 1 = 1$</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>A4'</td>
<td>AND/OR</td>
</tr>
<tr>
<td>$1 \cdot 1 = 1$</td>
<td>$0 + 0 = 0$</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>A5'</td>
<td>AND/OR</td>
</tr>
<tr>
<td>$0 \cdot 1 = 1 \cdot 0 = 0$</td>
<td>$1 + 0 = 0 + 1 = 1$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Dual</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T1'</td>
<td>Identity</td>
</tr>
<tr>
<td>$B \cdot 1 = B$</td>
<td>$B + 0 = B$</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>T2'</td>
<td>Null Element</td>
</tr>
<tr>
<td>$B \cdot 0 = 0$</td>
<td>$B + 1 = 1$</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>T3'</td>
<td>Idempotency</td>
</tr>
<tr>
<td>$B \cdot B = B$</td>
<td>$B + B = B$</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>$\overline{B} = B$</td>
<td>Involution</td>
</tr>
<tr>
<td>T5</td>
<td>T5'</td>
<td>Complements</td>
</tr>
<tr>
<td>$B \cdot \overline{B} = 0$</td>
<td>$B + \overline{B} = 1$</td>
<td></td>
</tr>
</tbody>
</table>
## Boolean theorems of multiple variables

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Dual</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T6</td>
<td>T6'</td>
<td>Commutativity</td>
</tr>
<tr>
<td>$B \cdot C = C \cdot B$</td>
<td>$B + C = C + B$</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>T7'</td>
<td>Associativity</td>
</tr>
<tr>
<td>$(B \cdot C) \cdot D = B \cdot (C \cdot D)$</td>
<td>$(B + C) + D = B + (C + D)$</td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>T8'</td>
<td>Distributivity</td>
</tr>
<tr>
<td>$(B \cdot C) + B \cdot D = B \cdot (C + D)$</td>
<td>$(B + C) \cdot (B + D) = B + (C \cdot D)$</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>T9'</td>
<td>Covering</td>
</tr>
<tr>
<td>$B \cdot (B + C) = B$</td>
<td>$B + (B \cdot C) = B$</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>T10'</td>
<td>Combining</td>
</tr>
<tr>
<td>$(B \cdot C) + (B \cdot \overline{C}) = B$</td>
<td>$(B + C) \cdot (B + \overline{C}) = B$</td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>T11'</td>
<td>Consensus</td>
</tr>
<tr>
<td>$(B \cdot C) + (\overline{B} \cdot D) + (C \cdot D)$</td>
<td>$(B + C) \cdot (\overline{B} + D) \cdot (C + D)$</td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>T12'</td>
<td>De Morgan’s Theorem</td>
</tr>
<tr>
<td>$B_0 \cdot B_1 \cdot B_2 \cdots$</td>
<td>$B_0 + B_1 + B_2 \cdots$</td>
<td></td>
</tr>
<tr>
<td>$= (\overline{B_0} + \overline{B_1} + \overline{B_2} \cdots)$</td>
<td>$= (\overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2})$</td>
<td></td>
</tr>
</tbody>
</table>
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:
  - \((X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = X \cdot Y + X' \cdot Z\)

  \[
  (X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) \\
  \text{identity} \quad (X \cdot Y) + (1) \cdot (Y \cdot Z) + (X' \cdot Z) \\
  \text{complementarity} \quad (X \cdot Y) + (X' + X) \cdot (Y \cdot Z) + (X' \cdot Z) \\
  \text{distributivity} \quad (X \cdot Y) + (X' \cdot Y \cdot Z) + (X \cdot Y \cdot Z) + (X' \cdot Z) \\
  \text{commutativity} \quad (X \cdot Y) + (X \cdot Y \cdot Z) + (X' \cdot Y \cdot Z) + (X' \cdot Z) \\
  \text{factoring} \quad (X \cdot Y) \cdot (1 + Z) + (X' \cdot Z) \cdot (1 + Y) \\
  \text{null} \quad (X \cdot Y) \cdot (1) + (X' \cdot Z) \cdot (1) \\
  \text{identity} \quad (X \cdot Y) + (X' \cdot Z) \checkmark
  \]
Proving theorems (perfect induction)

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

\[(X + Y)' = X' \cdot Y'\]

NOR is equivalent to AND
with inputs complemented

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X'</th>
<th>Y'</th>
<th>(X + Y)'</th>
<th>X' \cdot Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[(X \cdot Y)' = X' + Y'\]

NAND is equivalent to OR
with inputs complemented

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X'</th>
<th>Y'</th>
<th>(X \cdot Y)'</th>
<th>X' + Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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
    - $V = I \times R$ (Ohm’s Law)
The CMOS Switches

- CMOS circuit
  - Consists of N and PMOS transistors
  - Both N and PMOS are similar to basic switches
  - $R_p \sim 2 \ R_n \Rightarrow$ 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**

\[ Y = \overline{A} \]

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Truth Table**

<table>
<thead>
<tr>
<th>A</th>
<th>P1</th>
<th>N1</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CMOS Gates: NAND Gate

NAND

\[ Y = \overline{AB} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

A
B
Y

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P1</th>
<th>P2</th>
<th>N1</th>
<th>N2</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CMOS Gates: NAND Gate
Three input NOR gate

CMOS gate structure:

Three-input NOR

inputs

pMOS pull-up network

output

nMOS pull-down network
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: \( x \cdot y + x \cdot z + x \cdot v = x \cdot (y + z + v) \)
    
    \[
    \begin{array}{ll}
    \text{5 operations:} & \text{3 operations:} \\
    3 \text{ AND, 2 OR} & 1 \text{ AND, 2 OR} \\
    \# \text{ txs} = & \# \text{ txs} =
    \end{array}
    \]

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

• **Construct the function below in CMOS**
  \[ F = a + b \cdot (c + d); \text{ remember AND operations occur before OR} \]

• **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: $F = A \cdot (B + C)$
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
    • $R = \frac{r'}{L/W}$
    • $L$ 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: $\frac{dQ}{dt} = C \ \frac{dV}{dt}$

- **Timing estimate**
  - $D\ t = C \ \frac{dV}{i} = C \ \frac{dV}{(V/R_{\text{trans}})} = R_{\text{trans}}C \ \frac{dV}{V}$

- **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
  \[ V_{\text{out}} = V_{\text{dd}} \left( 1 - e^{-t/R_pC} \right) \]
- \( R_p \approx 2R_n \)

Source: Prof. Subhashish Mitra
Timing analysis: Inverter

\[ F = x' \]
Timing analysis in gates

OR

\[
\begin{array}{ccc}
  x & y & F \\
  0 & 0 & 1 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 0 \\
\end{array}
\]

AND

\[
\begin{array}{ccc}
  x & y & F \\
  0 & 0 & 1 \\
  0 & 1 & 1 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]

F = x or y

F' = x & y
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)

\[
P_{\text{dynamic}} = \frac{1}{2}CV_{DD}^2f
\]

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

\[
P_{\text{static}} = I_{DD}V_{DD}
\]
Power estimate example

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

\[
P = \frac{1}{2}CV_{DD}^2f + I_{DD}V_{DD}
\]
\[
= \frac{1}{2}(20 \text{ nF})(1.2 \text{ V})^2(1 \text{ GHz}) + (20 \text{ mA})(1.2 \text{ V})
\]
\[
= 14.4 \text{ W}
\]
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