EECS 150 - Components and Design Techniques for Digital Systems

Lec 02 – Gates and CMOS Technology
8-30-07

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://inst.eecs.berkeley.edu/~cs150

Outline

- Summary of last time
- Overview of Physical Implementations
- Boolean Logic
- CMOS devices
- Combinational Logic
- Announcements/Break
- CMOS transistor circuits
  - basic logic gates
  - tri-state buffers
  - flip-flops
    - » flip-flop timing basics
    - » example use
    - » circuits

L01 Summary: Digital Design

Given a functional description and performance, cost, & power constraints, come up with an implementation using a set of primitives.

- How do we learn how to do this?
  1. Learn about the primitives and how to generate them.
  2. Learn about design representation.
  3. Learn formal methods to optimally manipulate the representations.
  4. Look at design examples.
  5. Use trial and error - CAD tools and prototyping.
- *Digital design is in some ways more an art than a science. The creative spirit is critical in combining primitive elements & other components in new ways to achieve a desired function.*
- However, unlike art, we have objective measures of a design: *performance cost power & Time to Market*

The Boolean Abstraction

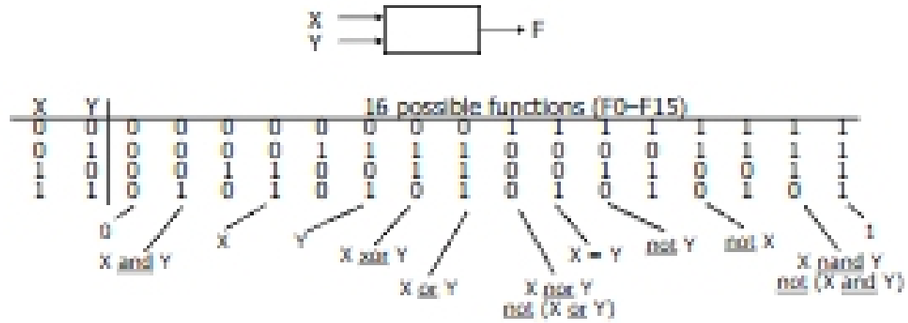
Mapping from physical world to binary world

Technology	State 0	State 1
Relay logic	Circuit Open	Circuit Closed
CMOS logic	0.0-1.0 volts	2.0-3.0 volts
Transistor transistor logic (TTL)	0.0-0.8 volts	2.0-5.0 volts
Fiber Optics	Light off	Light on
Dynamic RAM	Discharged capacitor	Charged capacitor
Nonvolatile memory (erasable)	Trapped electrons	No trapped electrons
Programmable ROM	Fuse blown	Fuse intact
Bubble memory	No magnetic bubble	Bubble present
Magnetic disk	No flux reversal	Flux reversal
Compact disc	No pit	Pit

Sense the logical value, manipulate in a systematic fashion.

Possible Logic Functions of Two Variables

- 16 possible functions of 2 input variables:
  - $2^{\#(2^{\#n})}$ functions of n inputs



Logic Functions and Boolean Algebra

- Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: ', +, and •

X	Y	X·Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X'	X'·Y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

X	Y	X'	Y'	X·Y	X'·Y'	(X·Y)+(X'·Y')
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

$(X \cdot Y) + (X' \cdot Y') = X \odot Y$

Boolean expression that is true when the variables X and Y have the same value and false, otherwise.

X, Y are Boolean algebra variables

Minimal set of functions

- Implement any logic functions from NOT, NOR, and NAND?
  - For example, implementing X and Y
    Is the same as implementing not (X nand Y)
- Do it with only NOR or only NAND
  - NOT is just a NAND or a NOR with both inputs tied together

X	Y	X nor Y
0	0	1
1	1	0

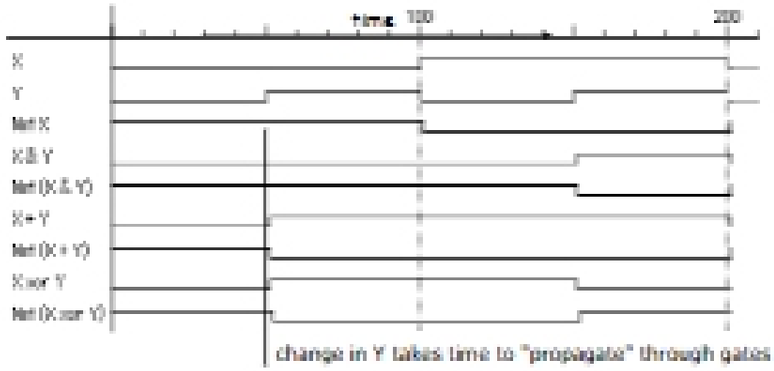
X	Y	X nand Y
0	0	1
1	1	0

  - and NAND and NOR are "duals", i.e., easy to implement one using the other

X nand Y ≡ not ((not X) nor (not Y))
X nor Y ≡ not ((not X) nand (not Y))

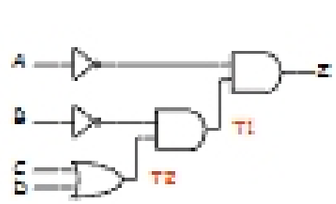
Waveform View of Logic Functions

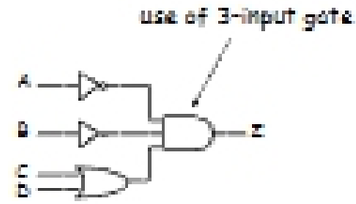
- Just a sideways truth table
  - But note how edges don't line up exactly
  - It takes time for a gate to switch its output!



From Boolean Expressions to Logic Gates

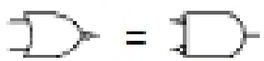
- More than one way to map expressions to gates
  - e.g., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot \overline{(C + D)}^{T2})^{T1})$





Proving theorems (perfect induction)

- De Morgan's Law
  - complete truth table, exhaustive proof



$(X + Y)' \cdot X' \cdot Y'$

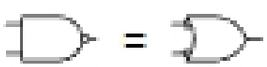
NOR is equivalent to AND with inputs complemented

X	Y	X'	Y'	(X + Y)'	X' · Y'
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

$(X \cdot Y)' \cdot X' \cdot Y'$

NAND is equivalent to OR with inputs complemented

X	Y	X'	Y'	(X · Y)'	X' + Y'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0



Push inv. bubble from output to input and change symbol

An algebraic structure

- An algebraic structure consists of
  - a set of elements B
  - binary operations { + , · }
  - and a unary operation { ' }
  - such that the following axioms hold:

More on this later

1. set B contains at least two elements, a, b, such that a ≠ b
2. closure: a + b is in B; a · b is in B
3. commutativity: a + b = b + a; a · b = b · a
4. associativity: a + (b + c) = (a + b) + c; a · (b · c) = (a · b) · c
5. identity: a + 0 = a; a · 1 = a
6. distributivity: a + (b · c) = (a + b) · (a + c); a · (b + c) = (a · b) + (a · c)
7. complementarity: a + a' = 1; a · a' = 0