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

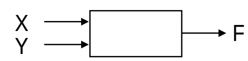
Mapping from physical world to binary world

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 s systematic fashion.

Possible Logic Functions of Two Variables

- **16 possible functions of 2 input variables:**
  - **2**(2**n) functions of n inputs**



16 possible functions (F0–F15)

X	Y	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	X and Y		X		Y	X xor Y	X or Y	X nor Y not (X or Y)	X = Y	not Y		not X		X nand Y not (X and Y)	1

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 • Y) + (X' • Y') ≡ X = 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

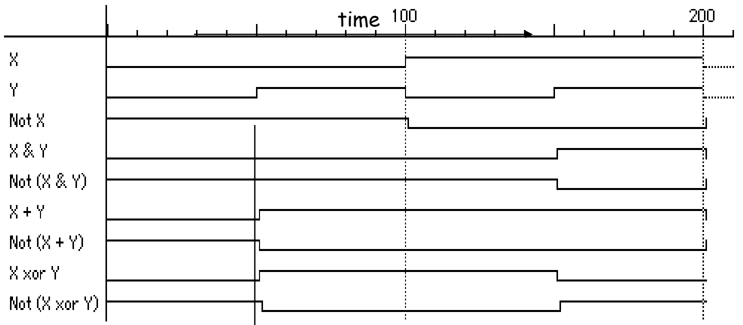
X	Y	X nand Y
0	0	1
1	1	0

  - **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

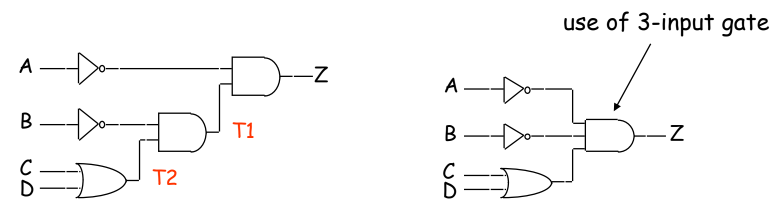
- Just a sideways truth table
  - But note how edges don't line up exactly
  - It takes time for a gate to switch its output!



change in Y takes time to "propagate" through gates

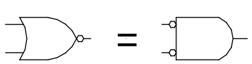
From Boolean Expressions to Logic Gates

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



Proving theorems (perfect induction)

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

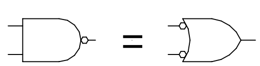


$(X + Y)' = X' \bullet Y'$
NOR is equivalent to AND with inputs complemented

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

More on this later

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

1. set B contains at least two elements, a, b, such that $a \neq b$
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

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 s systematic fashion.

Overview of Physical Implementations

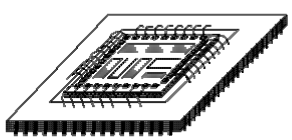
The stuff out of which we make systems.

- **Integrated Circuits (ICs)**
  - Combinational logic circuits, memory elements, analog interfaces.
- **Printed Circuits boards (PCBs)**
  - substrate for ICs and interconnection, distribution of CLK, Vdd, and GND signals, heat dissipation.
- **Power Supplies**
  - Converts line AC voltage to regulated DC low voltage levels.
- **Chassis (rack, card case, ...)**
  - holds boards, power supply, provides physical interface to user or other systems.
- **Connectors and Cables.**

Integrated Circuits

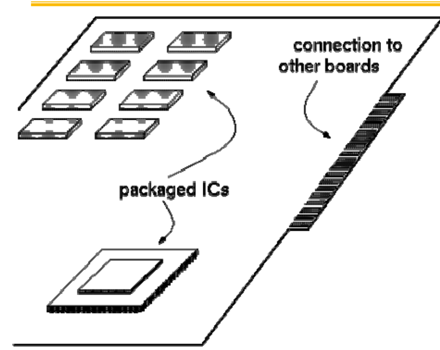
- Primarily Crystalline Silicon
- 1mm - 25mm on a side
- 100 - 200M transistors
- (25 - 50M "logic gates")
- 3 - 10 conductive layers
- 2002 - feature size ~ 0.13um = 0.13 x 10^{-6} m
- "CMOS" most common - complementary metal oxide semiconductor

Chip in Package



- Package provides:
  - spreading of chip-level signal paths to board-level
  - heat dissipation.
- Ceramic or plastic with gold wires.

Printed Circuit Boards



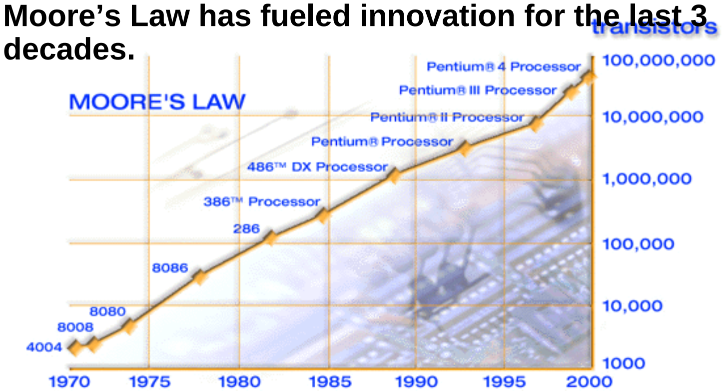
- fiberglass or ceramic
- 1-20 conductive layers
- 1-20in on a side
- IC packages are soldered down.

Multichip Modules (MCMs)

- Multiple chips directly connected to a substrate. (silicon, ceramic, plastic, fiberglass) without chip packages.

Integrated Circuits

- Moore's Law has fueled innovation for the last 3 decades.



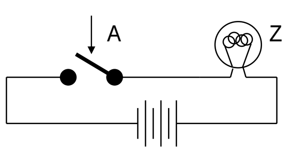
- "Number of transistors on a die doubles every 18 months."
- What are the side effects of Moore's law?

Integrated Circuits

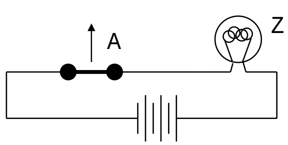
- Uses for digital IC technology today:
  - standard microprocessors
    - » used in desktop PCs, and embedded applications
    - » simple system design (mostly software development)
  - memory chips (DRAM, SRAM)
  - application specific ICs (ASICs)
    - » custom designed to match particular application
    - » can be optimized for low-power, low-cost, high-performance
    - » high-design cost / relatively low manufacturing cost
  - field programmable logic devices (FPGAs, CPLDs)
    - » customized to particular application after fabrication
    - » short time to market
    - » relatively high part cost
  - standardized low-density components
    - » still manufactured for compatibility with older system designs

Recall: Switches: basic element of physical implementations

- Implementing a simple circuit (arrow shows action if wire changes to "1"):



close switch (if A is "1" or asserted) and turn on light bulb (Z)



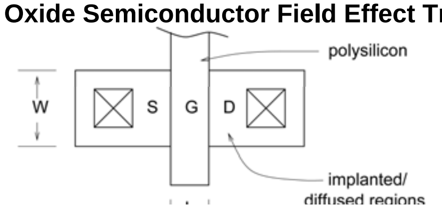
open switch (if A is "0" or unasserted) and turn off light bulb (Z)

Z ≡ A

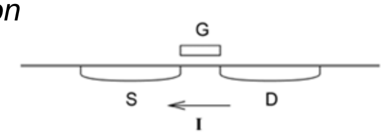
CMOS Devices

- MOSFET (Metal Oxide Semiconductor Field Effect Transistor).

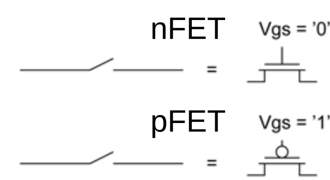
Top View



Cross Section



The gate acts like a capacitor. A high voltage on the gate attracts charge into the channel. If a voltage exists between the source and drain a current will flow. In its simplest approximation the device acts like a switch.



What's Complementary about CMOS

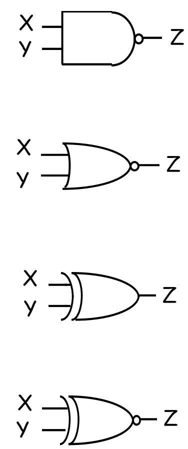
- **CMOS devices work in pairs**
- **Three terminals: drain, gate, and source**
  - **Switch action:**
    if voltage on gate terminal is (some amount) higher/lower than source terminal then conducting path established between drain and source terminals

n-channel
open when voltage at G is low
closes when:
voltage(G) > voltage (S) + ε

p-channel
closed when voltage at G is low
opens when:
voltage(G) < voltage (S) – ε

Building back up to our Boolean Abstraction

- **NAND**
- **NOR**
- **XOR**
  X ⊕ Y
- **XNOR**
  X = Y



NAND:

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR:

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR:

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

X xor Y = X Y' + X' Y
X or Y but not both
("inequality", "difference")

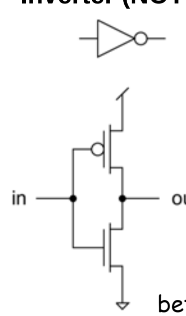
XNOR:

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

X xnor Y = X Y + X' Y'
X and Y are the same
("equality", "coincidence")

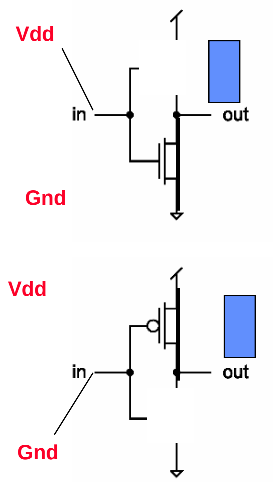
Transistor-level Logic Circuits (inv)

- **Inverter (NOT gate):**

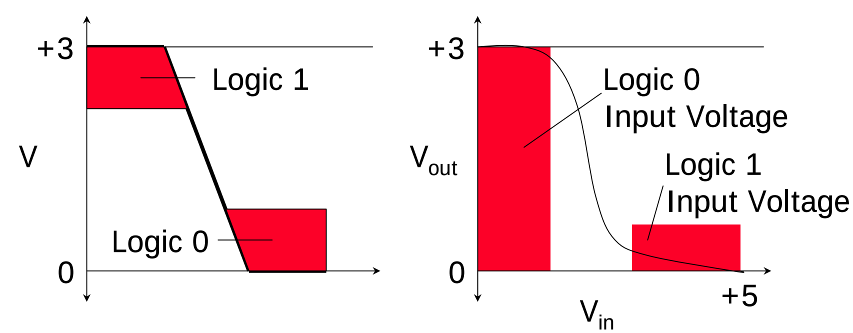


what is the relationship between in and out?

in	out
0 volts	
3 volts	



Logical Values



- **Threshold**
  - **Logical 1 (true) : V > Vdd –V th**
  - **Logical 0 (false) : V < Vth**
- **Noise margin?**

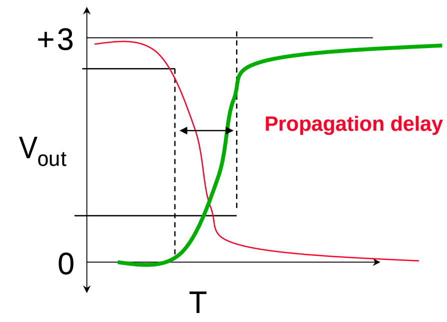
in	out
F	T
T	F

not(out, in)

Big idea: Self-restoring logic

- CMOS logic gates are *self-restoring*
  - Even if the inputs are imperfect, switching time is fast and outputs go rail to rail
  - Doesn't matter how many you cascade
    - » Although propagation delay increases
- Manage fan-out to ensure sharp and complete transition

Element of Time



- Logical change is not instantaneous
- Broader digital design methodology has to make it appears as such
  - Clocking, delay estimation, glitch avoidance

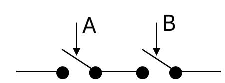
Announcements

- **Reading assignment for this week.**
  - Katz and Boriello, Chap 1
  - K&B 2.1-2.3, pp. 155-172
- **Homework 1 is posted - due week from friday**

Computing with Switches

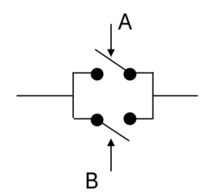
- Compose switches into more complex functions:

AND



Z ≡ A and B

OR

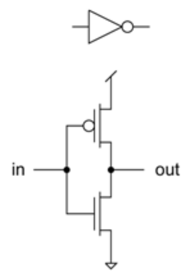


Z ≡ A or B

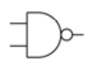
Two fundamental structures: series (AND) and parallel (OR)

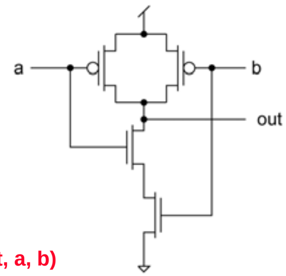
Transistor-level Logic Circuits - NAND

- **Inverter (NOT gate):**



- **NAND gate**





a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

nand (out, a, b)

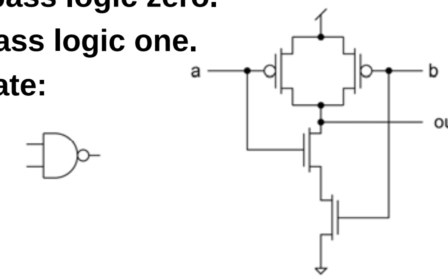
- **Logic Function:**
  - **out = 0 iff both a AND b = 1 therefore out = (ab)’**
  - **pFET network and nFET network are *duals* of one another.**

How about AND gate?

Transistor-level Logic Circuits

Simple rule for wiring up MOSFETs:

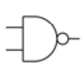
- **nFET is used only to pass logic zero.**
- **pFet is used only to pass logic one.**
- **For example, NAND gate:**

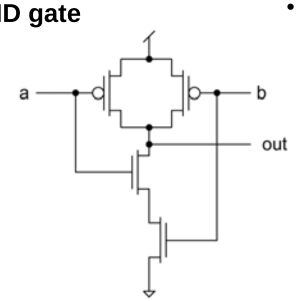


Note: This rule is sometimes violated by expert designers under special conditions.

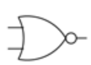
Transistor-level Logic Circuits - NOR

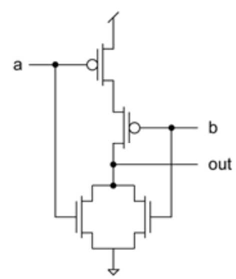
- **NAND gate**





- **NOR gate**





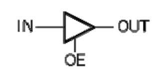
a	b	out
0	0	1
0	1	0
1	0	0
1	1	0

nor (out, a, b)

- **Function:**
  - **out = 0 iff both a OR b = 1 therefore out = (a+b)’**
  - **Again pFET network and nFET network are *duals* of one another.**
  - **Other more complex functions are possible. Ex: out = (a+bc)’**

Transistor-level Logic Circuits

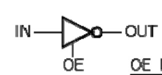
- **Tri-state Buffer**



OE	IN	OUT
0	0	Z
0	1	Z
1	0	0
1	1	1

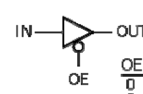
“high impedance” (output disconnected)

- **Variations**



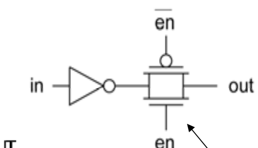
OE	IN	OUT
0	–	Z
1	0	1
1	1	0

Inverting buffer



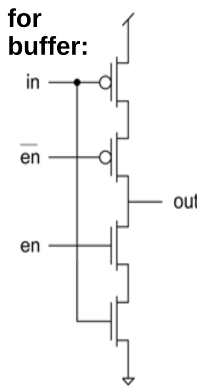
OE	IN	OUT
0	0	0
0	1	1
1	–	Z

Inverted enable



“transmission gate”

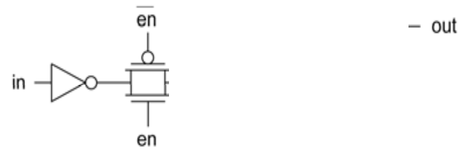
- **Transistor circuit for inverting tri-state buffer:**



Tri-state buffers are used when multiple circuits all connect to a common bus. Only one circuit at a time is allowed to drive the bus. All others “disconnect”.

Transmission Gate

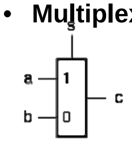
- Transmission gates are the way to build “switches” in CMOS.
- Both transistor types are needed:
  - nFET to pass zeros.
  - pFET to pass ones.
- The transmission gate is bi-directional (unlike logic gates and tri-state buffers).
- Functionally it is similar to the tri-state buffer, but does not connect to Vdd and GND, so must be combined with logic gates or buffers.



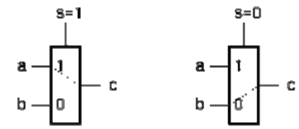
Is it self restoring?

Transistor-level Logic Circuits - MUX

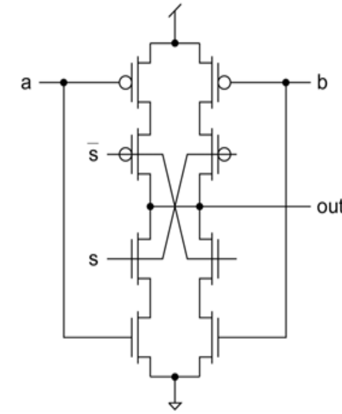
- Multiplexor



If s=1 then c=a else c=b

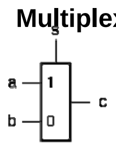


- Transistor Circuit for inverting multiplexor:

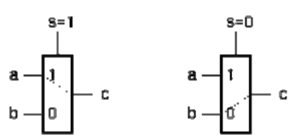


Transistor-level Logic Circuits - MUX

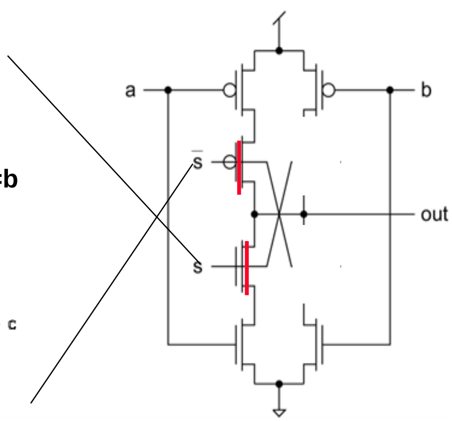
- Multiplexor



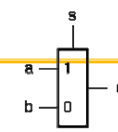
If s=1 then c=a else c=b



- Transistor Circuit for inverting multiplexor:

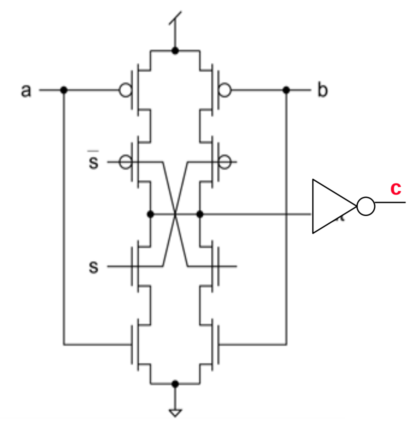


Interactive Quiz



mux (c, s, a, b)

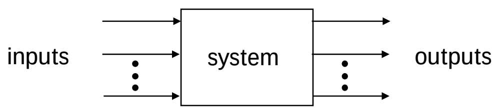
- Generate truth table for MUX
- Boolean expression?
- Can you build an inverter out of a MUX?
- How about AND?



universality

Combinational vs. Sequential Digital Circuits

- **Simple model of a digital system is a unit with inputs and outputs:**



- **Combinational means "memory-less"**
  - **Digital circuit is combinational if its output values only depend on its inputs**

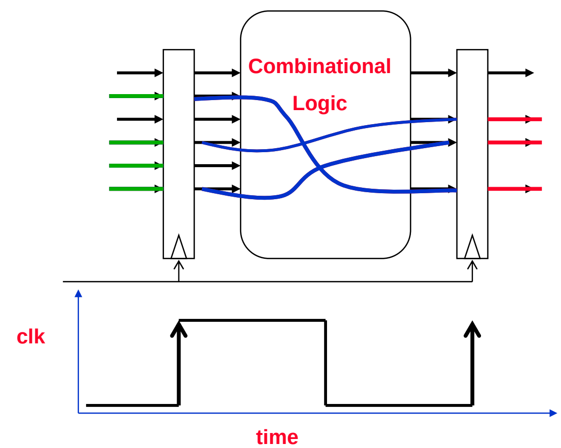
Sequential logic

- **Sequential systems**
  - **Exhibit behaviors (output values) that depend on current *as well as* previous inputs**
- **All real circuits are sequential**
  - **Outputs do not change instantaneously after an input change**
  - **Why not, and why is it then sequential?**
- **Fundamental abstraction of digital design is to reason (mostly) about steady-state behaviors**
  - **Examine outputs only after sufficient time has elapsed for the system to make its required changes and settle down**

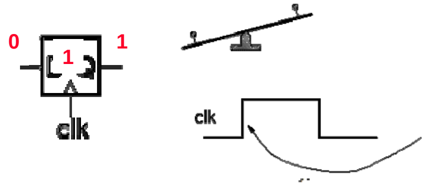
Synchronous sequential digital systems

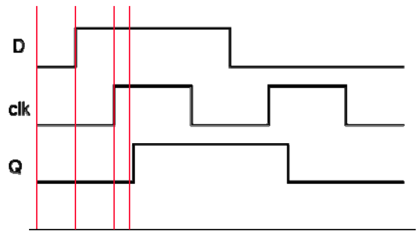
- **Combinational circuit outputs depend *only* on current inputs**
  - **After sufficient time has elapsed**
- **Sequential circuits have *memory***
  - **Even after waiting for transient activity to finish**
- **Steady-state abstraction: most designers use it when constructing sequential circuits:**
  - **Memory of system is its state**
  - **Changes in system state only allowed at specific times controlled by an external periodic signal (the *clock*)**
  - **Clock period is elapsed time between state changes sufficiently long so that system reaches steady-state before next state change at end of period**

Recall: What makes Digital Systems tick?

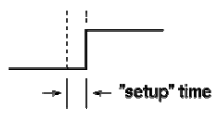


D-type edge-triggered flip-flop

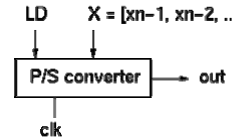




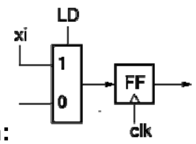
- The edge of the clock is used to *sample* the "D" input & send it to "Q" (positive edge triggering).
  - At all other times the output Q is independent of the input D (just stores previously sampled value).
  - The input must be stable for a short time before the clock edge.

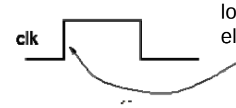


Parallel to Serial Converter Example



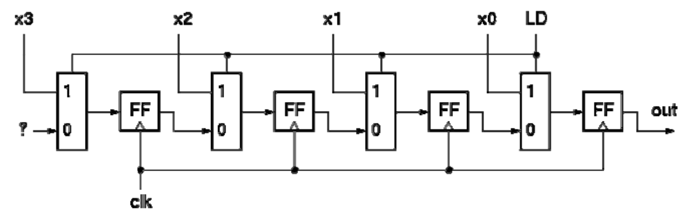
- Operation:
  - cycle 1: load x, output x_0
  - cycle i: output x_i
- Each stage:



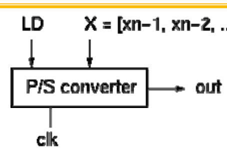


if LD=1
load FF from x_i
else from previous stage.

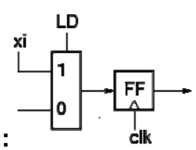
- 4-bit version:

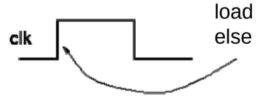


Parallel to Serial Converter Example



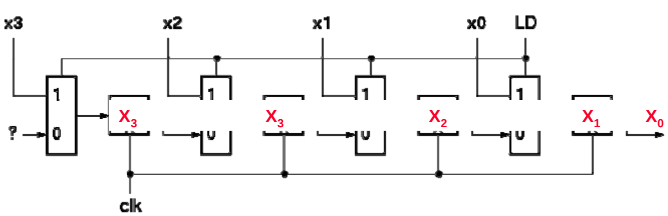
- Operation:
  - cycle 1: load x, output x_0
  - cycle i: output x_i
- Each stage:



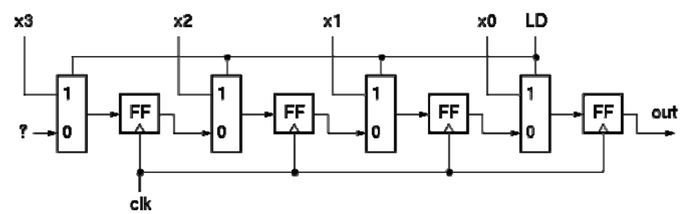


if LD=1
load FF from x_i
else from previous stage.

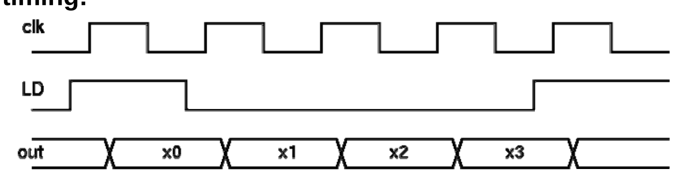
- 4-bit version:



Parallel to Serial Converter Example

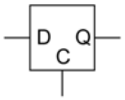


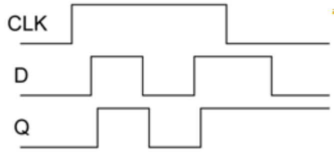
- timing:

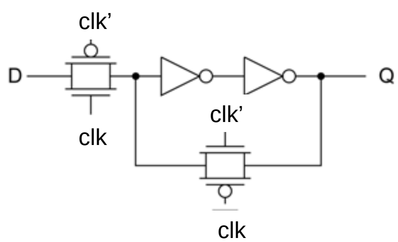


Transistor-level Logic Circuits - Latch

- **Positive Level-sensitive latch**







D FlipFlop

Summary: Representation of digital designs

- Physical devices (transistors, relays)
- ***Switches***
- ***Truth tables***
- ***Boolean algebra***
- ***Gates***
- Waveforms
- Finite state behavior
- Register-transfer behavior
- Concurrent abstract specifications

scope of CS 150
more depth than 61C
focus on building systems

Axioms & theorems of Boolean algebra

- **Identity**
  - 1. X + 0 = X 1D. X • 1 = X
- **Null**
  - 2. X + 1 = 1 2D. X • 0 = 0
- **Idempotency:**
  - 3. X + X = X 3D. X • X = X
- **Involution:**
  - 4. (X')' = X
- **Complementarity:**
  - 5. X + X' = 1 5D. X • X' = 0
- **Commutativity:**
  - 6. X + Y = Y + X 6D. X • Y = Y • X
- **Associativity:**
  - 7. (X + Y) + Z = X + (Y + Z) 7D. (X • Y) • Z = X • (Y • Z)

Axioms and theorems of Boolean algebra (cont'd)

- **Distributivity:**
  - 8. X • (Y + Z) = (X • Y) + (X • Z) 8D. X + (Y • Z) = (X + Y) • (X + Z)
- **Uniting:**
  - 9. X • Y + X • Y' = X 9D. (X + Y) • (X + Y') = X
- **Absorption:**
  - 10. X + X • Y = X 10D. X • (X + Y) = X
  - 11. (X + Y') • Y = X • Y 11D. (X • Y') + Y = X + Y
- **Factoring:**
  - 12. (X + Y) • (X' + Z) = X • Z + X' • Y 12D. X • Y + X' • Z = (X + Z) • (X' + Y)
- **Concensus:**
  - 13. (X • Y) + (Y • Z) + (X' • Z) = X • Y + X' • Z 13D. (X + Y) • (Y + Z) • (X' + Z) = (X + Y) • (X' + Z)

Axioms and theorems of Boolean algebra (cont')

- de Morgan's:
  14. (X + Y + ...)' = X' • Y' • ...   14D. (X • Y • ...)' = X' + Y' + ...
- generalized de Morgan's:
  15. f'(X1,X2,...,Xn,0,1,+,•) = f(X1',X2',...,Xn',1,0,•,+)
- establishes relationship between • and +

Axioms & theorems of Bool. Alg. - Duality

- Duality
  - Dual of a Boolean expression is derived by replacing • by +, + by •, 0 by 1, and 1 by 0, and leaving variables unchanged
  - Any theorem that can be proven is thus also proven for its dual!
  - Meta-theorem (a theorem about theorems)
- duality:
  16. X + Y + ... ⇔ X • Y • ...
- generalized duality:
  17. f (X1,X2,...,Xn,0,1,+,•) ⇔ f(X1,X2,...,Xn,1,0,•,+)
- Different than deMorgan's Law
  - this is a statement about theorems
  - this is not a way to manipulate (re-write) expressions

Proving theorems (rewriting)

- Using the axioms of Boolean algebra:
  - e.g., prove the theorem: X • Y + X • Y' = X

distributivity (8)	X • Y + X • Y'	=	X • (Y + Y')
complementarity (5)	X • (Y + Y')	=	X • (1)
identity (1D)	X • (1)	=	X ✓

  - e.g., prove the theorem: X + X • Y = X

identity (1D)	X + X • Y	=	X • 1 + X • Y
distributivity (8)	X • 1 + X • Y	=	X • (1 + Y)
identity (2)	X • (1 + Y)	=	X • (1)
identity (1D)	X • (1)	=	X ✓