

# **ECE357: Introduction to VLSI CAD**

Prof. Hai Zhou

Electrical Engineering & Computer Science  
Northwestern University

# Logistics

- Time & Location: MWF 11-11:50 TECH L160
- Instructor: Hai Zhou [haizhou@ece.northwestern.edu](mailto:haizhou@ece.northwestern.edu)
- Office: L461
- Office Hours: W 3-5P
- Teaching Assistant: Chuan Lin
- Texts:  
VLSI Physical Design Automation: Theory & Practice, Sait & Youssef, World Scientific, 1999.
- Reference:

An Introduction to VLSI Physical Design, Sarrafzadeh & Wong, McGraw Hill, 1996.

- Grading: Participation-10% Project-30% Midterm-30% Homework-30%
- Homework must be turned in before class on each due date, late: -40% per day
- Course homepage:  
[www.ece.northwestern.edu/~haizhou/ece357](http://www.ece.northwestern.edu/~haizhou/ece357)

## What can you expect from the course

- Understand modern VLSI design flows (but not the details of tools)
- Understand the physical design problem
- Familiar with the stages and basic algorithms in physical design
- Improve your capability to design algorithms to solve problems
- Improve your capability to think and reason

## What do I expect from you

- Active and critical participation
  - speed me up or slow me down if my pace mismatches yours
  - “Your role is not one of sponges, but one of whetstones; only then the spark of intellectual excitement can continue to jump over”
- Read the textbook
- Do your homeworks
  - You can discuss homework with your classmates, but need to write down solutions independently

## VLSI (Very Large Scale Integrated) chips

- VLSI chips are everywhere
  - computers
  - commercial electronics: TV sets, DVD, VCR, ...
  - voice and data communication networks
  - automobiles
- VLSI chips are artifacts
  - they are produced according to our will ...

## **Design: the most challenging human activity**

- Design is a process of creating a structure to fulfill a requirement
- Brain power is the scarcest resource
  - Delegate as much as possible to computers—CAD
- Avoid two extreme views:
  - Everything manual: impossible—millions of gates
  - Everything computer: impossible either

# Design is always difficult

A main task of a designer is to manage complexity

- **Silicon complexity:** physical effects no longer be ignored
  - resistive and cross-coupled interconnects; signal integrity; weak and faster gates
  - reliability; manufacturability
- **System complexity:** more functionality in less time
  - gap between design and fabrication capabilities
  - desire for system-on-chip (SOC)



## CAD: A tale of two designs

- Target–hardware design
  - How to create a structure on silicon to implement a function
- Aid–software design (programming)
  - How to create an algorithm to solve a design problem
- Be conscious of their similarities and differences

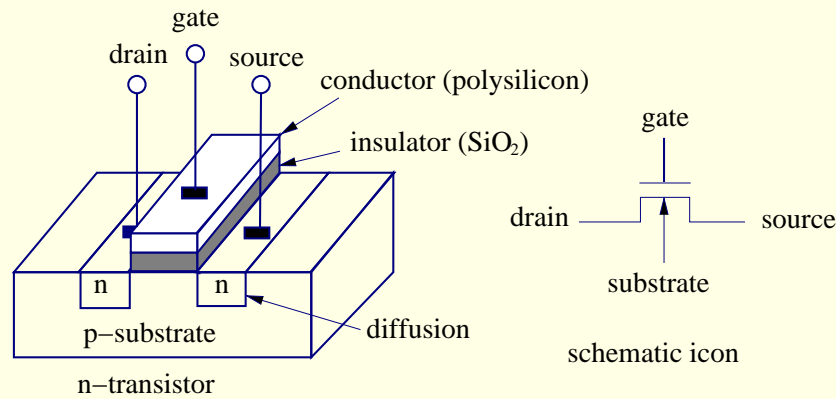
## Emphasis of the course

- Design flow
  - Understand how design process is decomposed into many stages
  - What are the problems need to be solved in each stage
- Algorithms
  - Understand how an algorithm solves a design problem
  - Consider the possibility to extend it
- Be conscious and try to improve problem solving skills

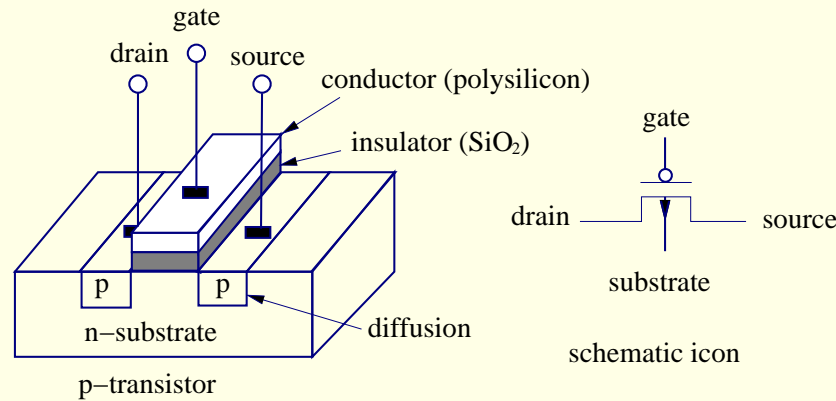
## Basics of MOS Devices

- The most popular VLSI technology: MOS (Metal-Oxide-Semiconductor).
- CMOS (Complementary MOS) dominates nMOS and pMOS, due to CMOS's lower power dissipation, high regularity, etc.
- Physical structure of MOS transistors and their schematic icons: nMOS, pMOS.
- Layout of basic devices:
  - CMOS inverter
  - CMOS NAND gate
  - CMOS NOR gate

# MOS Transistors

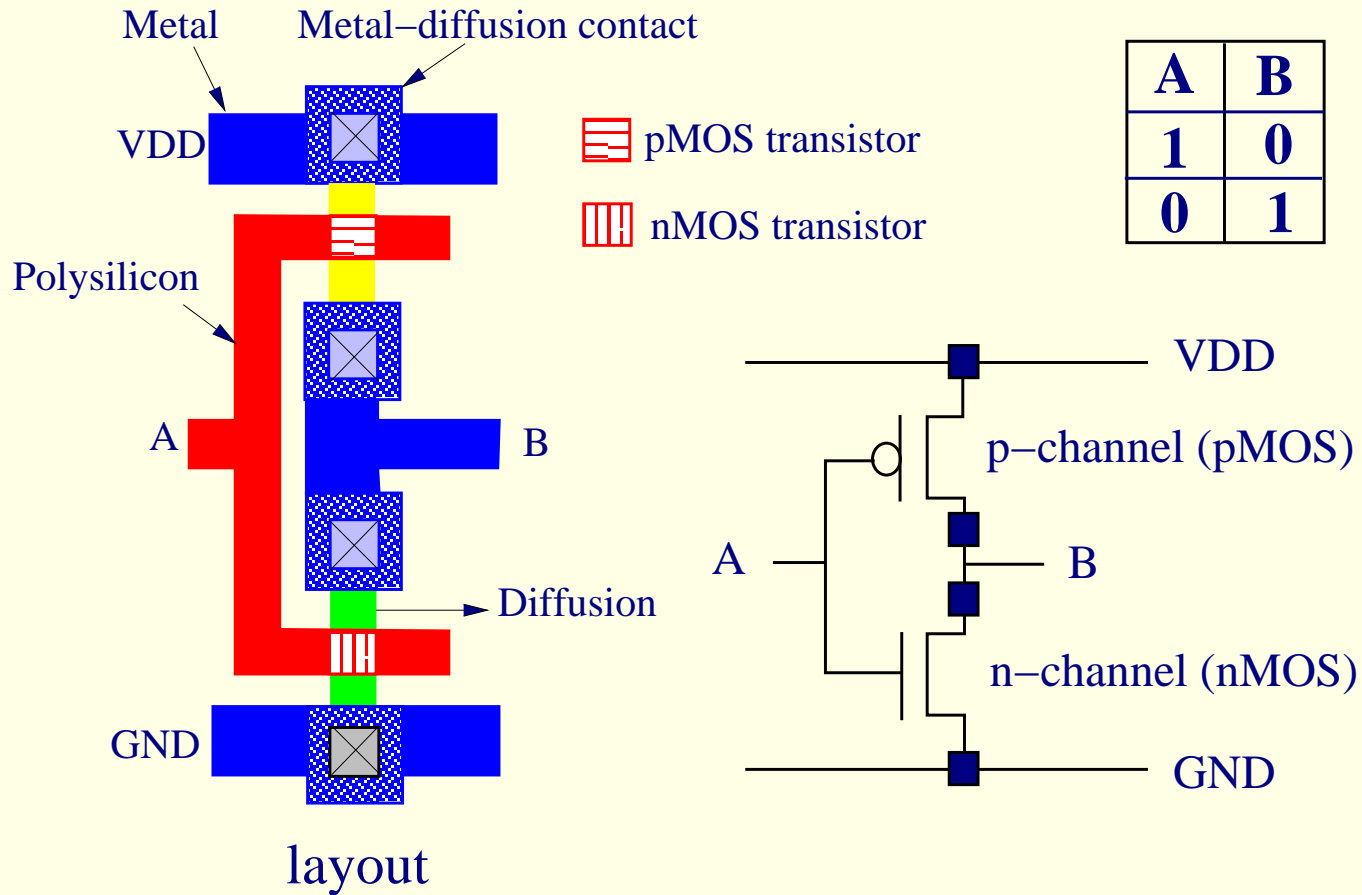


**The nMOS switch passes "0" well.**

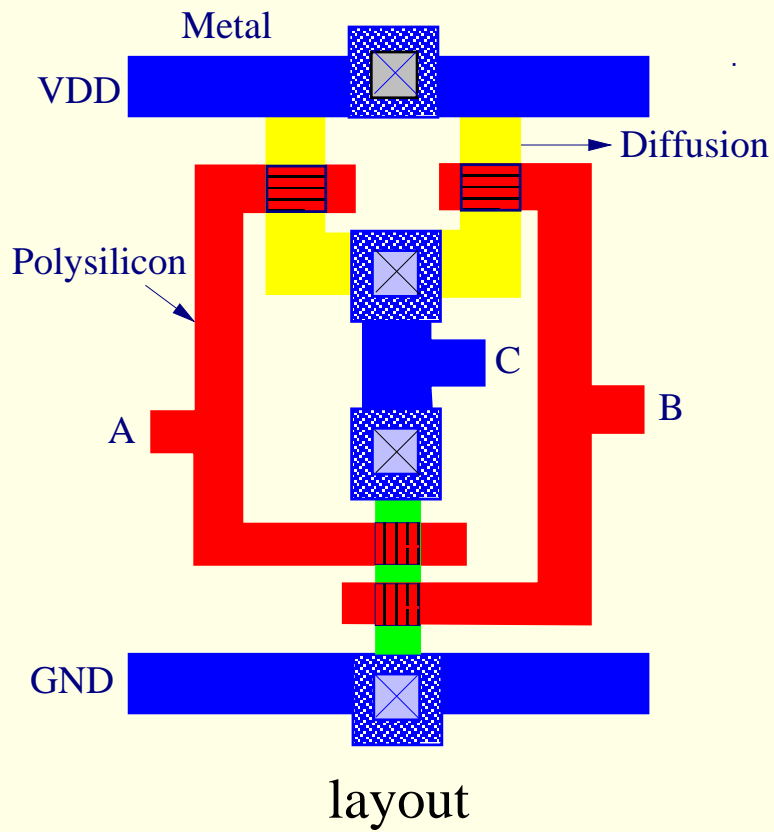


**The pMOS switch passes "1" well.**

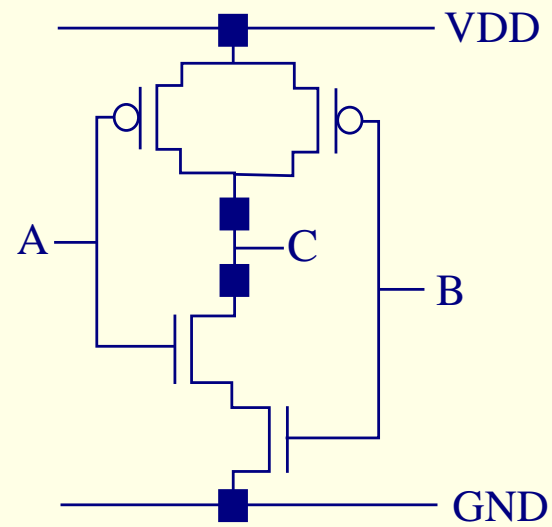
# A CMOS Inverter



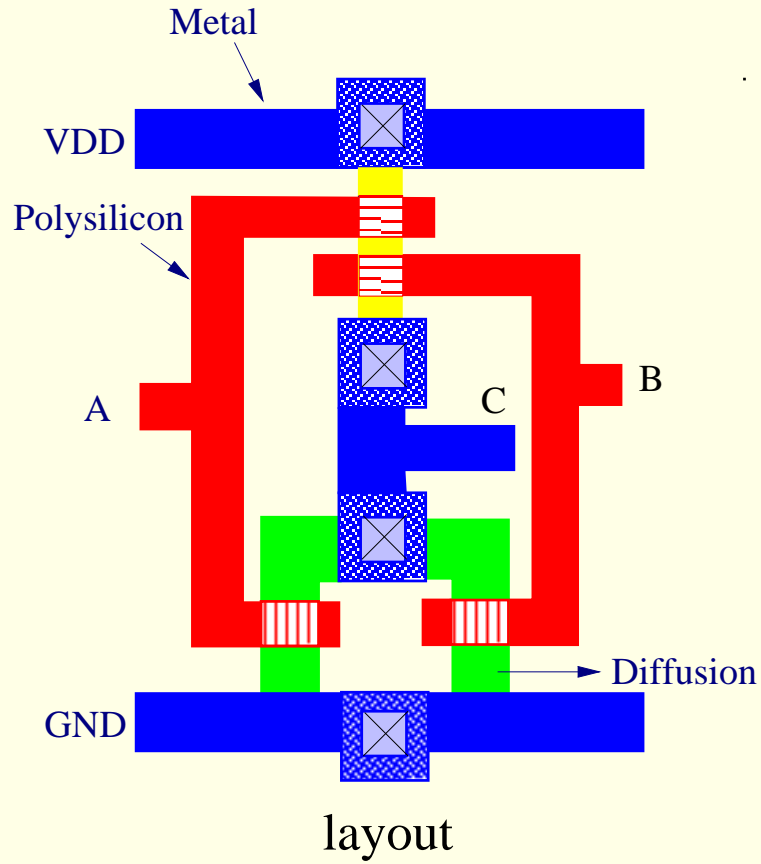
# A CMOS NAND Gate



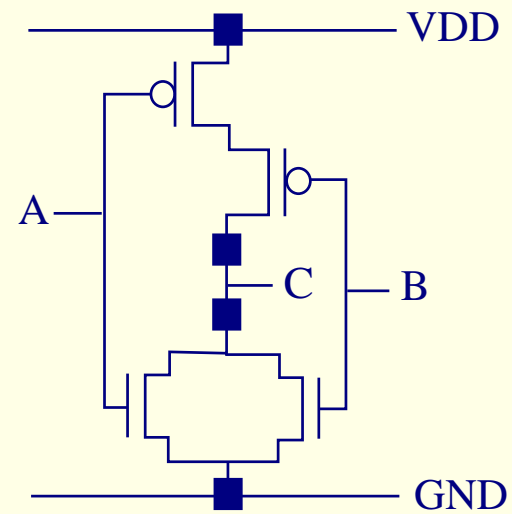
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



# A CMOS NOR Gate



A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



## Current VLSI design phases

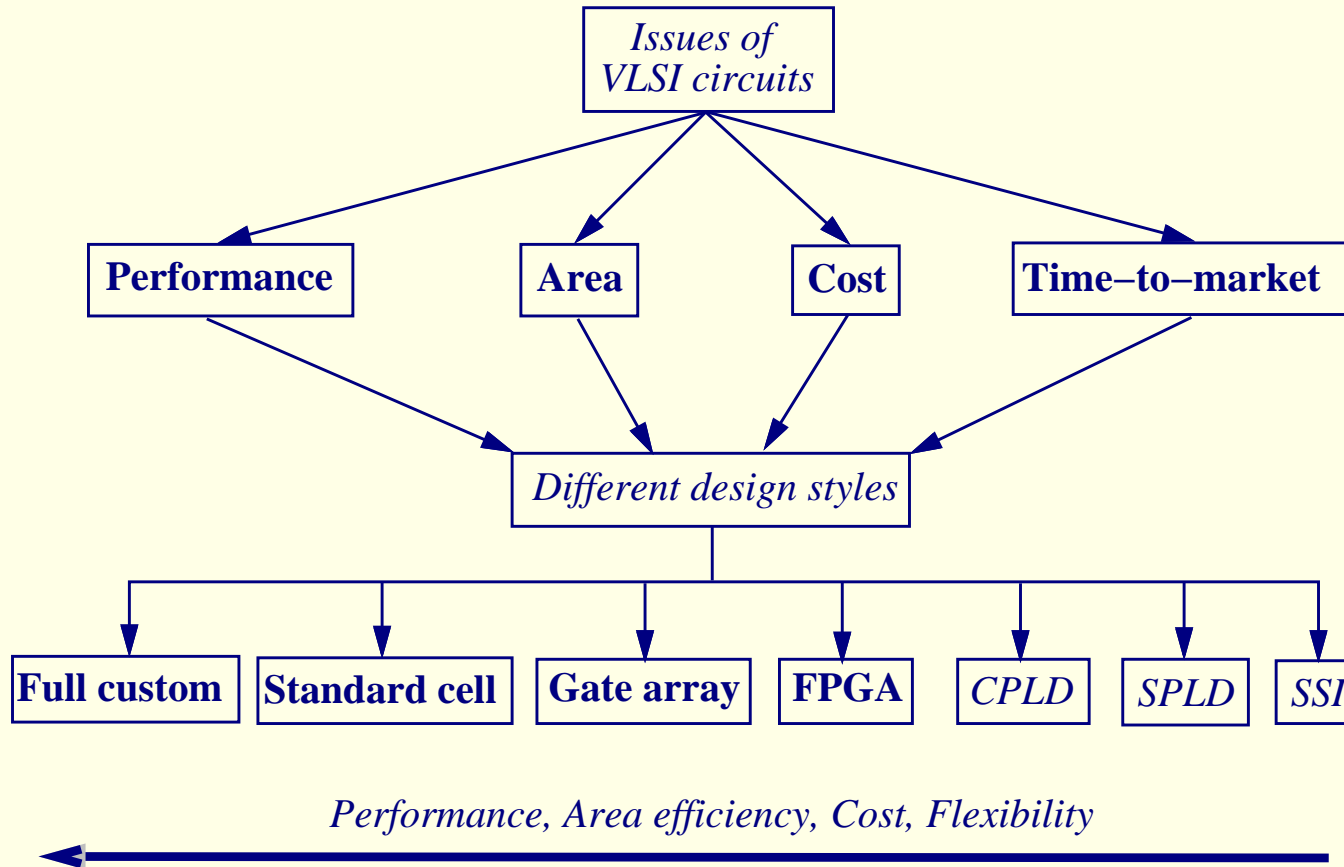
- Synthesis (i.e. specification → implementation)
  1. High level synthesis (459 VLSI Algorithmics)
  2. Logic synthesis (459 VLSI Algorithmics)
  3. Physical design (This course)
- Analysis (implementation → semantics)
  - Verification (design verification, implementation verification)
  - Analysis (timing, function, noise, etc.)
  - Design rule checking, LVS (Layout Vs. Schematic)



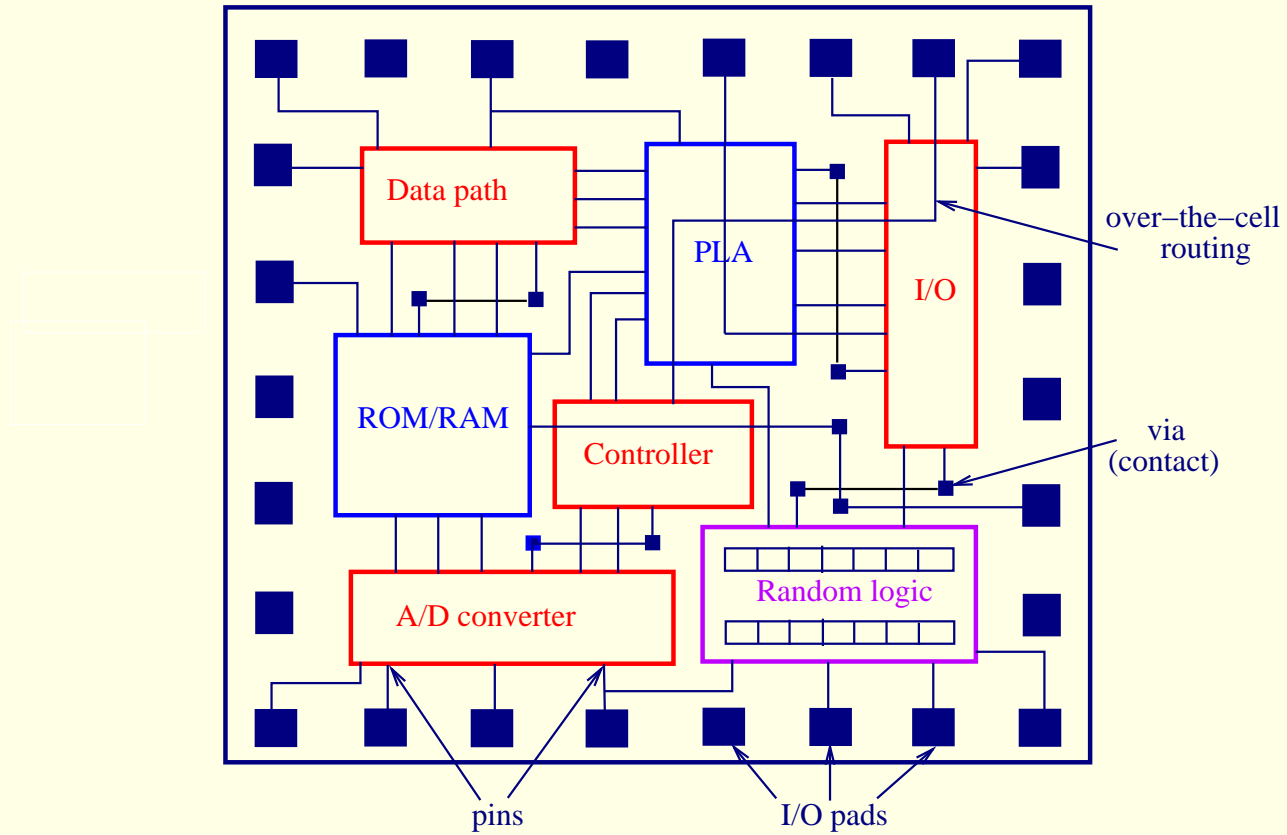
# Physical Design

- Physical design converts a structural description into a geometric description.
- Physical design cycle:
  1. Circuit partitioning
  2. Floorplanning
  3. placement, and pin assignment
  4. Routing (global and detailed)
  5. Compaction

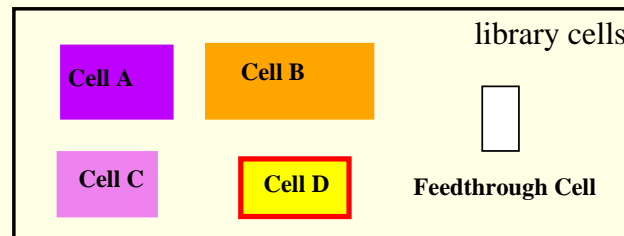
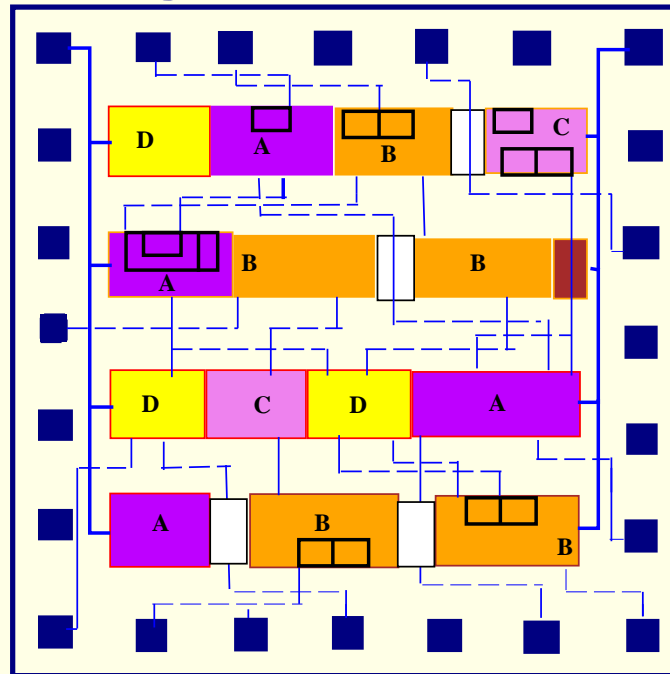
# Design Styles



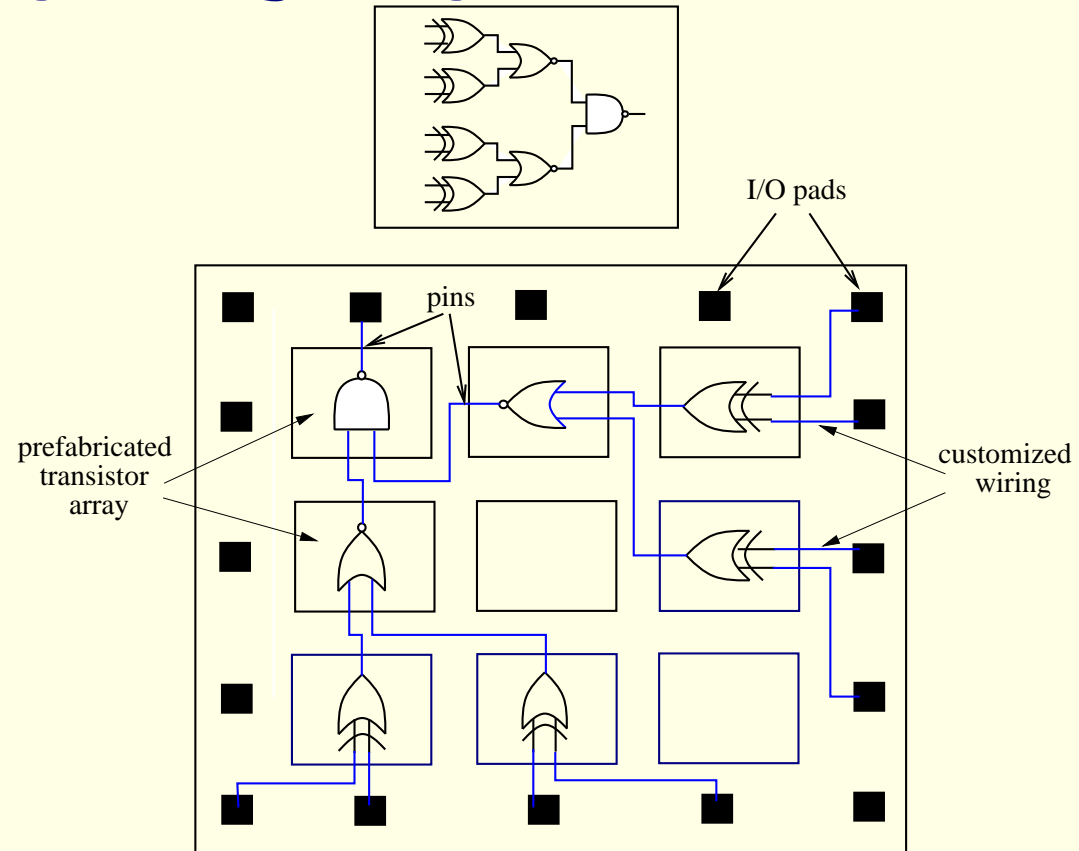
# Full Custom Design Style



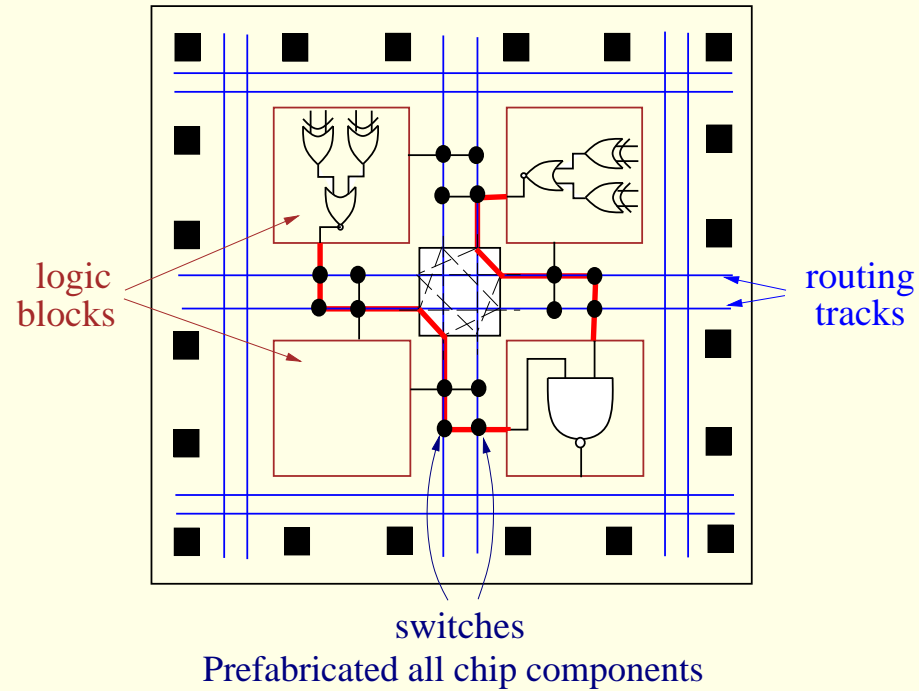
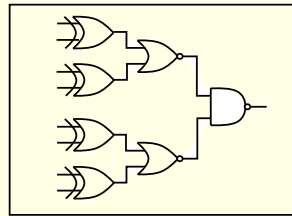
# Standard Cell Design Style



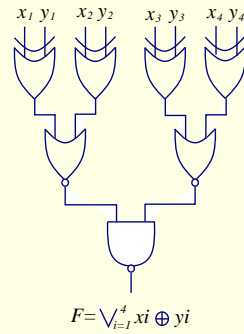
# Gate Array Design Style



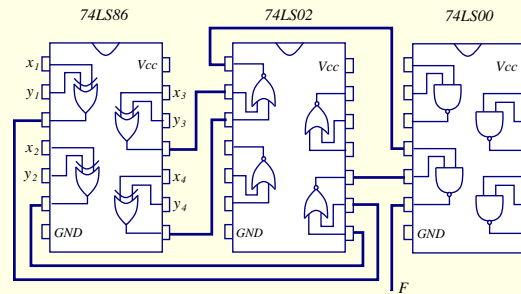
# FPGA Design Style



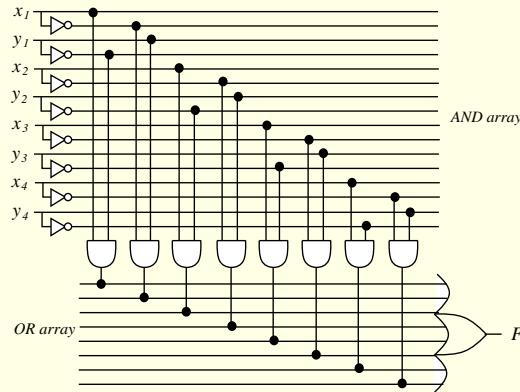
# SSI/SPLD Design Style



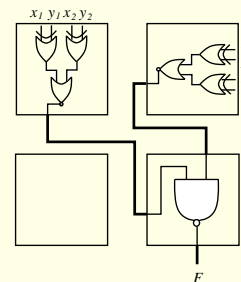
(a) 4-bit comparator.



(b) SSI implementation.



(c) SPLD (PLA) implementation.



(d) Gate array implementation.

## Comparisons of Design Styles

	Full custom	Standard cell	Gate array	FPGA	SPLD
Cell size	variable	fixed height*	fixed	fixed	fixed
Cell type	variable	variable	fixed	programmable	programmable
Cell placement	variable	in row	fixed	fixed	fixed
Interconnections	variable	variable	variable	programmable	programmable

\* Uneven height cells are also used.



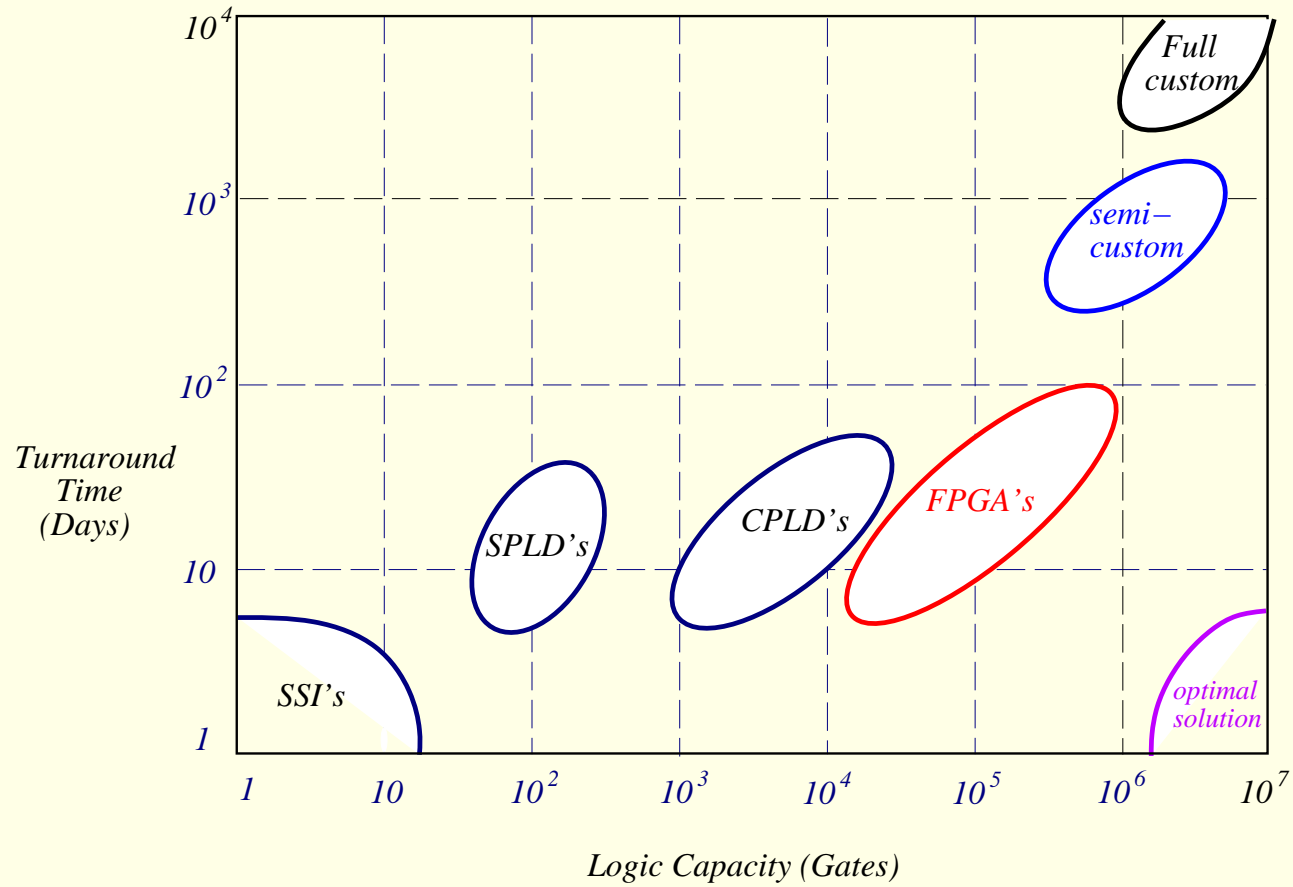
## Comparisons of Design Styles

	Full custom	Standard cell	Gate array	FPGA	SPLD
Fabrication time	---	--	+	+++	++
Packing density	+++	++	+	--	----
Unit cost in large quantity	+++	++	+	--	-
Unit cost in small quantity	---	--	+	+++	+
Easy design and simulation	---	--	-	++	+
Easy design change	---	--	-	++	++
Accuracy of timing simulation	-	-	-	+	+
Chip speed	+++	++	+	-	--

+ desirable

- not desirable

# Design-Style Trade-offs



# Algorithms 101

- Algorithm: a finite step-by-step procedure to solve a problem
- Requirements:
  - Unambiguity: can even be followed by a machine
  - Basic: each step is among a given primitives
  - Finite: it will terminate for every possible input

## A game

- An ECE major is sitting on the Northwestern beach and gets thirsty, she knows that there is an ice-cream booth along the shore of Lake Michigan but does not know where—not even north or south. How can she find the booth in the shortest distance?
- Primitives: walking a distance, turning around, etc.

## A first solution

- Select a direction, say north, and keep going until find the booth
- Suppose the booth is to the south, she will never stop... of course, with the assumption she follows a straight line, not lake shore or on earth

## Another solution

- Set the place she is sitting as the origin
- Search to south 1 yard, if not find, turn to north
- Search to north 1 yard, if not find, turn to south
- Search to south 2 yard, if not find, turn to north
- Search to north 2 yard, if not find, turn to south
- ... (follow the above pattern in geometric sequence 1, 2, 4, 8, ...)

OR

- $n = -1$ ;
- While (not find) do
  - $n = n + 1$ ;
  - Search to south  $2^n$ , and turn;
  - Search to north  $2^n$ , and turn;

## Correctness proof

- Each time when the while loop is finished, the range from south  $2^n$  to north  $2^n$  is searched. Based on the fact that the booth is at a constant distance  $x$  from the origin, it will be within a range from south  $2^N$  to north  $2^N$  for some  $N$ . With  $n$  to increment in each loop, we will find the booth in finite time.
- Is this the fastest (or shortest) way to find the booth?



## Analysis of algorithm

- Observation: the traveled distance depends on where is the booth
- Suppose the distance between the booth and the origin is  $x$
- When the algorithm stops, we should have  $2^n \geq x$  but  $2^{n-1} < x$

- The distance traveled is

$$3 \cdot 2^n + 2(2 \cdot 2^{n-1} + 2 \cdot 2^{n-2} + \dots + 2) \leq 7 \cdot 2^n$$

- which is smaller than  $14x$
- We know that the lower bound is  $x$ , can we do better?

## Complexity of an algorithm

- Two resources: running time and storage
- They are dependent on inputs: expressed as functions of input size
  - Why input size: lower bound (at least read it once)
- **Big-Oh** notation:  $f(n) = O(g(n))$  if there exist constants  $n_0$  and  $c$  such that for all  $n > n_0$ ,  $f(n) \leq c \cdot g(n)$ .
  - Make our life easy: is it  $13x$  instead of  $14x$  in our game
  - The solution is asymptotically optimal for our game

## Time complexity of an algorithm

- Run-time comparison: 1000 MIPS (Yr: 200x), 1 instr. /op.

Time	Big-Oh	$n = 10$	$n = 100$	$n = 10^3$	$n = 10^6$
500	$O(1)$	$5 \times 10^{-7}$ sec	$5 \times 10^{-7}$ sec	$5 \times 10^{-7}$ sec	$5 \times 10^{-7}$ sec
$3n$	$O(n)$	$3 \times 10^{-8}$ sec	$3 \times 10^{-7}$ sec	$3 \times 10^{-6}$ sec	0.003 sec
$n \log n$	$O(n \log n)$	$3 \times 10^{-8}$ sec	$2 \times 10^{-7}$ sec	$3 \times 10^{-6}$ sec	0.006 sec
$n^2$	$O(n^2)$	$1 \times 10^{-7}$ sec	$1 \times 10^{-5}$ sec	0.001 sec	16.7 min
$n^3$	$O(n^3)$	$1 \times 10^{-6}$ sec	0.001 sec	1 sec	$3 \times 10^5$ cent.
$2^n$	$O(2^n)$	$1 \times 10^{-6}$ sec	$3 \times 10^{17}$ cent.	-	-
$n!$	$O(n!)$	0.003 sec	-	-	-

- Polynomial-time complexity:  $O(p(n))$ , where  $n$  is the input size and  $p(n)$  is a polynomial function of  $n$ .

## Complexity of a problem

- Given a problem, what is the running time of the fastest algorithm for it?
- Upper bound: easy—find an algorithm with less time
- Lower bound: hard—every algorithm requires more time
- **P**: set of problems solvable in polynomial time
- **NP**(Nondeterministic P): set of problems whose solution can be proved in polynomial time
- Millennium open problem: **NP**  $\neq$  **P**?
  - Fact: there are a set of problems in **NP** resisting any polynomial solution for a long time (40 years)

## NP-complete and NP-hard

- Cook 1970: If the problem of boolean satisfiability can be solved in poly. time, so can all problems in **NP**.
- Such a problem with this property is called NP-hard.
- If a NP-hard problem is in **NP**, it is called NP-complete.
- Karp 1971: Many other problems resisting poly. solutions are NP-complete.

## How to deal with a hard problem

- Prove the problem is NP-complete:
  1. The problem is in NP (i.e. solution can be proved in poly. time)
  2. It is NP-hard (by polynomial reducing a NP-complete problem to it)
- Solve NP-hard problems:
  - Exponential algorithm (feasible only when the problem size is small)
    - \* Pseudo-polynomial time algorithms
  - Restriction: work on a subset of the input space

- Approximation algorithms: get a provable close-to-optimal solution
- Heuristics: get a as good as possible solution
- Randomized algorithm: get the solution with high probability

## Algorithmic Paradigms

- Divide and conquer: divide a problem into sub-problems, solve sub-problems, and combine them to construct a solution.
  - Greedy algorithm: optimal solutions to sub-problems will give optimal solution to the whole problem.
  - Dynamic programming: solutions to a larger problem are constructed from a set of solutions to its sub-problems.
- Mathematical programming: a system of optimizing an objective function under constraints functions.
- Simulated annealing: an adaptive, iterative, non-deterministic algorithm that allows “uphill” moves to escape from local optima.



- Branch and bound: a search technique with pruning.
- Exhaustive search: search the entire solution space.