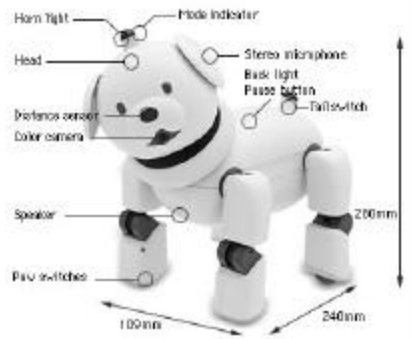


## Aibo

- 64 bit RISC CPU
- 32MB RAM
- 3 DOF head, 4 DOF legs
- 100,000 pixel image sensor
- Inferred distance sensor
- Acceleration sensor, inclination sensor, vibration sensor
- Voice recognition
- \$800



1

## Aibo Pal

- Download personality immediately
  - Happy mode
  - Media link mode
  - House guard mode



2

## Aibo Life

- Growing up
  - Baby, child, teen, adult
- Learning
  - Praise & Scolding
    - Petting/hitting head
- Training
  - Gesture
  - Motion



3

CS122A: Embedded System Design, Fall 02

## Furby Autopsy



4

CS122A: Embedded System Design, Fall 02

## Muscle

- One single reversible stepper motor
- A series of gears to move
  - Eyes, ears, eyelid, mouth, butt
  - Predetermined series of movement
  - But can move cams differently
- Dead zones
  - Can move butt without moving eyes
- Must cycle through to get to that dead zone

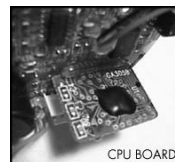


CS122A: Embedded System Design, Fall 02

5

## Nervous System

- A main PC board
  - Contain a few SSI chips, caps, resistors...
  - Pet sensors, inversion switch
- Two daughter boards
  - Each with a blob
  - Contain CPU and stuff...
- Pet switch, inversion switch, tummy switch, tongue switch, light sensor, IR send/receive, speaker, motor,



CS122A: Embedded System Design, Fall 02

6

## Head I.S18 Chip System

- Increase sweetspot
- Incredible stability
- Reduced vibration
- Greater power and control
- \$300



7

CS122A: Embedded System Design, Fall 02

## Chip system

- Self powered
- Intellifiber
  - Piezoelectric
  - Sit between electrodes, convert mechanical energy to electrical energy
  - 1ms timing constraints
- FlexCircuit
  - Transmit voltage and current to/from micro chip
- Microchip
  - Integrated in racquet handle
  - Send out signal to stiffen the raquet
  - Reduce 50% of vibration
  - Reduce twice as fast
  - ...So you can hit harder!!!



8

CS122A: Embedded System Design, Fall 02

## Administrative matter

---

- Free, catch-up, lab
  - New lab schedule online
- Design Technology (chap 11) will be taught first
  - IC Technology (chap 10) will be taught next
- Guest Lecture on 11/7
  - CS Ph.D. Candidate: Xi Chen
  - “Electronic Design Automation, System Level Design, and System Verification”
    - Potential extra credit questions



CS122A: Embedded System Design, Fall 02

9

## Design Technology

10

## Outline

---

- Automation: synthesis
- Verification: hardware/software co-simulation
- Reuse: intellectual property cores
- Design process models



## Introduction

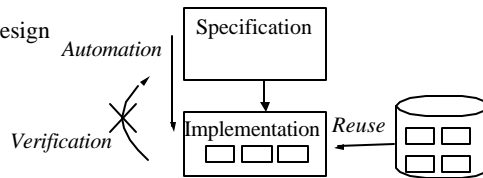
---

- Design task
  - Define system functionality
  - Convert functionality to physical implementation while
    - Satisfying constrained metrics
    - Optimizing other design metrics
- Designing embedded systems is hard
  - Complex functionality
    - Millions of possible environment scenarios
    - Competing, tightly constrained metrics
  - Productivity gap
    - As low as 10 lines of code or 100 transistors produced per day



## Improving productivity

- Design technologies developed to improve productivity
- Focus on technologies enabling hardware/software unified view
  - Automation
    - Program replaces manual design
    - Synthesis
  - Reuse
    - Predesigned components
    - Cores
    - Enable GPP and SPP on single IC
  - Verification
    - Ensuring correctness/completeness of each design step
    - Hardware/software co-simulation

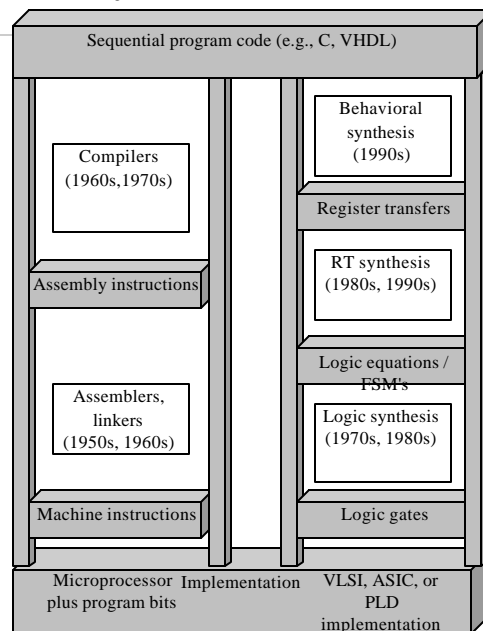


13

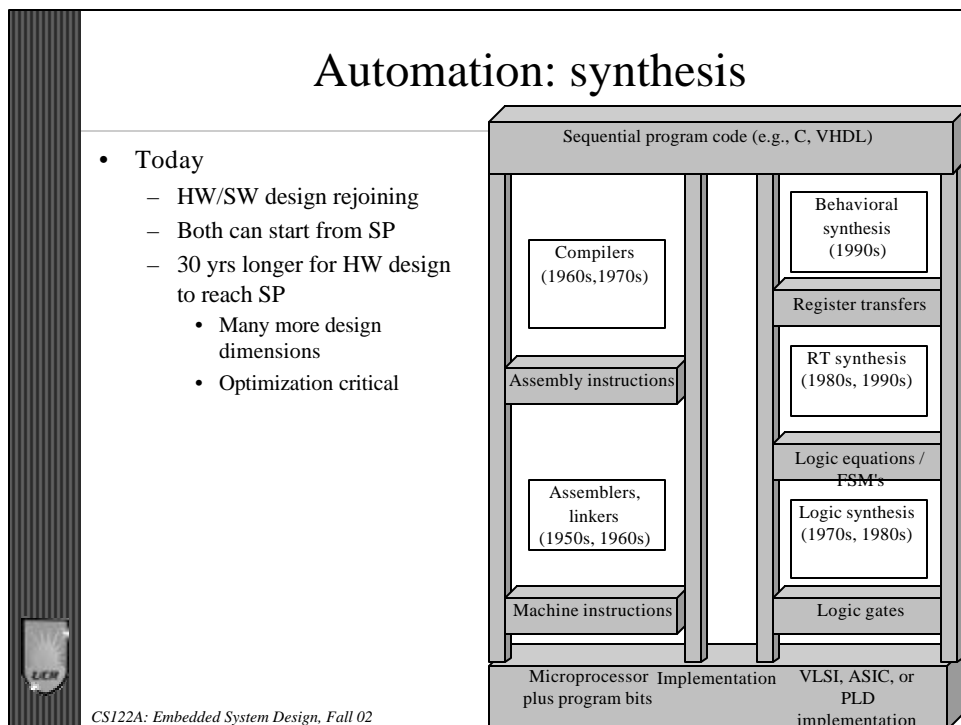
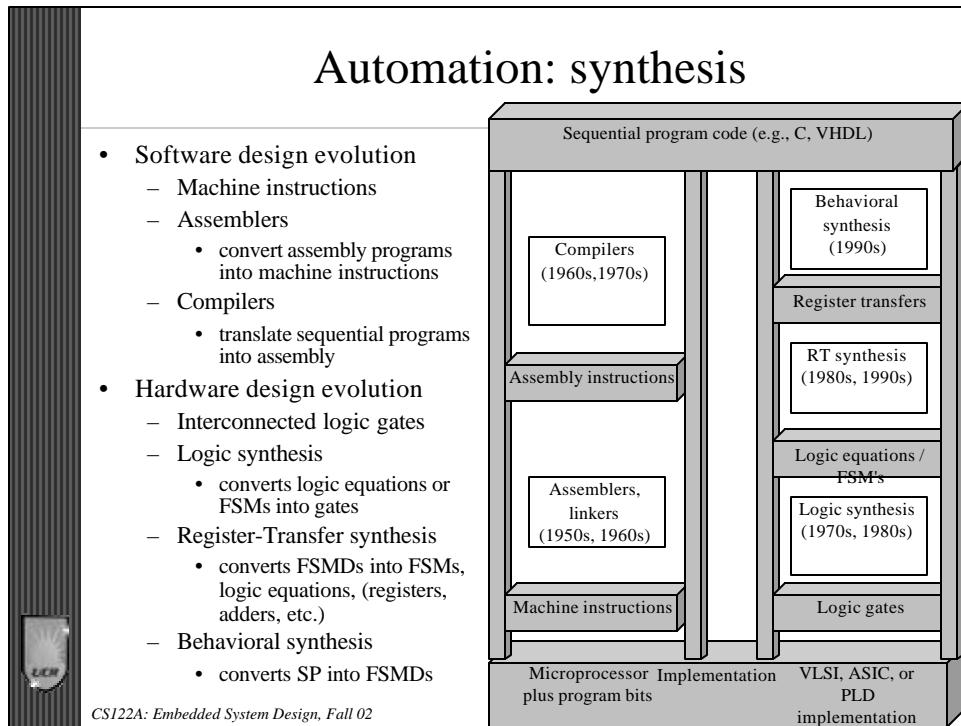
CS122A: Embedded System Design, Fall 02

## Automation: synthesis

- Early design mostly hardware
- Software complexity increased
  - advent of GPP
- SW/HW divide into 2 fields
  - Different techniques for design
- Design tools evolve
  - To higher levels of abstraction
  - Different rate in each field

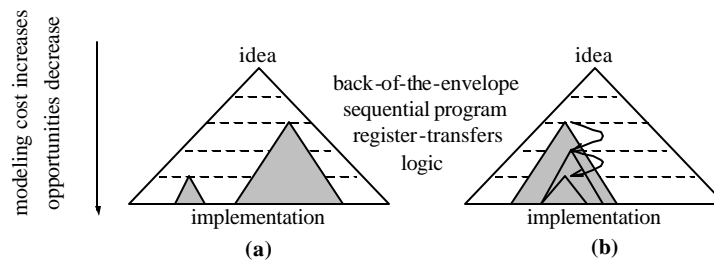


CS122A: Embedded System Design, Fall 02



## Increasing abstraction level

- Higher abstraction level focus of HW/SW design evolution
  - Description smaller/easier to capture
    - E.g., Line of sequential program code can translate to 1000 gates
  - Many more possible implementations available
    - SP designs may differ in performance/transistor count by 10 or 100
    - Logic-level designs may differ by only power of 2
- Design process proceeds to lower abstraction level
  - Narrowing in on single implementation



CS122A: Embedded System Design, Fall 02

17

## HW/SW Synthesis

- Automatically converting system's behavioral description to a structural implementation
  - Structural implementation must optimize design metrics
- More expensive, complex than pure SW compilers
  - Cost = \$100s to \$10,000s
  - User controls 100s of synthesis options
  - Optimization critical
    - Otherwise could use pure software
  - Optimizations different for each design
  - Run time = hours, days

CS122A: Embedded System Design, Fall 02

18

### Y-chart

- Axis represents type of description
  - Behavioral
    - Defines outputs as function of inputs
    - Algorithms but no implementation
  - Structural
    - Implements behavior by connecting components with known behavior
  - Physical
    - Gives size/locations of components and wires on chip/board

19

CS122A: Embedded System Design, Fall 02

### Y-chart

- Synthesis converts behavior at given level to structure at same level or lower
  - E.g.,
    - SP gates, flip-flops
    - SP transistors

20

CS122A: Embedded System Design, Fall 02

## Logic synthesis

- **Logic-level behavior to structural implementation**
  - Logic equations and/or FSM to connected gates
- **Combinational logic synthesis**
  - Two-level minimization (Sum of products/product of sums)
    - Best possible performance
    - Longest path = 2 gates (AND gate + OR gate/OR gate + AND gate)
    - Minimize size is the minimum cover that is prime
    - Exact solution is hard, heuristics are common
  - Multilevel minimization
    - Trade performance for size
    - Pareto-optimal solution for multi-objective optimization
    - Again, Exact solution is hard, heuristics are common
- **FSM synthesis**
  - State minimization
  - State encoding

21

CS122A: Embedded System Design, Fall 02

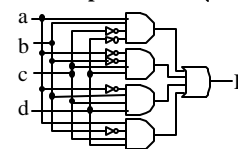
## Two-level minimization

- **Represent logic function as sum of products (or product of sums)**
  - AND gate for each product
  - OR gate for each sum
- **Gives best possible performance**
  - At most 2 gate delay
- **Goal: minimize size**
  - Minimum cover
    - Minimum # of AND gates (sum of products)
  - Minimum cover that is prime
    - Minimum # of inputs to each of the AND gates

**Sum of products**  

$$F = abc'd' + a'b'cd + a'bcd + ab'cd$$

**Direct implementation**



4 4-input AND gates  
and  
1 4-input OR gate

22

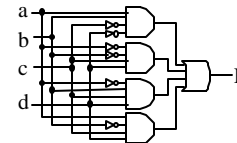
CS122A: Embedded System Design, Fall 02

## Terminology

- **Minimum # of AND gates (sum of products)** Sum of products
- **Literal: variable or its complement**
  - a or a', b or b', etc.
- **Minterm: product of literals**
  - Each literal appears exactly once
    - abc'd', ab'cd, a'bcd, etc.
- **Implicant: product of literals**
  - Each literal appears no more than once
    - abc'd', a'cd, etc.
  - Covers 1 or more minterms
    - a'cd covers a'bcd and a'b'cd
- **Cover: set of implicants that covers all minterms of function**
- **Minimum cover: cover with minimum # of implicants**

$$F = abc'd' + a'b'cd + a'bcd + ab'cd$$

Direct implementation



4 4-input AND gates  
and  
1 4-input OR gate

23

## Minimum cover: K-map approach

- **Karnaugh map (K-map)**
  - 1 represents minterm
  - Circle represents implicant
- **Minimum cover**
  - Covering all 1's with min # of circles
  - Example: direct vs. min cover
    - Less gates (4 vs. 5)
    - Less transistors

sum of products

		cd			
	ab	00	01	11	10
00		0	0	1	0
01		0	0	1	0
11		1	0	0	0
10		0	0	1	0

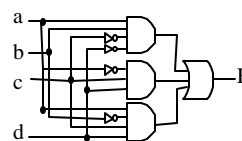
minimum cover

		cd			
	ab	00	01	11	10
00		0	0	1	0
01		0	0	1	0
11		1	0	0	0
10		0	0	1	0

Minimum cover

$$F = abc'd' + a'cd + ab'cd$$

Minimum cover implementation



2 4-input AND gates  
1 3-input AND gates  
1 4 input OR gate

24

## Minimum cover that is prime

- Minimum # of inputs to AND gates

- Prime implicant

- Implicant not covered by any other implicant
- Max-sized circle in K-map

K-map: minimum cover that is prime

		cd			
	ab	00	01	11	10
00		0	0	1	0
01		0	0	1	0
11		1	0	0	0
10		0	0	1	0

- Minimum cover that is prime

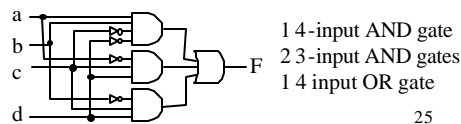
- Covering with min # of prime implicants
- Min # of max-sized circles
- Example: prime cover vs. min cover

Minimum cover that is prime

$$F = abc'd' + a'cd + b'cd$$

- Same # of gates (4 vs 4)
- Less transistors

Implementation



25

CS122A: Embedded System Design, Fall 02

## Minimum cover: heuristics

- K-maps give optimal solution every time

- Functions with > 6 inputs too complicated
- Use computer-based tabular method
  - Finds all prime implicants
  - Finds min cover that is prime
  - Also optimal solution every time
  - Problem:  $2^n$  minterms for n inputs
  - E.g. 32 inputs = 4 billion minterms, exponential complexity

- Heuristic

- Solution technique where optimal solution not guaranteed
- Hopefully comes close, at least in practice

26

CS122A: Embedded System Design, Fall 02

## Heuristics: iterative improvement

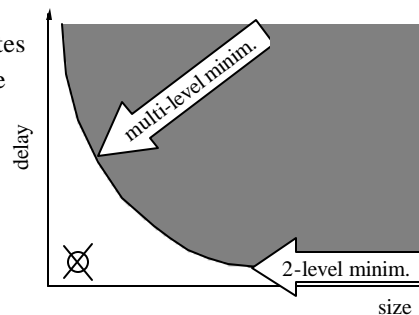
- Start with initial solution
  - ie., original logic equation (arrive perhaps with greedy methods)
- Repeatedly make modifications toward better solution
- Common modifications
  - Expand
    - Replace each nonprime implicant with a prime implicant covering it
    - Delete all implicants covered by new prime implicant
  - Reduce
    - Opposite
  - Reshape
    - Expands one implicant while reducing another
    - Maintains total # of implicants
  - Irredundant
    - A proper subset of a cover may still be a cover
    - Selects min # of implicants that cover from existing implicants
- Synthesis tools differ in modification “scripts”

27

CS122A: Embedded System Design, Fall 02

## Multilevel logic minimization

- Trade performance for size
  - Increase delay for lower # of gates
  - Gray area represents all possible solutions
  - Circle with X represents ideal solution
    - Generally not possible
  - 2-level gives best performance
    - max delay = 2 gates
    - Solve for smallest size
  - Multilevel gives pareto-optimal solution
    - Minimum delay for a given size
    - Minimum size for a given delay

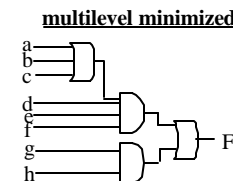
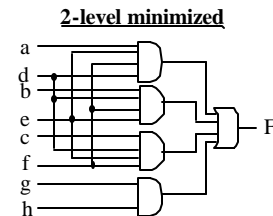


28

CS122A: Embedded System Design, Fall 02

## Example

- Minimized 2-level logic function:
  - $F = adef + bdef + cdef + gh$
  - Requires 5 gates with 18 total gate inputs
    - 4 ANDS and 1 OR
- After algebraic manipulation:
  - $F = (a + b + c)def + gh$
  - Requires only 4 gates with 11 total gate inputs
    - 2 ANDS and 2 ORs
    - Less inputs per gate
    - Assume gate inputs = 2 transistors
      - 36 ( $18 * 2$ ) down to 22 ( $11 * 2$ )
      - Reduced by 14 transistors
    - Sacrifices performance for size
      - Inputs a, b, and c now have 3-gate delay
- Iterative improvement heuristic commonly used



29

CS122A: Embedded System Design, Fall 02

## FSM synthesis

- FSM to gates
- State minimization  $O(n \log n)$ 
  - Reduce # of states
    - Identify and merge equivalent states
  - Equivalent states
    - Outputs, next states same for all possible inputs
  - Tabular method gives exact solution
    - Table of all possible state pairs
    - If n states,  $n^2$  table entries
    - Heuristics used with large # of states
- State encoding (NP-Complete)
  - Unique bit sequence for each state
  - If n states,  $\log_2(n)$  bits
  - $n!$  possible encodings
  - Heuristics common

30

CS122A: Embedded System Design, Fall 02

## Technology mapping

- Library of gates available for implementation
  - Simple
    - only 2-input AND,OR gates
  - Complex
    - various-input AND,OR,NAND,NOR,etc. gates
    - Efficiently implemented meta-gates (ie., AND-OR-INVERT,MUX)
- Final structure consists of specified library's components only
- If technology mapping integrated with logic synthesis
  - More efficient circuit
  - More complex problem
  - Heuristics required, may result in sub-optimal solution
- Crucial for “pre-fabricated technologies”
  - Gate array, standard cell, FPGA

31

CS122A: Embedded System Design, Fall 02

## Complexity impact on user

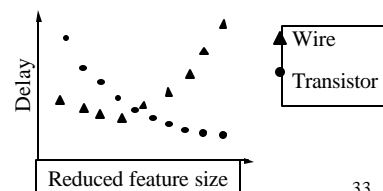
- As complexity grows, heuristics used
- Heuristics differ tremendously among synthesis tools
  - Computationally expensive (slow heuristics)
    - Higher quality results
    - Long run times (hours, days)
    - Requires huge amounts of memory
    - Typically needs to run on servers, workstations
  - Fast heuristics (computationally challenged)
    - Lower quality results
    - Shorter run times (minutes, hours)
    - Smaller amount of memory required
    - Could run on PC
- Super-linear-time (ie.  $n^3$ ) heuristics usually used
  - User can partition large systems to reduce run times/size
  - $100^3 > 50^3 + 50^3$  ( $1,000,000 > 250,000$ )
    - But the result may be suboptimal: can't optimize across boundary

32

CS122A: Embedded System Design, Fall 02

## Integrating logic design and physical design

- Past
  - Gate delay much greater than wire delay
  - Thus, performance evaluated as # of levels of gates only
- Today
  - Gate delay shrinking as feature size shrinking
  - Wire delay increasing
    - Performance evaluation needs wire length
  - Transistor placement impact wire length
    - Integrating with physical design
    - Wire-planning (simultaneous routing and logic synthesis)
  - Simultaneous logic synthesis and physical design required



33

CS122A: Embedded System Design, Fall 02

## Register-transfer synthesis

- Converts FSM to custom single-purpose processor
  - Datapath
    - Register units to store variables (Complex data types)
    - Functional units (Arithmetic operations)
    - Connection units (Busses, MUXs)
  - FSM controller
    - Controls datapath
- Key subproblems:
  - Allocation
    - Instantiate storage, functional, connection units
  - Binding
    - Mapping FSM operations to specific units

34

CS122A: Embedded System Design, Fall 02

## Behavioral synthesis

- High-level synthesis
- Converts single sequential program to GPP and SPP
- Key subproblems
  - Allocation
  - Binding
  - Scheduling
    - Assign sequential program's operations to states
    - Conversion template for SPP (earlier)
- Optimizations important
  - Compiler
    - Constant propagation, dead-code elimination, loop unrolling
  - Advanced techniques for allocation, binding, scheduling



CS122A: Embedded System Design, Fall 02

35

## System synthesis

- Convert 1 or more processes into 1 or more processors (system)
  - For complex embedded systems
    - Multiple processes may provide better performance/power
    - May be better described using concurrent sequential programs
- Tasks
  - Transformation
    - Can merge 2 exclusive processes into 1 process
    - Can break 1 large process into separate processes
    - Procedure inlining
    - Loop unrolling
  - Allocation
    - Design of system architecture
    - Select processors to implement processes
    - Also select memories and busses



CS122A: Embedded System Design, Fall 02

36

## System synthesis

---

- **Tasks (cont.)**
  - Partitioning
    - Mapping 1 or more processes to 1 or more processors
    - Mapping variables to memories
    - Mapping communications to buses
  - Scheduling
    - Multiple processes on a single processor
    - Memory accesses
    - Bus communications
- **Synthesis driven by constraints**
  - E.g., Meet performance requirements at minimum cost
    - Allocate as much behavior as possible to general-purpose processor
    - Low-cost/flexible implementation
    - Minimum # of SPPs used to meet performance



CS122A: Embedded System Design, Fall 02

37

## System synthesis

---

- **System synthesis for GPP only (software)**
  - Common for decades
    - Multiprocessing
    - Parallel processing
    - Real-time scheduling
- **Hardware/software codesign**
  - Simultaneous consideration of GPPs/SPPs during synthesis
  - Made possible by maturation of behavioral synthesis in 1990's



CS122A: Embedded System Design, Fall 02

38

## Temporal vs. spatial thinking

- Design thought process changed by evolution of synthesis
- Before synthesis
  - Designers worked primarily in structural domain
    - Connecting simpler components to build more complex systems
    - Connecting logic gates to build controller
    - Connecting registers, MUXs, ALUs to build datapath
  - “capture and simulate” era
    - Capture using CAD tools
    - Simulate to verify correctness before fabricating
  - Spatial thinking
    - Structural diagrams
    - Data sheets
  - Bottom-up methodologies



CS122A: Embedded System Design, Fall 02

39

## Temporal vs. spatial thinking

- After synthesis
  - Designers work primarily in behavioral domain
  - “describe and synthesize” era
    - Describe FSMs or sequential programs
    - Synthesize into structure
  - Top-down methodologies
  - Temporal thinking
    - States or sequential statements have relationship over time
- Strong understanding of hardware structure still important
  - Behavioral description must synthesize to efficient structural implementation



CS122A: Embedded System Design, Fall 02

40