CSE370: Introduction to Digital Design

- Course staff
  - Carl Ebeling
  - TAs: Corey Olson (grad), Steven Lockhart (ugrad)
  - Student lab assistants in 003
- Course web
  - www.cs.washington.edu/370/
  - You should already be on the mailing list
    - cse370a_sp10@u.washington.edu
- Course text
  - Contemporary Logic Design, 2e, Katz/Borriello, Prentice-Hall
- Today's agenda
  - Class administration and overview of course web
  - Course objectives and approach
  - A brief introduction to the course

Why are you here?

- Obvious reasons
  - this course is part of the CS/CompE requirements
  - it is the implementation basis for all modern computing devices
    - building large things from small components
      - computers = transistors + wires - it's all in how they are interconnected
    - provide a model of how a computer works
- More important reasons
  - the inherent parallelism in hardware is your first exposure to parallel computation
  - it offers an interesting counterpoint to programming and is therefore useful in furthering our understanding of computation

What will we learn in CSE370?

- The language of logic design
  - Boolean algebra, logic minimization, state, timing, CAD tools
- The concept of state in digital systems
  - analogous to variables and program counters in software systems
- How to specify/simulate/compile/realize our designs
  - hardware description languages
  - tools to simulate the workings of our designs
  - logic compilers to synthesize the hardware blocks of our designs
  - mapping onto programmable hardware
- Contrast with programming
  - sequential and parallel implementations
  - specify algorithm as well as computing/storage resources it will use

What is logic design?

- What is design?
  - given a specification of a problem, come up with a way of solving it choosing appropriately from a collection of available components
  - while meeting some criteria for size, cost, power, beauty, elegance, etc.
- What is logic design?
  - determining the collection of digital logic components to perform a specified control and/or data manipulation and/or communication function and the interconnections between them
  - which logic components to choose? – there are many implementation technologies (e.g., off-the-shelf fixed-function components, programmable devices, transistors on a chip, etc.)
  - the design may need to be optimized and/or transformed to meet design constraints

Applications of logic design

- Conventional computer design
  - CPUs, busses, peripherals
- Networking and communications
  - phones, modems, routers
- Embedded products
  - in cars, toys, appliances, entertainment devices
- Scientific equipment
  - testing, sensing, reporting

What is digital hardware?

- Physical way to represent different states (typically two: “0” and “1”)
  - example: digital logic where voltage < 0.8v is a “0” and > 2.0v is a “1”
  - example: pair of transmission wires where a “0” or “1” is distinguished by which wire has a higher voltage (differential)
  - example: orientation of magnetization signifies a “0” or a “1”
- A way to save values
  - Change a saved value
  - Sense a stored value
- A way to perform logical functions on values
  - example: if two wires are both “1”, make another be “1” (AND)
  - example: if at least one of two wires is “1”, make another be “1” (OR)
  - example: if a wire is “1”, make another be “0” (NOT)
- A way to send values from one function to another
  - examples: wire, optical fiber, radio, chemical pathway

Binary numbers in C/Java

- 00010101
- 10000101

- Count the number of 1's in the binary representation of a C/ Java integer
  - int foo;

- Count the number of bits in a C/Java variable
  - char foo;
  - short foo;
  - long foo;

- Find the most significant 1 in a C/Java integer

What is happening now in digital design?

- Important trends in how industry does hardware design
  - larger and larger designs
  - shorter and shorter time to market
  - cheaper and cheaper products
  - design time often dominates cost
- Scale
  - pervasive use of computer-aided design tools over hand methods
  - multiple levels of design representation
- Time
  - emphasis on abstract design representations
  - programmable rather than fixed function components
  - automatic synthesis techniques
  - importance of sound design methodologies
- Cost
  - higher levels of integration
  - use of simulation to debug designs
  - simulate and verify before you build

CSE 370: concepts/skills/abilities

- Understanding the basics of logic design (concepts)
- Understanding sound design methodologies (concepts)
- Modern specification methods (concepts)
- Familiarity with a full set of CAD tools (skills)
- Realize digital designs in an implementation technology (skills)
- Appreciation for the differences and similarities (abilities) in hardware and software design

<u>New ability:</u> to accomplish the logic design task with the aid of computer-aided design tools and map a problem description into an implementation with programmable logic devices after validation via simulation and understanding of the advantages/disadvantages as compared to a software implementation

Representation of digital designs

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

scope of CSE 370

Computation: abstract vs. implementation

- Up to now, computation has been a mental exercise (paper, programs)
- This class is about physically implementing computation using physical devices that use voltages to represent logical values
- Basic units of computation are:
  - representation: "0", “1" on a wire
set of wires (e.g., for binary ints)
  - assignment: x = y
  - data operations: x + y – 5
  - control:

sequential statements:	A; B; C;
conditionals:	if x == 1 then y;
loops:	for (i = 1 ; i == 10, i++) {…}
procedures:	A; proc(...); B;

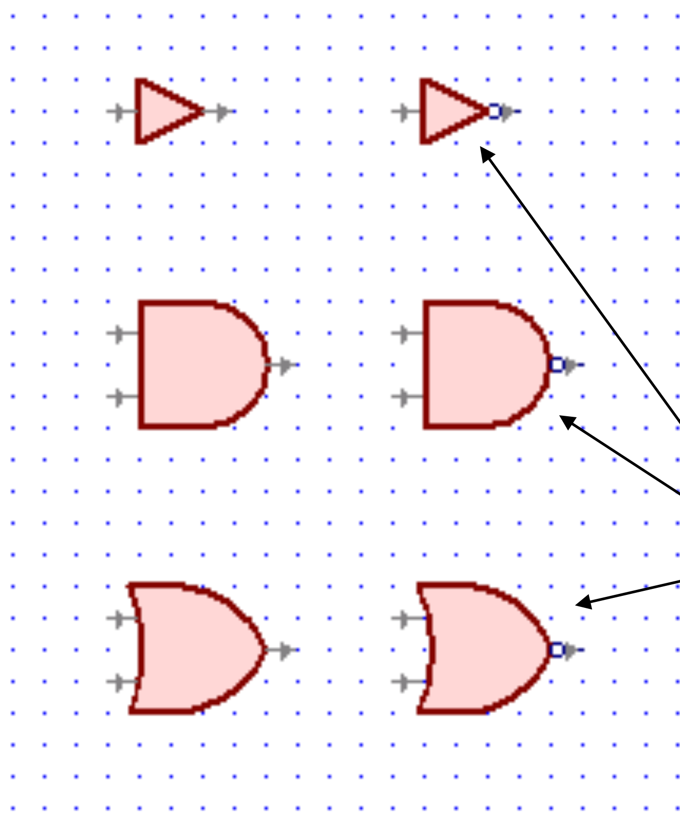
- We will study how each of these are implemented in hardware and composed into computational structures

Class components

- Combinational logic
  - $output_t = F(input_t)$
- Sequential logic
  - $output_t = F(output_{t-1}, input_t)$
    - output dependent on history
    - concept of a time step (clock)
- Basic computer architecture
  - how a CPU executes instructions
- Tools to make our job easier/efficient
  - designs that work the first time
  - designs that are efficient and easy to change/maintain

Combinational logic

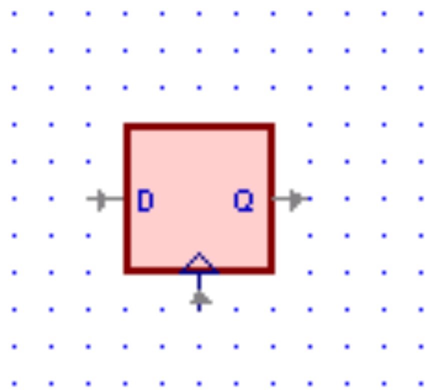
- Common combinational logic elements are called logic gates
  - Buffer, NOT
  - AND, NAND
  - OR, NOR



easy to implement
with CMOS transistors

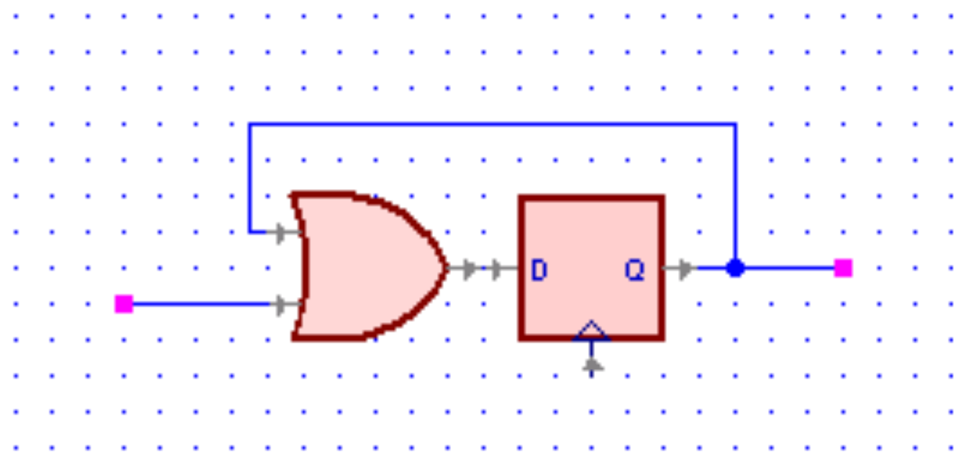
Sequential logic

- Common sequential logic elements are called flip-flops
  - Flip-flops only change their output after a clocking event



Mixing combinational and sequential logic

- What does this very simple circuit do?



Combinational or sequential?

assignment:	x = y;
data operations:	x + y – 5
sequential statements:	A; B; C;
conditionals:	if x == 1 then y;
loops:	for (i = 1 ; i == 10, i++) {…}
procedures/methods:	A; proc(...); B;

A quick combinational logic example

- Calendar subsystem: number of days in a month (to control watch display)
  - used in controlling the display of a wrist-watch LCD screen

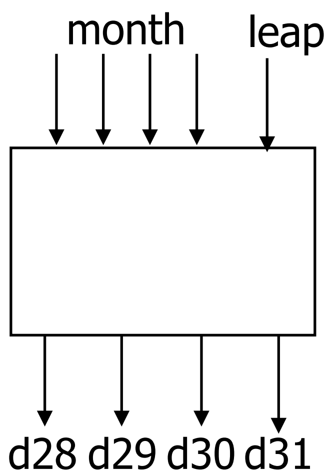
  - inputs: month, leap year flag
  - outputs: number of days

Implementation in software

```
integer number_of_days ( month, leap_year_flag) {
   switch (month) {
      case 1: return (31);
      case 2: if (leap_year_flag == 1) then return (29)
         else return (28);
      case 3: return (31);
      ...
      case 12: return (31);
      default: return (0);
   }
}
```

Implementation as a combinational digital system

- Encoding:
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31
- Behavior:
  - combinational
  - truth table specification



month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0	0	1
1011	–	0	0	1	0
1100	–	0	0	0	1
1101	–	–	–	–	–
1110	–	–	–	–	–
1111	–	–	–	–	–

Combinational example (cont'd)

- Truth-table to logic to switches to gates
  - d28 = "1 when month=0010 and leap=0"
  - d28 = m8'•m4'•m2•m1'•leap'
  - d31 = "1 when month=0001 or month=0011 or ... month=1100"
  - d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) + ... (m8•m4•m2'•m1')
  - d31 = can we simplify more?

symbol for <u>not</u>

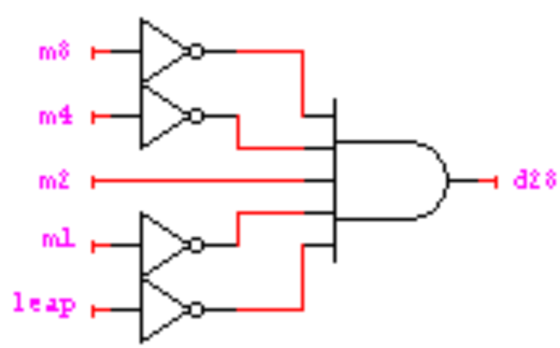
symbol for <u>and</u>

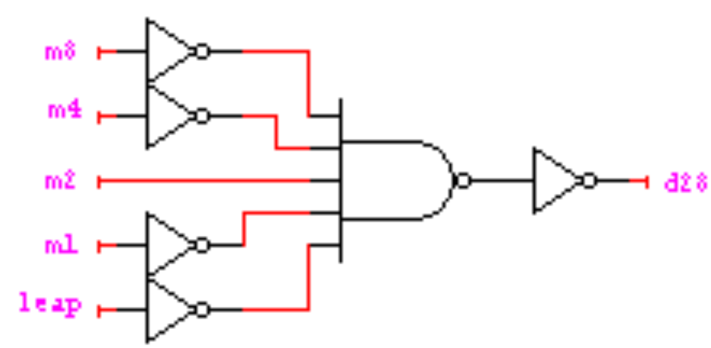
symbol for <u>or</u>

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
...					
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–

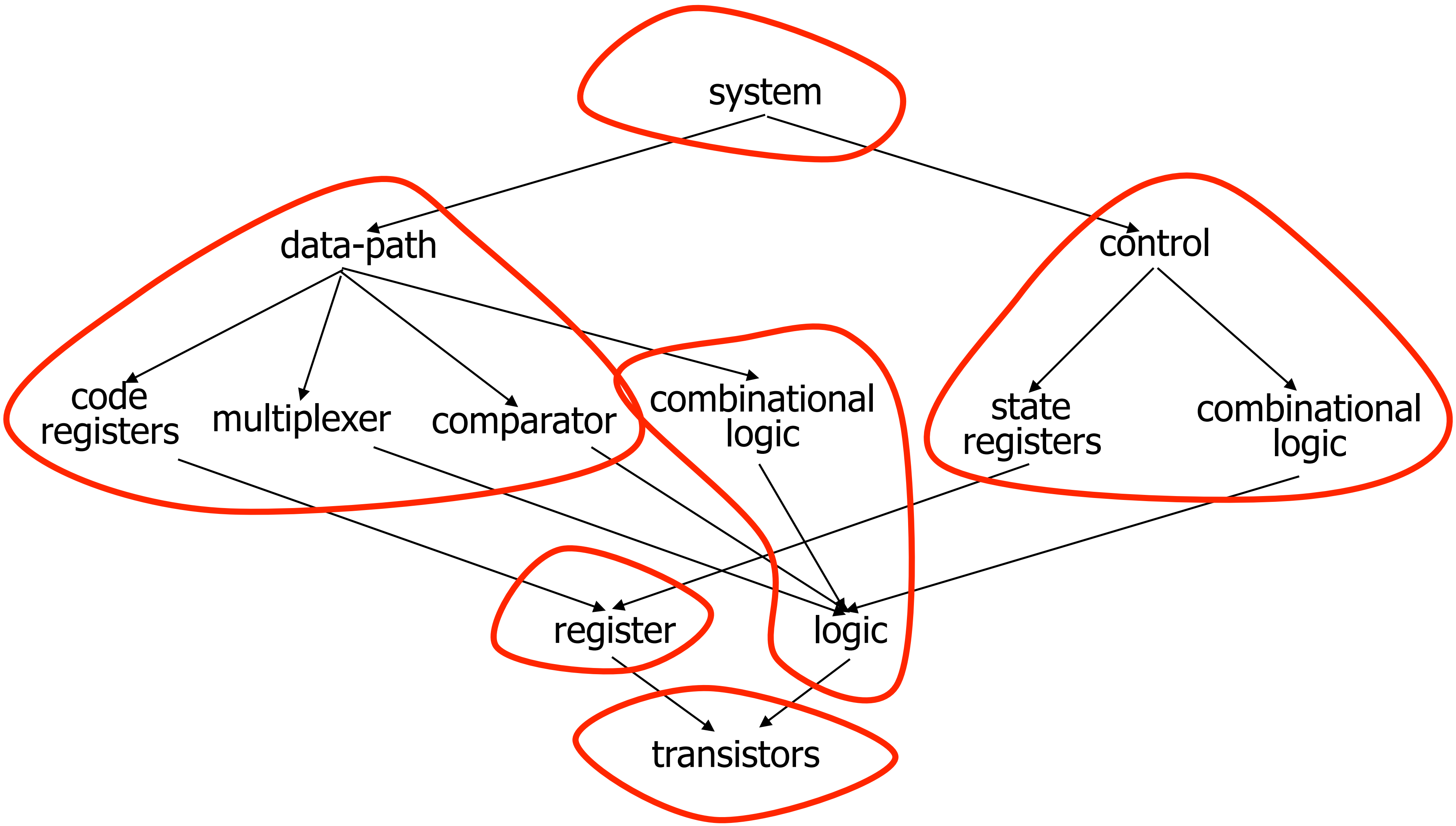
Combinational example (cont'd)

- d28 = m8'•m4'•m2•m1'•leap'
- d29 = m8'•m4'•m2•m1'•leap
- d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
  (m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
  = (m8'•m4•m1') + (m8•m4'•m1)
- d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
  (m8'•m4•m2'•m1) + (m8'•m4•m2•m1) +
  (m8•m4'•m2'•m1') + (m8•m4'•m2•m1') +
  (m8•m4•m2'•m1')





Design hierarchy



Challenge problems

- Write fast code for the following:
  - Count the number of 1’s in a C/Java integer
  - Find the most significant 1 in a C/Java integer
- Write code that swaps two int’s without any extra temp variable