# Welcome to EECS 150: Components and Design Techniques for Digital Systems

- Course staff
  - Randy Katz (Instructor), Po-Kai Chen (Head TA)
  - Teaching Assistants: Bryan Brady, Jay Chen, Brian Gawalt, Jack Tzeng
  - Readers: David Lin, Kevin Lin
- Course web
  - inst.eecs.Berkeley.edu/~eecs150 (coming soon)
- This week
  - What is logic design?
  - What is digital hardware?
  - What will we be doing in this class?
  - Quick Review
  - Class administration, overview of course web, and logistics



# Why Are We Here?

- Implementation basis for modern computing devices
  - Constructing large systems from small components
  - Another view of a computer: controller + datapath
- Inherent parallelism in hardware
  - Parallel computation beyond 61C
- Counterpoint to software design
  - Furthering our understanding of computation

## We Will Learn in EECS 150 …

- Language of logic design
  - Logic optimization, state, timing, CAD tools
- Concept of state in digital systems
  - Analogous to variables and program counters in software systems
- Hardware system building
  - Datapath + control = digital systems
- Hardware system design methodology
  - Hardware description languages: *Verilog*
  - Tools to simulate design behavior: output = function (inputs)
  - Logic compilers synthesize hardware blocks of our designs
  - Mapping onto programmable hardware (code generation)
- Contrast with software design
  - Both map specifications to physical devices
  - Both must be flawless…the price we pay for using discrete math

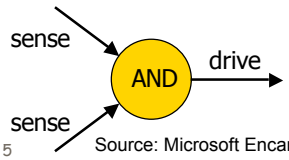


## What is Logic Design?

- What is design?
  - Given problem spec, solve it with available components
  - While meeting quantitative (size, cost, power) and qualitative (beauty, elegance)
- What is logic design?
  - Choose digital logic components to perform specified control, data manipulation, or communication function and their interconnection
  - Which logic components to choose?
    Many implementation technologies (fixed-function components, *programmable devices*, individual transistors on a chip, etc.)
  - Design optimized/transformed to meet design constraints

## What is Digital Hardware?

- Devices that sense/control wires carrying digital values (physical quantity interpreted as "0" or "1")
  - Digital logic: voltage < 0.8v is "0", > 2.0v is "1"
  - Pair of wires where "0"/"1" distinguished by which has higher voltage (differential)
  - Magnetic orientation signifies "0" or "1"
- Primitive digital hardware devices
  - Logic computation devices (sense and drive)
    - Two wires both "1" - make another be "1" (AND)
    - At least one of two wires "1" - make another be "1" (OR)
    - A wire "1" - then make another be "0" (NOT)
  - Memory devices (store)
    - Store a value
    - Recall a value previously stored



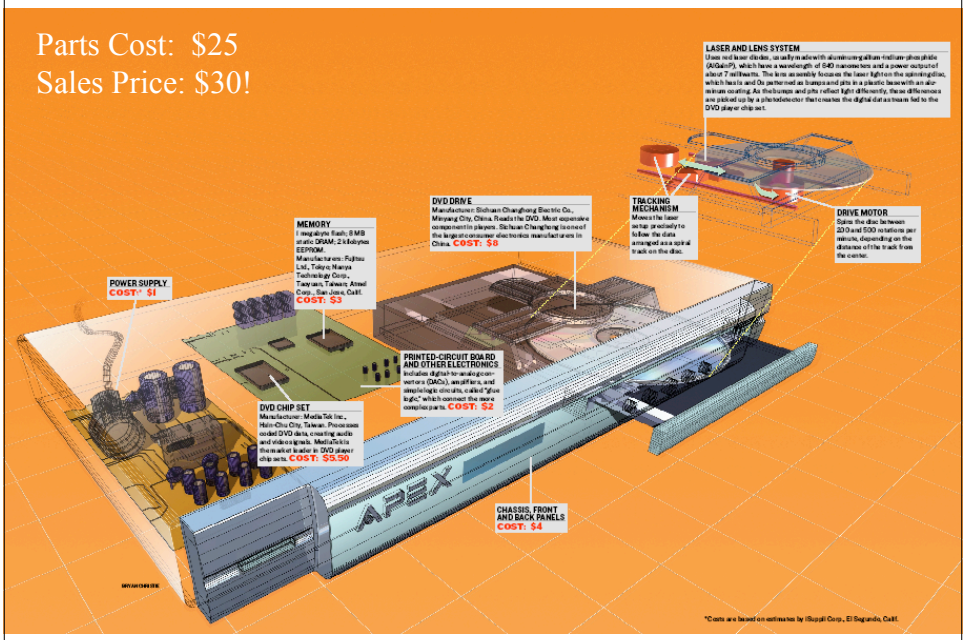

Source: Microsoft Encarta



## What is the Current State of Digital Design?

- Changes in industrial practice
  - Larger designs
  - Shorter time to market
  - Cheaper products
- 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

$39 DVD Player@Amazon.com






# CS 150: Concepts/Skills/Abilities

- Basics of logic design (concepts)
- Sound design methodologies (concepts)
- Modern specification methods (concepts)
- Familiarity with full set of CAD tools (skills)
- Appreciation for differences and similarities (abilities) in hardware and software design

New ability: perform logic design with computer-aided design tools, validating that design via simulation, and mapping its implementation into programmable logic devices;
Appreciating the advantages/disadvantages hw vs. sw implementation

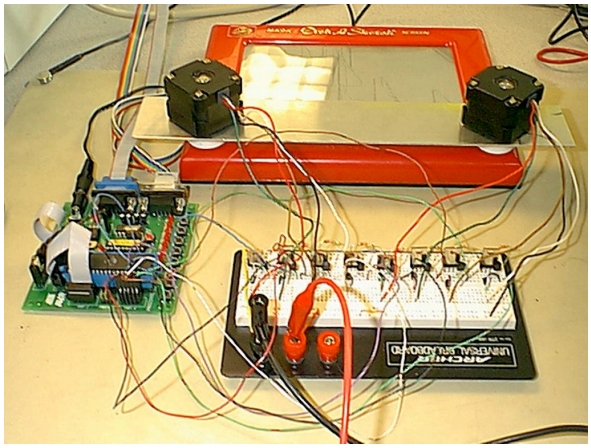
## Administrative Details

- See course web page for gory details!
  - MW 1-2:30 course lecture, F 2-3 lab lecture
  - 1x3 hour lab, 1x1=hour discussion per week
  - No labs or discussions first week!
- Grading
  - Midterm Exams (28 Sep, 9 Nov): 20%
  - Final Exam (16 Dec): 20%
  - Labs (1-5): 15%
  - Project (Etch-a-Sketch): 30%
  - Homeworks (10 problem sets): 10%
  - In-class pop quizzes: 5%
    - First one NOW: Diagnostic Quiz (not graded!)

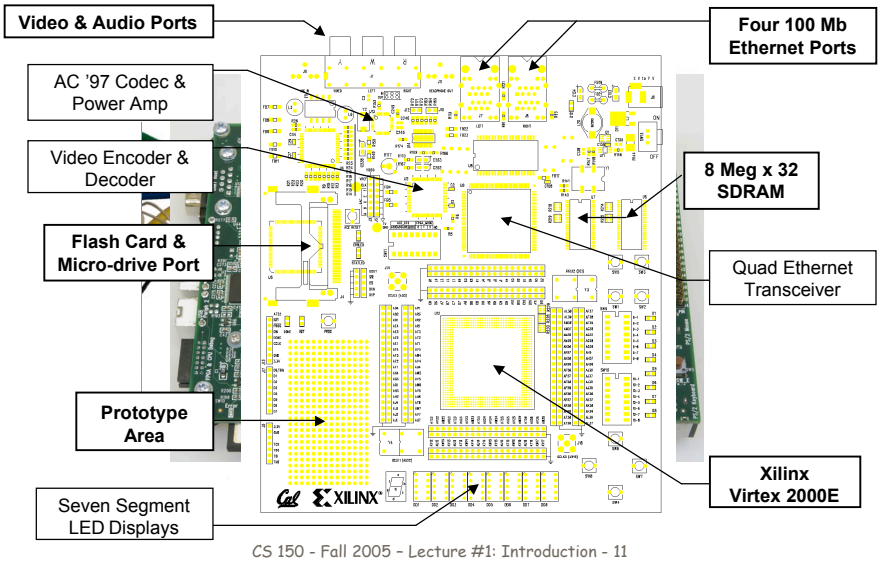



## Course Project: Electronic Etch-a-Sketch

- Not quite this … but:
  - Game controller interface
  - CRT video I/f
  - Pen effects
    - E.g., Color
    - E.g., Width
  - Implemented in a Xilinx FPGA on the Calinx boards you will use in lab
  - Groups of two



## Calinx EECS 150 Lab/Project Protoboard



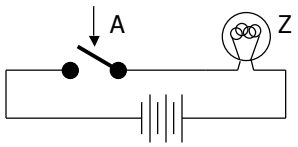



## Computation: Abstract vs. Implementation

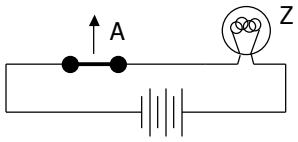
- Computation as a mental exercise (paper, programs)
- vs. implementation with physical devices using voltages to represent logical values
- Basic units of computation:
  - Representation: "0", "1" on a wire; set of wires (e.g., for binary integers)
  - Assignment: x = y
  - Data operations: x + y – 5
  - Control:
    - Sequential statements: A; B; C
    - Conditionals: if x == 1 then y
    - Loops: for ( i = 1 ; i == 10, i++)
    - Procedures: A; proc(...); B;
- Study how these are implemented in hardware and composed into computational structures

## 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)

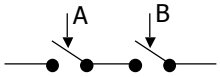
Z = A



## Switches (cont'd)

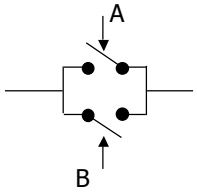
- Compose switches into more complex ones (Boolean functions):

AND




Z = A and B

OR




Z = A or B

## Switching Networks

- Switch settings
  - Determine whether conducting path exists to light the bulb
- To build larger computations
  - Use bulb (output of the network) to set other switches (inputs to another network)
- Interconnect switching networks
  - Construct larger switching networks, i.e., connect outputs of one network to the inputs of the next.

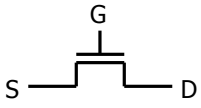


## Transistor Networks

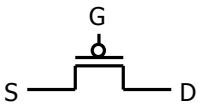
- Modern digital systems designed in CMOS
  - MOS: Metal-Oxide on Semiconductor
  - C for complementary: normally-open and normally-closed switches
- MOS transistors act as voltage-controlled switches
  - Similar, though easier to work with, than relays.

## MOS Transistors

- 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



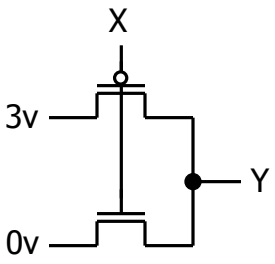

n-channel
open when voltage at G is low
closes when:
voltage(G) > voltage (S) + ε




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



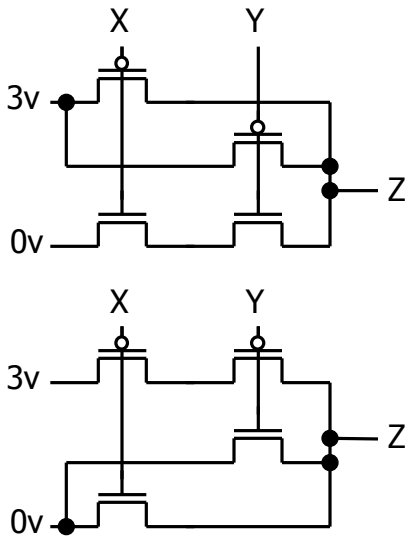
## MOS Networks




what is the relationship between x and y?

| x | y |
|---|---|
| 0 volts | |
| 3 volts | |

## Two Input Networks




what is the relationship between x, y and z?

| x | y | z |
|---|---|---|
| 0 volts | 0 volts | |
| 0 volts | 3 volts | |
| 3 volts | 0 volts | |
| 3 volts | 3 volts | |



## 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

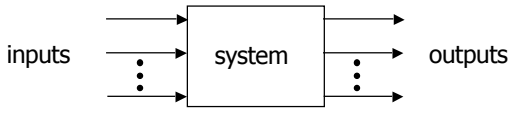
## Mapping Physical to Binary World

| 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 |



## Combinational vs. Sequential Digital Circuits

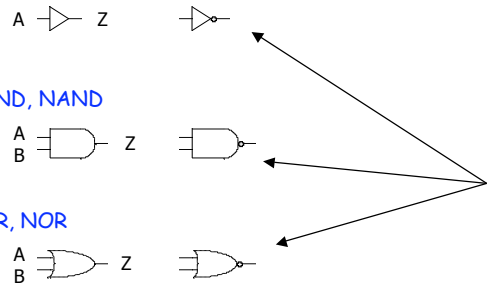
- Simple model of a digital system is a unit with inputs and outputs:




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

## Combinational Logic Symbols

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT
  - AND, NAND
  - OR, NOR




Easy to implement with CMOS transistors (the switches we have available and use most)



## Sequential Logic

- Sequential systems
  - Exhibit behaviors (output values) that depend on current as well as previous inputs
- Time response of 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

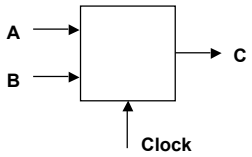
# Synchronous Sequential Digital Systems

- Combinational 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 external periodic signal (the *clock*)
  - Clock period is time between state changes sufficiently long so that system reaches steady-state before next state change



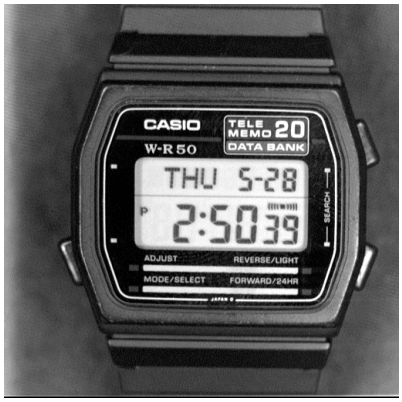
# Distinction: Combinational vs. Sequential Logic

- Combinational:
  - Input A, B
  - Wait for clock edge
  - Observe C
  - Wait for another clock edge
  - Observe C again: will stay the same
- Sequential:
  - Input A, B
  - Wait for clock edge
  - Observe C
  - Wait for another clock edge
  - Observe C again: *may be different*



## Example: Combinational Design

- 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



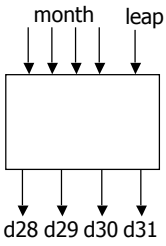



## Implementation in Software

```
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 | – | – | – | – | – |
| 111– | – | – | – | – | – |



# 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 or

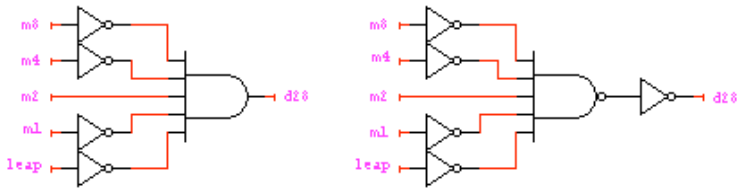
symbol for and

symbol for not

| month | leap | d28 | d29 | d30 | d31 |
|---|---|---|---|---|---|
| 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– | – | – | – | – | – |
| 0000 | – | – | – | – | – |

## Combinational Example (cont'd)

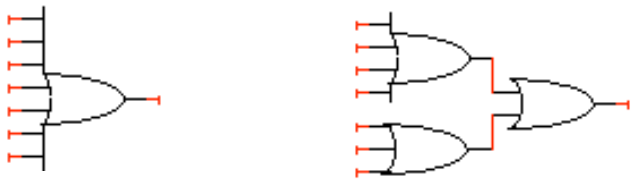
- d28 = m8'·m4'·m2·m1'·leap'
- d29 = m8'·m4'·m2·m1'·leap
- d30 = (m8'·m4·m2'·m1') + (m8'·m4·m2·m1') + (m8·m4'·m2'·m1) + (m8·m4'·m2·m1)
- d31 = (m8'·m4'·m2'·m1) + (m8'·m4'·m2·m1) + (m8'·m4·m2'·m1) + (m8'·m4·m2·m1) + (m8·m4'·m2'·m4') + (m8·m4'·m2·m1') + (m8·m4·m2'·m1')






## 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)
- d31 = (m8'·m4'·m2'·m1) + (m8'·m4'·m2·m1) + (m8'·m4·m2'·m1) + (m8'·m4·m2·m1) + (m8·m4'·m2'·m4') + (m8·m4'·m2·m1') + (m8·m4·m2'·m1')



## Example: Sequential Design

- Door combination lock:
  - Punch in 3 values in sequence and the door opens; if there is an error the lock must be reset; once the door opens the lock must be reset
  - Inputs: sequence of input values, reset
  - Outputs: door open/close
  - Memory: must remember combination
    or always have it available as an input

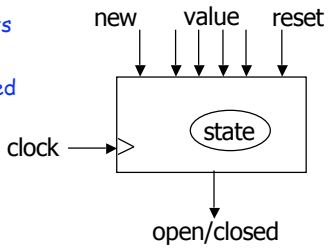


## Implementation in Software

```
integer combination_lock ( ) {
    integer v1, v2, v3;
    integer error = 0;
    static integer c[3] = 3, 4, 2;

    while (!new_value( ));
    v1 = read_value( );
    if (v1 != c[1]) then error = 1;

    while (!new_value( ));
    v2 = read_value( );
    if (v2 != c[2]) then error = 1;

    while (!new_value( ));
    v3 = read_value( );
    if (v2 != c[3]) then error = 1;

    if (error == 1) then return(0); else return (1);
}
```

# Implementation as a Sequential Digital System

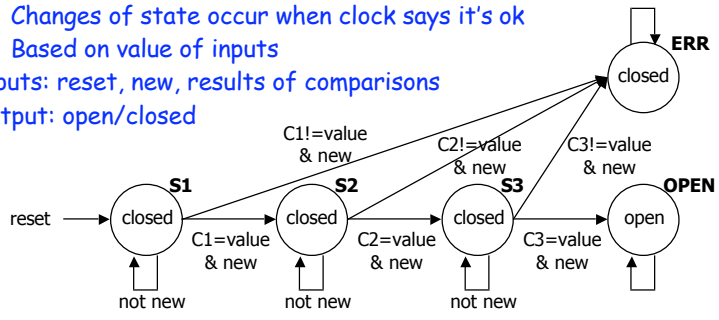
- Encoding:
  - How many bits per input value?
  - How many values in sequence?
  - How do we know a new input value is entered?
  - How do we represent the states of the system?
- Behavior:
  - Clock wire tells us when it's ok to look at inputs (i.e., they have settled after change)
  - Sequential: sequence of values must be entered
  - Sequential: remember if an error occurred
  - Finite-state specification





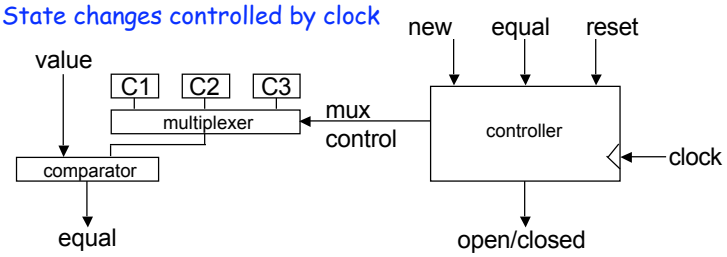

# Sequential Example (cont'd): Abstract Control

- Finite state diagram
  - States: 5 states
    - Represent point in execution of machine
    - Each state has outputs
  - Transitions: 6 from state to state, 5 self transitions, 1 global
    - Changes of state occur when clock says it's ok
    - Based on value of inputs
  - Inputs: reset, new, results of comparisons
  - Output: open/closed



# Sequential Example (cont'd): Datapath vs. Control

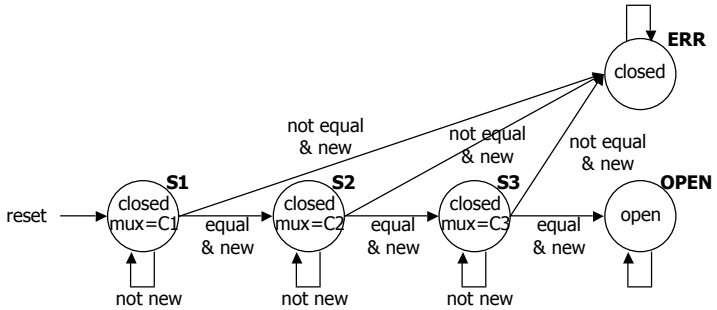
- Internal structure
  - Data-path
    - Storage for combination
    - Comparators
  - Control
    - Finite state machine controller
    - Control for data-path
    - State changes controlled by clock





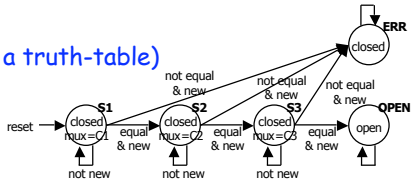

# Sequential Example (cont'd): Finite State Machine

- Finite-state machine
  - Refine state diagram to include internal structure



# Sequential Example (cont'd): Finite State Machine

- Finite State Machine
  - Generate state table (much like a truth-table)




| reset | new | equal | state | next state | mux | open/closed |
|---|---|---|---|---|---|---|
| 1 | - | - | - | S1 | C1 | closed |
| 0 | 0 | - | S1 | S1 | C1 | closed |
| 0 | 1 | 0 | S1 | ERR | - | closed |
| 0 | 1 | 1 | S1 | S2 | C2 | closed |
| 0 | 0 | - | S2 | S2 | C2 | closed |
| 0 | 1 | 0 | S2 | ERR | - | closed |
| 0 | 1 | 1 | S2 | S3 | C3 | closed |
| 0 | 0 | - | S3 | S3 | C3 | closed |
| 0 | 1 | 0 | S3 | ERR | - | closed |
| 0 | 1 | 1 | S3 | OPEN | - | open |
| 0 | - | - | OPEN | OPEN | - | open |
| 0 | - | - | ERR | ERR | - | closed |



# Sequential Example (cont'd): Encoding

- Encode state table
  - State can be: S1, S2, S3, OPEN, or ERR
    - needs at least 3 bits to encode: 000, 001, 010, 011, 100
    - and as many as 5: 00001, 00010, 00100, 01000, 10000
    - choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - Output mux can be: C1, C2, or C3
    - needs 2 to 3 bits to encode
    - choose 3 bits: 001, 010, 100
  - Output open/closed can be: open or closed
    - needs 1 or 2 bits to encode
    - choose 1 bits: 1, 0

# Sequential Example (cont'd): Encoding

- Encode state table
  - State can be: S1, S2, S3, OPEN, or ERR
    - Choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - Output mux can be: C1, C2, or C3
    - Choose 3 bits: 001, 010, 100
  - Output open/closed can be: open or closed
    - Choose 1 bits: 1, 0

| reset | new | equal | state | next state | mux | open/closed |
|---|---|---|---|---|---|---|
| 1 | - | - | - | 0001 | 001 | 0 |
| 0 | 0 | - | 0001 | 0001 | 001 | 0 |
| 0 | 1 | 0 | 0001 | 0000 | - | 0 |
| 0 | 1 | 1 | 0001 | 0010 | 010 | 0 |
| 0 | 0 | - | 0010 | 0010 | 010 | 0 |
| 0 | 1 | 0 | 0010 | 0000 | - | 0 |
| 0 | 1 | 1 | 0010 | 0100 | 100 | 0 |
| 0 | 0 | - | 0100 | 0100 | 100 | 0 |
| 0 | 1 | 0 | 0100 | 0000 | - | 0 |
| 0 | 1 | 1 | 0100 | 1000 | - | 1 |
| 0 | - | - | 1000 | 1000 | - | 1 |
| 0 | - | - | 0000 | 0000 | - | 0 |

good choice of encoding!

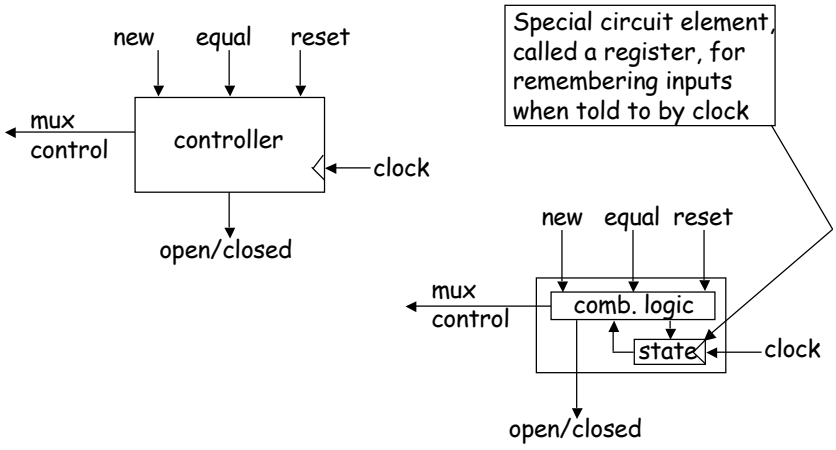
mux is identical to last 3 bits of state

open/closed is identical to first bit of state

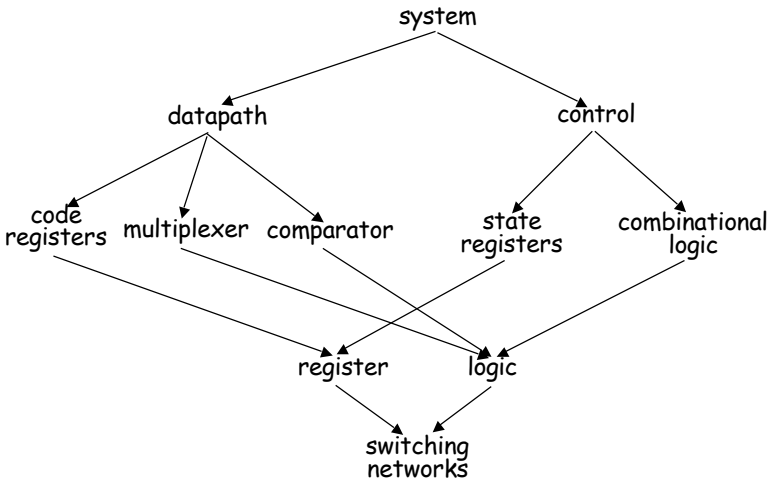


# Sequential Example (cont'd): Controller Implementation

- Controller Implementation



## Design Hierarchy






## Summary

- What the entire course is about
  - Converting solutions to problems into combinational and sequential networks effectively organizing the design hierarchically
  - Doing so with a modern set of design tools that lets us handle large designs effectively
  - Taking advantage of optimization opportunities
- Now let's do it again
  - this time we'll take the rest of the semester!