Chapter 1
Introduction

Why study logic design?

- it is the implementation basis for all modern computing devices
  - building large things from small components
  - provide a model of how a computer works
- More important reasons
  - the inherent parallelism in hardware is often our first exposure to parallel computation
  - it offers an interesting counterpoint to software design and is therefore useful in furthering our understanding of computation, in general

What will we learn in this class? (1/2)

- The basics 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

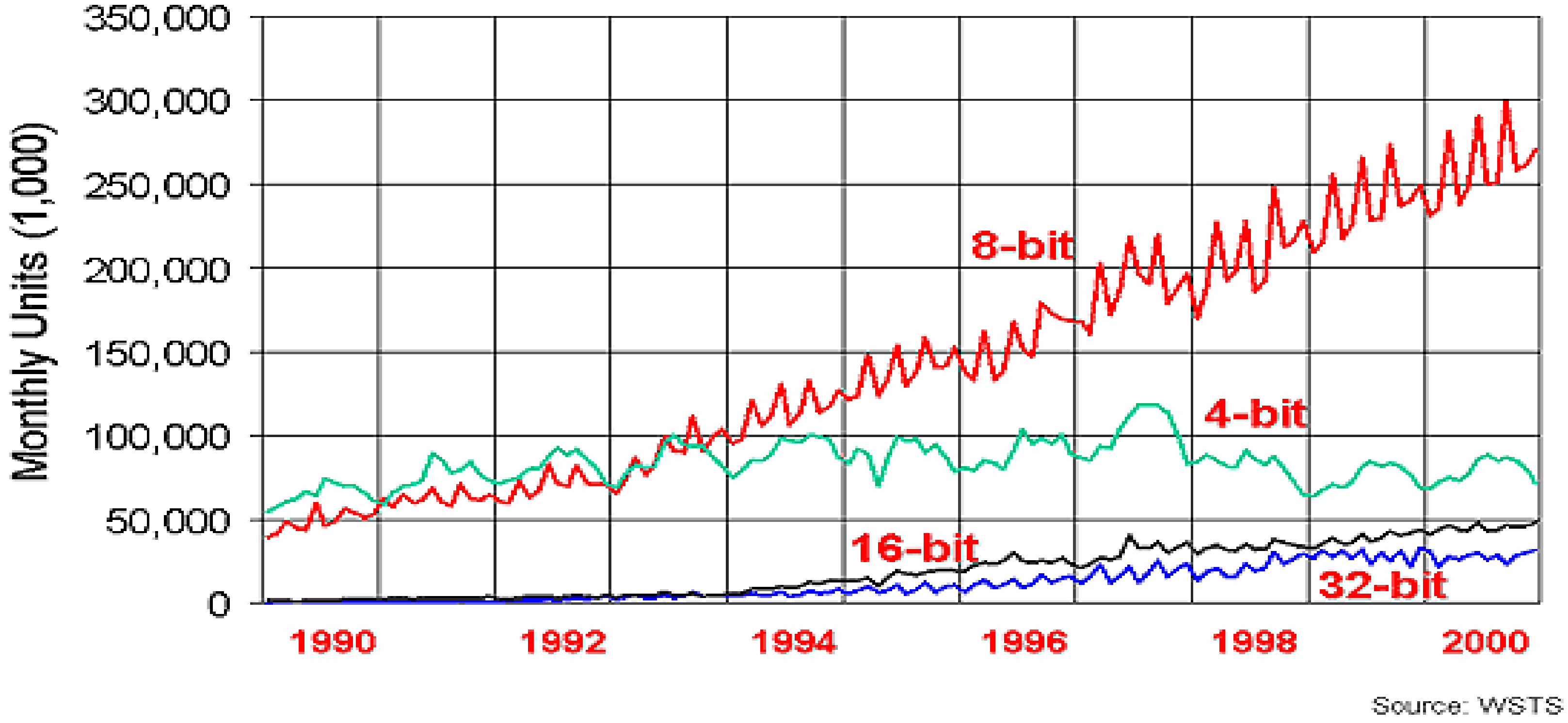
What will we learn in this class? (2/2)

- How to specify/simulate/compile/realize our designs
  - hardware description languages (HDLs)
  - tools to simulate the workings of our designs
  - logic compilers to synthesize the hardware blocks of our designs
  - mapping onto programmable hardware
- Contrast with software design
  - sequential and parallel implementations
  - specify algorithm as well as computing/storage resources it will use

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
- The world of computing is much bigger than just PCs!

Global processor/controller market



PC CPUs constitute less than 2% of the market

What is logic design? (1/2)

- Digital hardware consists of components
- Components or building blocks
  - Switches built from semiconductor transistors
    - Most basic element
  - Higher level circuits such as logic gates and memories
- A logic designer should choose the right component to solve logic design problems
- Constraints: size, cost, performance, power consumption
  - Cost vs. size

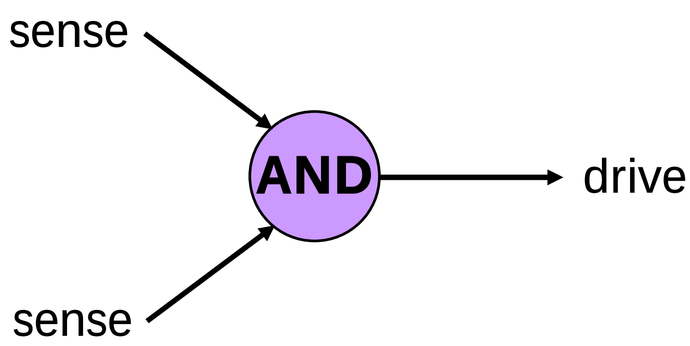
* A circuit is an interconnected collection of switches

What is logic design? (2/2)

- Each component has
  - a set of input wires
  - a set of output wires
  - Each wire is set to some analog voltage value
    - But will be interpreted as either 1 or 0 (digital abstraction)
- Transistors react to the voltage levels on the input wires
  - Switch their state and cause a change in output wires
  - At macro scale, a component that contains transistors reacts to input voltage values
- Depending on the way a circuit reacts to the input voltages
  - Combinational logic circuits
  - Sequential logic circuits

What is digital hardware?

- Collection of devices that sense and/or control wires, which carry a digital value (i.e., a physical quantity that can be interpreted as a "0" or "1")
  - example: digital logic where voltage < 0.8v is a "0" and > 2.0v is a "1"
- Primitive digital hardware devices
  - logic computation devices (sense and drive)
    - are two wires both "1" - make another be "1" (AND)
    - is at least one of two wires "1" - make another be "1" (OR)
    - is a wire "1" - then make another be "0" (NOT)
  - memory devices (store)
    - store a value
    - recall a previously stored value



Computation: abstract vs. implementation

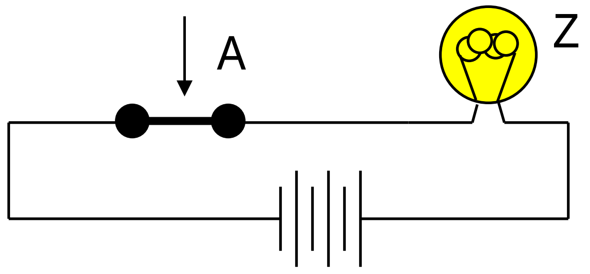
- Computation has been a mental exercise (paper, programs)
- This class is about physically implementing computation using physical devices that use voltages to represent logical values
- Basic units of computation are:
  - representation: "0", "1" on a wire
set of wires (e.g., binary integer)
  - assignment: 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;

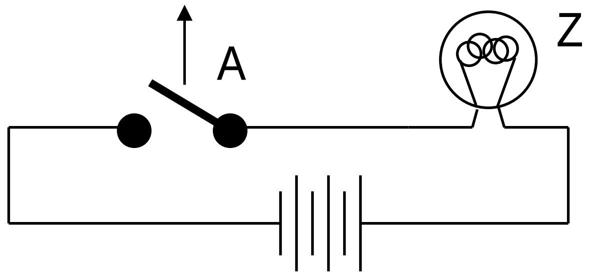
- We will study how each of these are implemented in hardware and composed into computational structures

Switches: basic element of physical implementations

- Implementing a simple circuit



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)

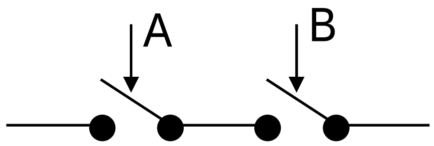
$$Z \equiv A$$

Z and A are equivalent boolean variables

Switches (cont'd)

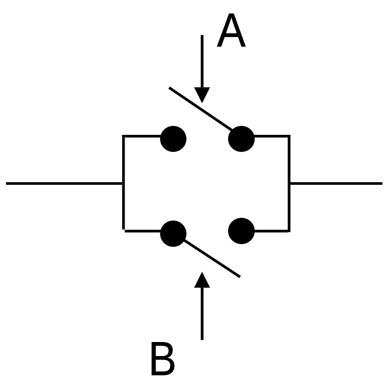
- Compose switches into more complex ones (Boolean functions):

AND



$Z \equiv$ A and B

OR



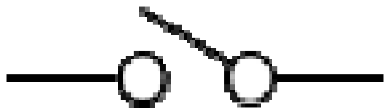
$Z \equiv$ A or B

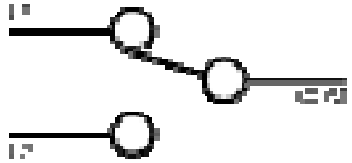
Switching networks

- Switch settings
  - determine whether or not a conducting path exists to light the light bulb
- To build larger computations
  - use a light bulb (output of the network) to set other switches (inputs to another network).
- Connect together switching networks
  - to construct larger switching networks, i.e., there is a way to connect outputs of one network to the inputs of the next.

A switch?

- A mechanical switch





- A semiconductor switch or transistor

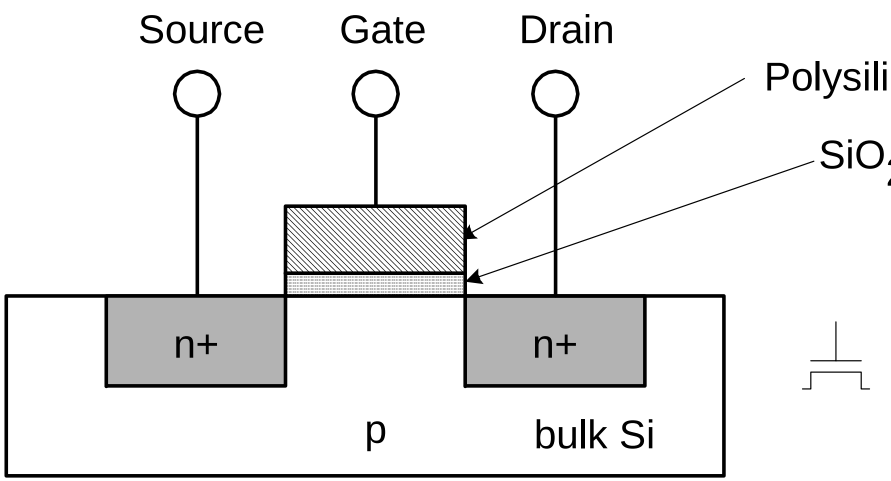
Transistor networks

- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and normally-closed switches: nMOS and pMOS
- MOS transistors act as voltage-controlled switches

* CMOS: complementary metal-oxide semiconductor

n-channel (or n-type) MOS (nMOS) circuit

- Three terminals: source-gate-drain (or S-G-D for short)
- three layers: polysilicon (used to be metal) – SiO2 - substrate



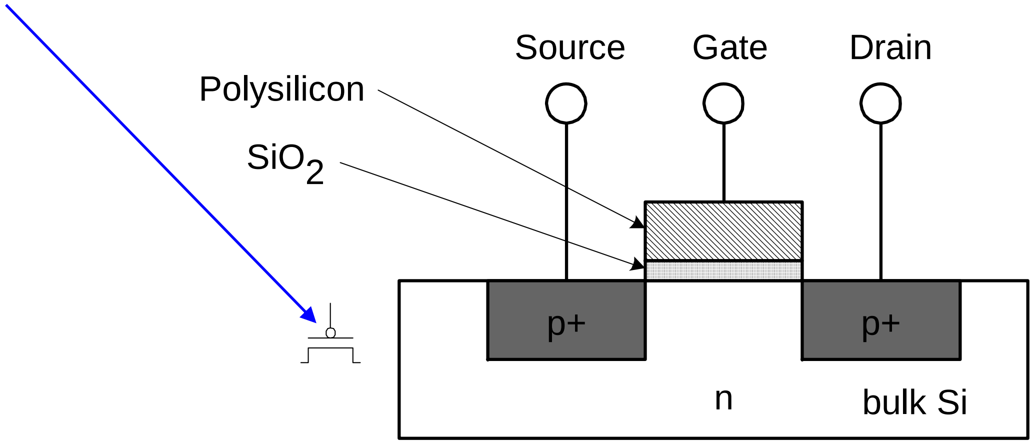
* oxide 산화물

- If G is at positive voltage, electrons in the substrate will move toward G terminal, which sets up a channel between S and D
  - And D is at high voltage, current will flow from drain to source
- Metal is replaced by polysilicon which is more adhesive

* n+: heavily doped n-type semiconductor

p-channel (or p-type) MOS (pMOS) circuit

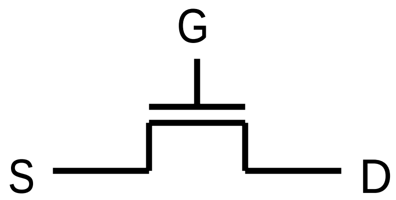
- Three terminals: source-gate-drain (or S-G-D for short)
- Same principle, but reverse doping and voltage
  - Source (Vss) is positive with regard to drain (Vdd)
- Bubble indicates the inverted behavior



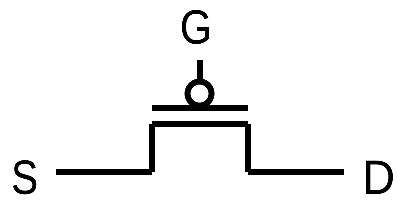
- If G is at positive voltage, the current does not flow
- If G is at ground level, the current flows

MOS transistors

- MOS transistors have three terminals: drain, gate, and source
  - they act as switches in the following way:
    if the voltage on the gate terminal is (some amount) higher/lower than the source terminal, then a conducting path will be established between the drain and source terminals

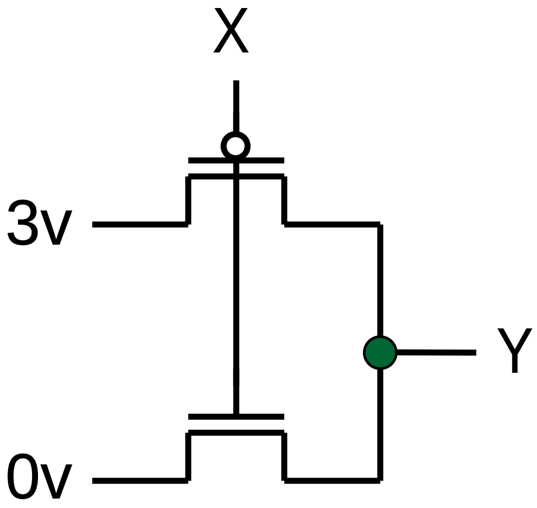


n-channel
open when voltage at G is low
closed when voltage at G is high



p-channel
closed when voltage at G is low
open when voltage at G is high

MOS networks

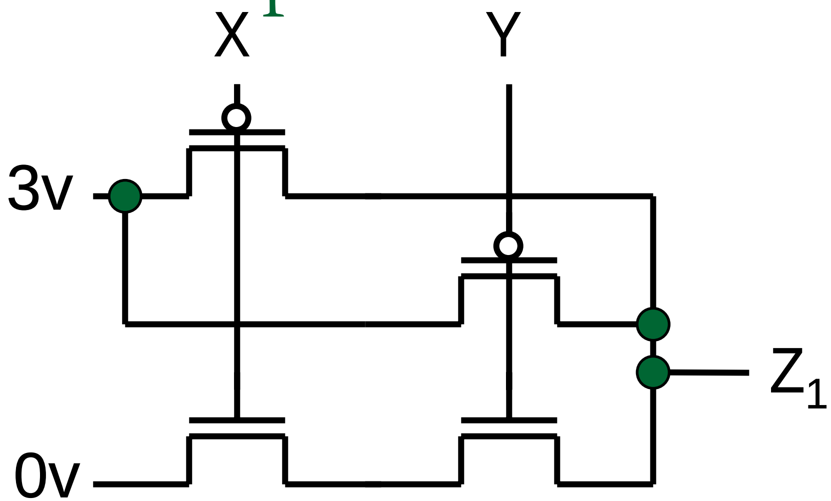


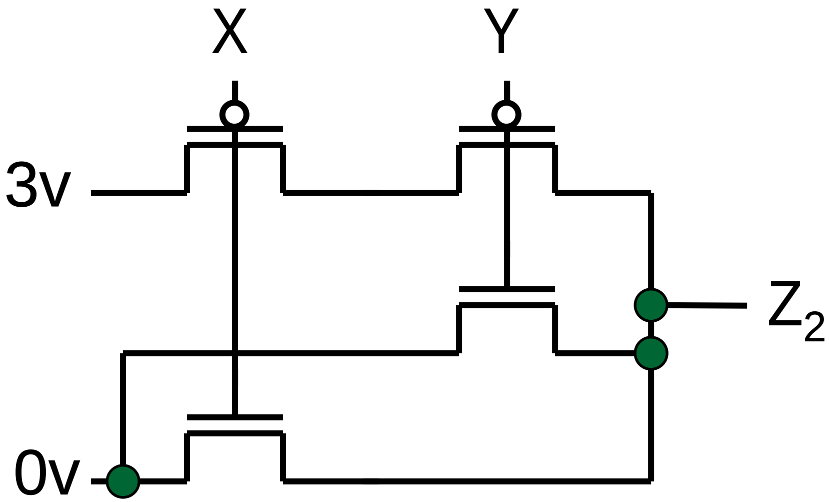
what is the relationship between x and y?

x	y
0 volts	
3 volts	

- A simple component is made up of two transistors
  - X: input
  - Y: output
  - What is this function?
- In CMOS circuits, pMos and nMOS are used in pair

Two input networks



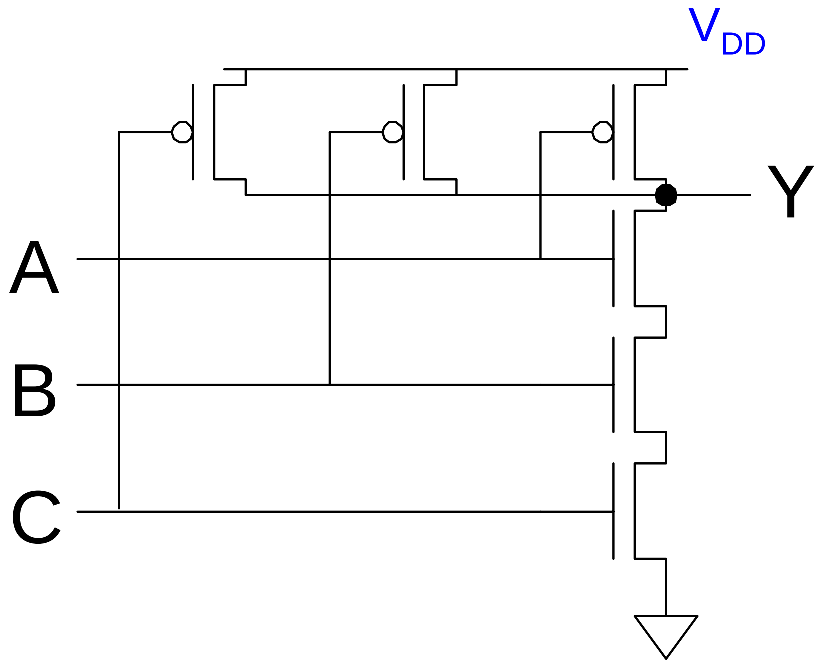


what is the relationship between x, y and z1/z2?

x	y	z1	z2
0 volts	0 volts		
0 volts	3 volts		
3 volts	0 volts		
3 volts	3 volts		

Three input NAND gate

- Y pulls low if ALL inputs are 1
- Y pulls high if ANY input is 0



- In general, the more inputs, the more transistors (TRs)
- In CMOS, a variable requires a pair of TRs

Digital vs. analog

- Convenient to think of digital systems as having only discrete, digital, input/output values
- In reality, real electronic components exhibit continuous, analog behavior

P.14

- Why do we make the digital abstraction anyway?
  - switches operate this way
  - easier to think about a small number of discrete values
  - Quantization error, though
- Why does it work?
  - does not propagate small errors in values
  - always resets to 0 or 1

Combinational logic symbols

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT

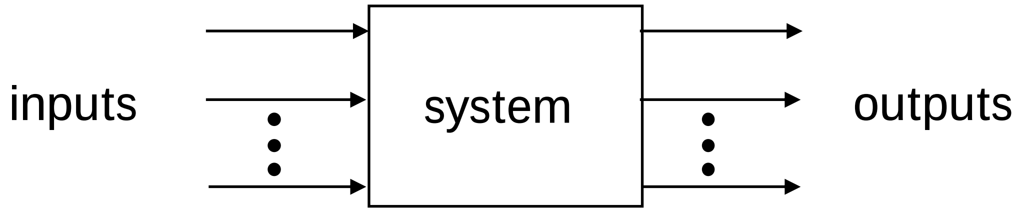
  - AND, NAND

  - OR, NOR

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

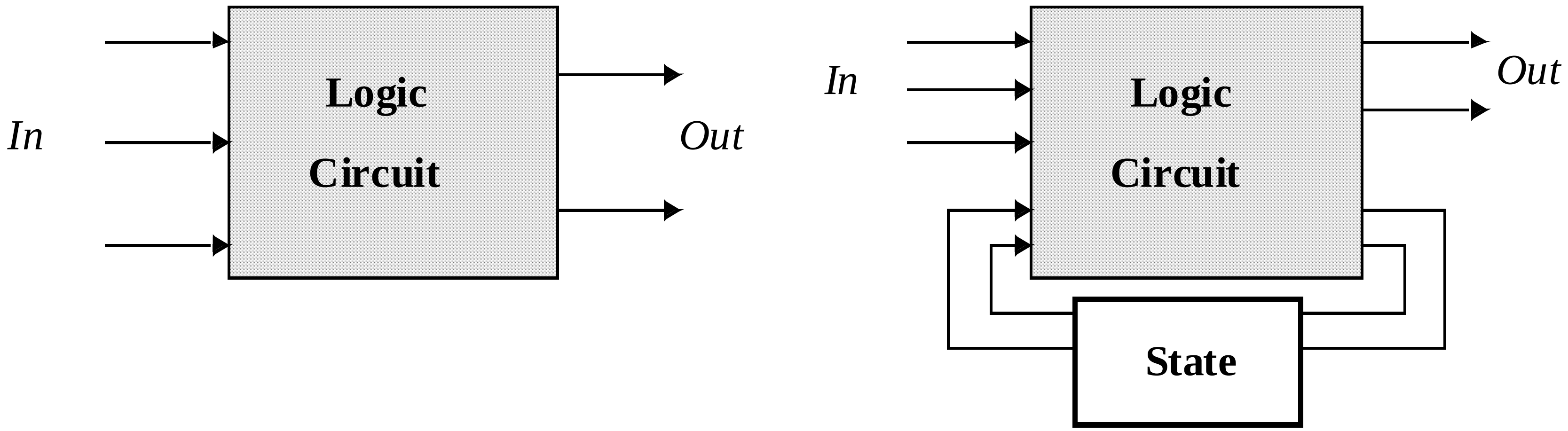
Combinational vs. sequential digital circuits

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



- Combinational means "memory-less"
  - a digital circuit is combinational if its output values only depend on its (current) input values

Combinational vs. sequential digital circuits



(a) Combinational

(b) Sequential

Output = *f*(*In*)

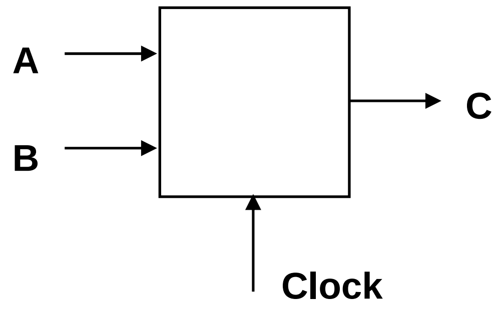
Output = *f*(*In*, *Previous In*)

Sequential logic

- Sequential systems
  - exhibit behaviors (output values) that depend not only on the current input values, but also on previous input values

Example of combinational and 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



Abstractions

- Some we've seen already
  - digital interpretation of analog values
  - transistors as switches
  - switches as logic gates
  - use of a clock to realize a synchronous sequential circuit
- Some others we will see
  - truth tables and Boolean algebra to represent combinational logic
  - encoding of signals with more than two logical values into binary form
  - state diagrams to represent sequential logic
  - hardware description languages to represent digital logic
  - waveforms to represent temporal behavior

An example

- Calendar subsystem: number of days in a month (to control watch display)
  - Combinational logic
  - 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);
  }
}
```

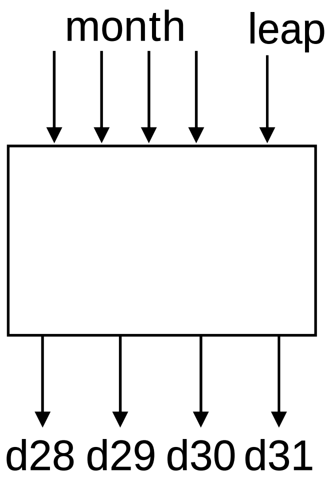
P.16

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

Don't care

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 not

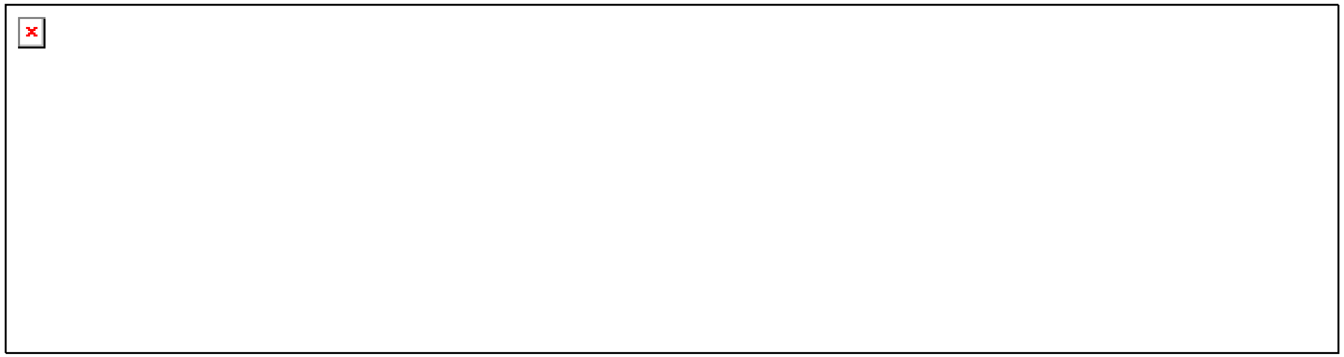
symbol for and

symbol for or

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	–	–	–	–	–

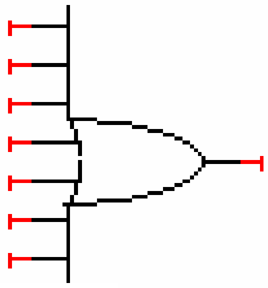
Combinational example (cont'd)

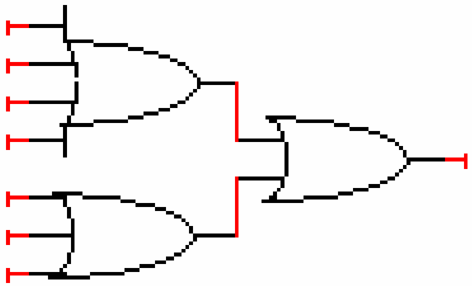
- $d28 = m8' \bullet m4' \bullet m2 \bullet m1' \bullet leap'$
- $d29 = m8' \bullet m4' \bullet m2 \bullet m1' \bullet leap$
- $$\begin{aligned} d30 &= (m8' \bullet m4 \bullet m2' \bullet m1') + (m8' \bullet m4 \bullet m2 \bullet m1') + \\ &\quad (m8 \bullet m4' \bullet m2' \bullet m1) + (m8 \bullet m4' \bullet m2 \bullet m1) \\ &= (m8' \bullet m4 \bullet m1') + (m8 \bullet m4' \bullet m1) \end{aligned}$$
- $$\begin{aligned} d31 &= (m8' \bullet m4' \bullet m2' \bullet m1) + (m8' \bullet m4' \bullet m2 \bullet m1) + \\ &\quad (m8' \bullet m4 \bullet m2' \bullet m1) + (m8' \bullet m4 \bullet m2 \bullet m1) + \\ &\quad (m8 \bullet m4' \bullet m2' \bullet m1') + (m8 \bullet m4' \bullet m2 \bullet m1') + \\ &\quad (m8 \bullet m4 \bullet m2' \bullet m1') \end{aligned}$$



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'•m1') + (m8•m4'•m2•m1') +
  (m8•m4•m2'•m1')





Another example (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
  - Sequential logic
  - inputs: sequence of input values, reset
    - Numeric number: 4 wires
  - outputs: door open/close
  - memory: must remember combination

    or always have it available as an input

Implementation in software

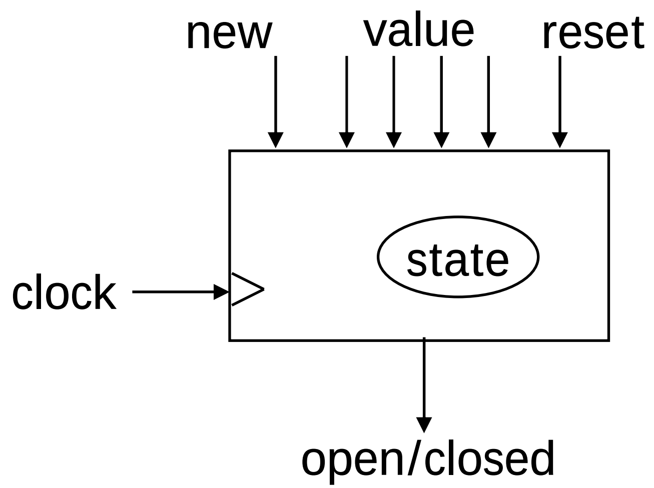
```
integer combination_lock ( ) {
   integer v1, v2, v3;
   integer error = 0;
   static integer c[3] = 3, 4, 2;

   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);
}
```

Array index starts from 1

P.19, 20

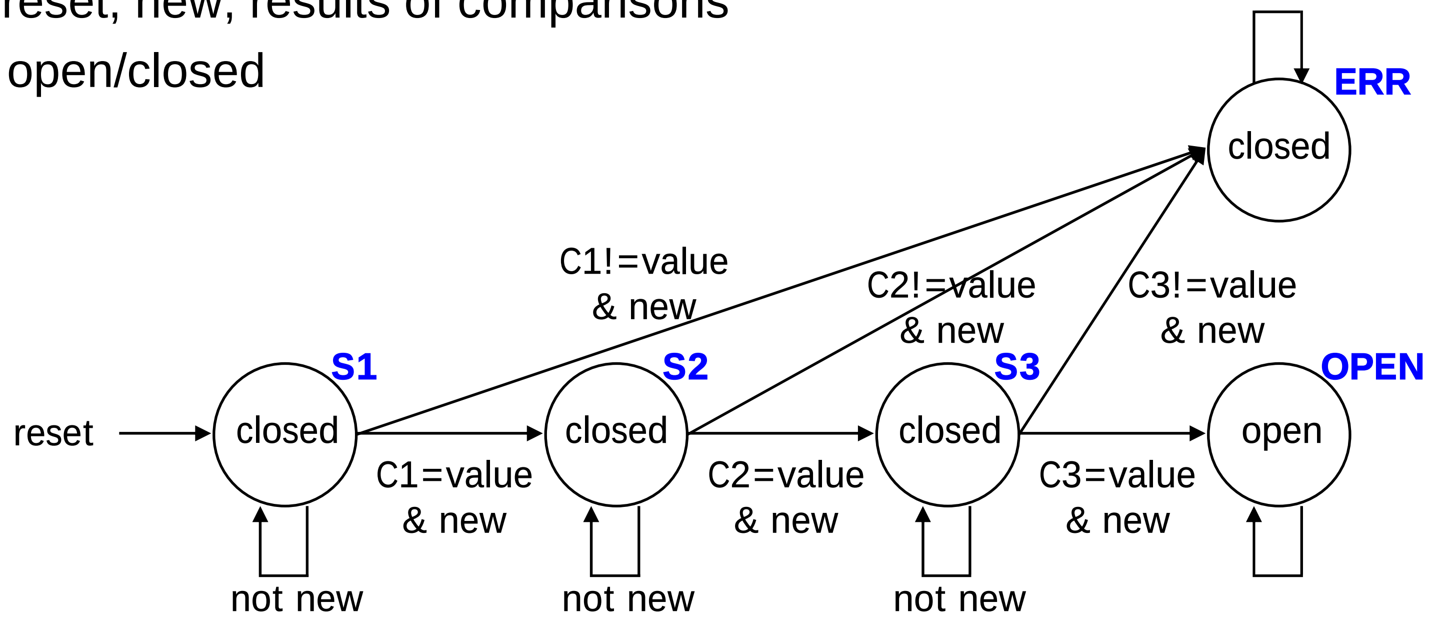
Implementation as a sequential digital system

- 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



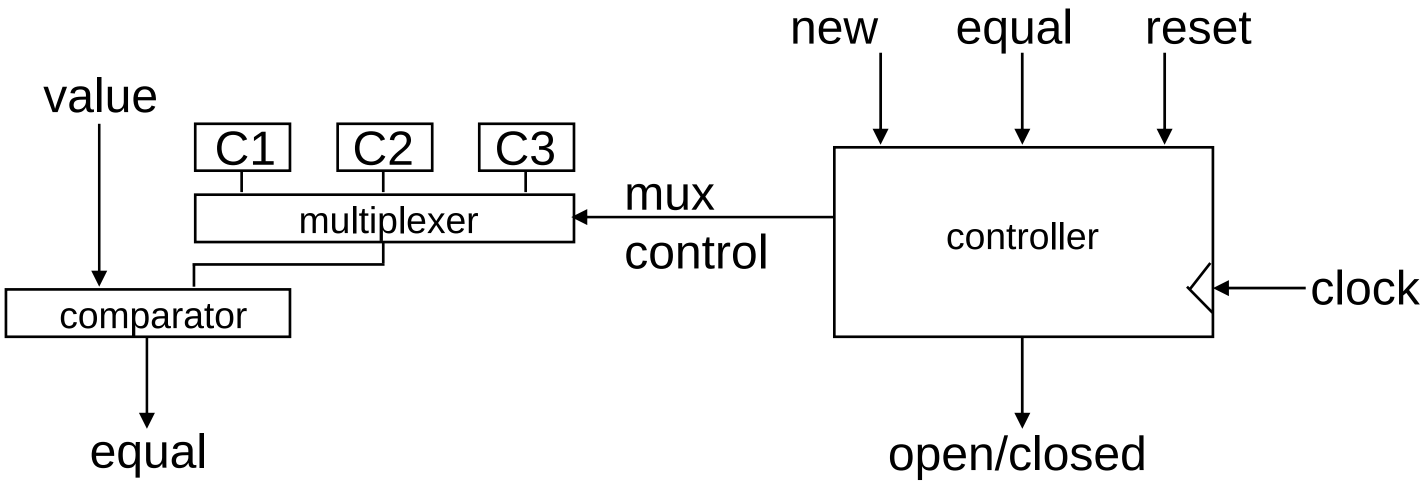
Sequential example (cont'd): abstract control

- Finite-state diagram
  - states: 5 states
    - represent point in execution of machine
    - each state has inputs and 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): data-path vs. control

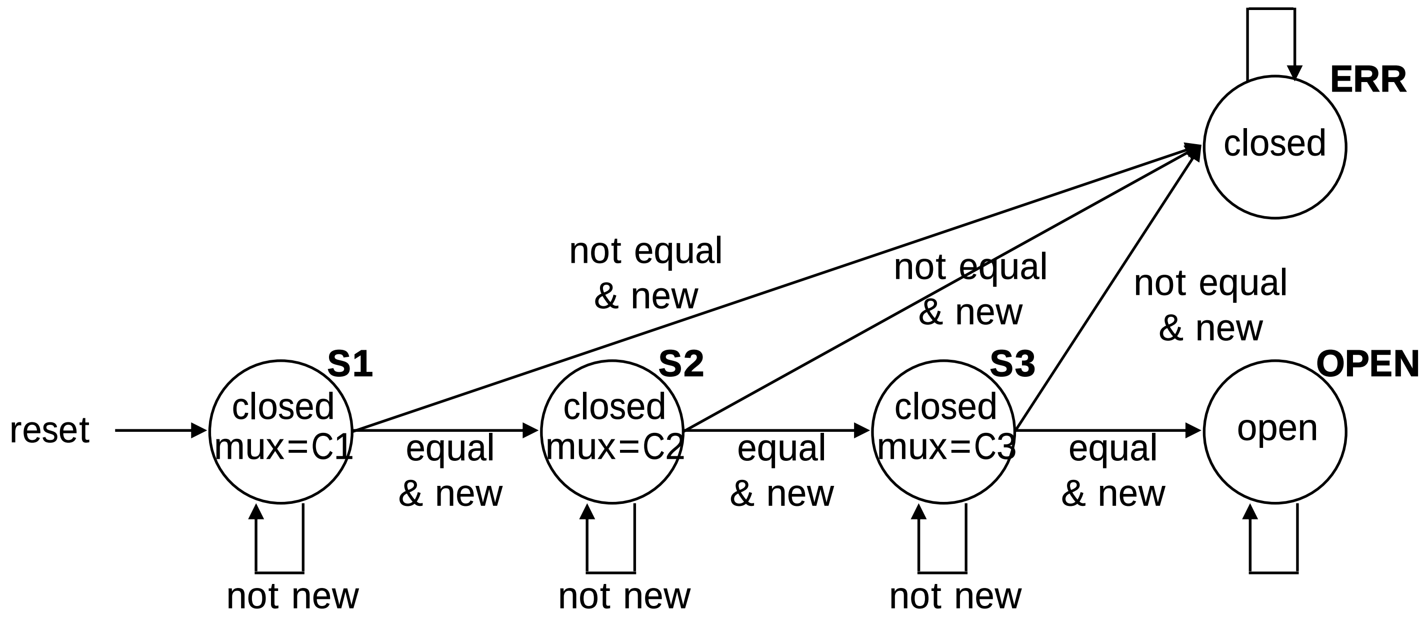
- Internal structure
  - data-path
    - storage for combination
    - comparators
  - control
    - finite-state machine controller
    - control for data-path
    - state changes controlled by clock



* Multiplexer (MUX)

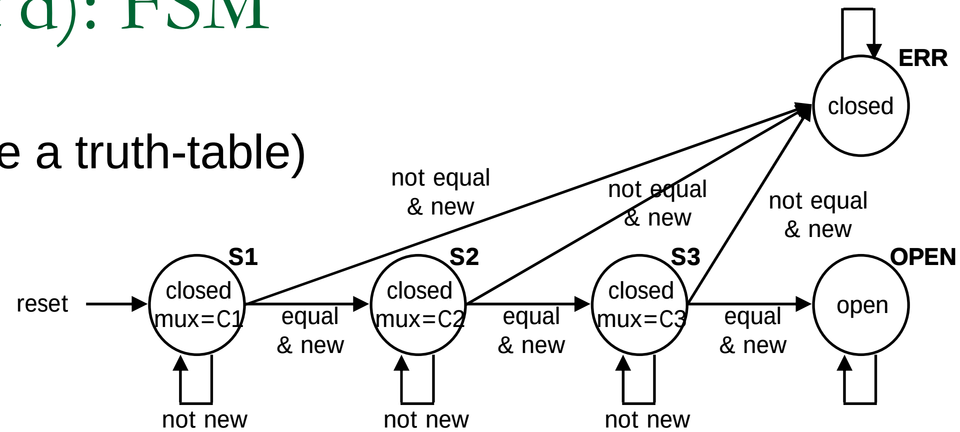
Sequential example (cont'd): FSM

- Finite-state machine (FSM)
  - refine state diagram to include internal structure



Sequential example (cont'd): FSM

- 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

* state is not input, but internal variable

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

Internal input

Internal output

system's or external output

external input

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

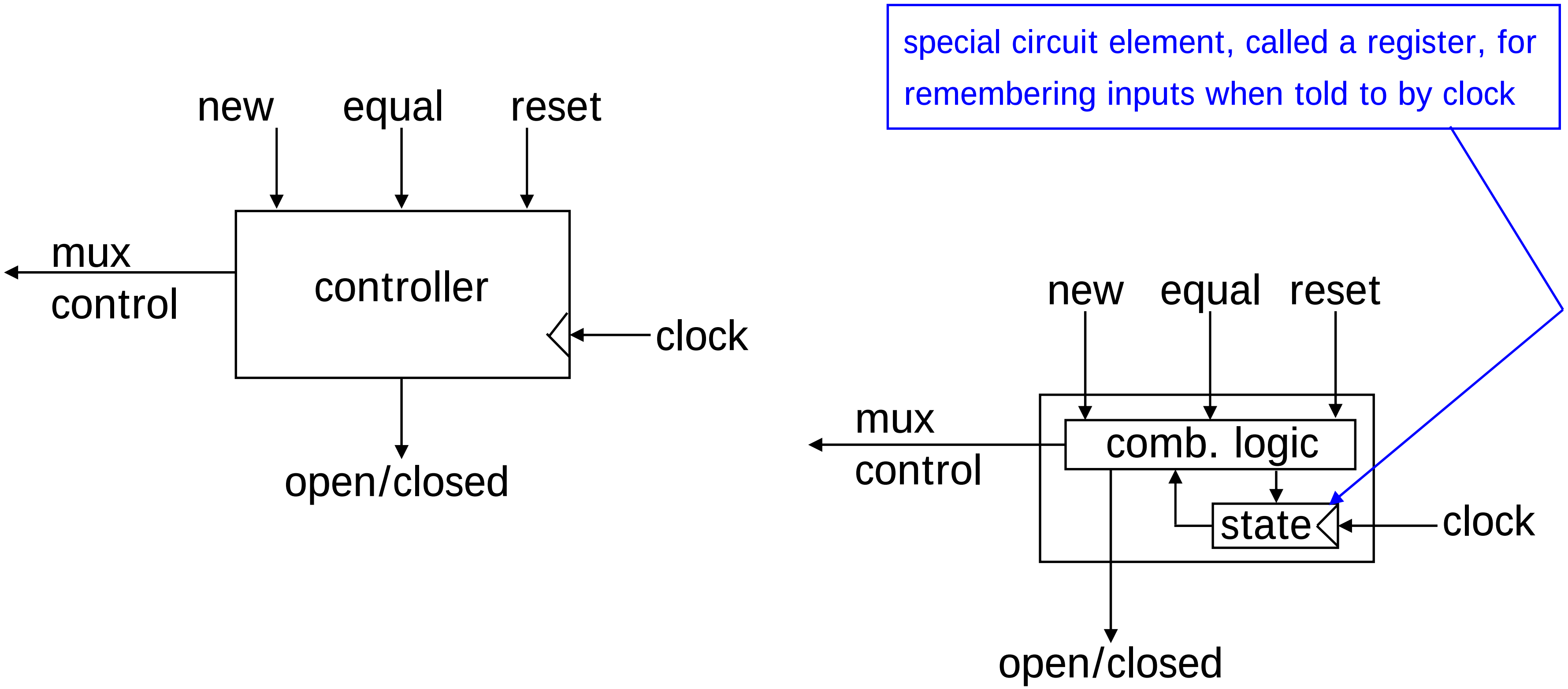
good choice of encoding!

mux is identical to last 3 bits of next state

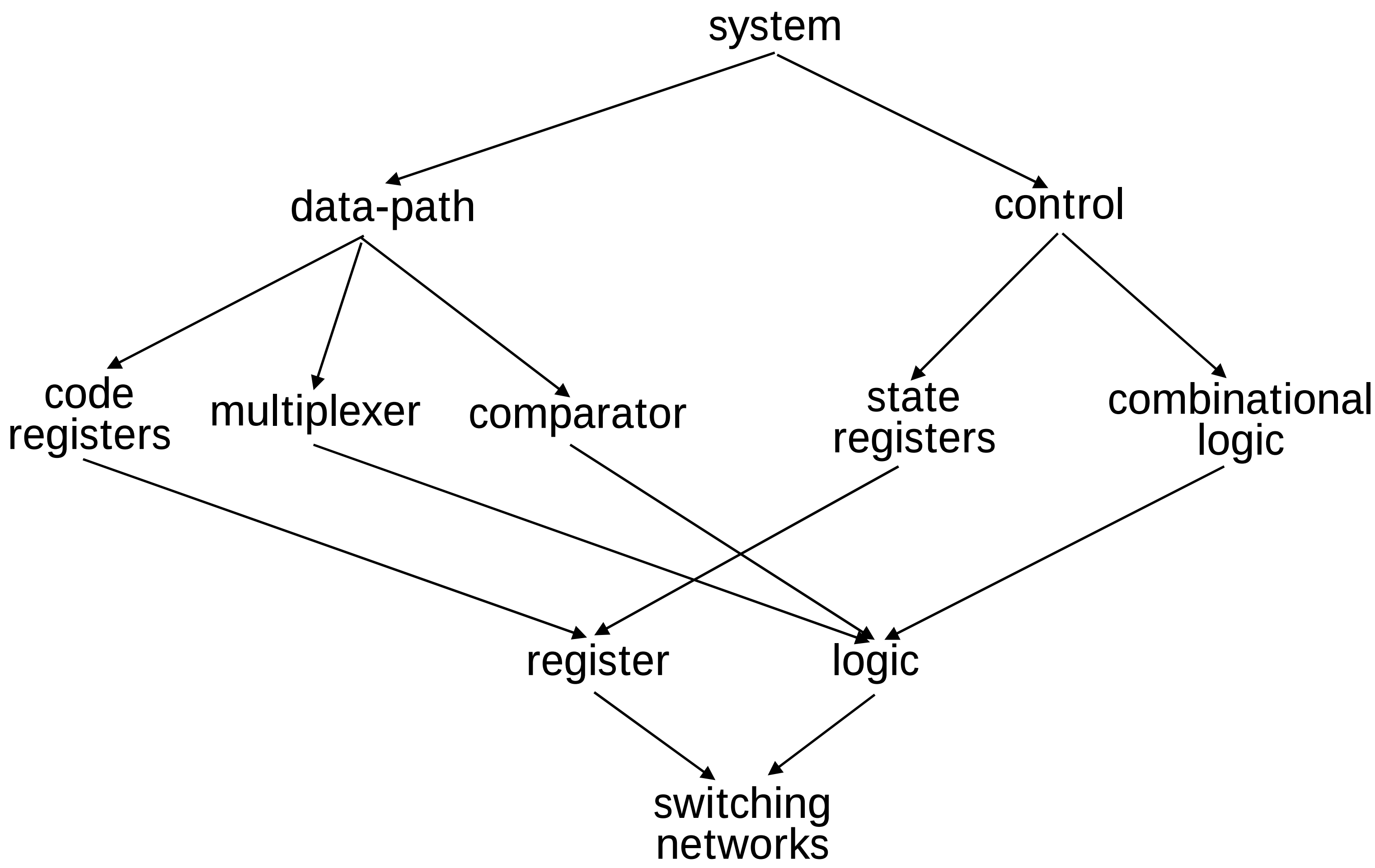
open/closed is identical to first bit of state

Sequential example (cont'd): controller implementation

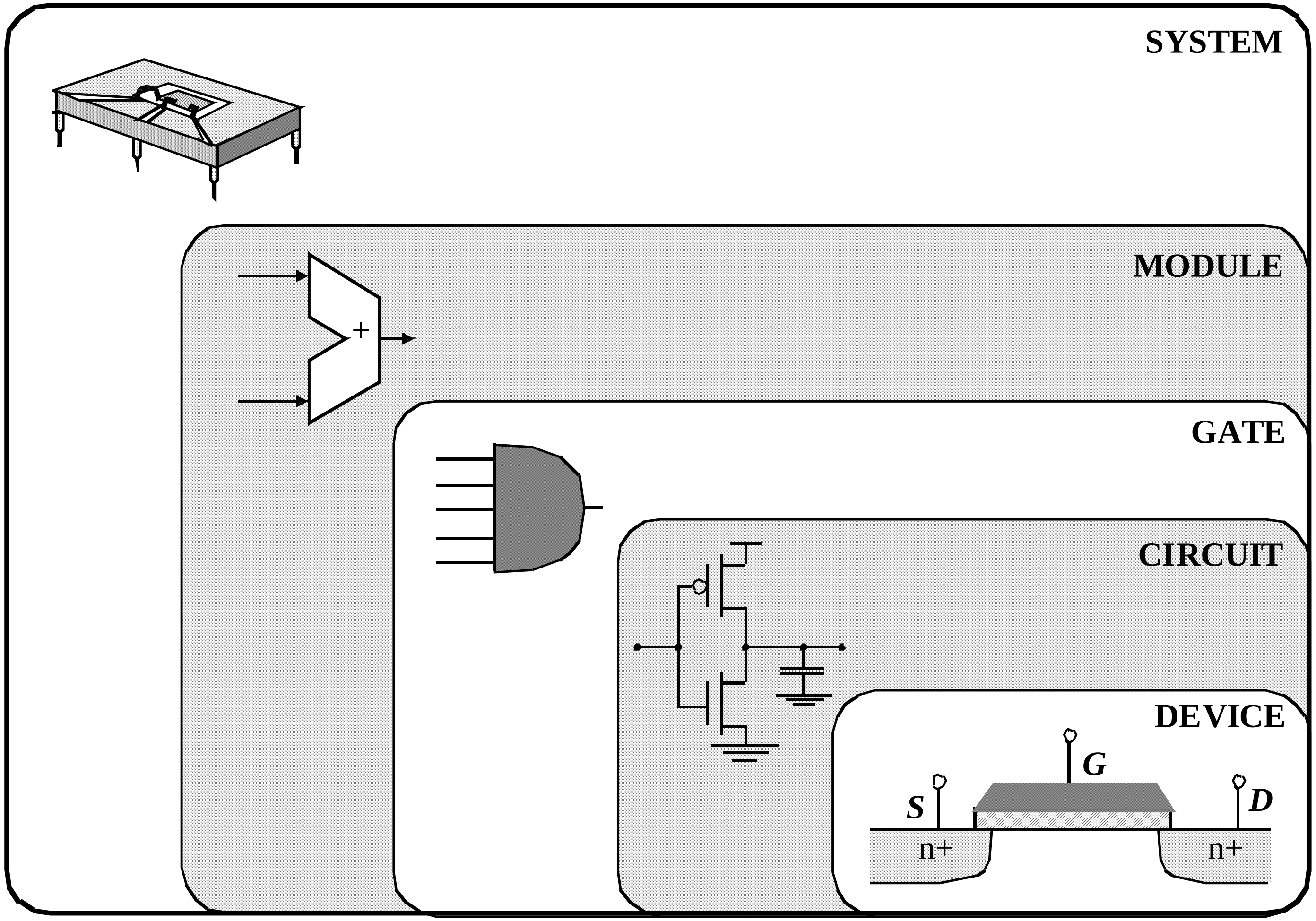
- Implementation of the controller



Design hierarchy



terminologies



Summary

- That was 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