Digital Logic Design

Ch 1

Introduction

General information

- Instructor: 백 윤 흥
  - Office hour: by prior appointment thru e-mail
  - ☎: 880-1748, Email: ypaek@snu.ac.kr
- TA: 황동일 (Head)
  - ☎: 880-1742, Email: Logicdesign-ta@sor.snu.ac.kr
- References
  - R. Katz and G. Borriello, Contemporary Logic Design
- Lecture notes or other class materials will be available online

  Login with your portal ID into http://etl.snu.ac.kr

Grading policy

- Two quizzes and 1 exam: 60 %
  - Quizzes (10+20%): Early Oct and Nov
  - Exam (30%): Mid Dec
- Lab assignments: 35%
  - Details will be given by the TAs in lab hours
- Class attendance: 5%
  - Attendance sheets will be handed out during the class.
  - Please sign it up for your attendance verification.

Lecture information

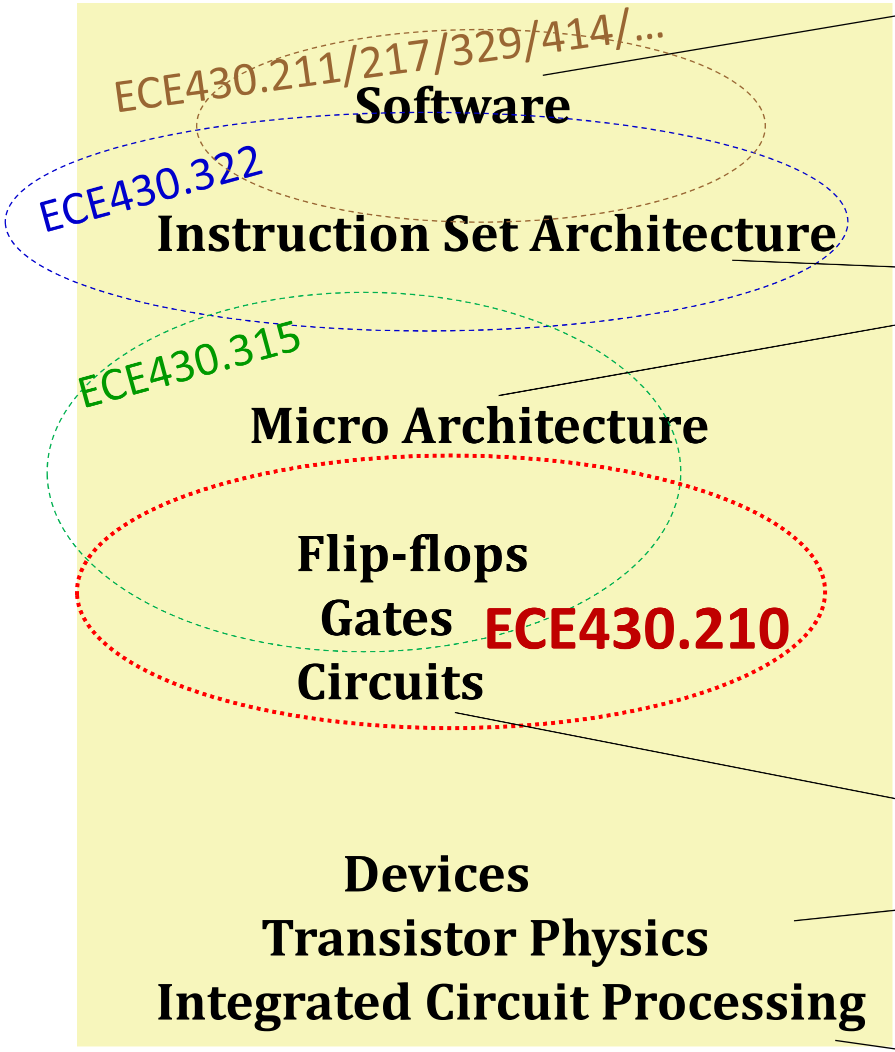
- Organization of the lecture
  - 70~80 min classes
  - Regular lectures
    - 11:00 am ~12:15 pm, MoWe
    - 302-408
  - Programming lab hours
    - 6:30 pm ~ 8:20 pm, Monday
    - 301 Computer Lab
- Expected grading distribution for this course
  - A: 20~30%, B: 30~40%, C: 30~40%, D: ~10%

Tentative class schedule

구 분	강의내용
1 & 2주	Introduction (Chapter 1)
3 주	Combinational Logic Design (Chapter 2)
4 주	Combinational Logic Design (Chapter 2)
5 주	Combinational Logic Design (Chapters 3)
6 주	Combinational Logic Design (Chapters 3~4)
7 주	Combinational Logic Design (Chapter 4)
8 주	Sequential Logic Design (Chapter 6)
9 주	Sequential Logic Design (Chapters 6~7)
10 주	Sequential Logic Design (Chapters 7~8)
11 주	Sequential Logic Design (Chapter 8)
12 주	Sequential Logic Design (Chapters 8~9)
13 주	Sequential Logic Design (Chapter 9)
14 주	Case studies (Chapter 5)
15 주	Case studies (Chapter 10)
16 주	Exam

Digital computer design courses in SNUECE

Hierarchical structure of computer design problem



High-level Programming languages
Compiler, Machine languages
CPU, Memory, Cache Architectures
Communication, Networks
I/O Devices, HDD, SSD, DMA

Hardware Description Languages
Arithmetic units
Encoding, Framing
Boolean Logic
Combinational logic circuits
Sequential logic circuits
Timing & Clocking
Synchronous circuits
Finite State Machines

IC Fabrication, Wafers
IC masking,
Lithography, Etching

Introduction to Digital Logic

- Computer hardware has experienced the most dramatic improvement in capabilities and costs for the past decades
  - **Logic components** are basic building blocks of all today's digital computers.
  - **Logic design** is one of the disciplines that has enabled the digital revolution which has dramatically altered our lives.
- Design
  - the process of coming up with a solution to a problem while meeting some criteria (ex: size, cost, power, performance)
  - Divide-and-conquer approach has been developed to handle the complexity of the design process by breaking down the problem into smaller pieces, dealing with constraints beyond their control and putting all the pieces together to solve the bigger problem.

Logic Design

- The process of choosing the **logic components** that solve a logic design problem while meeting constraints (e.g., size, cost, performance, and power consumption)
- Digital (logic) components
  - They have input and output wires which carry digital logic values (i.e., 0 and 1).
  - Arbitrary information can be represented using this digital abstraction.
  - Transistors react to the voltage levels on the input wires.
  - Sequential logic circuits' outputs react to the current values on the input wires and to the past history of values on those same input wires.

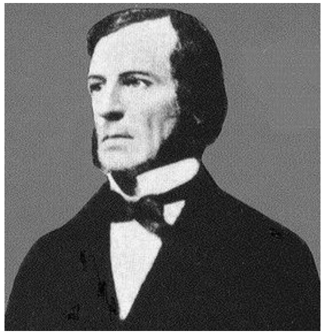
Contemporary Logic Design

- Important trends in contemporary Logic Design
  - larger and larger designs
  - shorter and shorter time to market
  - cheaper and 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 → *More emphasis on ISA/microarchitecture specification*
    Relying on logic compilers for logic/circuit implementation
  - use of simulation to debug designs
  - simulate and verify before you build

To fill the performance gap between designer's specification and actual implementation by tools

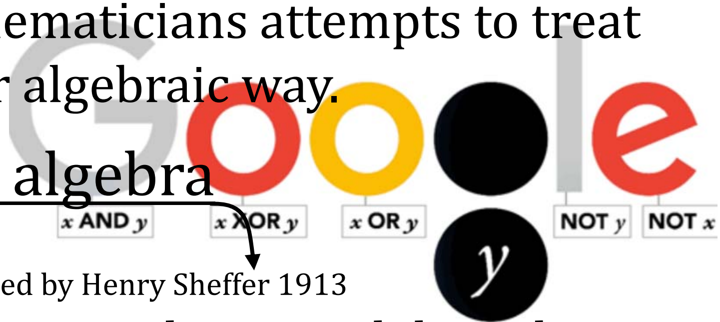
History of Logic Design

조지 불 탄생 200주년



1815-1864

- Before the 19th century
  - Theories of logic were studied with rhetoric, through the syllogism, and with philosophy in many cultures in history, including China, India, Greece.
  - In the 18th-century, some philosophical mathematicians attempts to treat the operations of formal logic in a symbolic or algebraic way.
- 1850's: **George Boole** invented <u>Boolean algebra</u>



  - *The Mathematical Analysis of Logic* (1847) named by Henry Sheffer 1913
  - (Mathematical) logic was finally established as a mathematical discipline.
  - maps logical propositions to symbols
  - permits manipulation of logic statements using mathematics.

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

logic ➔ algebra

x	y	$x+y$	$x'\cdot y'$
1	1	1	0
1	0	1	0
0	1	1	0
0	0	0	1

$x + y = (x'\cdot y')' \;\Rightarrow\; x' + y = (x\cdot y')'$

$(x + y)' = x'\cdot y' \;\Rightarrow\; (x' + y)' = x\cdot y'$

$(x + y) + x'\cdot y' = 1$

$= x + y + x'\cdot y' = x + (x\cdot y')'$

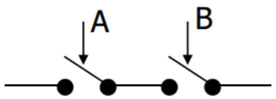
$= x + x' + y = 1 + y = 1$

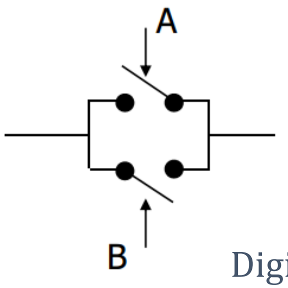
History of Logic Design

1916-2001

- 1938: **Claude Shannon** links Boolean algebra to switches
  - *A Symbolic Analysis of Relay and Switching Circuits*, Master thesis in MIT

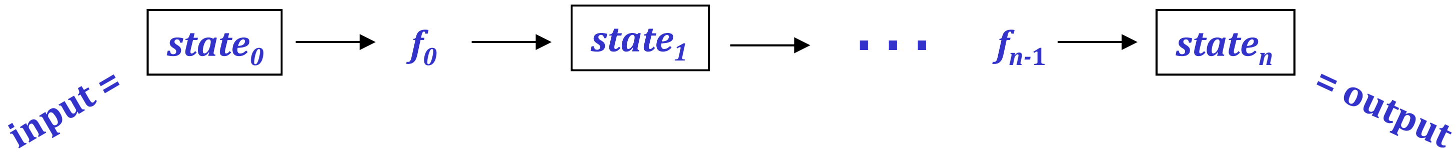
    "In the control and protective circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. Examples of these circuits occur in almost any circuits designed to perform complex operations automatically. In this paper a mathematical analysis of certain of the properties of such networks will be made. ..."
  - revolutionizes the study of **switches** and **relays**, which in turn form the circuitry behind the binary arithmetic of modern computers
    - Any **circuit** is represented by a set of equations corresponding to the various relays and switches in the circuit.
    - These equations are manipulated to accomplish certain **computations** by simple mathematical processes, exactly analogous to the calculus of propositions used in Boolean algebra.

$A \cdot B \equiv$ 

$A+B \equiv$ 

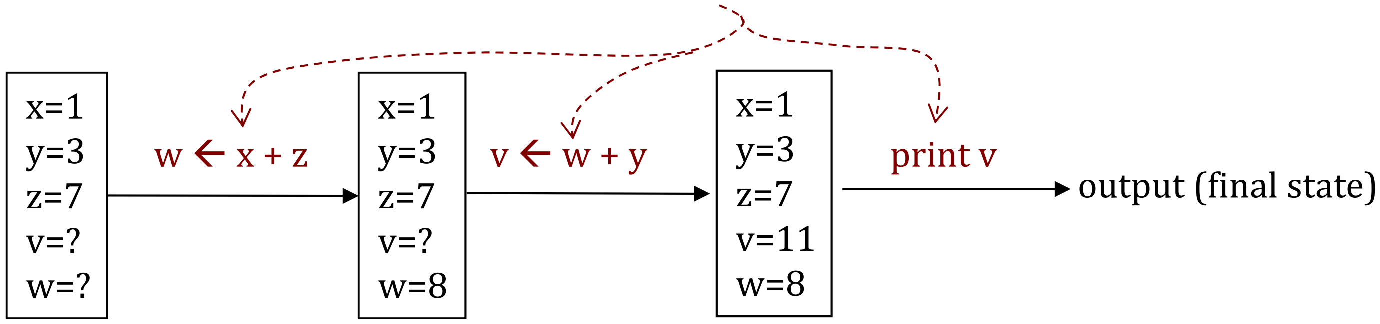
Computation: abstract concept

- Abstract view: $output = F(input) = f_n(f_{n-1}(\ldots(f_0(input))))$



Functions are a sequence of mutators of states!

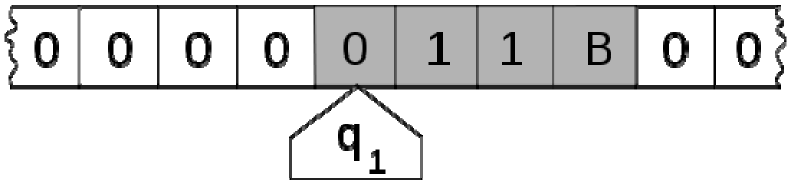
- Example
  - print the summation of three integers 1, 3 and 7
  - A program: a sequence of three functions

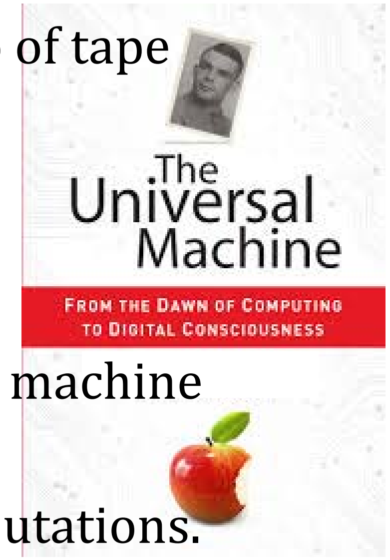


A universal machine that…

1912-1954

- … does computations automatically by performing functions and storing states.
- 1936: **Alan Turing** invented the machine (*Turing machine*)!
  - A hypothetical device that manipulates symbols on a strip of tape according to a table of rules.





  - The Turing machine mathematically models an automatic machine that mechanically operates on a tape.
  - The machine can be adapted to simulate the logic of computations.
- Now, people understood what a machine can and cannot do.
  - How to realize this hypothetical machine in a physical form?

Turing meets Shannon

In 1943, Turing left England and spent two months at Bell Labs talking to Shannon in USA

- Functions can be implemented by switches.
- How?
  - Example: 2 + 3 = 5
  - To perform logic operations for arithmetic functions, the digital number must be transformed into binary forms ➔ *encoding!*
  - Now a function can be performed by the calculus of Boolean algebra.

010 + 011 = 101

$$\begin{array}{rcccl} & 0 & 1 & 0 & \\ + & 0 & 1 & 1 & \\ \hline & 0 & 0 & 1 & \leftarrow \text{Sum} \\ + & 0 & 1 & 0 & 0 \;\leftarrow \text{Carry} \\ \hline & 1 & 0 & 1 & \leftarrow \text{Result} \end{array}$$

x	y	S	C
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

$S = (x \wedge \neg y) \vee (\neg x \wedge y)$

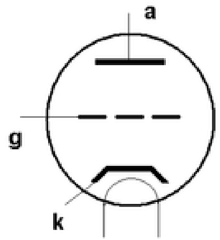
$\quad = xy' + x'y$

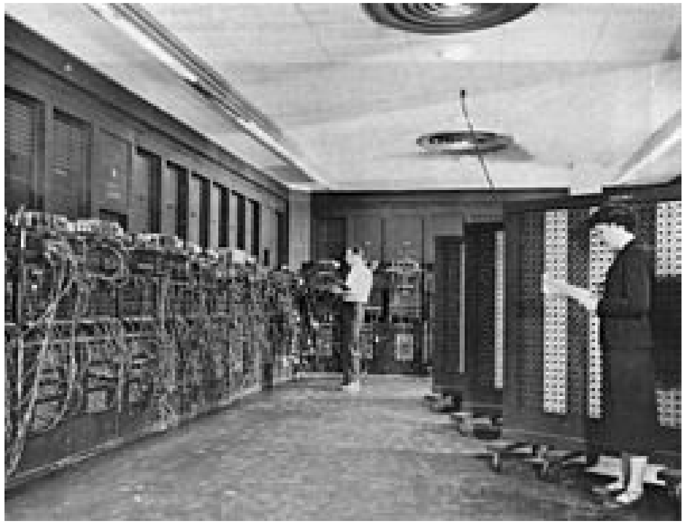
$C = x \wedge y = xy$

  - *Boolean algebra can be used to perform Turing-complete computations!*

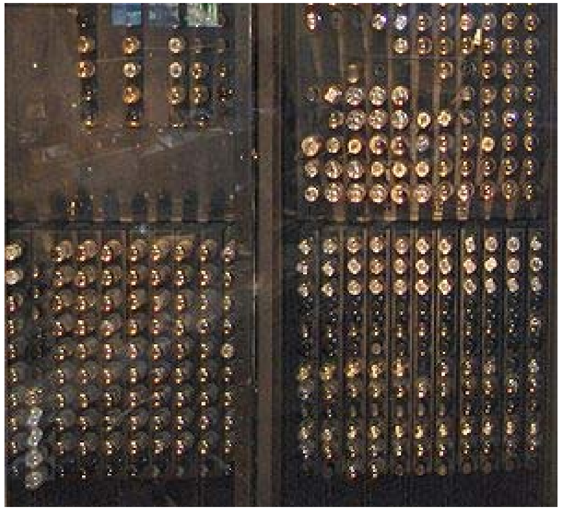
History of Logic Design (continued)

- 1946: ENIAC . . . World's first automatic machine to perform Turing-complete computations ➔ called a ***computer*!**
  - inspired by Turing's theory
  - completely electronic computer
  - 17,468 **vacuum tubes** as electrical *switches*
  - several hundred multiplications per minute



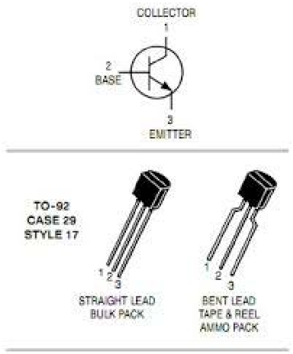


weighed 27t, 2.4m×1m×30m in size, consumed 150 kW

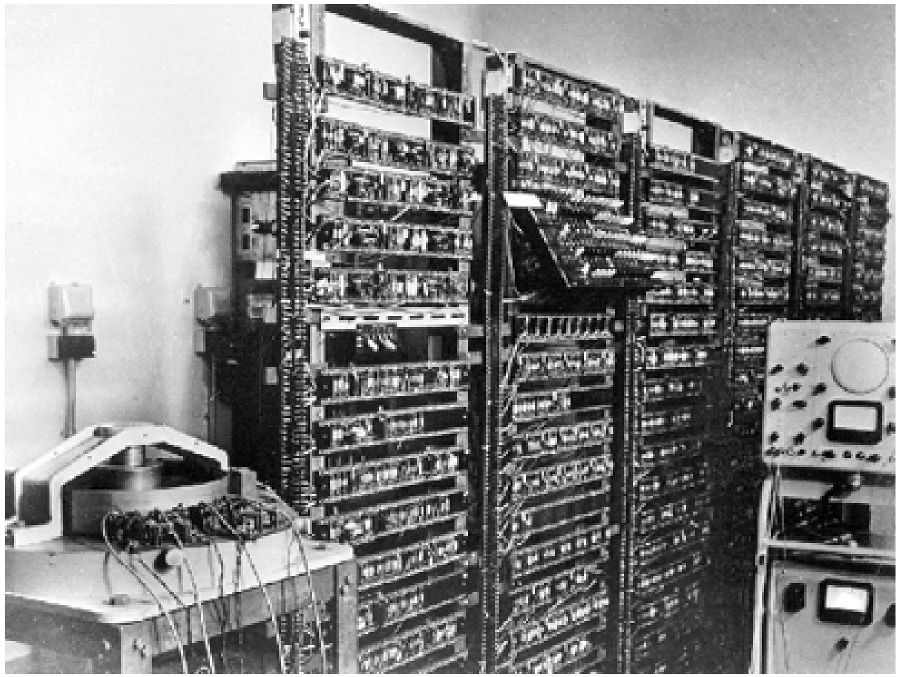


vacuum tubes in ENIAC

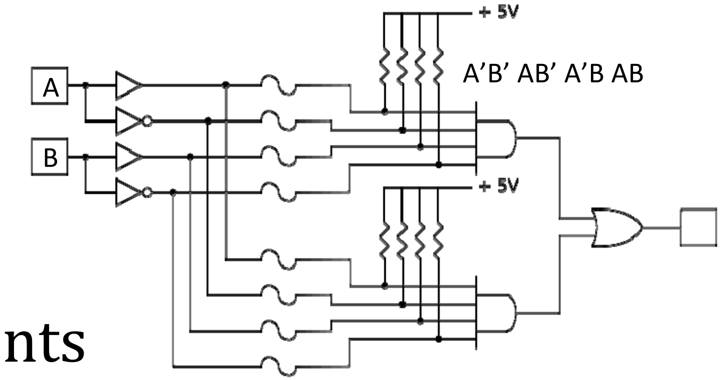
History of Logic Design



- 1947: Shockley, Brittain and Bardeen invent the **transistor**
  - replaces vacuum tubes
  - enable integration of multiple devices into one package
  - gateway to modern electronics
- 1953: Manchester TC
  - World first Transistor Computer
  - Univ. of Manchester
  - 48-bit word
  - 92 transistors and 550 diodes
  - consumes less than 100w



History of Logic Design

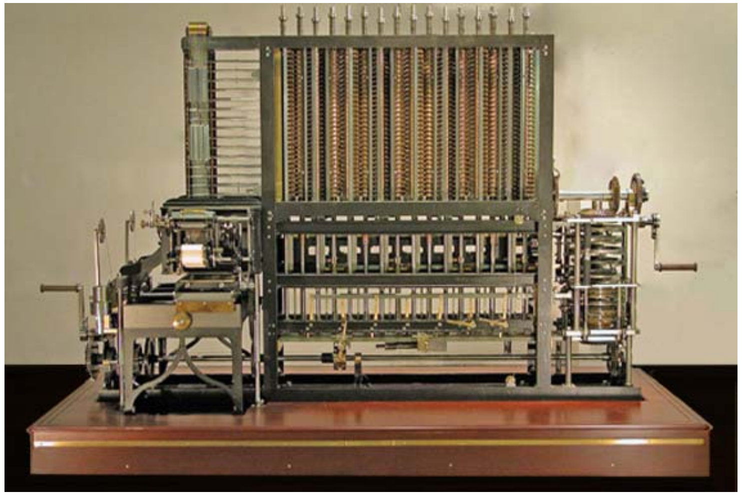


Simplified programmable logic device

- 1960s: A large catalog of logic components
  - Texas Instruments TTL data book
  - Arbitrary logic circuits could be built from these basic primitives.
- 1978: Programmable Array Logic (PAL) by Monolithic Memories
  - A logic gate has a fixed function, but a PAL has an undefined function at the time of manufacture. → Before being used, it must be reconfigured.
  - collections of switches in regular arrangements
  - increase levels of integration
  - make it easier for designers to change the wiring pattern
- 1984: Field-programmable gate arrays introduced by Xilinx
  - internally based on Look-up tables (LUTs), meant for more complex designs.
  - Logic circuits can be altered over times.
  - Synthesis tools have followed with the appropriate compilation.

Computation: implemented w/ switches

- In the past, computation has been a mental exercise on paper.
  - There were primitive trials to build a physical machine for computation, though…
  - 1830: Mechanical computer (invented by Charles Babbage, England) to compute polynomials $f(x) = \Sigma\, c_k x^k$



- From the history of logic design, we have learned that computation can be implemented electrically with switches
  - Combining Turing machine model with Shannon's electrical switches
  - This class is about physically implementing computation using physical devices that use voltages to represent logical values.

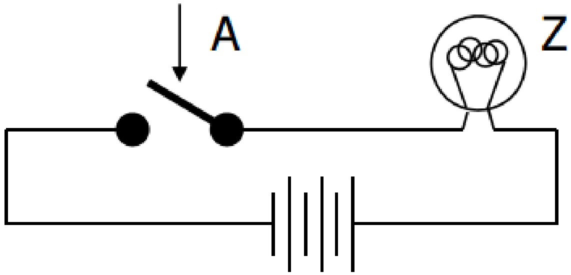
Basic function units of computation

- Representation: "0", "1" on a wire
  set of wires (e.g., for binary numbers)
- 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; foo(...); B;

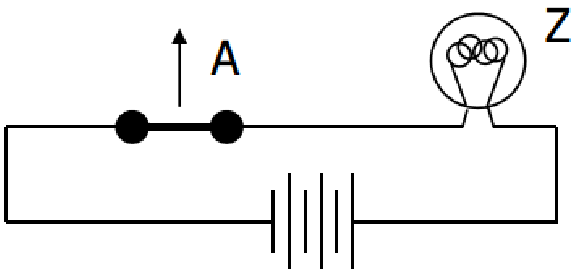
> *We will study how each of these is implemented in hardware and composed into computational structures*

Switches

- Basic building blocks of digital computing machines
- Implementing a simple circuit (arrow shows action if wire changes to "1")

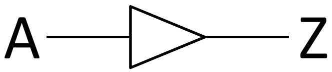


close switch (if A is "1" or asserted)
and turn on light bulb (Z)



open switch (if A is "0" or unasserted)
and turn off light bulb (Z)

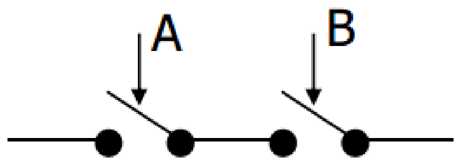
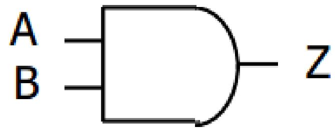
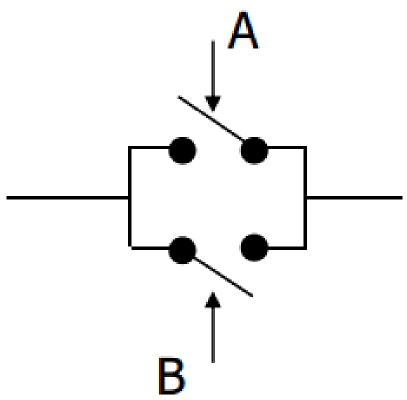
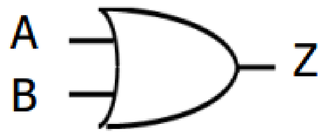
Z = A



schematic symbol
indicating direction (value of Z does not affect A)

Switches

- Compose switches into more complex Boolean functions

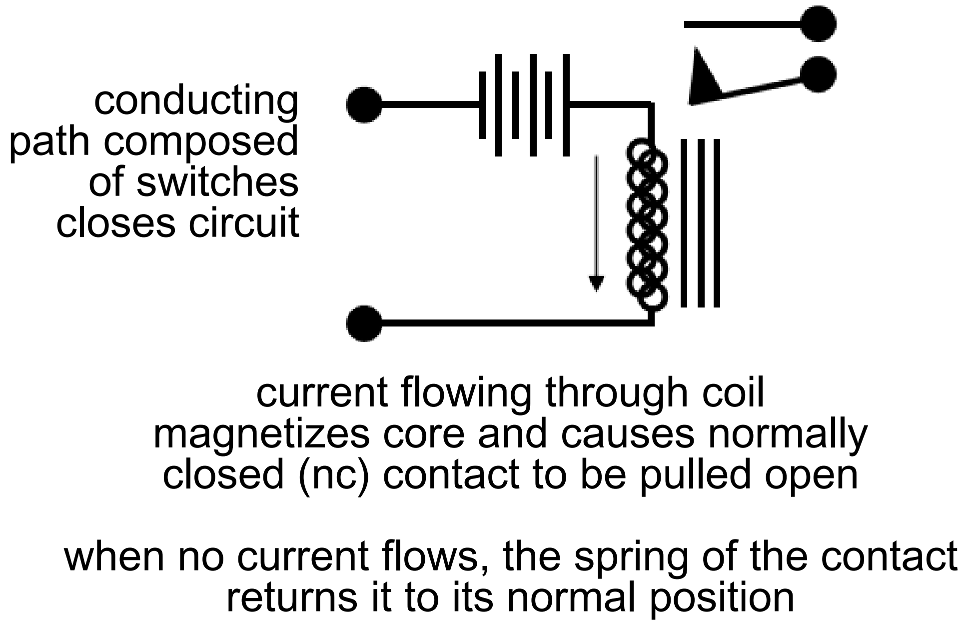
	Circuits	*Boolean expressions*	*Schematic symbols*
AND		$Z = A \wedge B$	
OR		$Z = A \vee B$	

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.

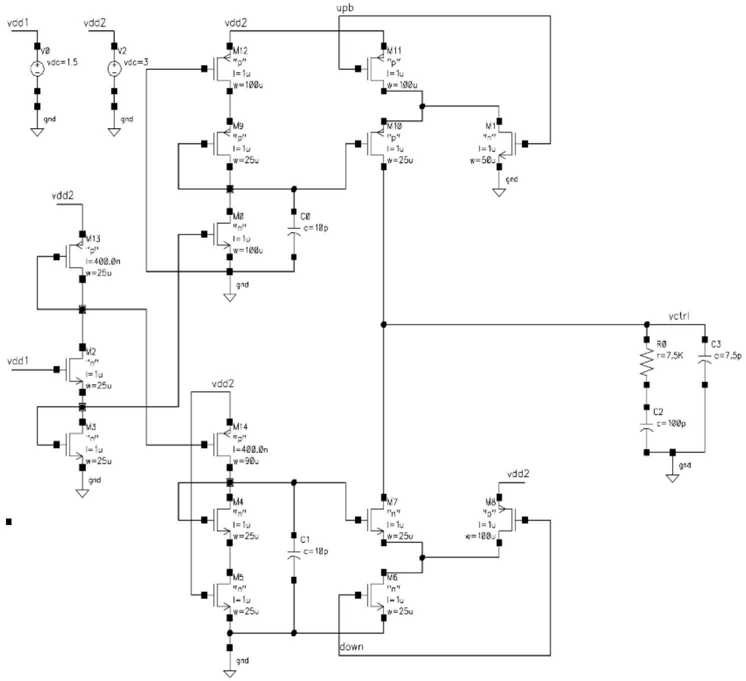
Relay networks

- A simple way to convert between conducting paths and switch settings is to use (electro-mechanical) relays.
- What is a relay?



What determines the switching speed of a relay network?

Transistor networks

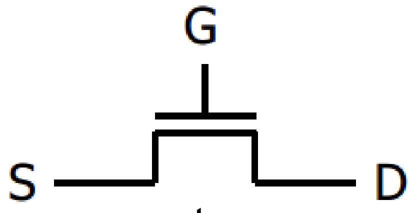


- Relays aren't used much anymore
  - Relay circuits were large and slow (inadequate for large computing machines).
  - Magnets take time to charge.
  - Also mechanical switches are slow to move
- Vacuum tubes
  - Electronic devices that could be used for switches
  - They are faster than mechanical switches, but still too slow
- 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)
  - MOS transistors act as voltage-controlled switches

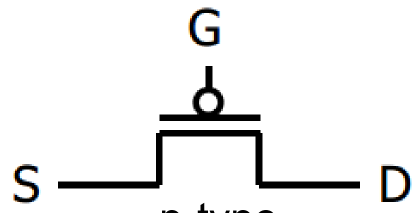
    ➔ similar, though easier and faster to work with than relays.

MOS transistors

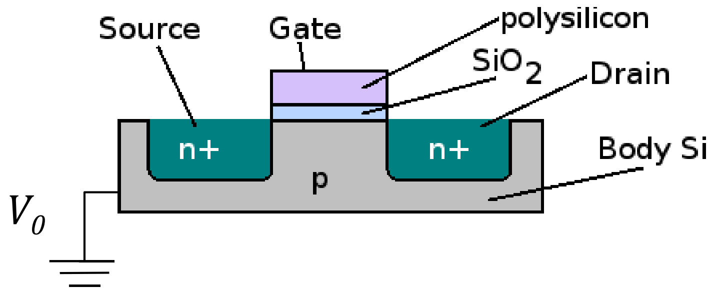
- MOS transistors have three terminals: drain, gate, and source
- They act as switches in the following way.
  - Check the voltage on the gate terminal.
  - If it is (some amount) higher/lower than the source terminal, then a conducting path will be established between the drain and source.

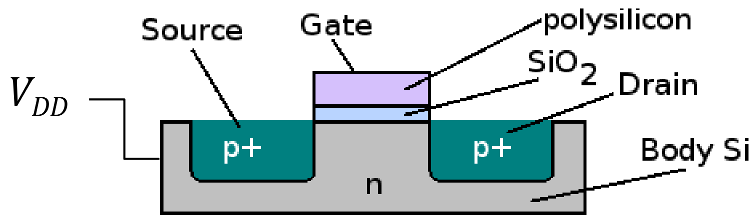


n-type
open when voltage at G is low
closed when voltage at G is high



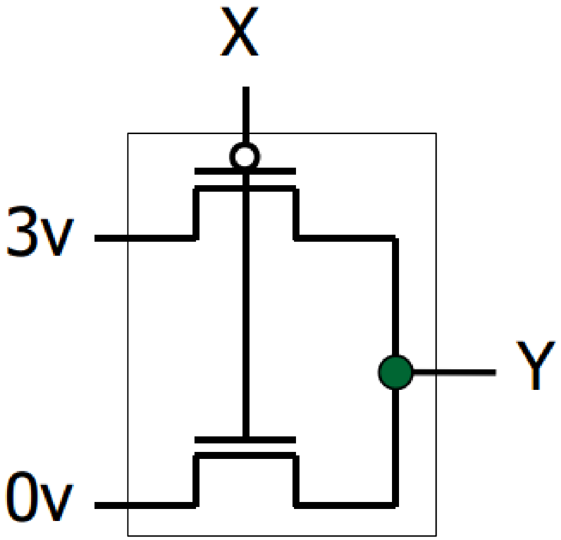
p-type
closed when voltage at G is low
open when voltage at G is high





CMOS network

- A switch consists of both normally-open and normally-closed switches.

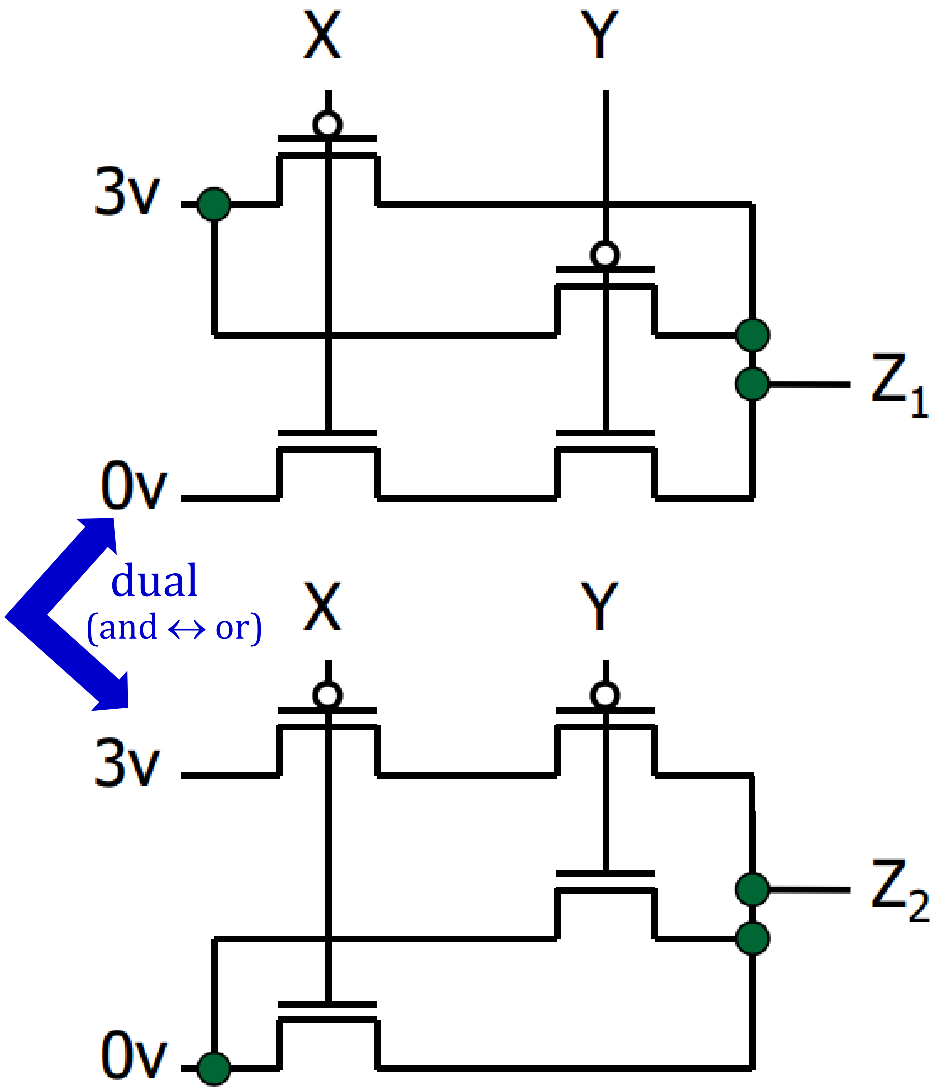


what is the relationship between X and Y?

X	Y
0 volts	
3 volts	

- CMOS is more stable than single MOS since it ensures that …
  - when X is off, the voltage of Y stays high enough by being connected to 3V
  - when X is on, the voltage of Y stays low enough by being grounded

Two CMOS transistors networks

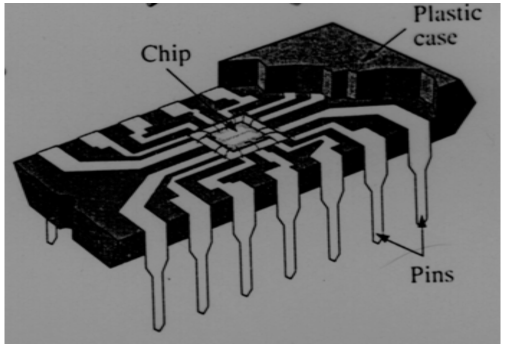


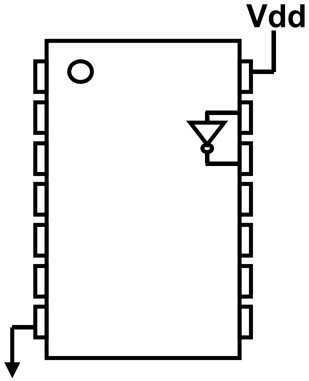
what is the relationship between X, Y and Z?

X	Y	Z1	Z2
0 volts	0 volts		
0 volts	3 volts		
3 volts	0 volts		
3 volts	3 volts		

Combinational logic symbols

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





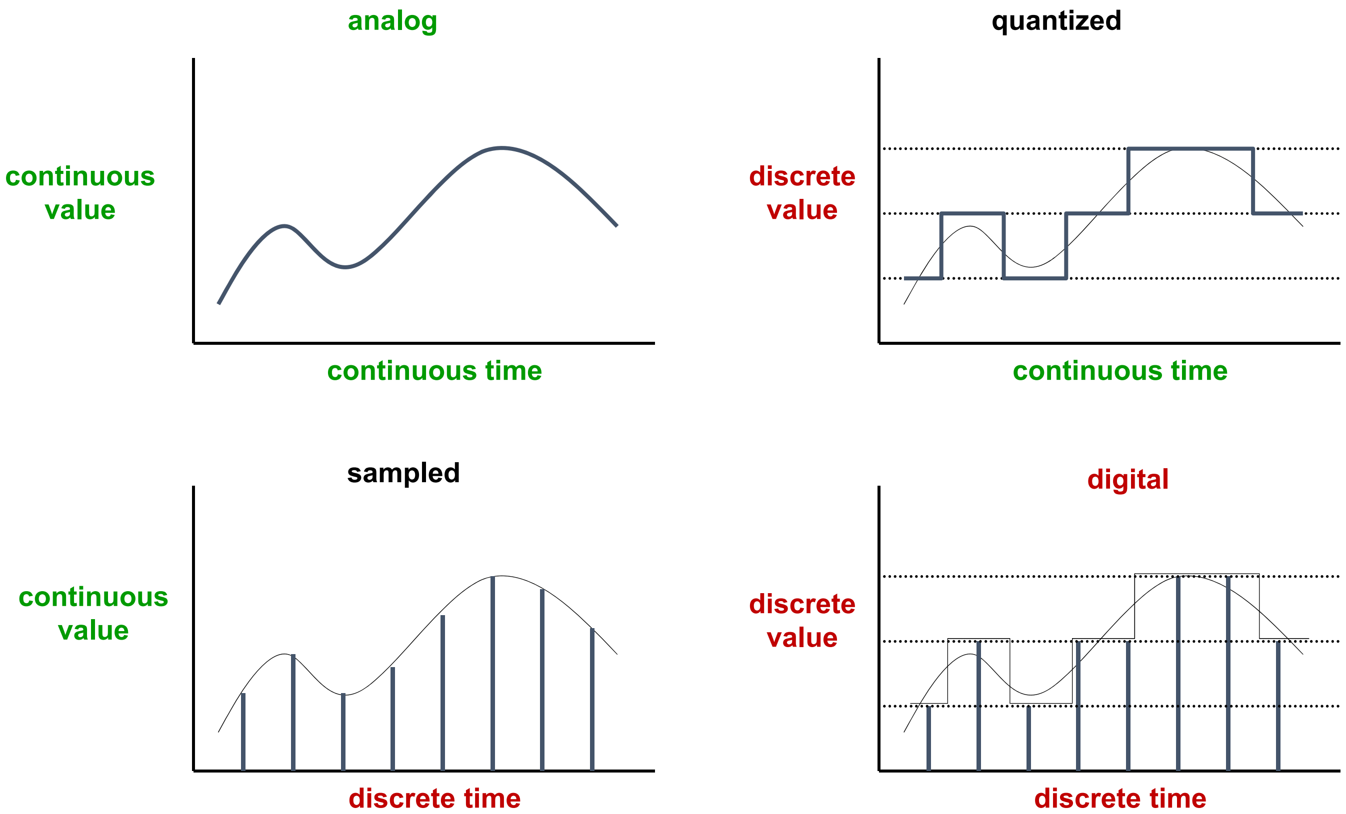
  - AND, NAND

  - OR, NOR

In CMOS technology, NAND/NOR functions are easier and cheaper to implement than AND/OR functions.

Digital vs. Analog

- Digital systems have only discrete input/output values (0 and 1)
- In reality, real electronic components exhibit continuous, analog behavior

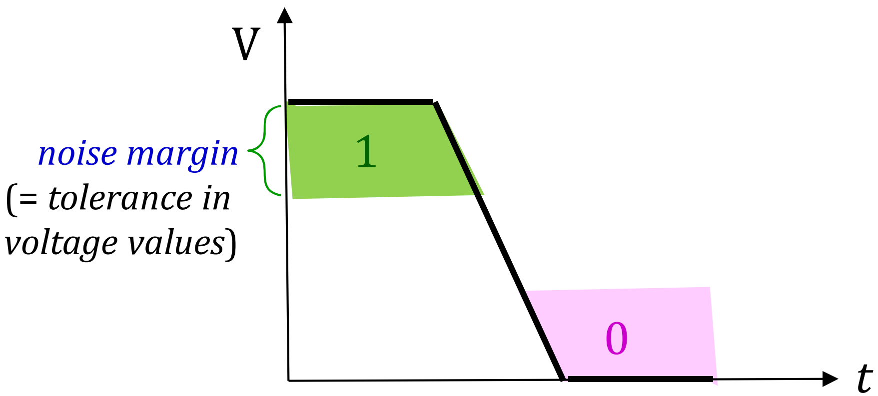


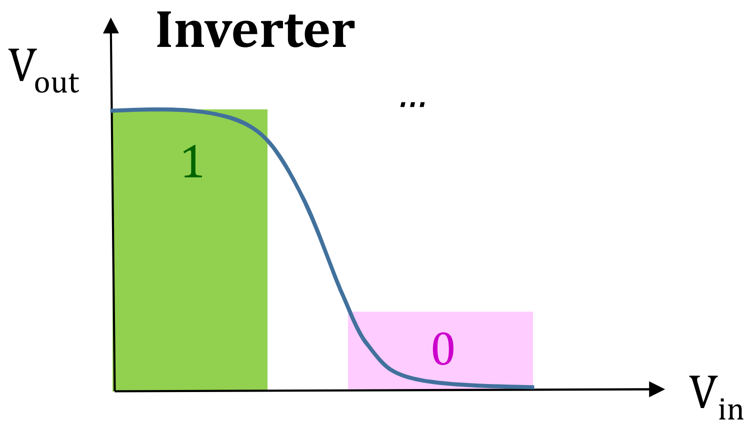
Digital processing

- Why digital?
  - easier to think about a small number of discrete values
  - Human processes in digital
    - e.g. analog vs. digital watch
  - Robust : immune to noise by reshaping
    - e.g. LP vs. CD
- Binary processing
  - The quantization levels are simply two (represented by two numbers: 0 and 1)
  - can use simple switches (on and off)
  - Regarded as decision making (true and false) → simple logical model
  - Reliability (big noise margin)

Digital representations

- Problems with real-world digital systems
  - Digital circuits are built from imperfect switches like transistors.
  - Slight variations in a transistor could alter the operating voltages and fail the circuit to recognize the voltage value as logic 0 or 1 correctly.
- Digital logic eliminates these real-world problems by …
  - not recognize a single voltage as 0 or 1 by restoring degraded imperfect values as logic value 0 or 1
  - not propagating small errors in voltage values





Encoding

- Digital representations exist for every object in the world (i.e., numbers, characters, images, sound).
- A value must be encoded into a binary string of 0s and 1s with an agreed-upon interpretation.
  - Switches are to compute logic expressions: $T \vee F = T \equiv 1+0=1$
  - Examples of binary encoding: 2 → 010, 3 → 011, 5→ 101
  - Computations on numbers are done in digital (binary) representation.
    ➔ 2 + 3 = 5 → 010 + 011 = 101 (*decimal-to-binary encoding*)
- Each binary digit is called a **bit**.
  - The length of a binary string to represent a decimal number n is $\lceil \lg n \rceil$.
  - A 32-bit binary representation can represent up to
    4 billion ($2^{32} = 4 \times 10^9$) distinct numbers.

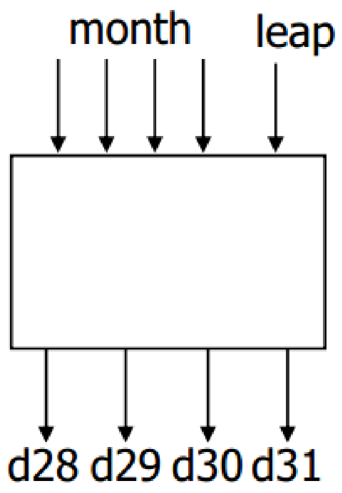
Example : Calendar subsystem

- 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 ‘january’: return (31);
      case ‘february’: if (leap_year_flag == 1) then
         return (29) else return (28);
      case ‘march’: return (31);
      ...
      case ‘december’: return (31);
      default: return (0);
   }
}
```

Implementing 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-	-	-	-	-	-

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?

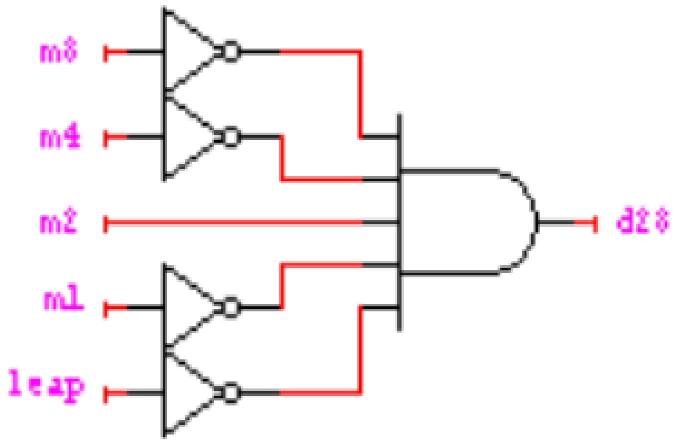
input don't care

output don't care

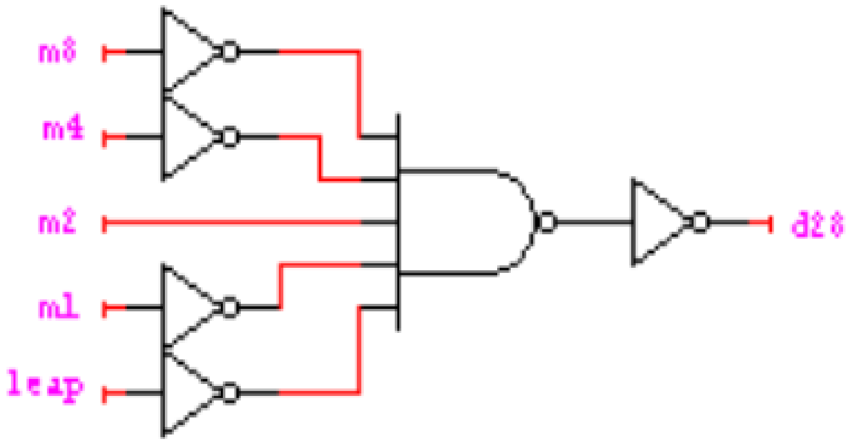
month	leap	d28	d29	d30	D31
0001	-	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	-	0	0	0	1
0100	-	0	0	1	0
...					
1100	-	0	0	0	1
1101	-	-	-	-	-
111-	-	-	-	-	-

Boolean expressions for outputs

- d28 = m8'•m4'•m2•m1'•leap'



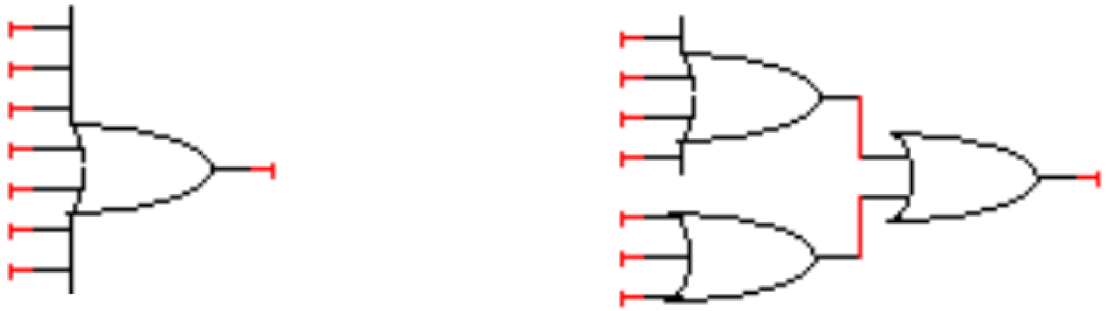
more realistic design for CMOS



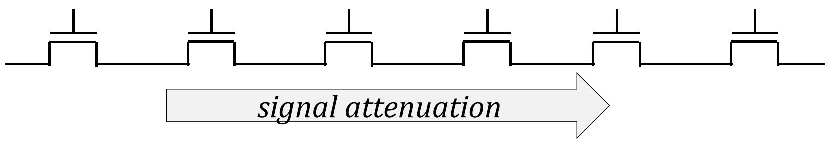
- d29 = m8'•m4'•m2•m1'•leap
- d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
  (m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
  = (m8'•m4•m1') + (m8•m4'•m1)

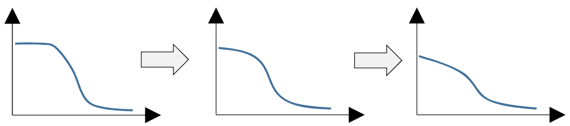
Boolean expressions for outputs

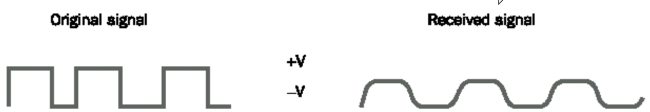
- $d31 = (m8' \bullet m4' \bullet m2' \bullet m1) + (m8' \bullet m4' \bullet m2 \bullet m1) + (m8' \bullet m4 \bullet m2' \bullet m1) + (m8' \bullet m4 \bullet m2 \bullet m1) + (m8 \bullet m4' \bullet m2' \bullet m1') + (m8 \bullet m4' \bullet m2 \bullet m1') + (m8 \bullet m4 \bullet m2' \bullet m1')$



Alternate realizations for a 7-input OR function (gates with more than 4 fan-ins are not practical considering CMOS physical structures)







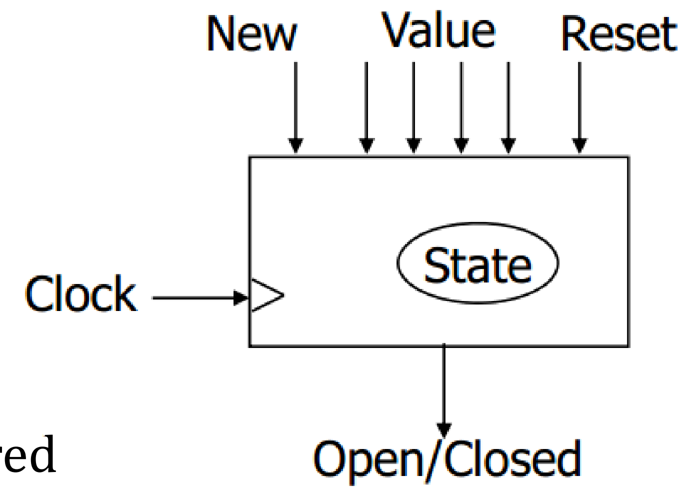
Another example : Combination Lock

- 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 (v3 != c[3]) then error = 1;
   if (error == 1) then return(0);
                   else return (1);
}
```

Implementing a sequential digital system

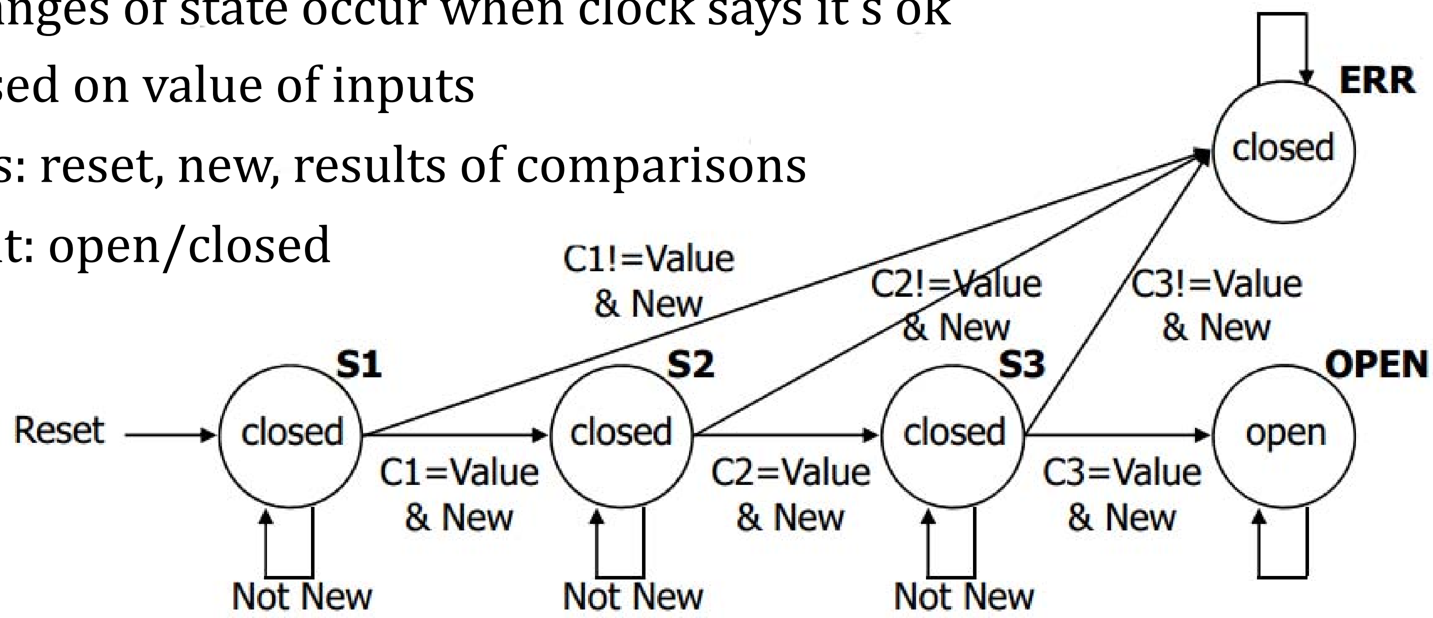
- Encoding:
  - how many bits per input value?
  - how many values in sequence?
  - how do we know a new input value is entered?
  - how do we represent the states of the system?
- Behavior:
  - clock wire tells us when it's ok
    to look at inputs
    (i.e., they have settled after change)
  - sequential: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



to remember all these, we maintain a ***finite*** *set of* ***states*** *in memory*

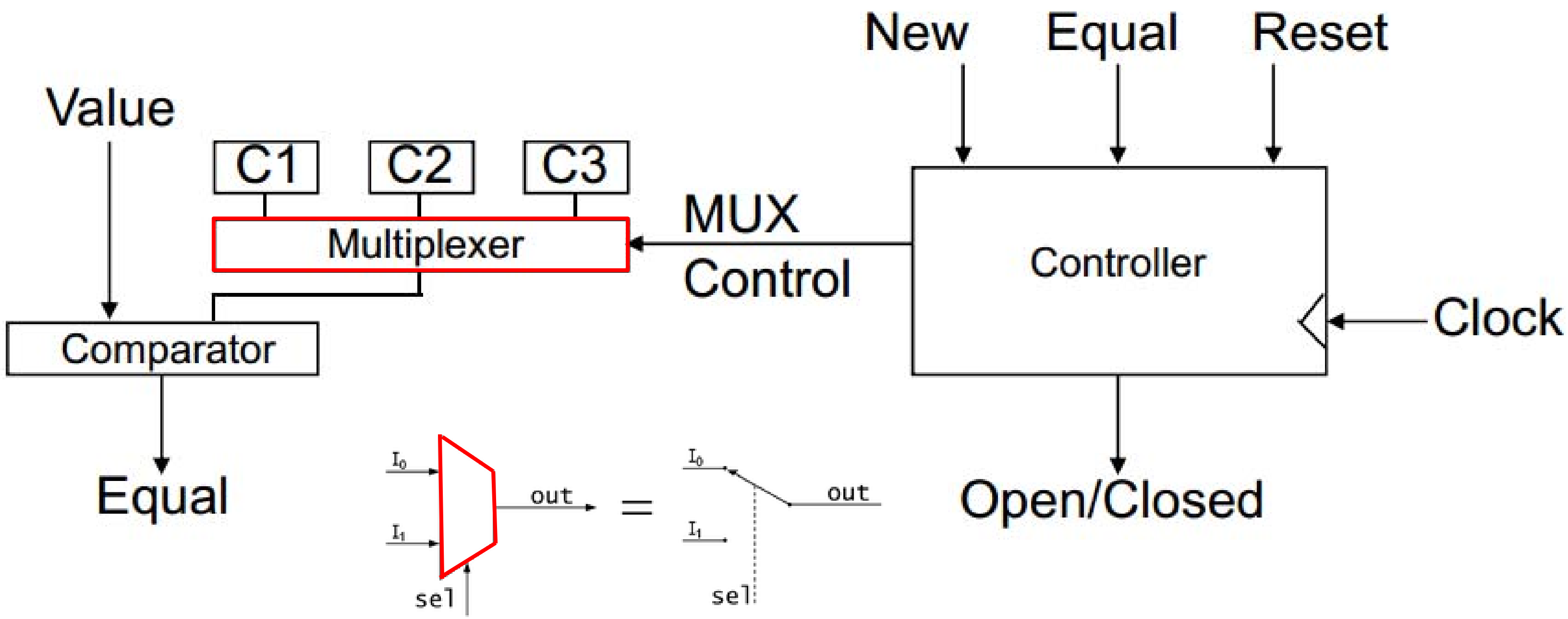
Abstract control

- 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



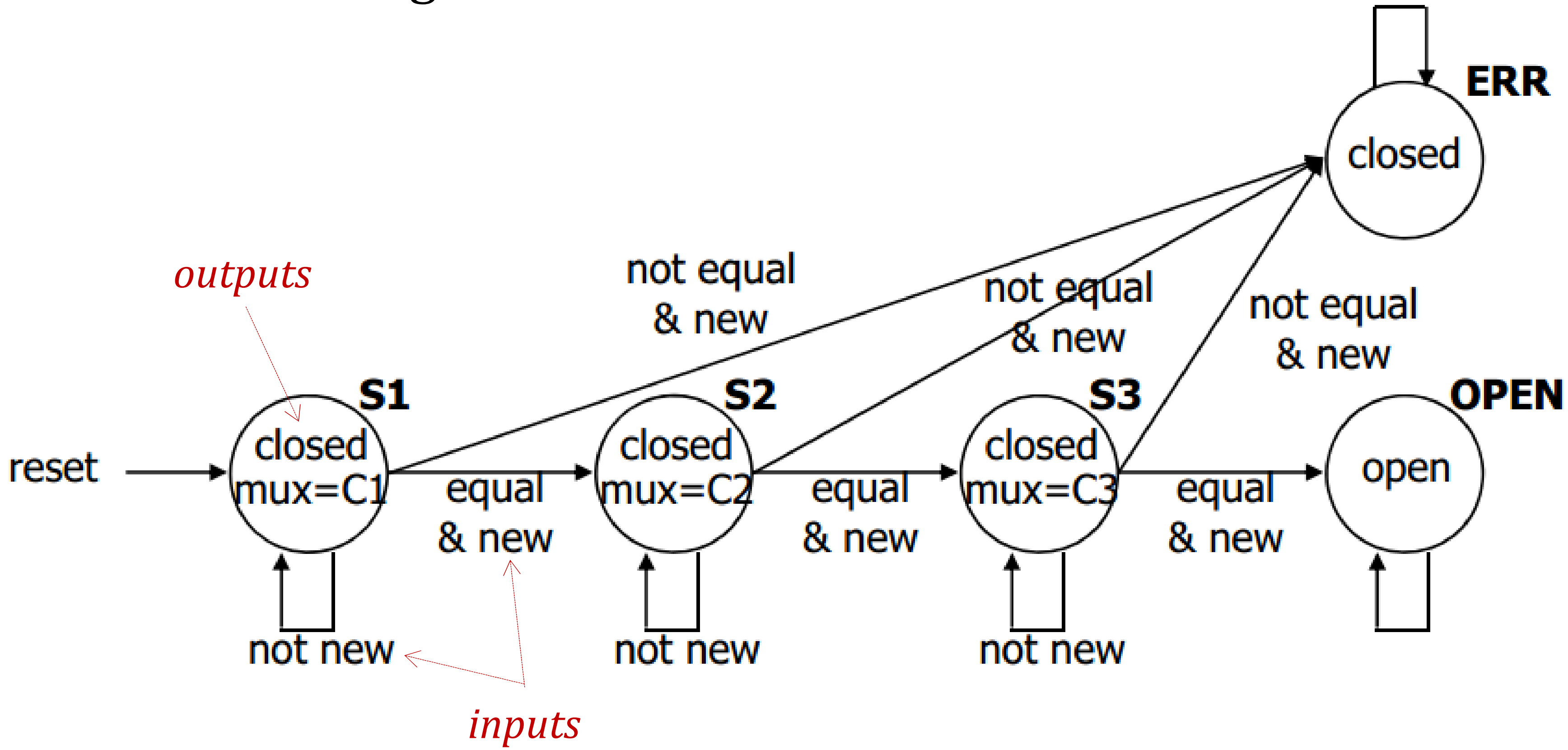
Internal structure: data-path vs. control

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



Finite-state machine

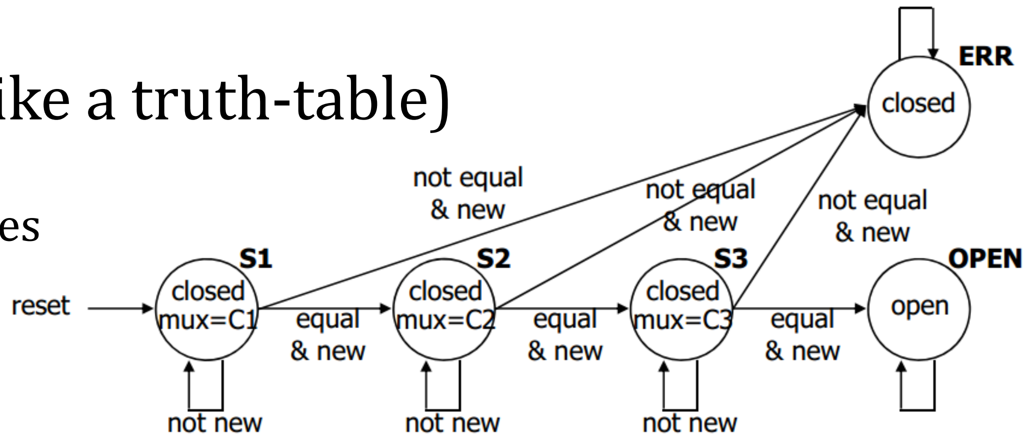
- refine state diagram to include internal structure



Finite-state machine

- generate state table (much like a truth-table)

It has additional state names



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

Encoding 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
    - 4 bits are more wasteful but more flexible than 3 bits
    - Such flexibility often simplifies circuits.

choose 5 codes from 2^3
choose 5 codes from 2^4

- 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

Encoding 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

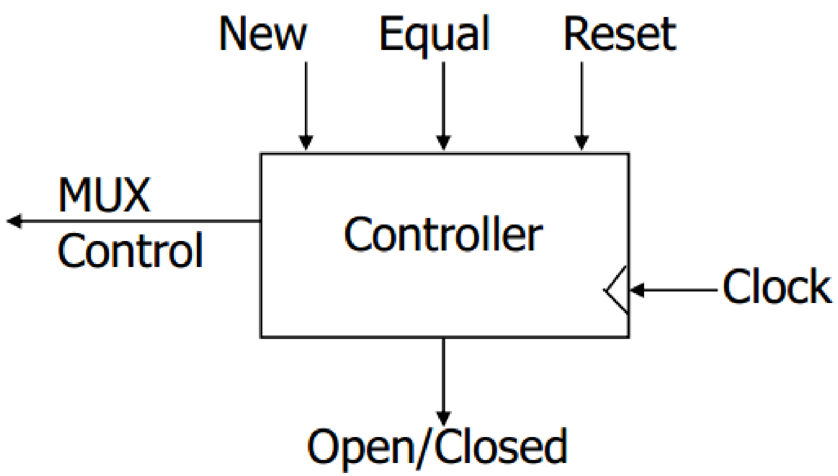
reset	new	equal	state	next state	mux	open/closed
1	-	-	-	0001	001	0
0	0	-	0001	0001	001	0
0	1	0	0001	0000	-	0
0	1	1	0001	0010	010	0
0	0	-	0010	0010	010	0
0	1	0	0010	0000	-	0
0	1	1	0010	0100	100	0
0	0	-	0100	0100	100	0
0	1	0	0100	0000	-	0
0	1	1	0100	1000	-	1
0	-	-	1000	1000	-	1
0	-	-	0000	0000	-	0

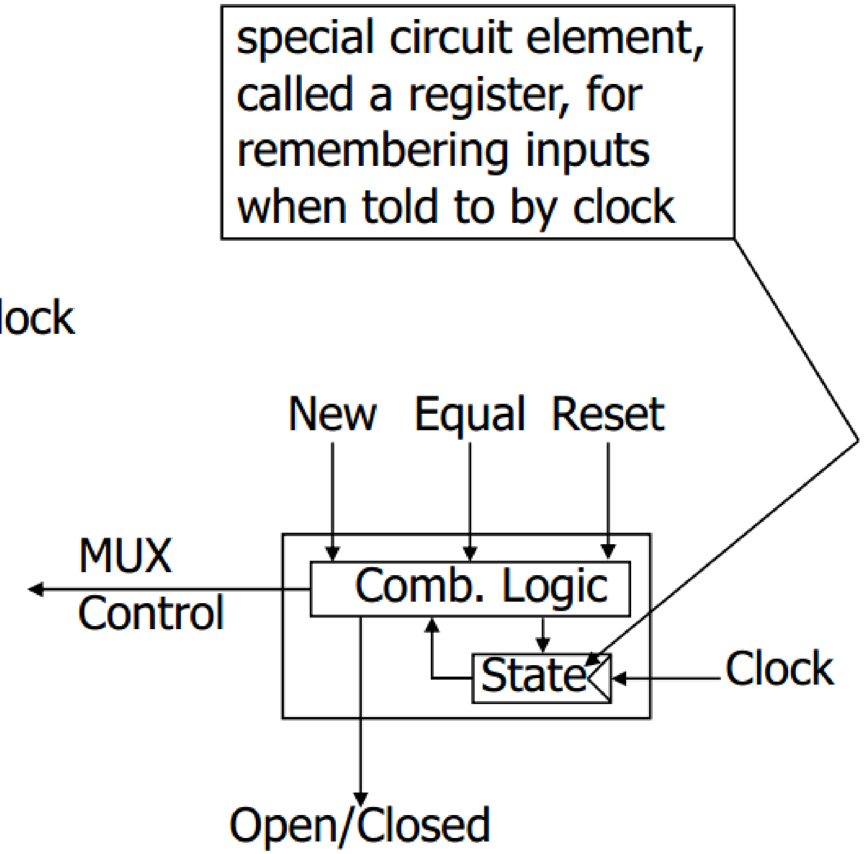
good choice of encoding!

mux is identical to last 3 bits of state

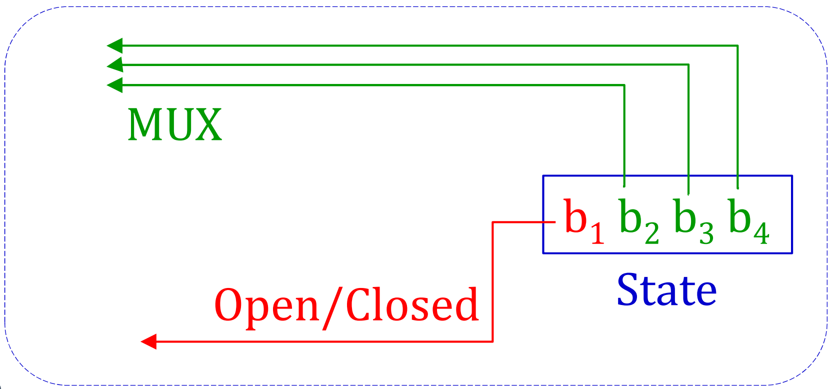
open/closed is identical to first bit of state

Controller implementation





Thanks to clever encoding choices, we can greatly simplify output circuits by using the same wires used to represent our current state



Design hierarchy