

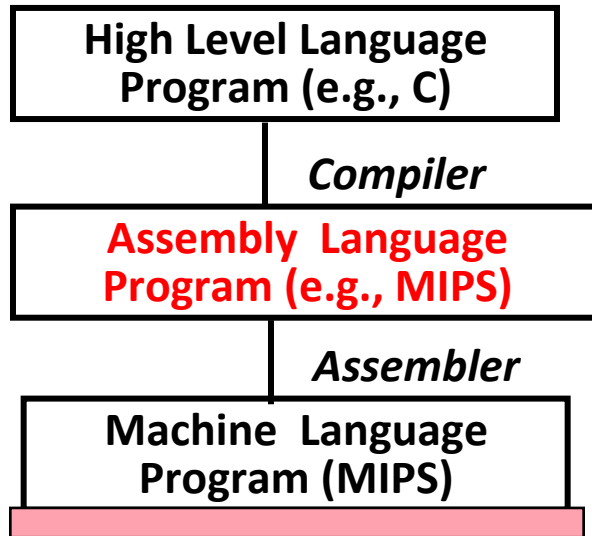
CS 61C:
Great Ideas in Computer Architecture
Synchronous Digital Systems

Instructors:

John Wawrzynek & Vladimir Stojanovic

<http://inst.eecs.Berkeley.edu/~cs61c/fa15>

Levels of Representation/ Interpretation

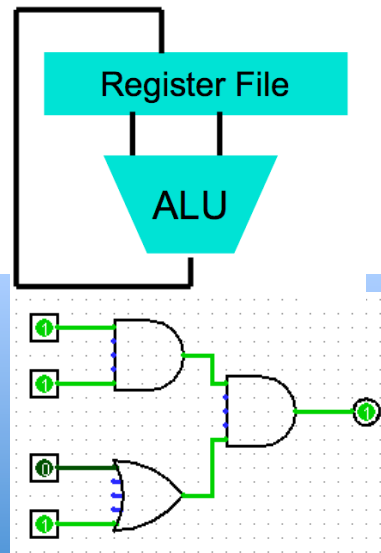
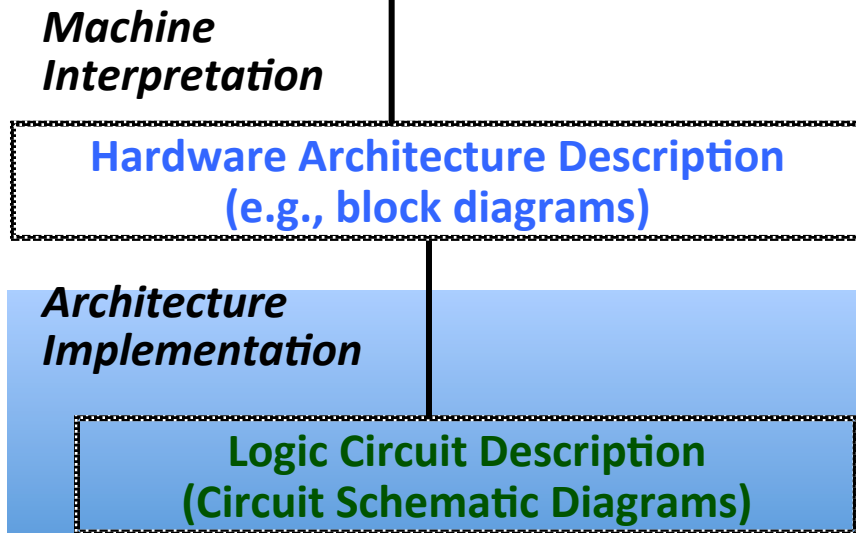


```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
lw $t0, 0($2)  
lw $t1, 4($2)  
sw $t1, 0($2)  
sw $t0, 4($2)
```

Anything can be represented
as a *number*,
i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```



Hardware Design

- Next several weeks: how a modern processor is built, starting with basic elements as building blocks
- Why study hardware design?
 - Understand capabilities and limitations of HW in general and processors in particular
 - What processors can do fast and what they can't do fast (avoid slow things if you want your code to run fast!)
 - Background for more in-depth HW courses (EECS 151, CS 152)
 - Hard to know what you'll need for next 30 years
 - There is only so much you can do with standard processors: you may need to design own custom HW for extra performance
 - Even some commercial processors today have customizable hardware!

Synchronous Digital Systems

Hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

Synchronous:

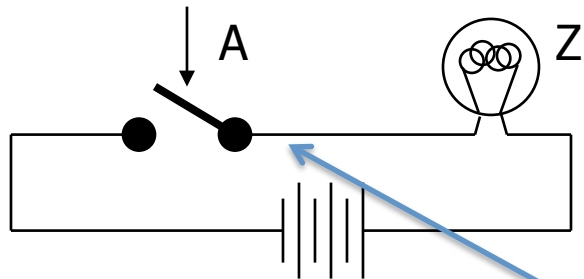
- All operations coordinated by a central clock
 - “Heartbeat” of the system!

Digital:

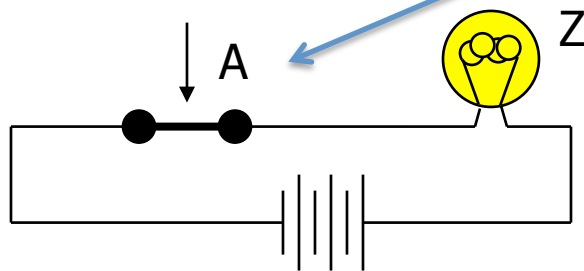
- Represent all values by discrete values
- Two binary digits: 1 and 0
- Electrical signals are treated as 1's and 0's
 - 1 and 0 are complements of each other
- High /low voltage for true / false, 1 / 0

Switches: Basic Element of Physical Implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1” or is *asserted*):



On-switch (if A is “1” or asserted) turns-on light bulb (Z)

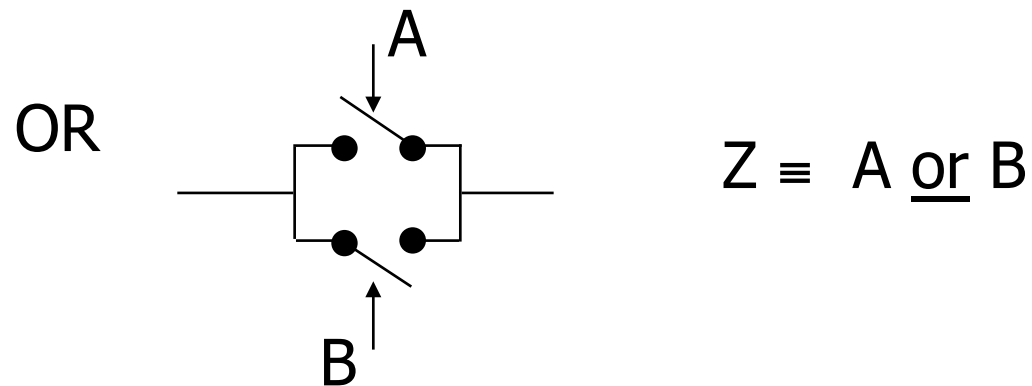
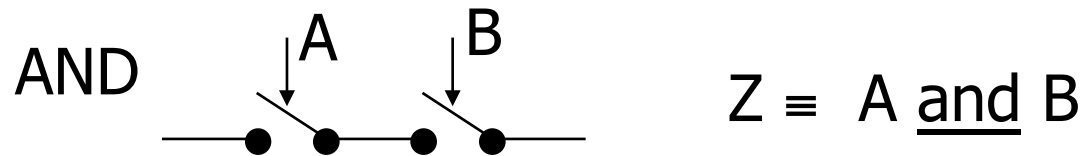


Off-switch (if A is “0” or unasserted) turns-off light bulb (Z)

$$Z \equiv A$$

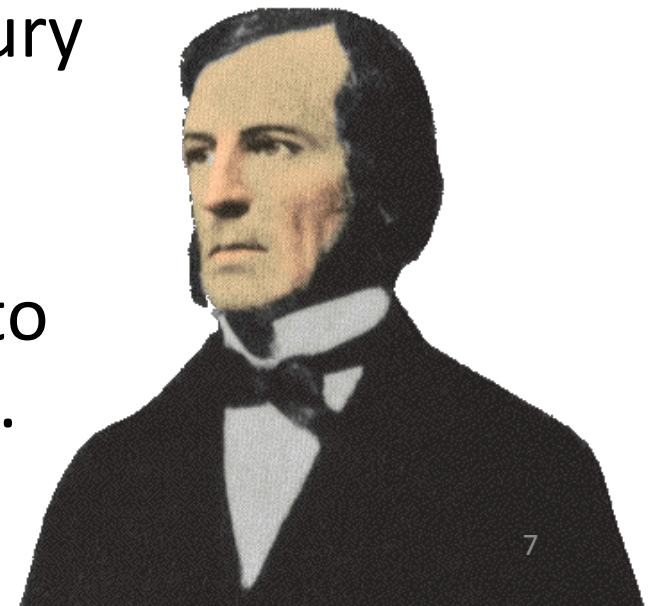
Switches (cont'd)

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



Historical Note

- Early computer designers built ad hoc circuits from switches
- Began to notice common patterns in their work: ANDs, ORs, ...
- Master's thesis (by Claude Shannon, 1940) made link between work and 19th Century Mathematician George Boole
 - Called it “Boolean” in his honor
- Could apply math to give theory to hardware design, minimization, ...



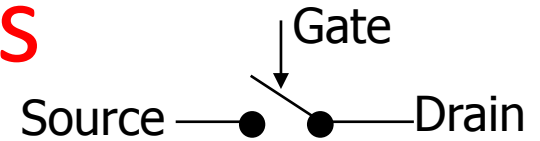
Transistors

- High voltage (V_{dd}) represents 1, or true
 - In modern microprocessors, $V_{dd} \sim 1.0$ Volt
- Low voltage (0 Volt or Ground) represents 0, or false
- Pick a midpoint voltage to decide if a 0 or a 1
 - Voltage greater than midpoint = 1
 - Voltage less than midpoint = 0
 - This removes noise as signals propagate – a big advantage of digital systems over analog systems
- If one switch can control another switch, we can build a computer!
- Our switches: CMOS transistors

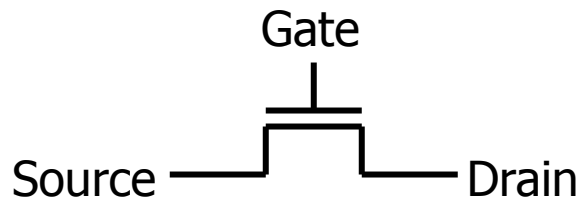
CMOS Transistor Networks

- Modern digital systems designed in CMOS
 - MOS: Metal-Oxide on Semiconductor
 - C for complementary: use *pairs* of normally-*on* and normally-*off* switches
- CMOS transistors act as voltage-controlled switches
 - Similar, though easier to work with, than electro-mechanical relay switches from earlier era
 - Use energy primarily when switching

CMOS Transistors



- Three terminals: source, gate, and drain
 - Switch action:
if voltage on gate terminal is (some amount) higher/lower than source terminal then conducting path established between drain and source terminals (switch is closed)

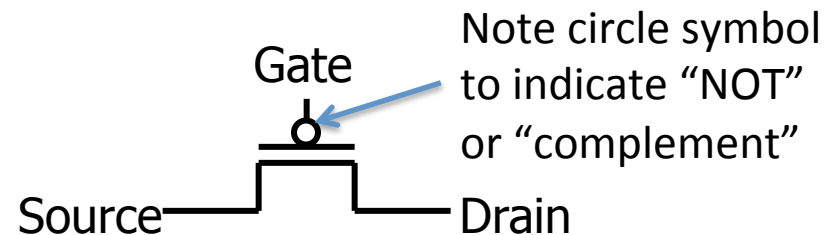


n-channel transistor

off when voltage at Gate is low

on when:

voltage(Gate) > voltage (Threshold)



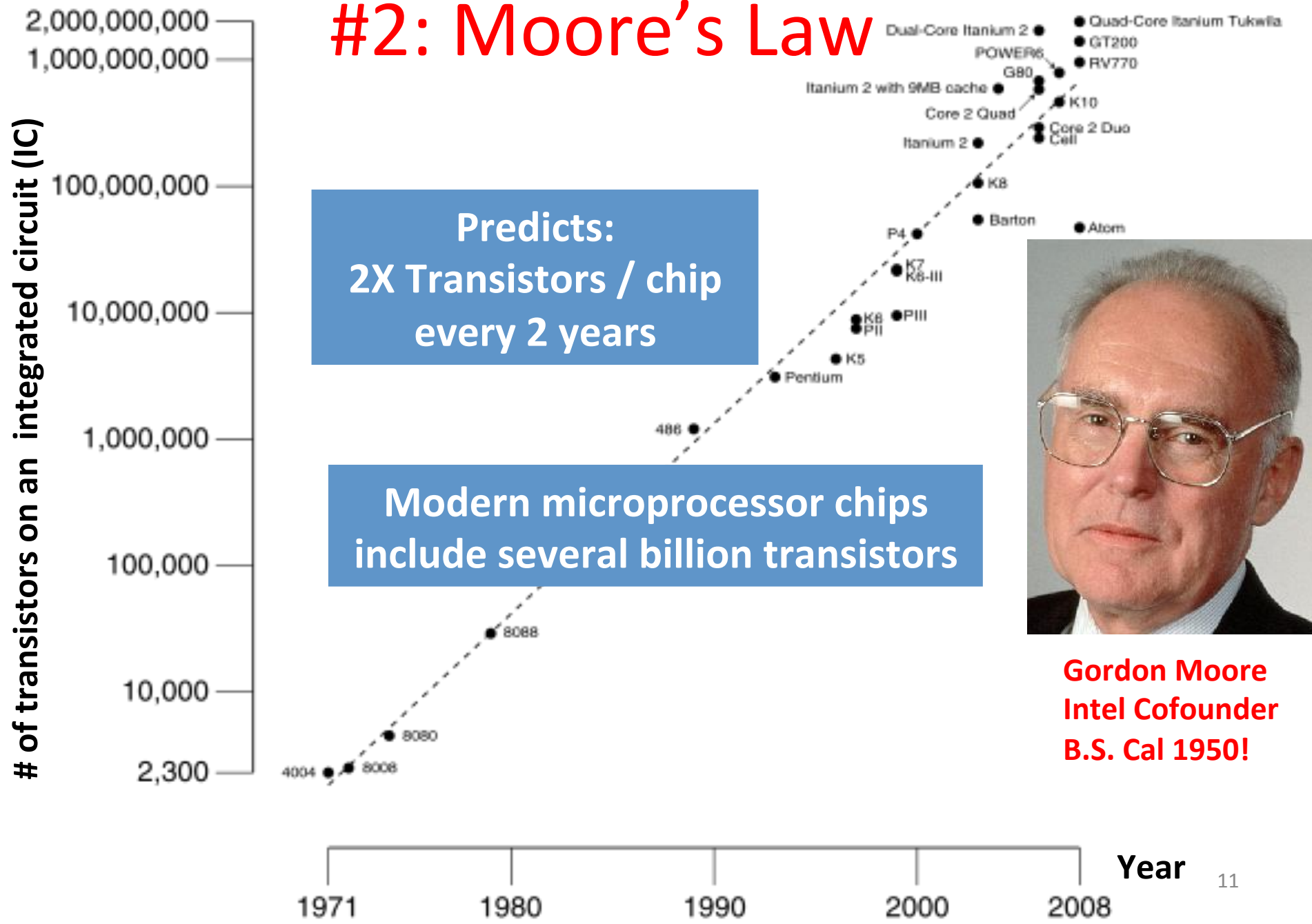
p-channel transistor

on when voltage at Gate is low

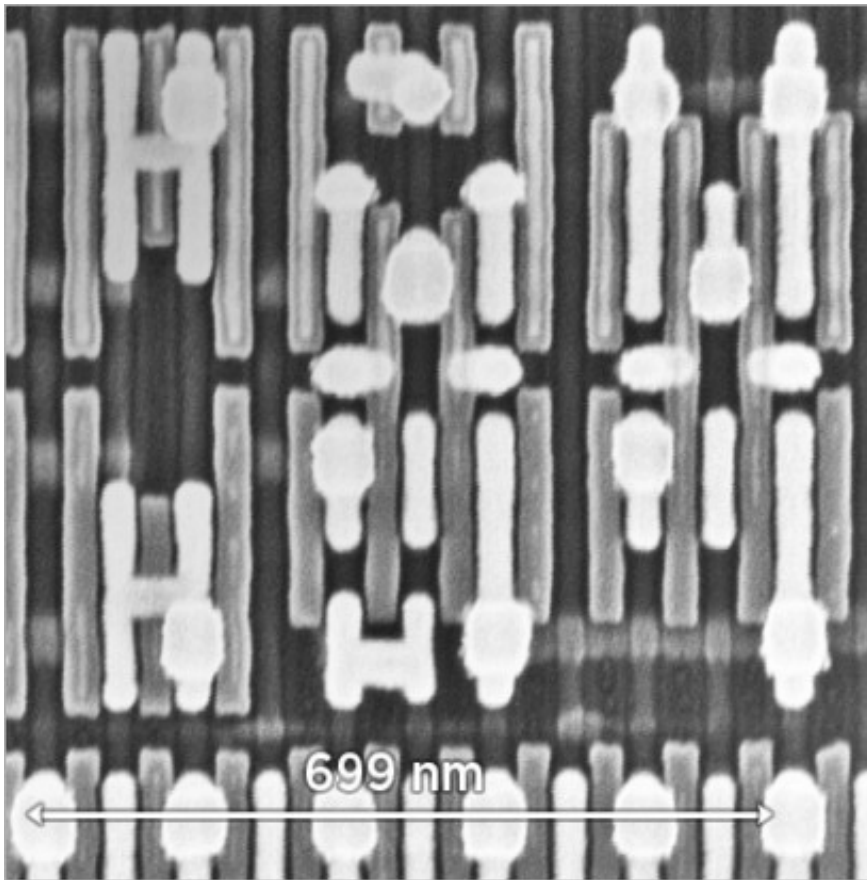
off when:

voltage(Gate) > voltage (Threshold)

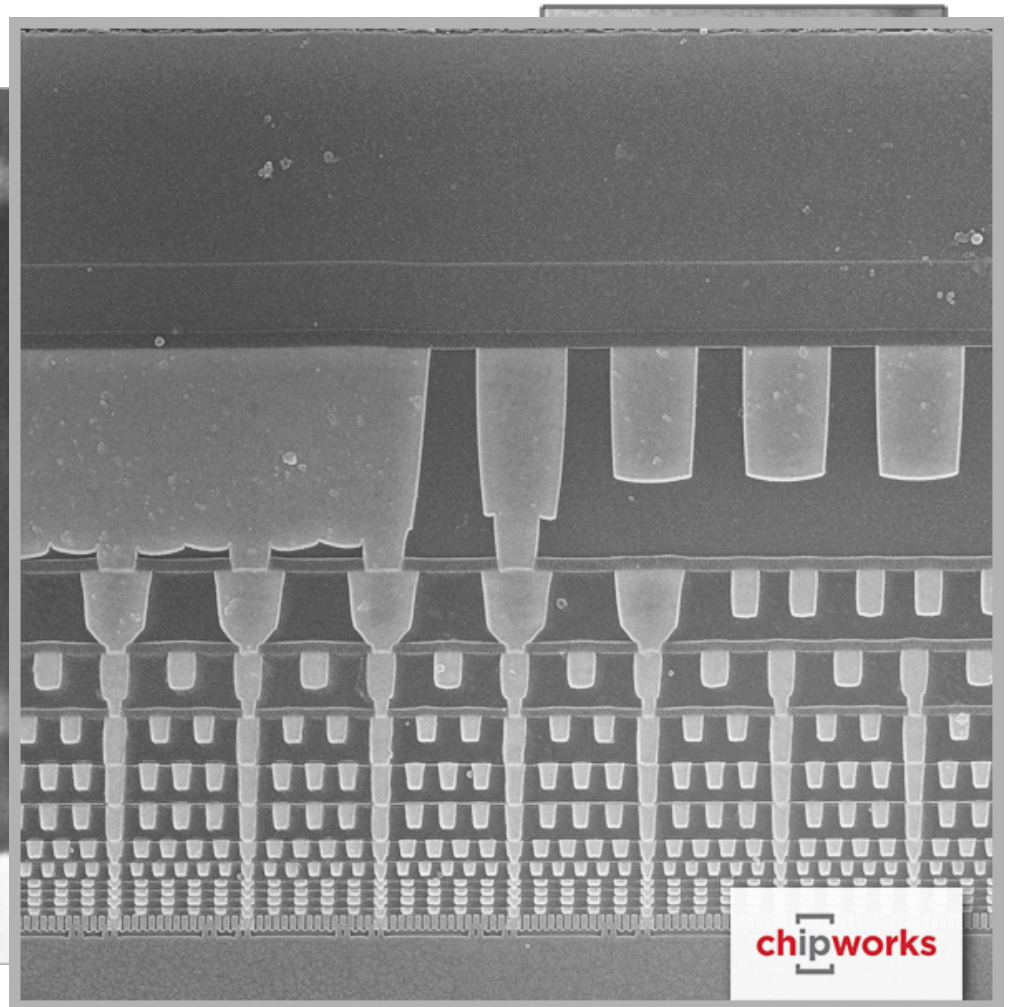
#2: Moore's Law



Intel 14nm Technology



Plan view of transistors



Side view of wiring layers

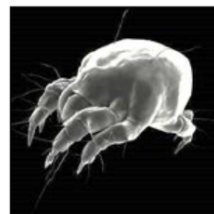
Sense of Scale



Mark
1.66 m



Fly
7 mm



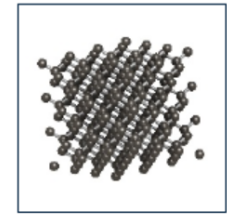
Mite
300 μm



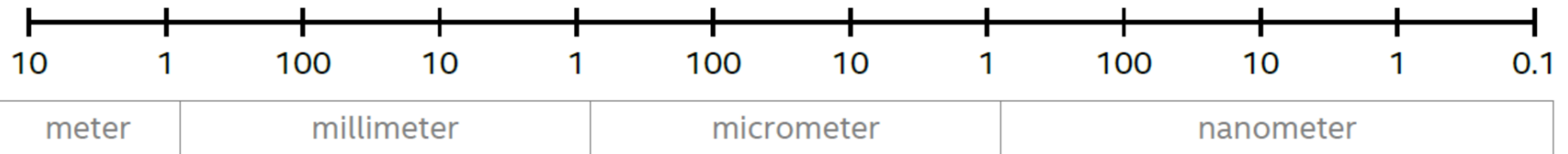
Blood Cell
7 μm



Virus
100 nm



Silicon Atom
0.24 nm



Source: Mark Bohr, IDF14

CMOS Circuit Rules

- Don't pass weak values => Use Complementary Pairs
 - N-type transistors pass weak 1's ($V_{dd} - V_{th}$)
 - N-type transistors pass strong 0's (ground)
 - Use N-type transistors only to pass 0's (N for negative)
 - Converse for P-type transistors: Pass weak 0s, strong 1s
 - Pass weak 0's (V_{th}), strong 1's (V_{dd})
 - Use P-type transistors only to pass 1's (P for positive)
 - Use pairs of N-type and P-type to get strong values
- Never leave a wire undriven
 - Make sure there's always a path to V_{dd} or GND
- Never create a path from V_{dd} to GND (ground)
 - This would short-circuit the power supply!

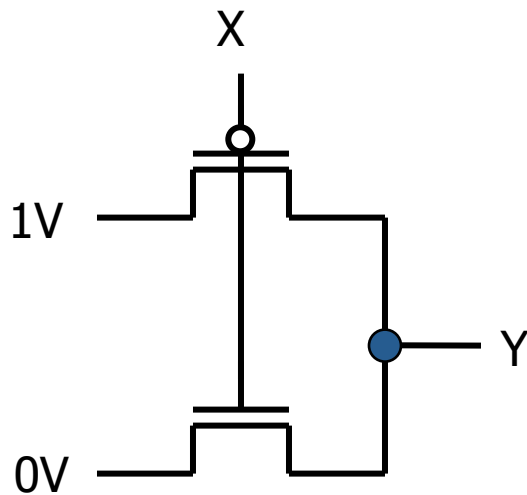
CMOS Networks

p-channel transistor

on when voltage at Gate is low

off when:

voltage(Gate) > voltage (Threshold)



n-channel transistor

off when voltage at Gate is low

on when:

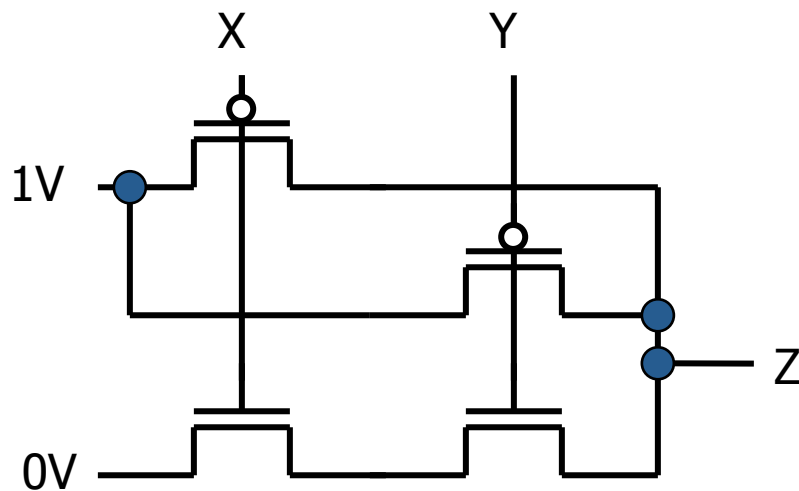
voltage(Gate) > voltage (Threshold)

what is the relationship between x and y?

x	y
0 Volt (GND)	1 Volt (Vdd)
1 Volt (Vdd)	0 Volt (GND)

Called an *inverter* or *not gate*

Two-Input Networks

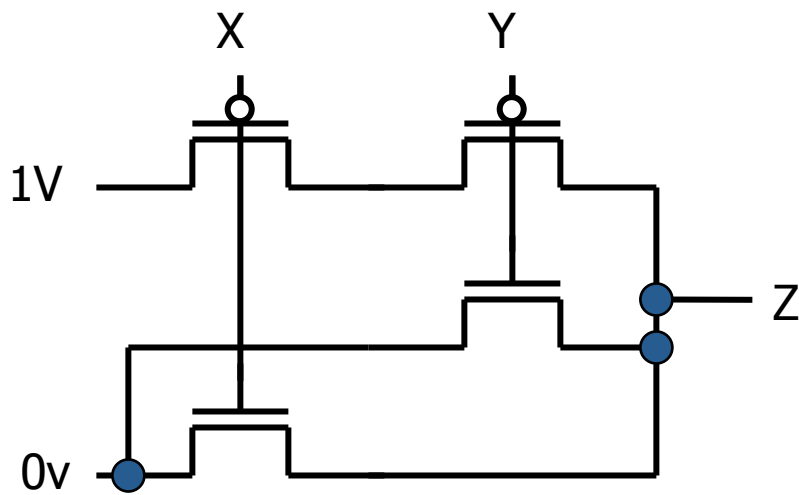


what is the relationship between x, y and z?

x	y	z
0 Volt	0 Volt	1 Volt
0 Volt	1 Volt	1 Volt
1 Volt	0 Volt	1 Volt
1 Volt	1 Volt	0 Volt

Called a *NAND gate*
(*NOT AND*)

Clickers/Peer Instruction



x	y	z				
		A	B	C	D	
0 Volt	0 Volt	0	0	1	1	Volts
0 Volt	1 Volt	0	1	0	1	Volts
1 Volt	0 Volt	0	1	0	1	Volts
1 Volt	1 Volt	1	1	0	0	Volts

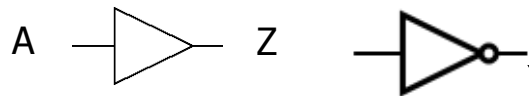
Administrivia

- Hopefully everyone turned-in Project 1
 - How many hours spent?
- Start projects early!
 - Project 2 releasing soon
- Guerilla Session: MIPS, Sat 9/26 noon-2pm @ 306 Soda

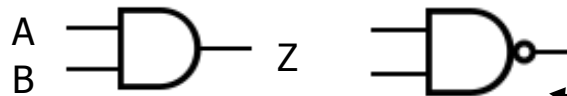
Combinational Logic Symbols

- Common combinational logic systems have standard symbols called logic gates

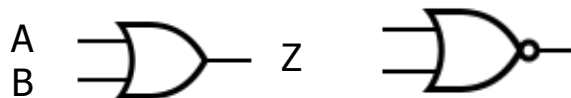
- Buffer, NOT



- AND, NAND



- OR, NOR



Inverting versions (NOT, NAND, NOR) easiest to implement with CMOS transistors (the switches we have available and use most)

Boolean Algebra

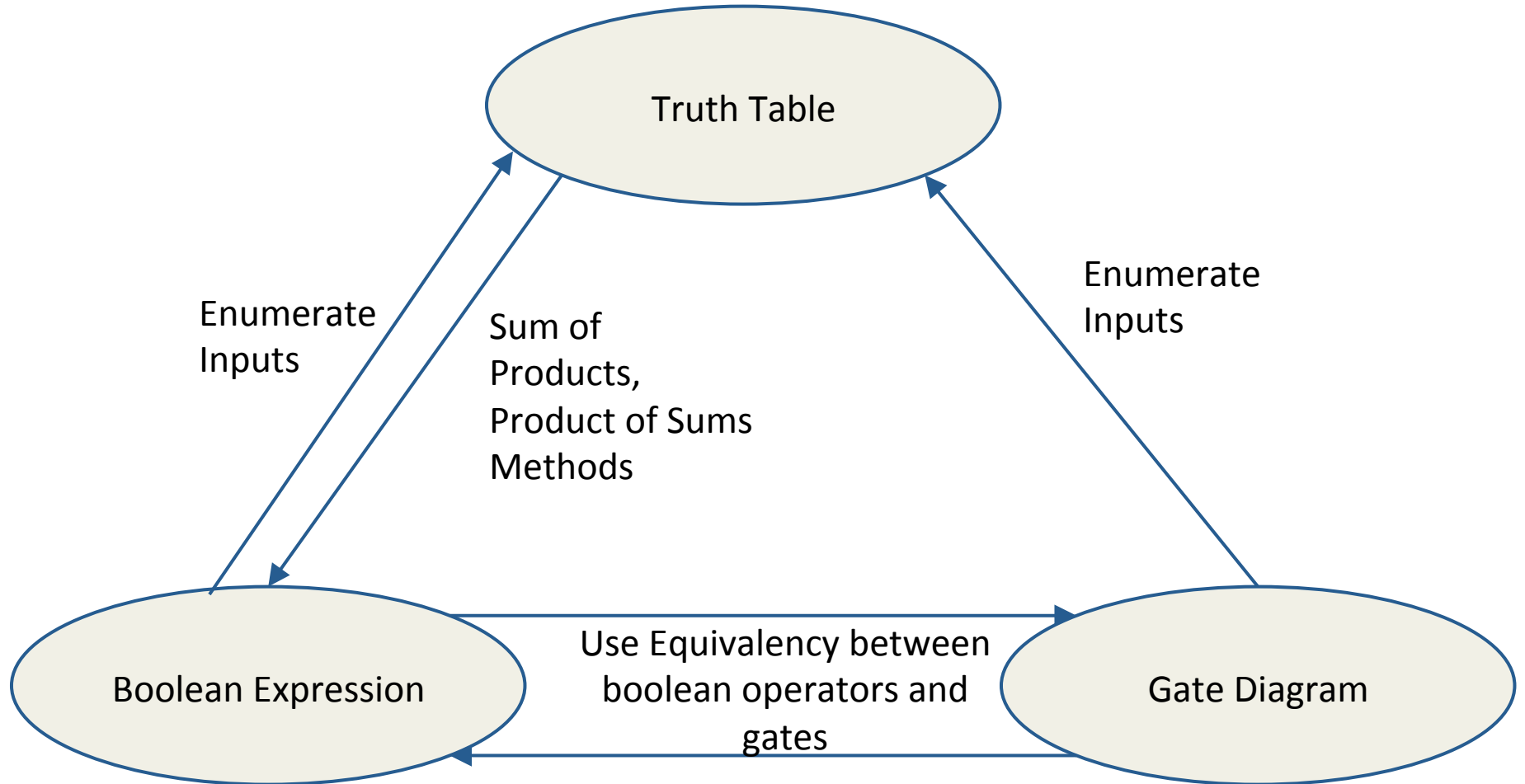
- Use plus “+” for OR
 - “logical sum”
- Use product for AND ($a \bullet b$ or implied via ab)
 - “logical product”
- “Hat” to mean complement (NOT)

• Thus

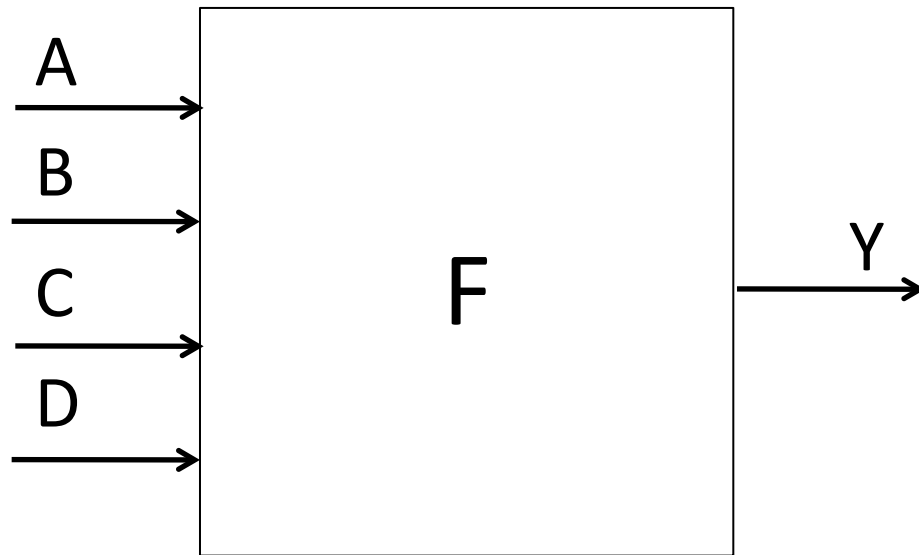
$$\begin{aligned} & ab + a + \bar{c} \\ = & a \bullet b + a + \bar{c} \\ = & (a \text{ AND } b) \text{ OR } a \text{ OR } (\text{NOT } c) \end{aligned}$$



Representations of Combinational Logic (groups of logic gates)



Truth Tables for Combinational Logic



Exhaustive list of the output value generated for each combination of inputs

How many logic functions can be defined with N inputs?

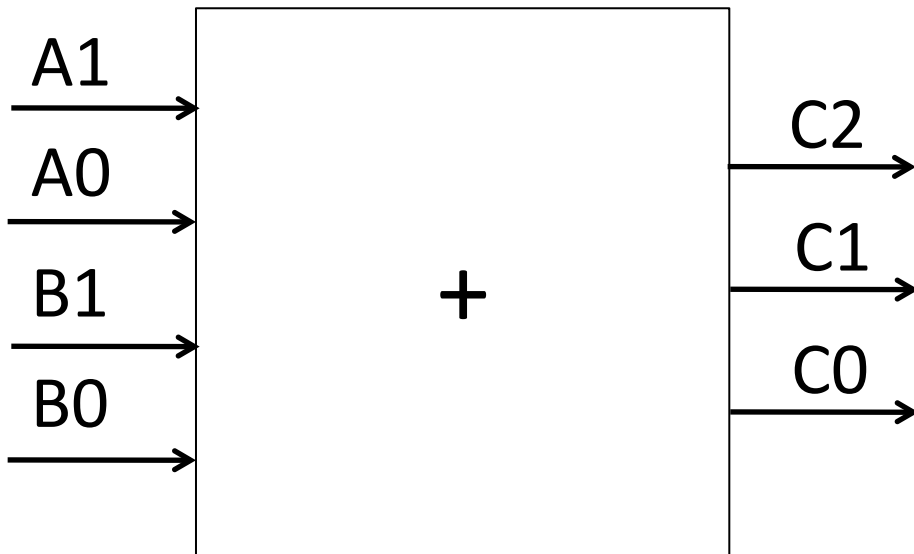
a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

Truth Table Example #1:

$y = F(a,b)$: 1 iff $a \neq b$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table Example #2: 2-bit Adder



How
Many
Rows?

A a_1a_0	B b_1b_0	C $c_2c_1c_0$

Truth Table Example #3: 32-bit Unsigned Adder

A	B	C
000 ... 0	000 ... 0	000 ... 00
000 ... 0	000 ... 1	000 ... 01
.	.	.
.	.	.
.	.	.
111 ... 1	111 ... 1	111 ... 10

How
Many
Rows?

Truth Table Example #4: 3-input Majority Circuit

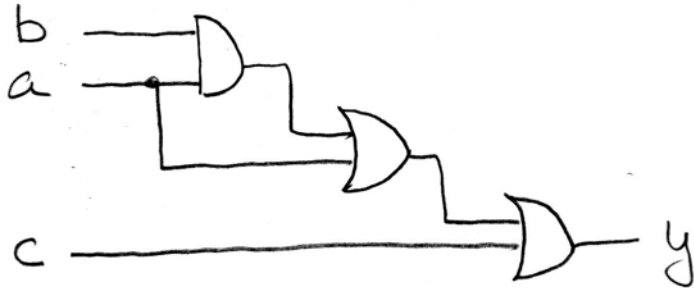
$Y =$

This is called *Sum of Products* form;
Just another way to represent the TT
as a logical expression

More simplified forms
(fewer gates and wires)

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Algebra: Circuit & Algebraic Simplification



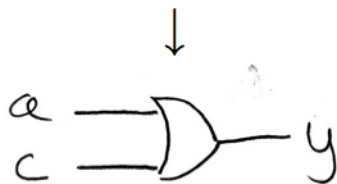
original circuit

$$y = ((ab) + a) + c$$

equation derived from original circuit

$$\begin{aligned} &\downarrow \\ &= ab + a + c \\ &\downarrow \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

algebraic simplification



simplified circuit

Laws of Boolean Algebra

$$X \bar{X} = 0$$

$$X 0 = 0$$

$$X 1 = X$$

$$X X = X$$

$$X Y = Y X$$

$$(X Y) Z = Z (Y Z)$$

$$X (Y + Z) = X Y + X Z$$

$$X Y + X = X$$

$$\bar{X} Y + X = X + Y$$

$$\overline{X Y} = \bar{X} + \bar{Y}$$

$$X + \bar{X} = 1$$

$$X + 1 = 1$$

$$X + 0 = X$$

$$X + X = X$$

$$X + Y = Y + X$$

$$(X + Y) + Z = Z + (Y + Z)$$

$$X + Y Z = (X + Y) (X + Z)$$

$$(X + Y) X = X$$

$$(\bar{X} + Y) X = X Y$$

$$\overline{X + Y} = \bar{X} \bar{Y}$$

Complementarity

Laws of 0's and 1's

Identities

Idempotent Laws

Commutativity

Associativity

Distribution

Uniting Theorem

Uniting Theorem v. 2

DeMorgan's Law

Boolean Algebraic Simplification Example

$$y = ab + a + c$$

Boolean Algebraic Simplification

Example

$$y = ab + a + c$$

$$a \ b \ c \ y = a(b + 1) + c \quad \text{distribution, identity}$$

$$0 \ 0 \ 0 \ 0 = a(1) + c \quad \text{law of 1's}$$

$$0 \ 0 \ 1 \ 1 = a + c \quad \text{identity}$$

$$0 \ 1 \ 0 \ 0$$

$$0 \ 1 \ 1 \ 1$$

$$1 \ 0 \ 0 \ 1$$

$$1 \ 0 \ 1 \ 1$$

$$1 \ 1 \ 0 \ 1$$

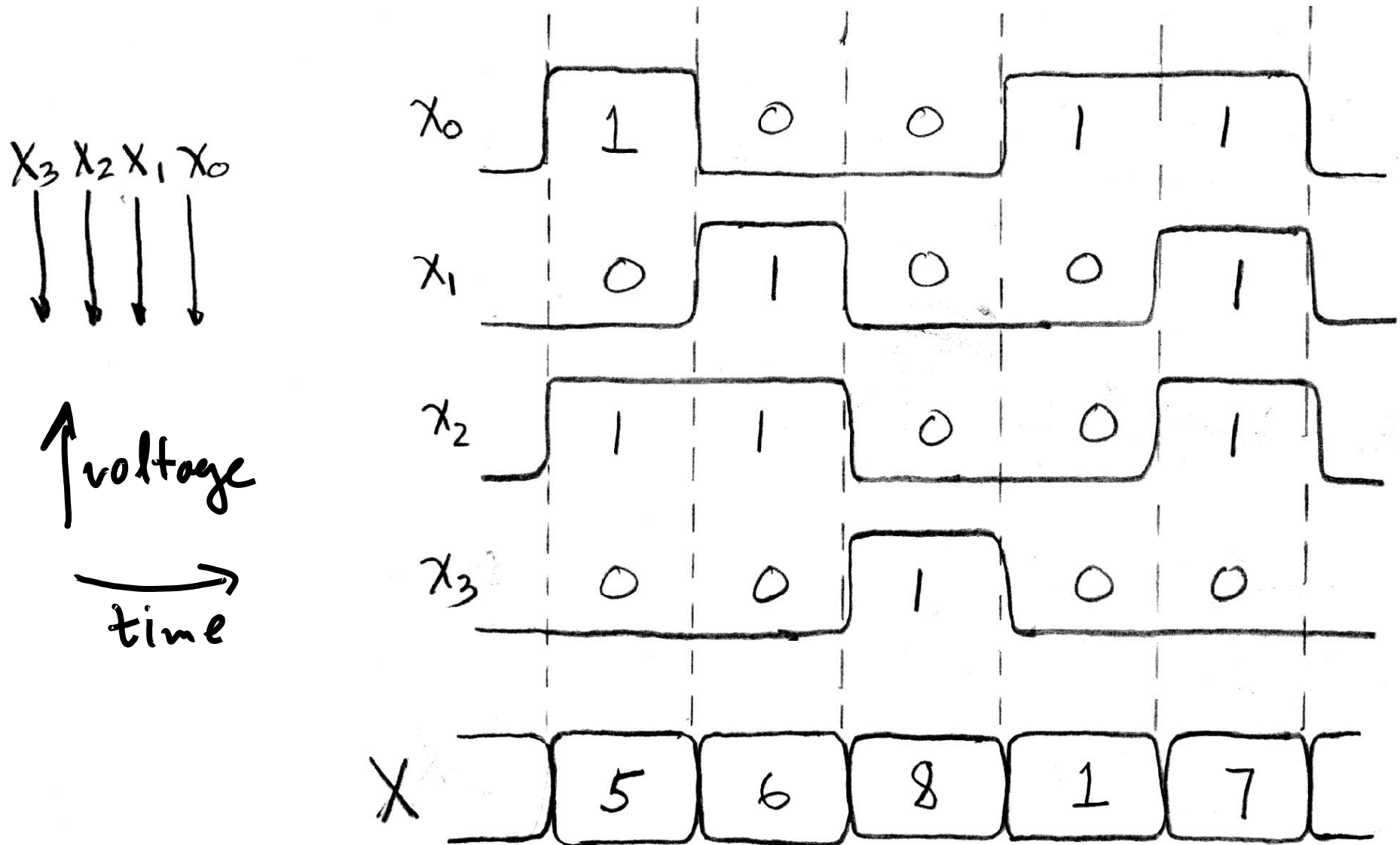
$$1 \ 1 \ 1 \ 1$$

Clickers/Peer Instruction

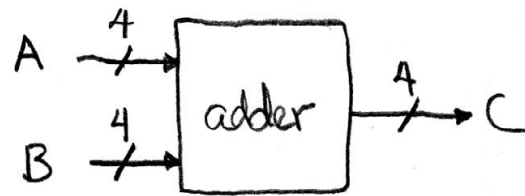
- Simplify $Z = A + BC + \overline{A}(\overline{BC})$
- A: $Z = 0$
- B: $Z = \overline{A(1 + BC)}$
- C: $Z = (A + BC)$
- D: $Z = BC$
- E: $Z = 1$

Break

Signals and Waveforms: Grouping

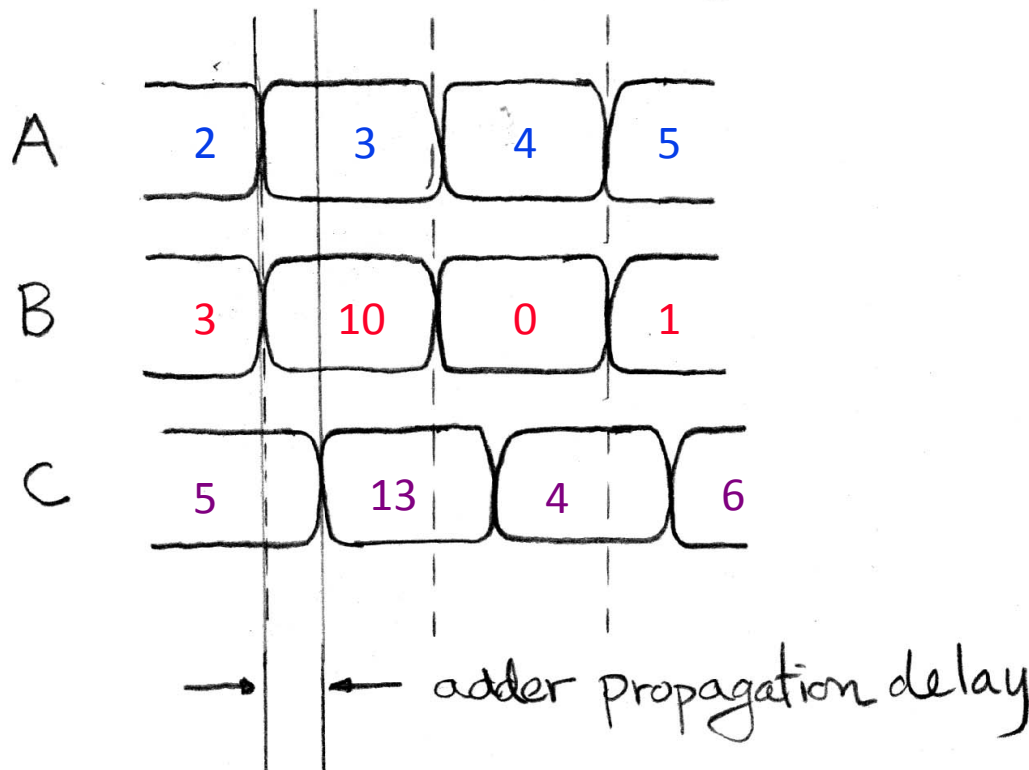
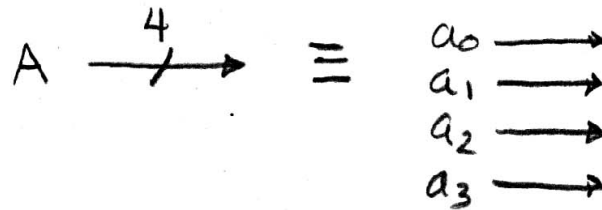


Signals and Waveforms: Circuit Delay

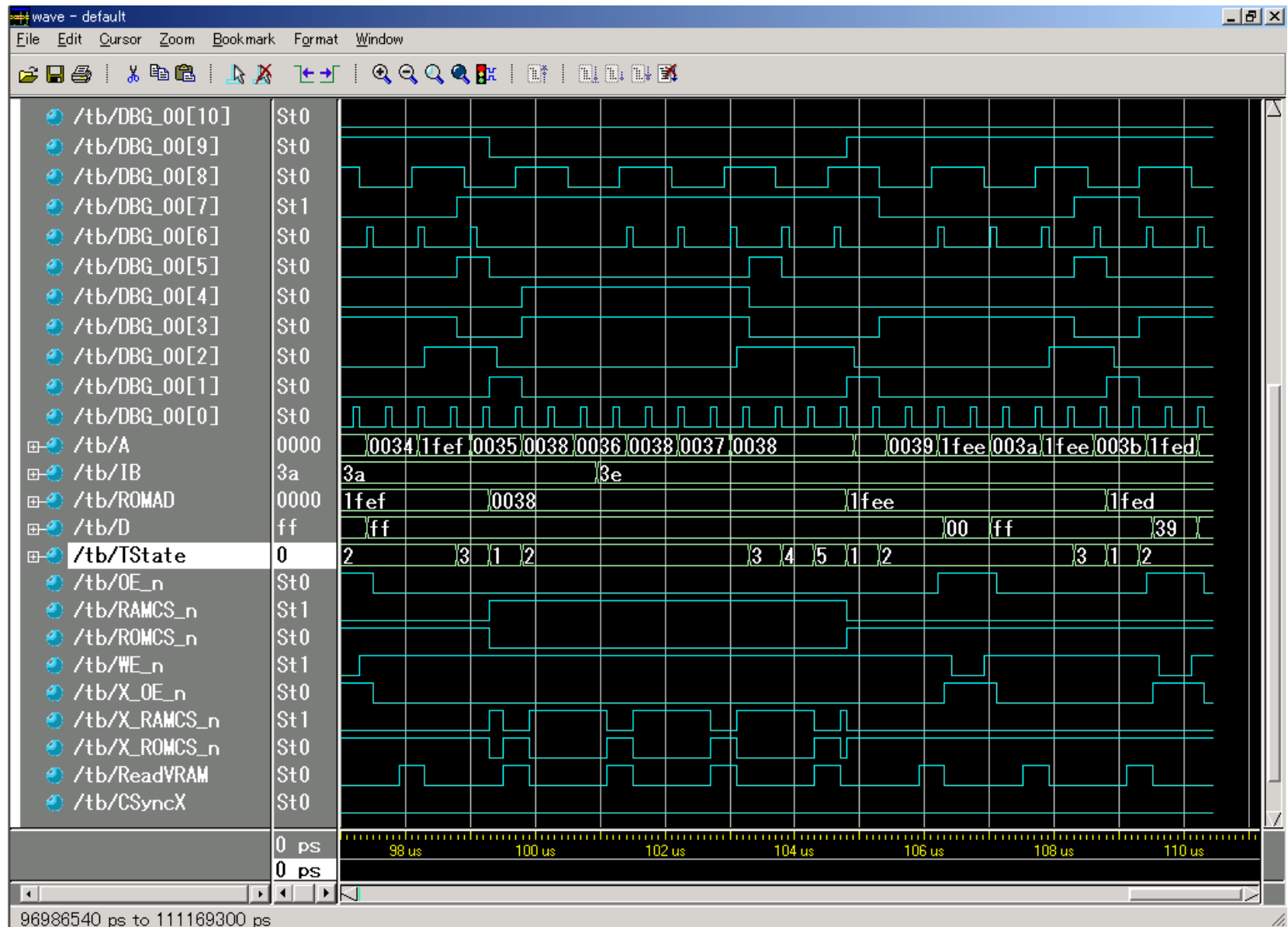


$$A = [a_3, a_2, a_1, a_0]$$

$$B = [b_3, b_2, b_1, b_0]$$



Sample Debugging Waveform



Type of Circuits

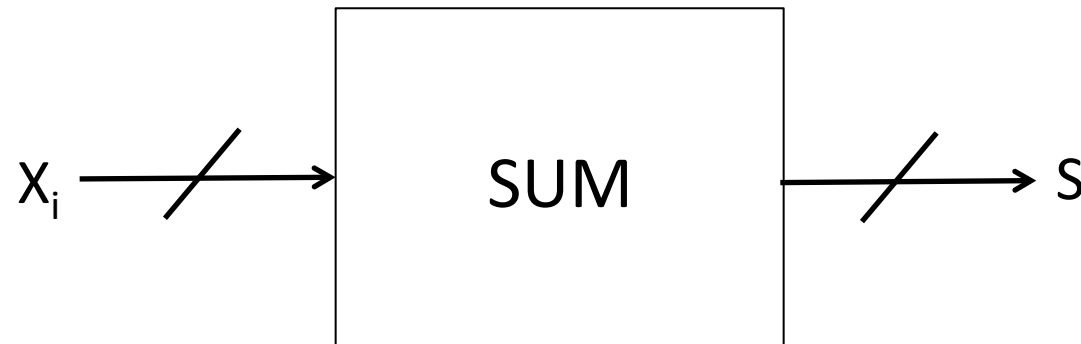
- *Synchronous Digital Systems* consist of two basic types of circuits:
 - Combinational Logic (CL) circuits
 - Output is a function of the inputs only, not the history of its execution
 - E.g., circuits to add A, B (ALUs)
 - Sequential Logic (SL)
 - Circuits that “remember” or store information
 - aka “State Elements”
 - E.g., memories and registers (Registers)

Uses for State Elements

- Place to store values for later re-use:
 - Register files (like \$1-\$31 in MIPS)
 - Memory (caches and main memory)
- *Help control flow of information between combinational logic blocks*
 - State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage

Accumulator Example

Why do we need to control the flow of information?



Want: $S=0;$

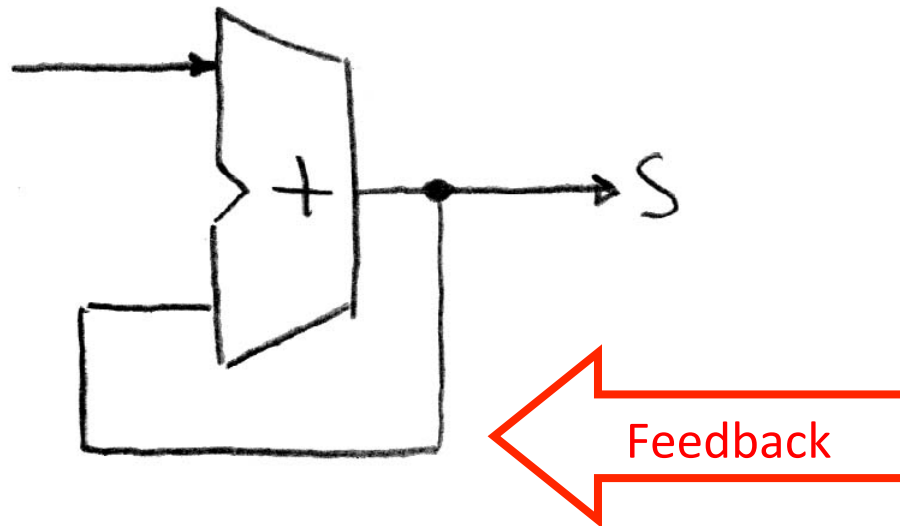
```
for (i=0; i<n; i++)
```

```
    S = S + Xi
```

Assume:

- Each X value is applied in succession, one per cycle
- After n cycles the sum is present on S

First Try: Does this work?

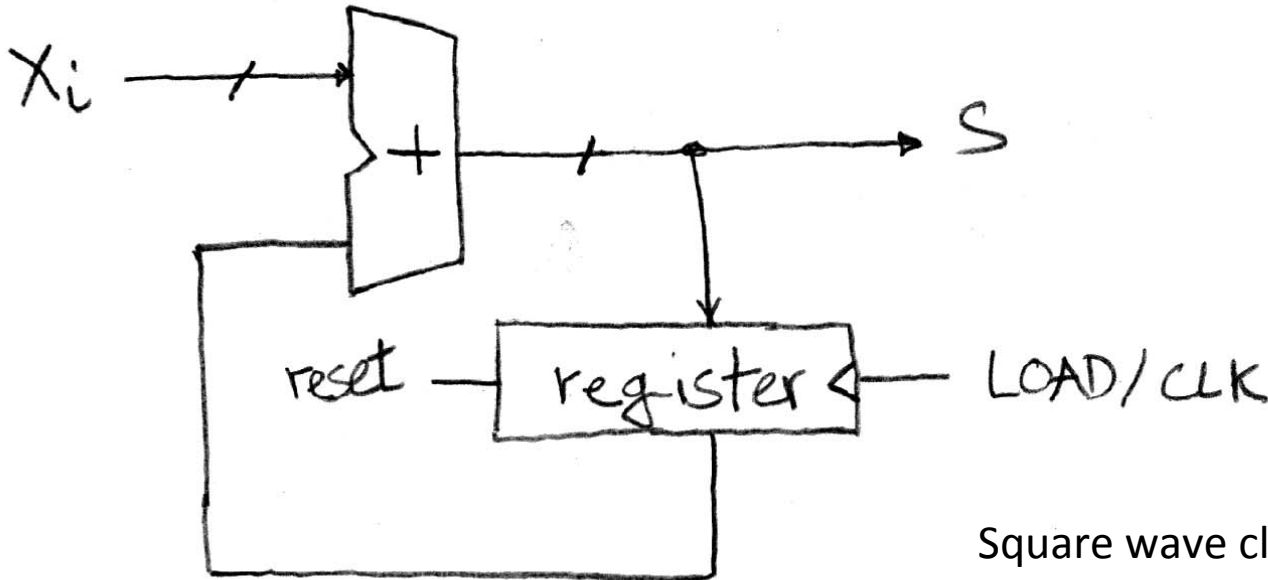


No!

Reason #1: How to control the next iteration of the 'for' loop?

Reason #2: How do we say: 'S=0'?

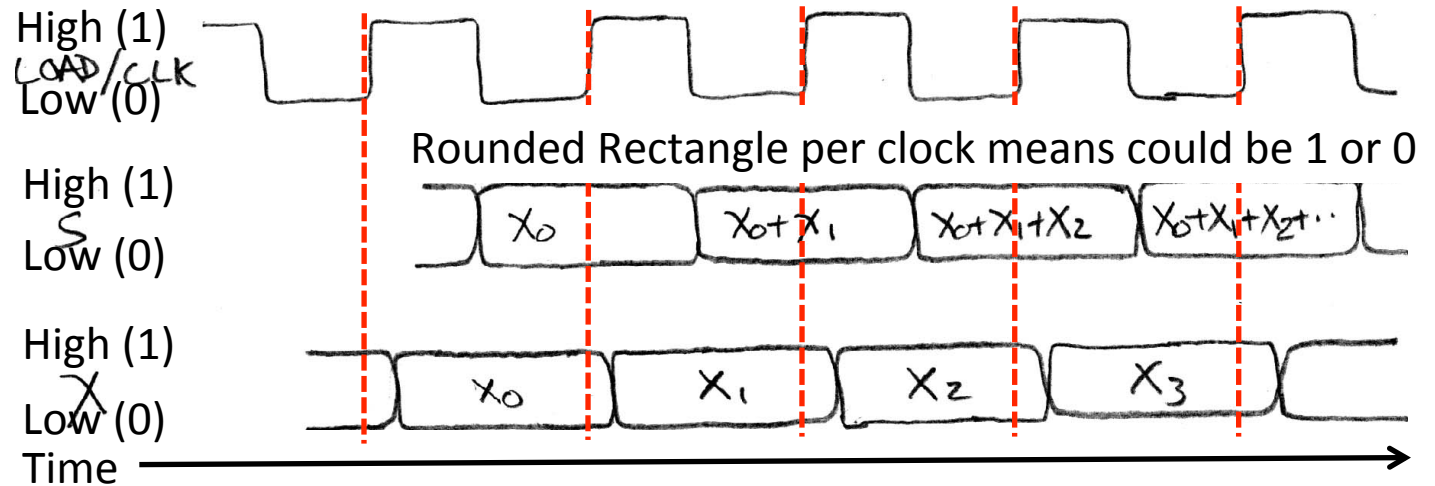
Second Try: How About This?



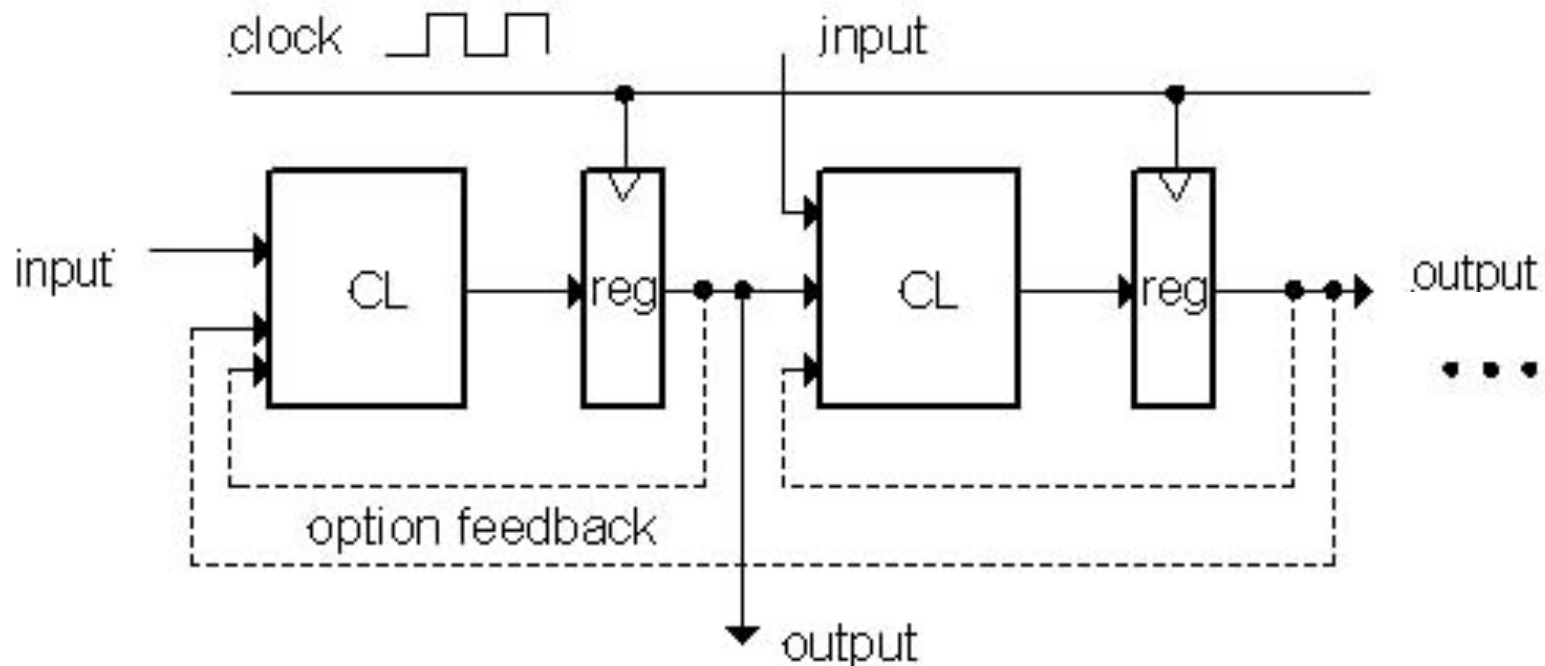
Register is used to hold up the transfer of data to adder

Square wave clock sets when things change

Rough timing ...

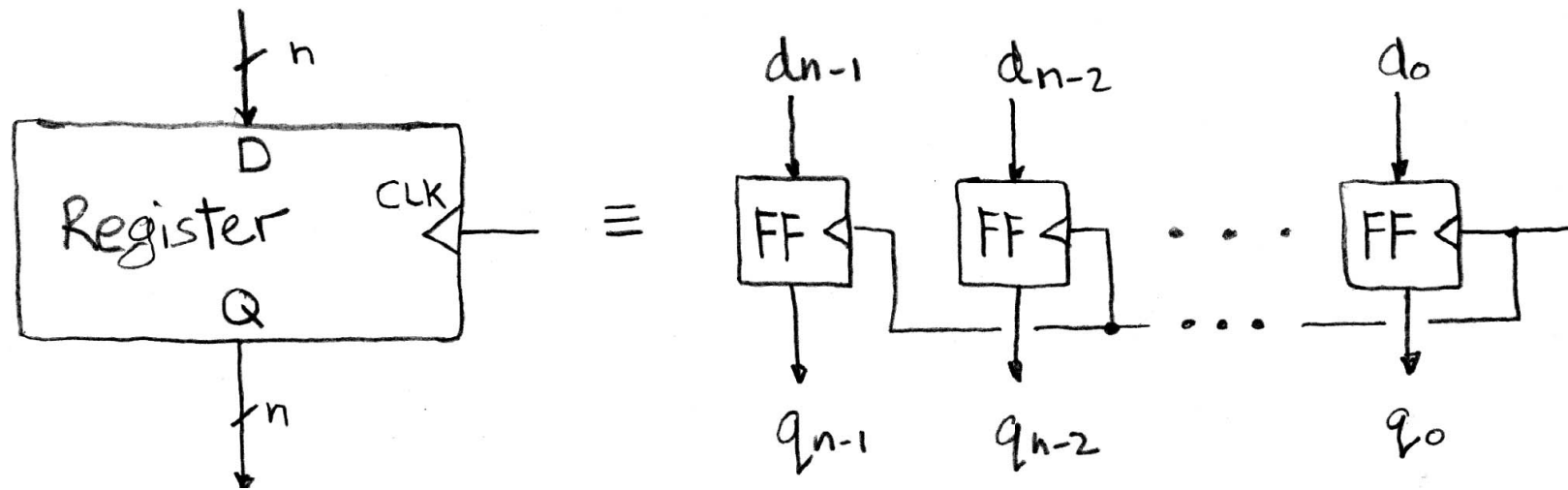


Model for Synchronous Systems



- Collection of Combinational Logic blocks separated by registers
- Feedback is optional
- Clock signal(s) connects only to clock input of registers
- Clock (CLK): steady square wave that synchronizes the system
- Register: several bits of state that samples on rising edge of CLK (positive edge-triggered) or falling edge (negative edge-triggered)

Register Internals



- n instances of a “Flip-Flop”
- Flip-flop name because the output flips and flops between 0 and 1
- D is “data input”, Q is “data output”
- Also called “D-type Flip-Flop”

Camera Analogy Timing Terms

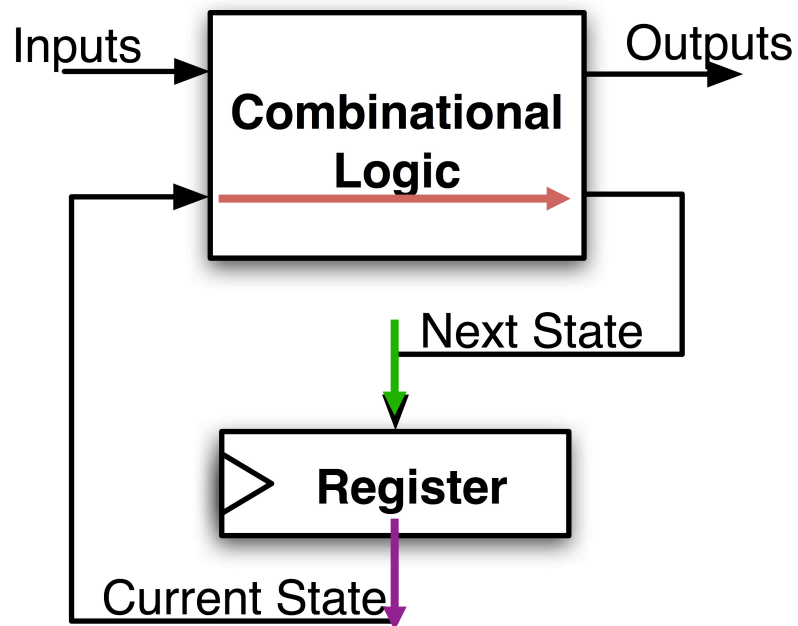
- Want to take a portrait – timing right before and after taking picture
- *Set up time* – don't move since about to take picture (open camera shutter)
- *Hold time* – need to hold still after shutter opens until camera shutter closes
- *Time click to data* – time from open shutter until can see image on output (viewscreen)

Hardware Timing Terms

- **Setup Time:** when the input must be stable *before* the edge of the CLK
- **Hold Time:** when the input must be stable *after* the edge of the CLK
- **“CLK-to-Q” Delay:** how long it takes the output to change, measured from the edge of the CLK

Maximum Clock Frequency

- What is the maximum frequency of this circuit?



Hint:
Frequency = $1/\text{Period}$

$$\text{Max Delay} = \text{Setup Time} + \text{CLK-to-Q Delay} + \text{CL Delay}$$

And in Conclusion, ...

- Multiple Hardware Representations
 - Analog voltages quantized to represent logic 0 and logic 1
 - Transistor switches form gates: AND, OR, NOT, NAND, NOR
 - Truth table mapped to gates for combinational logic design
 - Boolean algebra for gate minimization
- State Machines
 - Finite State Machines: made from *Stateless* combinational logic and *Stateful* “Memory” Logic (aka Registers)
 - Clocks synchronize D-FF change (Setup and Hold times important!)