CMOS Design, Gates & PLAs

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema
Andrew Rigas
Anisha Bhatia
Audrey Yang
Craig Lee
Daniel Duan
David LuoZhang
Eddy Yang
Ernest Ng
Heyi Liu
Janavi Chadha
Jason Hom
Katherine Wang
Kyrie Dowling
Mohamed Abaker
Noam Elul
Patricia Agnes
Patrick Kehinde Jr.
Ria Sharma
Sarah Luthra
Sofia Mouchtaris

How are you feeling about CMOS?

Logistics

- HW01 bits.c: **This Friday** 9/16 @ 11:59 pm
  - Will require VM setup
  - Has you "program" in C
  - Should have everything you need
  - Terminal & starting demo in Recitations this week

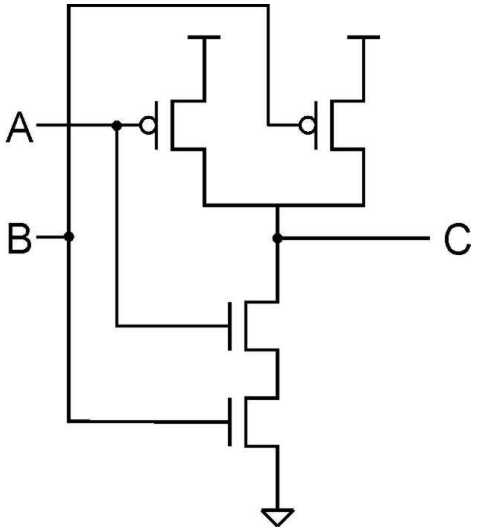
- HW02 Combinational Logic: to be released this week

Lecture Outline

- **CMOS Circuit Design**
- Gates & PLAs
- Transistors in the Real World

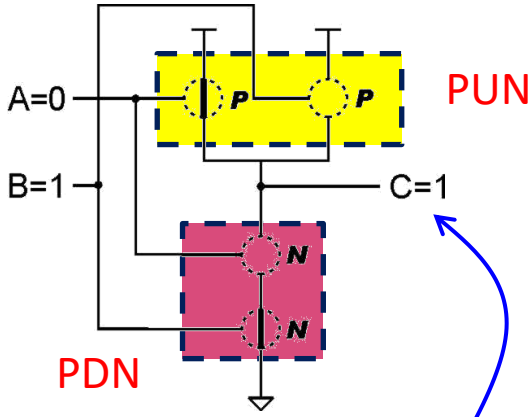
The NAND (NOT-AND) Circuit

Note: parallel structure on top, series on bottom



NAND Transistor Level

Sample Input: A=0, B=1



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NAND Gate Truth Table

Rules & Suggestions For Design

- Rules:
  - You cannot have pMOS transistors in the PDN
  - You cannot have nMOS in the PUN
  - Exactly one of PDN/PUN must be “on” at a time
    - Cannot have neither connected to output
    - Cannot have both connected to output
  - Every transistor in the PDN must have a complimentary transistor in the PUN
    - When any transistor is PDN is ON, its compliment transistor is OFF

- Suggestions
  - Start with the PDN and then do the PUN (most find it easier)
  - Simplify logic before you design any part of the circuit

NOR Circuit Design Walkthrough

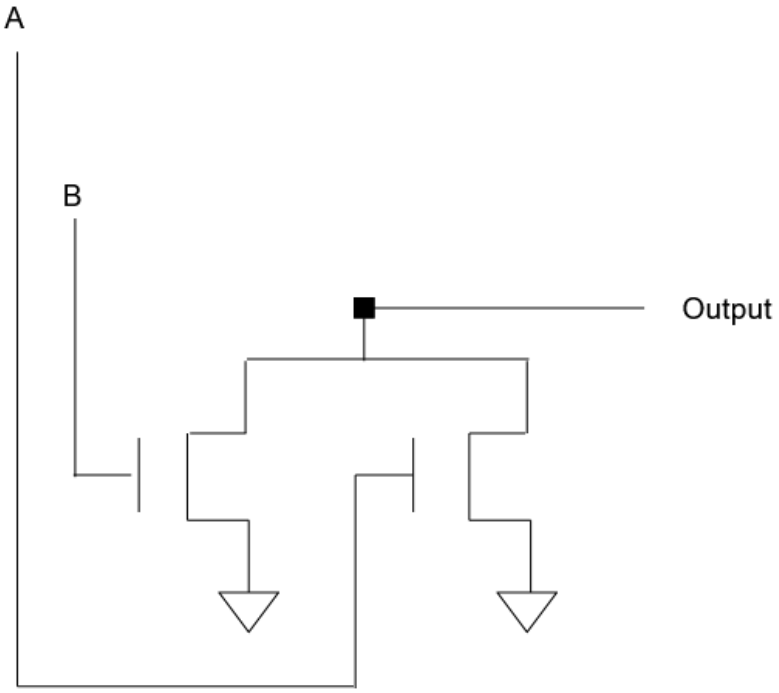
- First, start with the Boolean Expression
  - NOR is NOT-OR, which is
    ~(A | B) <- statement for when output is 1 (True)

- Since I like to start with PDN, find the cases when output is FALSE (output connected to GND), then simplify
  - This can be done by negating the original expression
  - ~~(A | B) <- statement for when output is 0 (False)
  - (A | B) <- simplified by identity property

NOR Circuit Design Walkthrough

- With the expression for the PDN (when output is 0), translate it to a circuit diagram for the PDN
  - OR's become transistors in parallel
  - AND's become transistors in series

(A | B) becomes →



NOR Circuit PUN (Strategy 1)

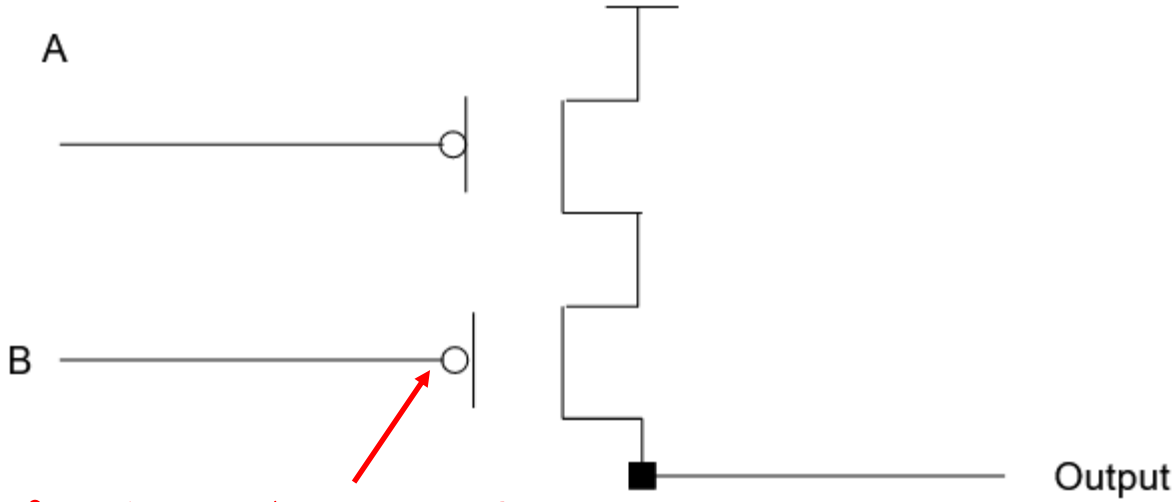
- With the simplified expression for PDN, negate it to get the expression for PUN
  - (A | B) becomes ~(A | B)
  - Simplify
    - ~(A | B)
    - (~A & ~B) // by De Morgan's Law

In this example, initial PUN expression is the same as the original expression. This is not always the case

- From here, we can convert (~A & ~B) into a PUN

NOR Circuit PUN (Strategy 1)

- From here, we can convert (~A & ~B) into a PUN
  - OR's become transistors in parallel
  - AND's become transistors in series
  - pMOS transistors have an implicit "NOT"
    - pMOS only turns on when input is low (0)

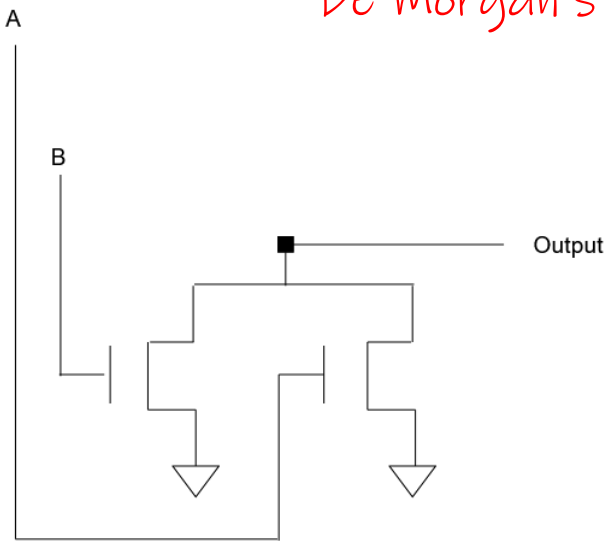


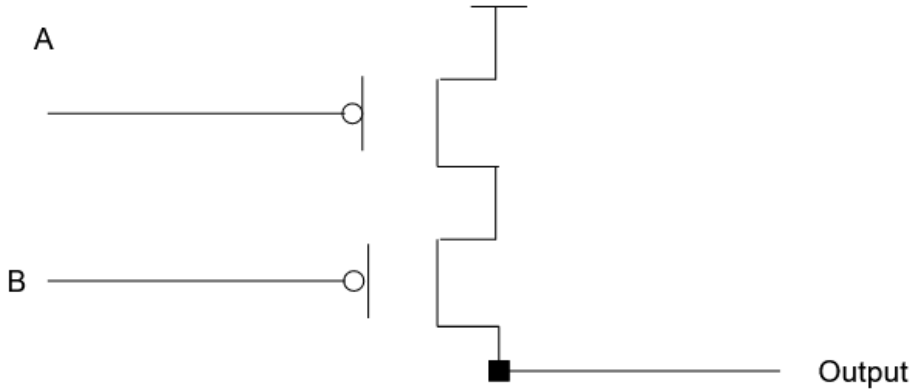
The reason for the circle in pMOS:
Circle represents a "Not" or negation

NOR Circuit PUN (Strategy 2)

- Alternative way to get the PUN from the PDN
  - Series relations in PDN become Parallel relations in PUN
  - Parallel relations in PDN become Series relations in PUN
  - nMOS transistors become pMOS transistors

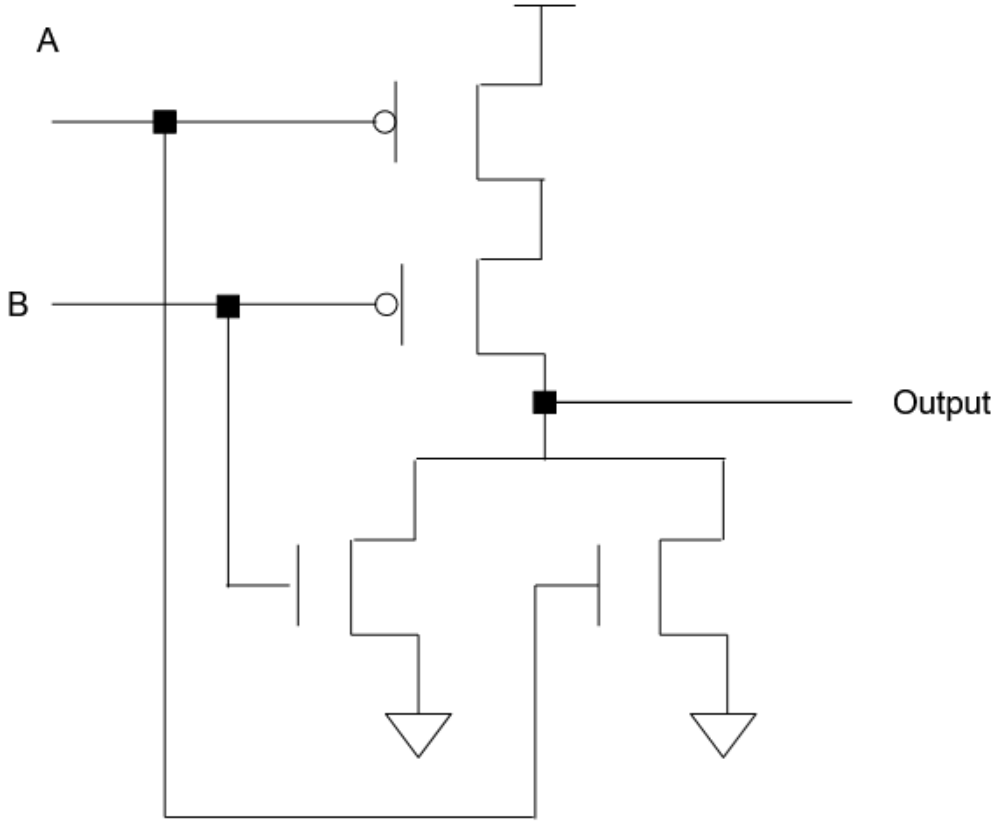
This is just a "short-cut" for applying De Morgan's to the PDN





Completed NOR Circuit

- Finally, all you have to do is combine the PDN and PUN
  - To be safe, check your work and create a truth table



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Complex Practice

- Let's Practice, create the CMOS circuit that takes in 3 inputs (A, B, C) and has a high output (1) if C is 0 and at least one of A or B is 0.

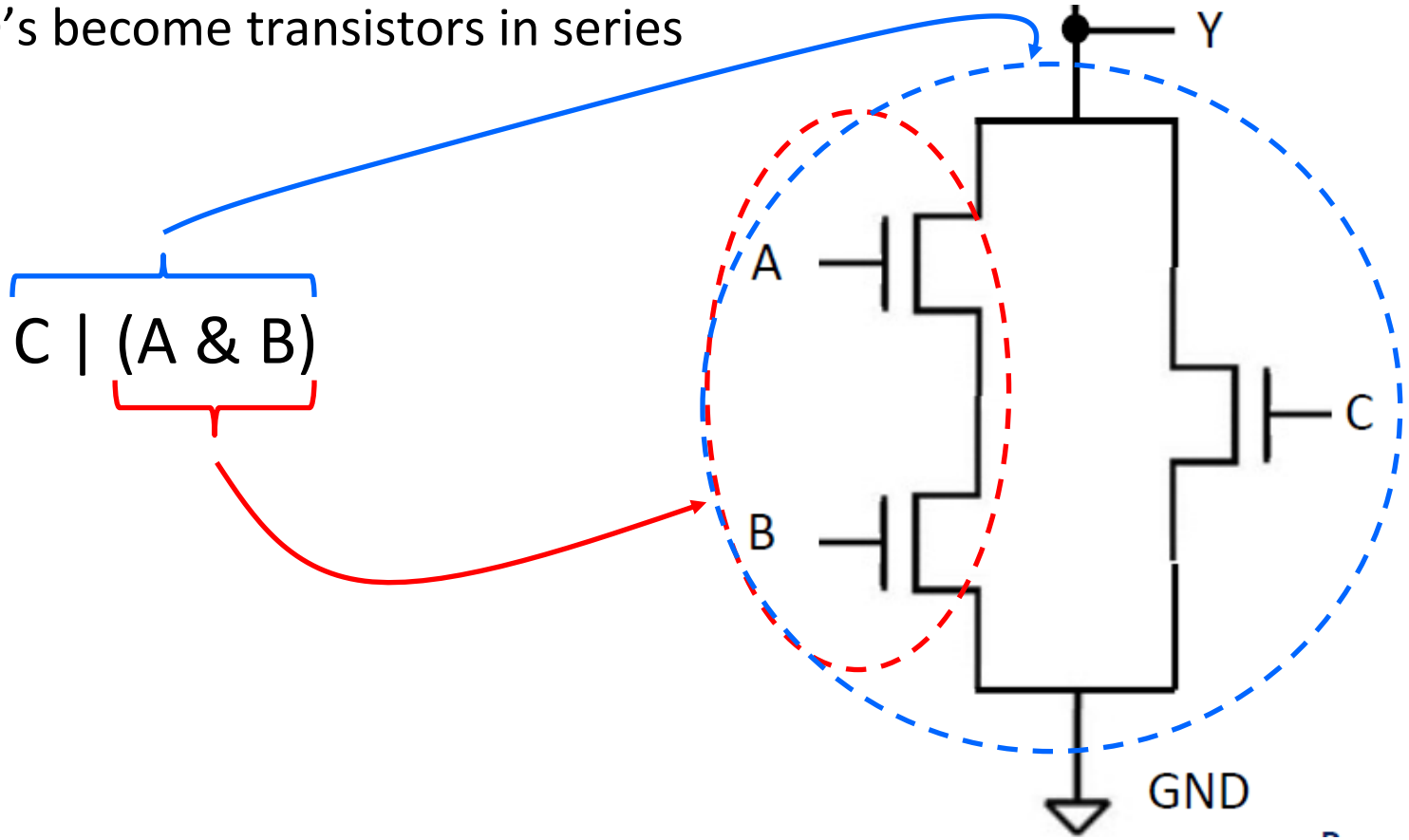
- First, what is the expression?
  - ~C & (~A | ~B)

Complex Practice

- Second, what is the expression for the PDN?
  - ~C & (~A | ~B) // original overall expression
  - ~(~C & (~A | ~B)) // Negate for PDN expression
  - ~~C | ~(~A | ~B) // by De Morgan's
  - C | ~(~A | ~B) // By identity property
  - C | (~~A & ~~B) // by De Morgan's
  - C | (A & B) // by identity property

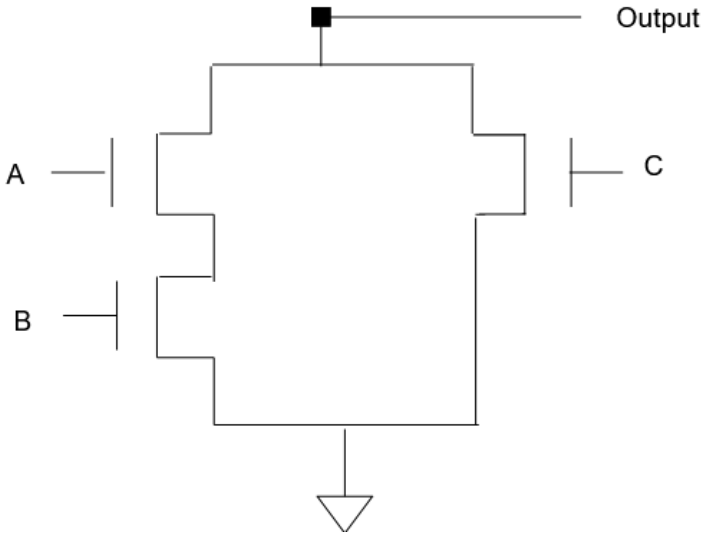
Complex Practice

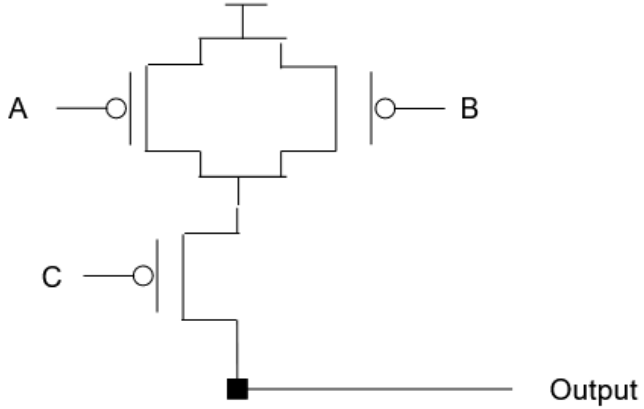
- Third, convert the PDN expression into a circuit
  - OR's become transistors in parallel
  - AND's become transistors in series



Complex Practice

- Fourth get the PUN from the PDN (I'll use the short-cut)
  - Series relations in PDN become Parallel relations in PUN
  - Parallel relations in PDN become Series relations in PUN
  - nMOS transistors become pMOS transistors

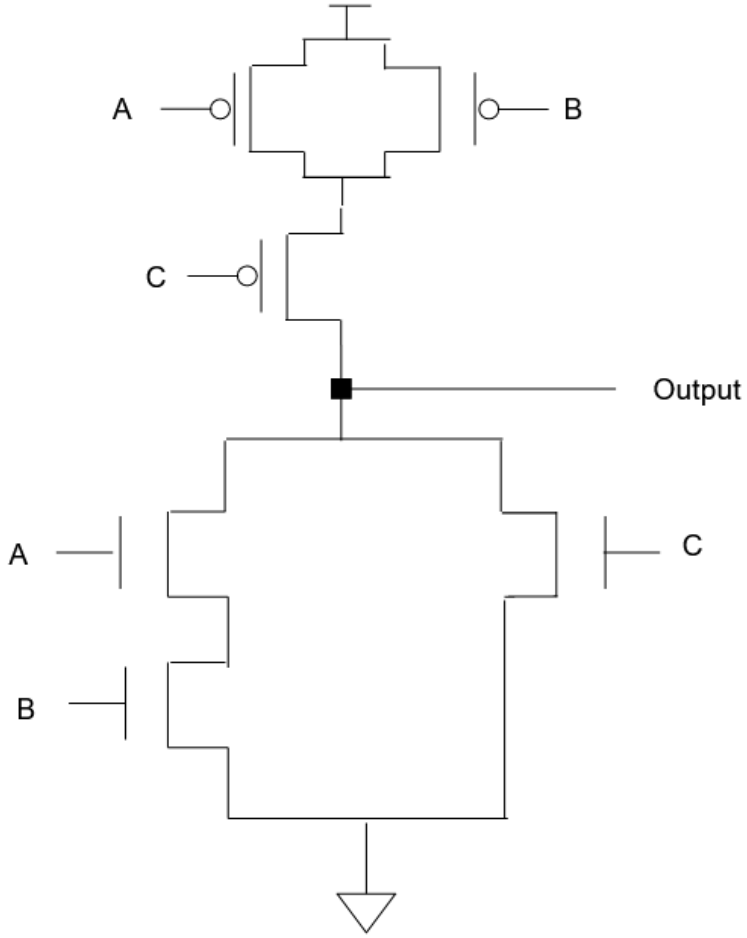




Complex Practice

- Last step:
  combine PDN and PUN

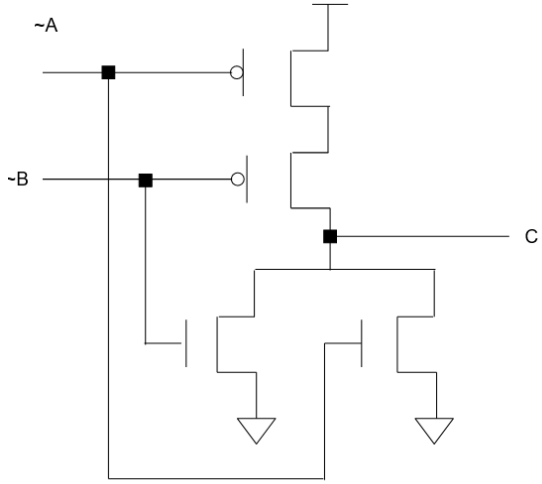
- Suggested:
  Check your work



Closing Details on CMOS

- Sometimes, there will be 3 or more inputs (A, B, C)

- Sometimes you will also have the complements of the inputs (~A, ~B, ~C) that you can use as inputs in a circuit
  - Example AND circuit:



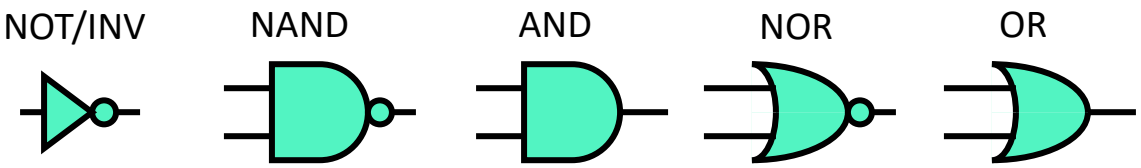
- Highly suggest you simplify before creating circuits
  - More practice available in recitation

Lecture Outline

- CMOS Circuit Design
- **Gates & PLAs**
- Transistors in the Real World

Logic Gates

- Basic Logic Gates



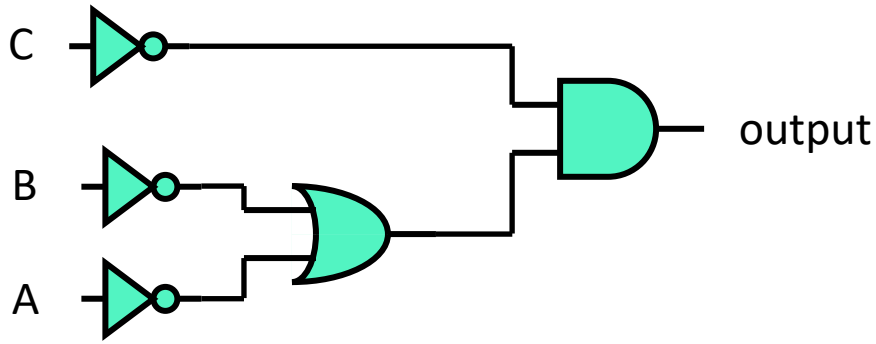
- Moving an Abstraction layer "up"
  - + Abstracting away messy details of CMOS Circuits!
  - + Technology independent at this level, doesn't have to be CMOS
  - - Designing only at this level can be less efficient than CMOS

- Gates of more complex operations exist (e.g. an XOR gate)

- Gates can be combined to make more complex functions

Logic to Gates Practice

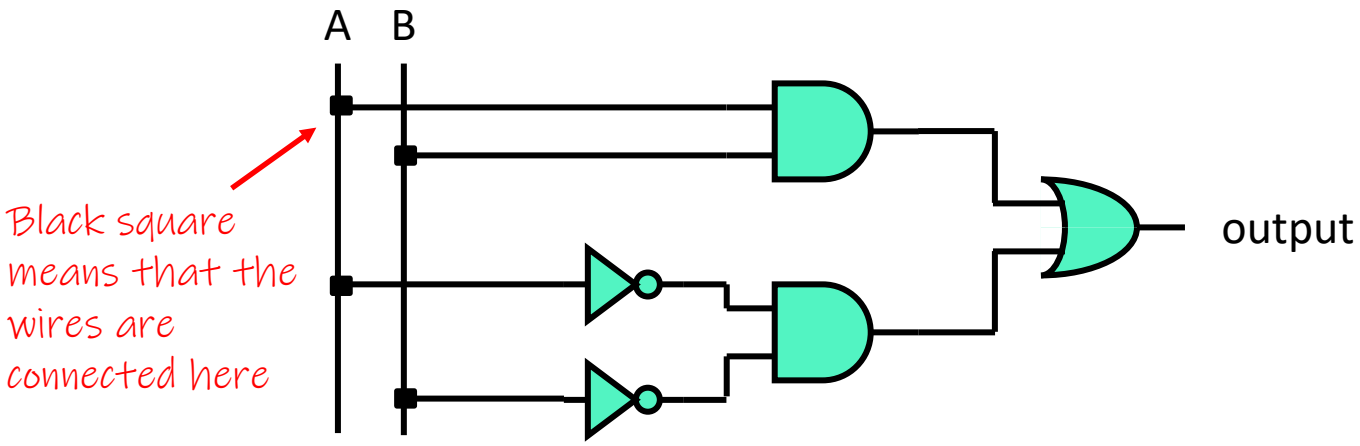
- Let's Practice, create a Gate-level circuit that takes in 3 inputs (A, B, C) and has a high output (1) if C is 0 and at least one of A or B is 0.
- First, what is the expression?
  - ~C & (~A | ~B)
- Second, translate it to gates:



Logic to Gates Practice 2

- Create a gate-level circuit that takes two inputs (A and B) and output 1 if and only if A and B are the same.
  - Use only the basic logic gates I've shown you so far

- First, what is the expression?
  - (A & B) | (~A & ~B)



Poll Everywhere

pollev.com/tqm

- What is the equivalent expression of this gate circuit

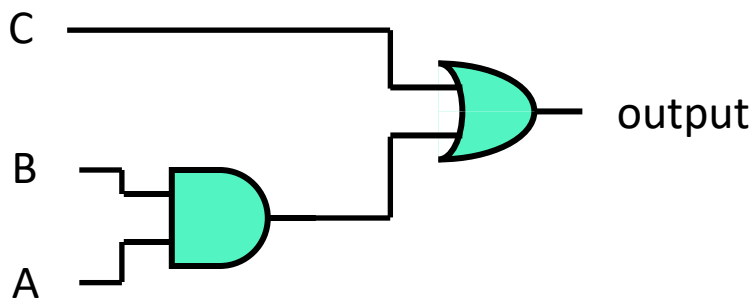
A. (A & B) | C

B. C & (B | A)

C. (C | B) & A

D. (B & C) | A

E. I'm not sure



Poll Everywhere

pollev.com/tqm

- What is the equivalent expression of this gate circuit

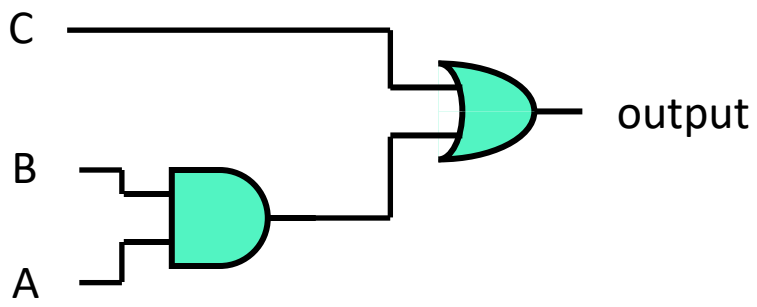
A. (A & B) | C

B. C & (B | A)

C. (C | B) & A

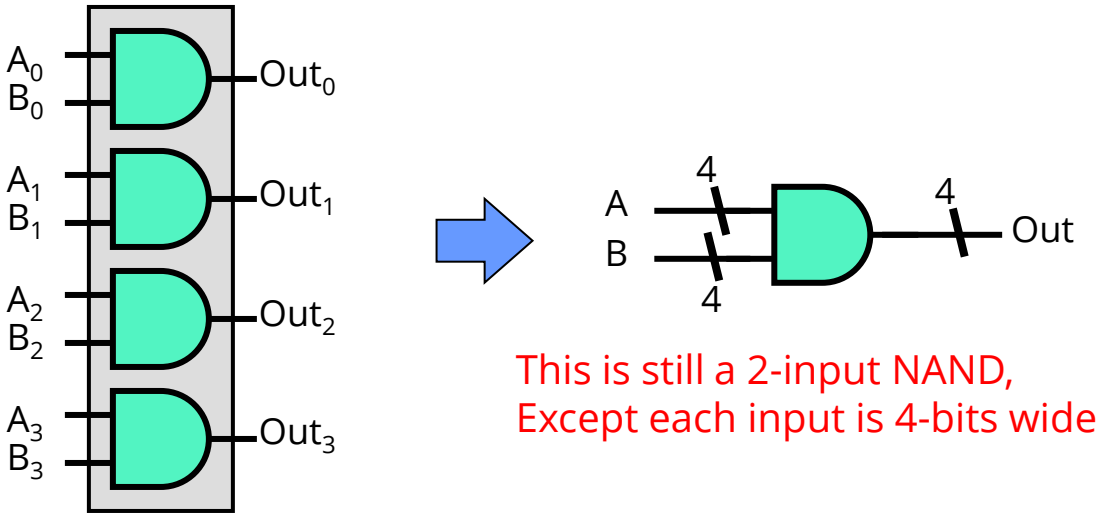
D. (B & C) | A

E. I'm not sure



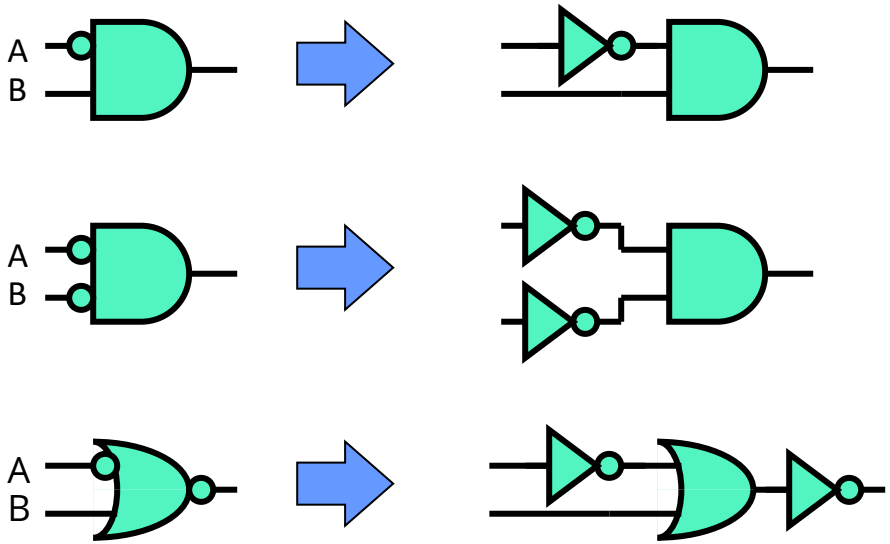
Shortcuts: Multi-Bit Gates

- Use a cross-hatch mark groups of wires:
  - Example, Say we had two 4-bit integers we wanted to AND
  - A_3 is the most-significant bit, A_0 is the least significant bit



Shortcuts: Inverting Signals

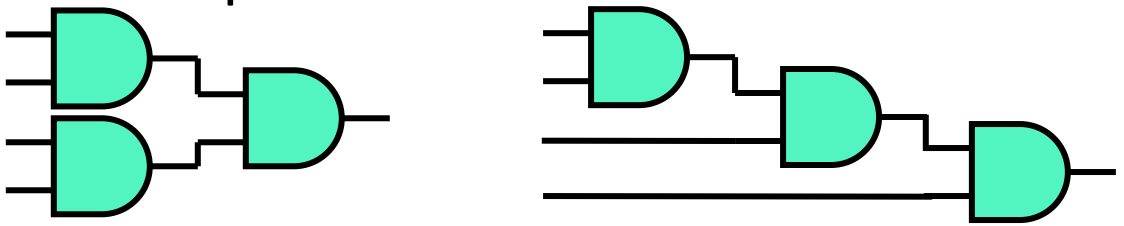
- Instead of always drawing out a NOT gate
  - A ○ represents a negation/inversion of a signal



Boolean algebra rules on gates

- Boolean rules still apply to gate-level circuits

- Associative Example:



- De Morgan's law Example:

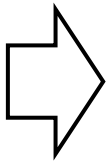
- **NOTE**: There are actual differences between these circuits, but "logically" they are the same. More on this later in lecture.

PLA's

- What if we only had a truth table to create a gate circuit?

- PLA: **P**rogrammable **L**ogic **A**rray
  - A device where we can configure AND, OR and NOT gates to implement a function

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

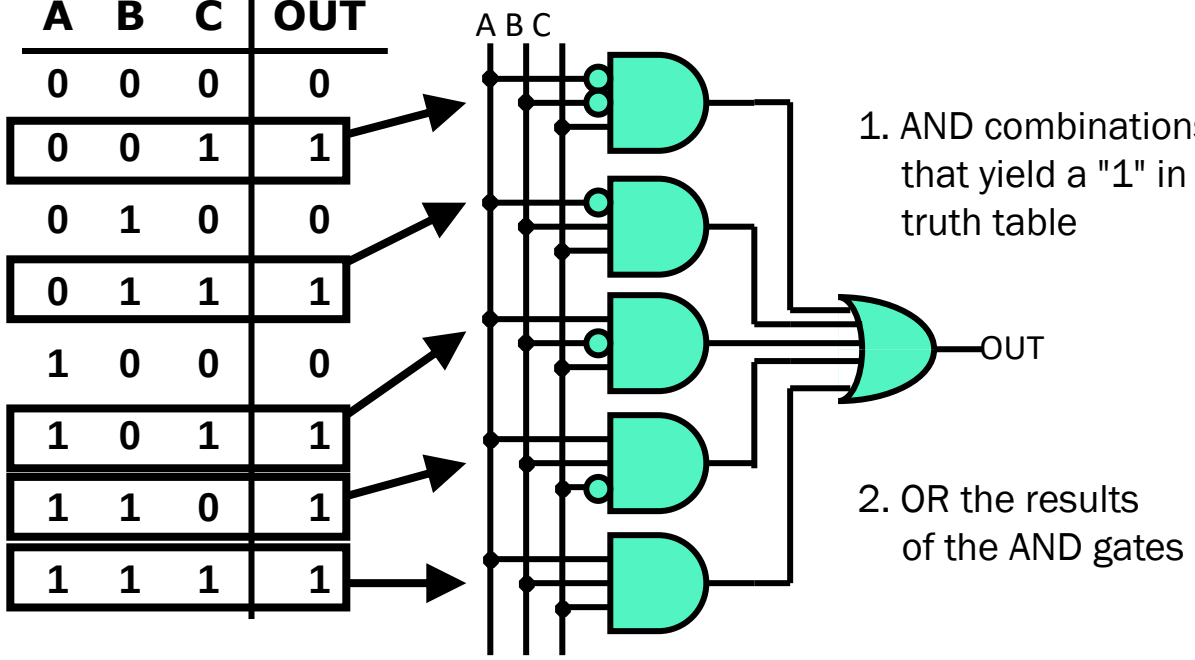


GATES

Implementing a PLA From a Truth Table

- NOT, AND, OR can implement any truth table function

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



In a PLA, this structure is always followed

Notice, 5 rows that cause a “1” in the output…5 AND gates
Notice, 1 output, only 1 OR gate
Notice, negations always happen before the AND gates

PLAs Pros & Cons

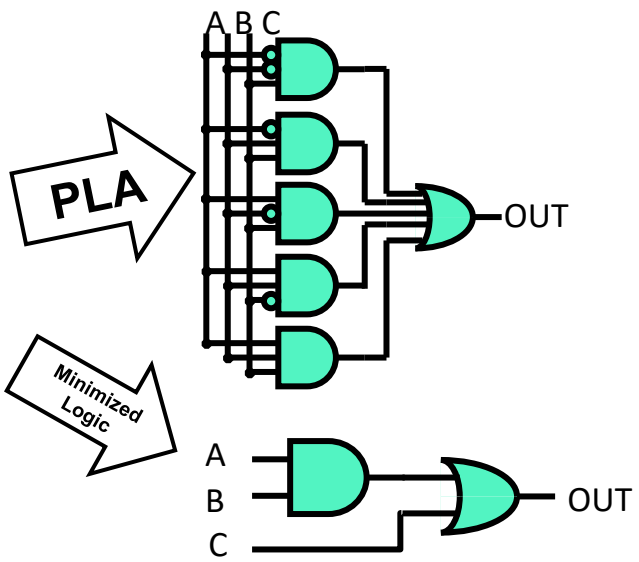
- A PLA can be used to implement ANY logical function
  - Provides you with an incredibly easy tool to use
  - If you can generate a truth table to model desired behavior
    - PLA gives you a way generate the gate level implementation
  - *However, PLAs don't give the most efficient solution*
    - *In terms of "run-time" and transistor cost*

Logic Function

F=(A AND B) OR C

Truth Table

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

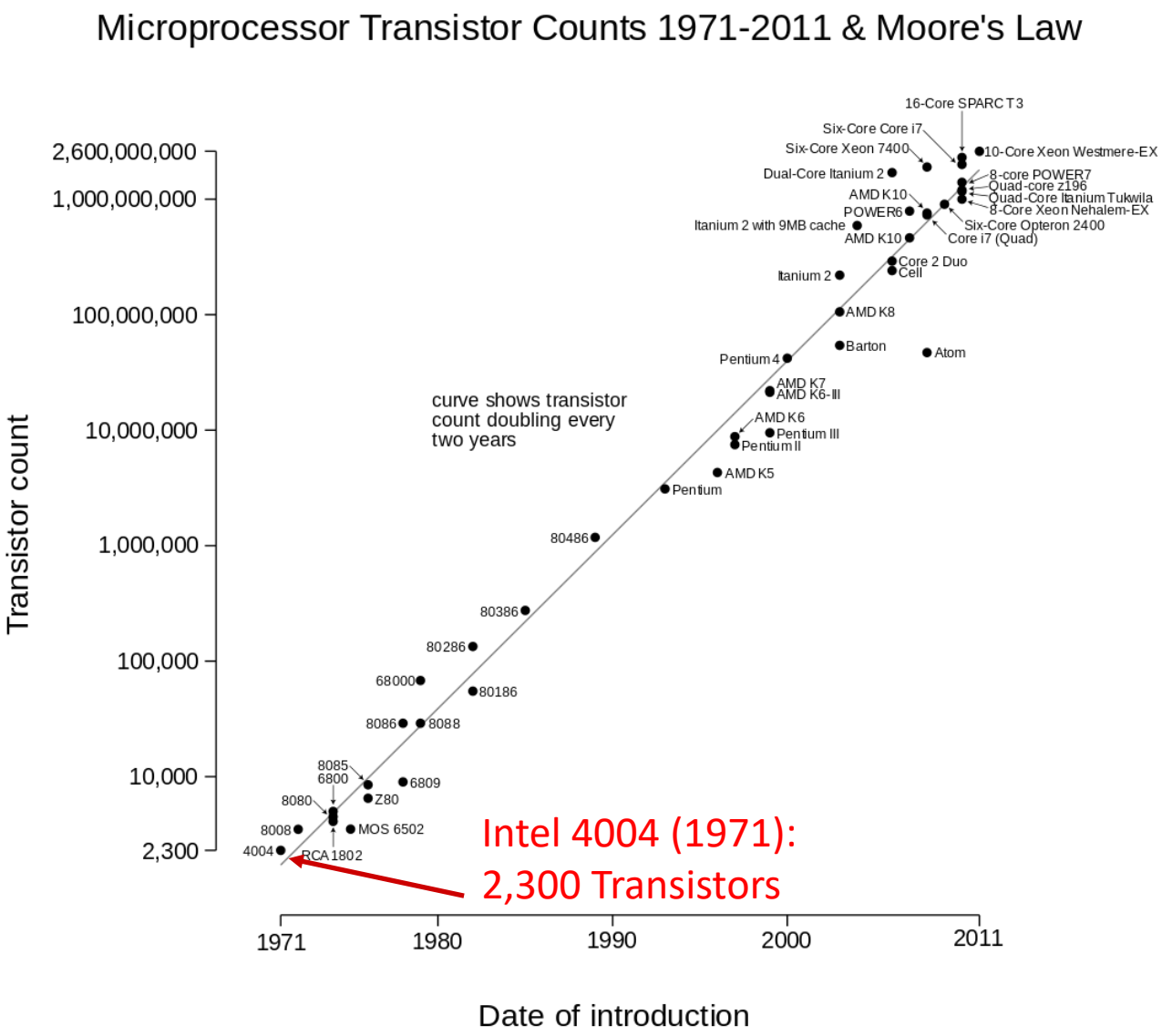


Lecture Outline

- CMOS Circuit Design
- Gates & PLAs
- **Transistors in the Real World**

Moore's Law

- Observation that number of transistors that can be put in an integrated circuit doubles every 18 months-ish



Intel 15-core Xeon IvyBridge-EX (2014): 4.3 billion (not pictured)

Transistors are Small

- The Intel Skylake chip uses transistors that are 60 times smaller than the wavelength of light.
  - Skylake transistor size 14nm.
  - Wavelength of visible light 400-700nm

- There are now more transistors at work in the world (15 quintillion : 15,000,000,000,000,000,000) than there are leaves on all the trees in the world
  - From (The Perfectionists, Simon Winchester)

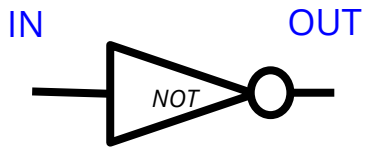
Transistors limitations

- After switching an input, it takes time for current to flow, and output to match changed output

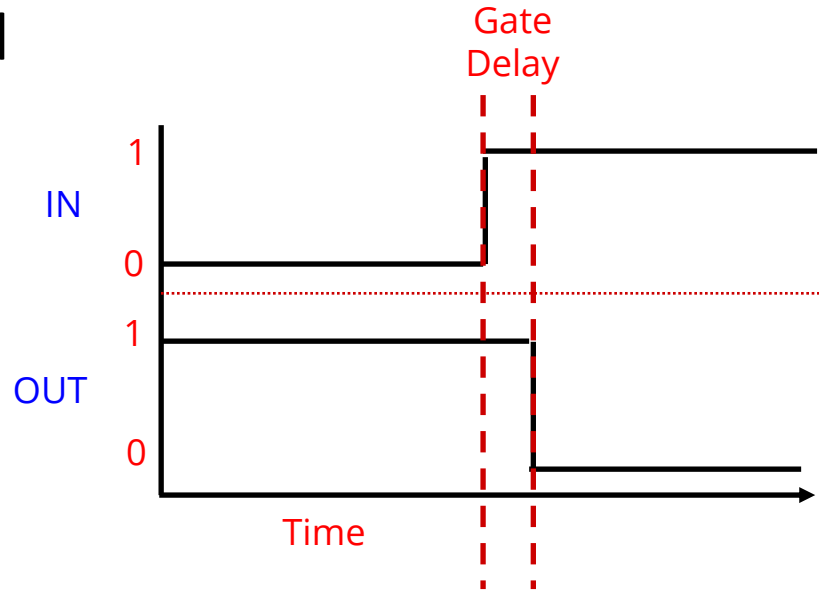
- The faster you switch the circuit, the more current flows, the more heat is generated, the hotter your laptop gets.
  - This has proven to be an important barrier to speeding up CMOS circuitry

Gate Delays

- With any logic circuit there will be a short delay between when an input changes and when the output updates to it
  - This time is referred to as the gate delay
- For modern circuitry, gate delays are on the order of nanoseconds (10^{-9} seconds) or picoseconds (10^{-12} seconds)
- These delays ultimately limit the number of operations you can perform per second



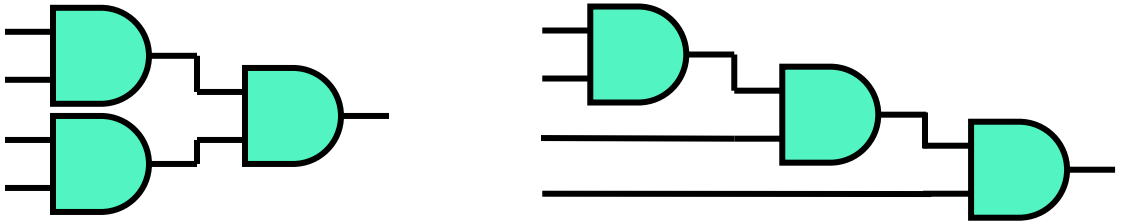
Inverter Logic Gate



Poll Everywhere

pollev.com/tqm

- Consider the following logically equivalent circuits:



Circuit A Circuit B

Which is the better implementation of 4-input AND?

A. Circuit A

B. Circuit B

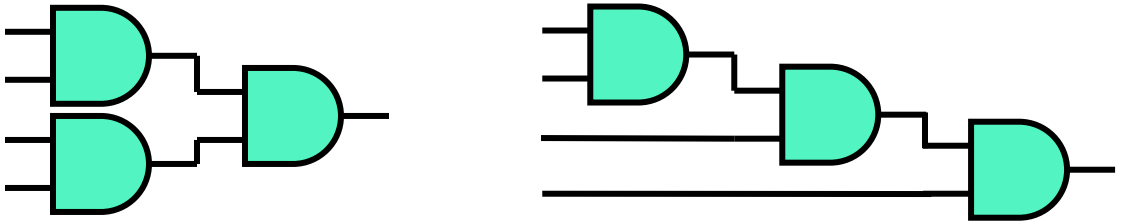
C. They are equally good

D. I'm not sure

Poll Everywhere

pollev.com/tqm

- Consider the following logically equivalent circuits:



Circuit A Circuit B

Which is the better implementation of 4-input AND?

A. Circuit A

B. Circuit B

C. They are equally good

D. I'm not sure

Circuit A is better:
Why? It's faster, 2 "gate delays" instead of 3

- **Gate delays: longest path (in gates) through a circuit**
  - Grossly over-simplified, ignores gate differences, wires
  - Good enough for our purposes

Abstraction

- Moving from CMOS to Gates is going to a higher abstraction level.

- Transistor Level:
  - More control over how many transistors there are and how they are oriented (orientation could affect how much delay there is)
  - More hardware details that must be worried about (PDN vs PUN, pMOS vs nMOS, etc)

- Gate level:
  - Abstract a way the hardware details, focus only on logic.
  - Multiple gates -> longer delay

Multiple Approaches to things

- Lots of ways to implement things. Implementation choice depends on many factors:
  - Ease of Implementation (Readability, dev time, …)
  - Kinds of gates available
  - Number of transistors available
  - Energy Consumption
  - Circuit delay
  - …

Lecture Schedule

- This Lecture
  - CMOS
  - Gates
  - PLAs
  - Gate Delays

- Next Lecture:
  - Use Gates & PLAs to start building more complex circuits and components of a computer