University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Two-Level Logic

SOP Form Gives Good Performance

As you know, one can **use a K-map to obtain an SOP form**.

If one chooses

- a minimal number of loops of maximal size
- **the resulting SOP form has optimal area***

But what about speed?

The speed of an SOP form is typically optimal.

*See caveats in slides on K-maps.

The Best Case is One Gate Delay*

Recall our delay heuristic:
the number of gate delays from any input.

Let's assume that complemented literals are available with no delay.

What can we express with one gate delay in CMOS?

Only NAND and NOR
(NOT is a 1-input NAND/NOR).

*Ignoring the functions 0 and 1 and functions consisting of a single literal, all of which have zero gate delays.

K-Maps Can Identify Single-Gate Functions

A single NAND is an SOP expression.*

So is a single NOR.

An expression using a single gate is also optimal by our area heuristic.

So **if a function can be built with a single gate, the K-map will give us that expression**.

*And a POS expression.

Is Counting AND/OR Gates Realistic?

Most functions cannot be expressed as a single NAND/NOR gate.

So **how fast is an SOP expression**?

Two gate delays.

AND, followed by OR.

But in CMOS, we only have NAND and NOR.

How many gate delays do we get if we only use NAND/NOR?

Let's Introduce Some Algebra

A little Boolean algebra will help us:

DeMorgan's Laws

$$(AB)' = A' + B' \qquad\qquad (A+B)' = A'B'$$

Want a proof? Use a truth table (4 lines each).

They also generalize to more than two inputs.

For example,

$$(ABC)' = A' + B' + C' \qquad (A+B+C)' = A'B'C'$$

DeMorgan's Laws Relate NAND/NOR to AND/OR

What do DeMorgan's Laws mean?

Here's one way to think about them:

- **(AB)' = A' + B'** NAND is the same as OR on the complements of the inputs.
- **(A+B)' = A'B'** NOR is the same as AND on the complements of the inputs.

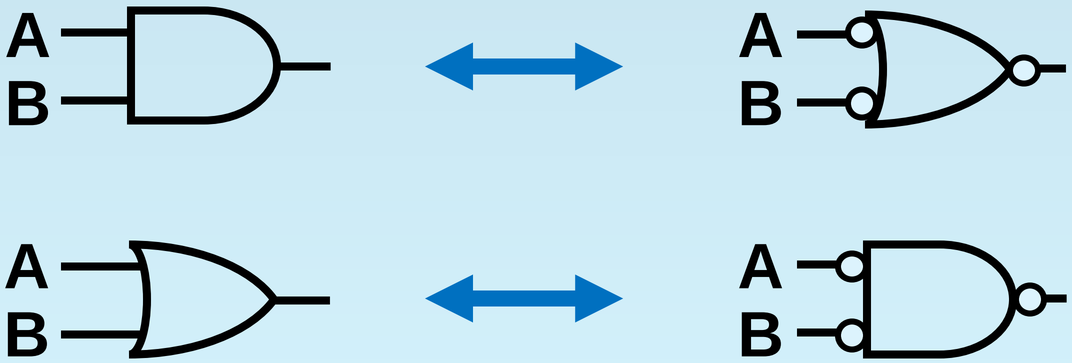
A Graphical Representation Can Be Useful, Too

Let's also think about them graphically.

Complement both sides first, so we have…

AB = (A' + B')' **A+B = (A'B')'**

and now we can draw gates…



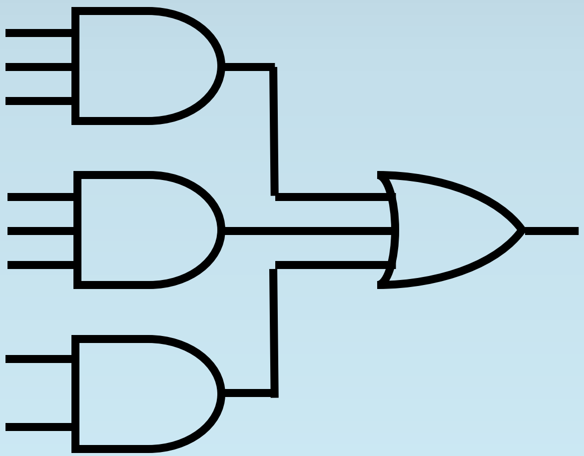
How Do We Draw an SOP Form? AND, then OR

What were we talking about?

Ah, speed of SOP forms.

SOP is AND followed by OR.

Something like this…

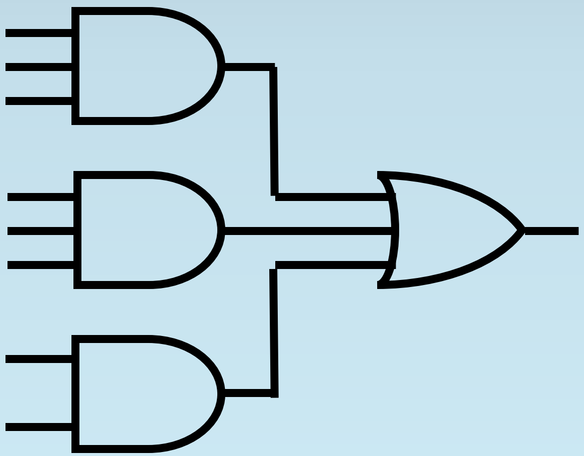


(with some number of AND gates,
 each with some number of inputs)

Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

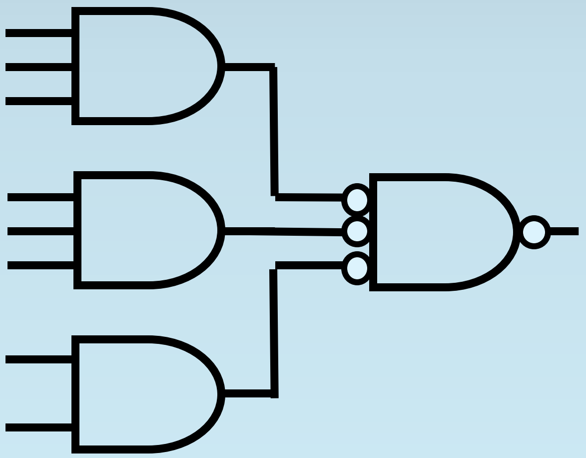


Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

Remember that the **input bubbles mean inverters** (NOT).



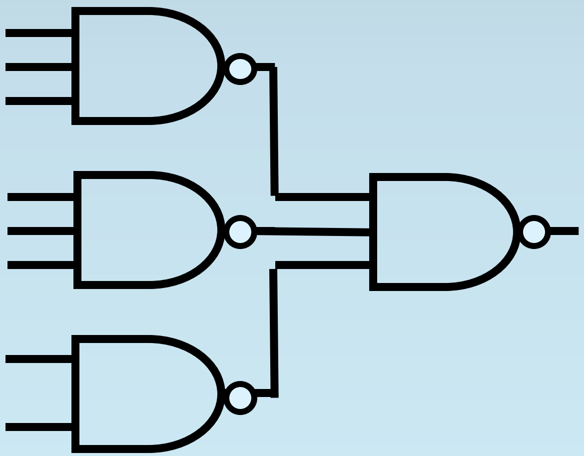
Now **slide them down the wires to the left** until they sit in front of the ANDs.

Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

Remember that the **input bubbles mean inverters** (NOT).



Now **slide them down the wires to the left** until they sit in front of the ANDs.

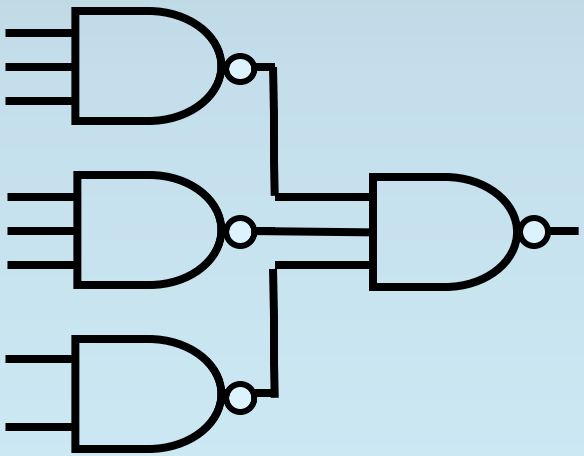
SOP Form Speed is Two Gate Delays

We didn't change the function of the circuit.

But now all of the gates are NAND gates.

So **we can build any SOP function using two levels of NAND**.

And the speed? **Two gate delays.**



SOP and POS Forms Give Us Two-Level Logic

We can use two levels of NANDs to build any SOP expression.

We refer to this approach as **two-level logic**.

For a POS expression
- **one can do exactly the same thing**
- replacing OR followed by AND
- **with NOR followed by NOR**.

So **any POS expression also requires two gate delays** (again, assuming that complemented inputs are free).

Use a K-Map to Find POS Expressions

But **how can we find a POS form?**

Again, **use a K-map**.

1. Given a function **F**, draw a K-map for **F'**.
2. Use K-map to **find an SOP form for F'**.
3. **Complement the result** to find **F**
   - and apply DeMorgan's laws a few times,
   - **complement** of SOP form **is POS form**.

In Practice, Form Loops Around 0s to Find POS

In practice, **just circle 0s instead of 1s**.

Recall that a box in a K-map
- when filled with a 1
- corresponds to a **minterm**.

The same box
- when filled with a 0
- corresponds to a **maxterm**
- an expression that produces exactly one 0 row in its truth table.

Complement Literals When Reading POS Factors

But be careful: the **maxterm** has all variables complemented relative to the **minterm**.

For example,

- a box corresponding to **minterm ABC'** (equal to 1 when **A=1** and **B=1** and **C=0**)
- corresponds to **maxterm A' + B' + C** (equal to 0 when **A=1** and **B=1** and **C=0**)

SOP and POS Forms Give Us Two-Level Logic

To **find a POS form** that has optimal area (among POS forms),
- **follow the same approach** as before,
- but instead of drawing loops around 1s,
- **draw loops around 0s**.

Again, **do not forget to complement the literals relative to their form for implicants!**

(And write each loop as a sum, not as a product.)

Which Form is Better? Solve Both and Compare

Which gives better area, SOP or POS?

That depends on the function.

Solve both ways and compare.

You will have some experience finding POS forms in discussion section.

You can also use the online tool, but the exercises are not as direct as for SOP.

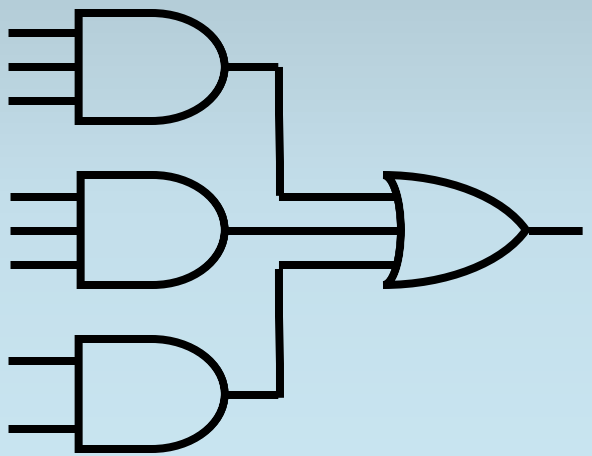
Summary of 2-level Design

Every Boolean function can be expressed as

Minimal SOP (or POS) expression

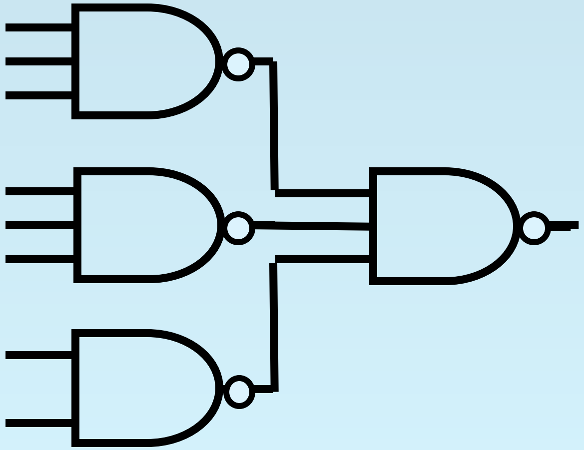
And can be implemented as

AND-to-OR (or OR-to-AND) two-level network



Or as

NAND-to-NAND (or NOR-to-NOR) two-level network



University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Pareto Optimization*

* *

What's Best When We Have Several Metrics?

As engineers, you will rarely have the luxury of a single metric.

How does one choose between metrics?

Imagine the following …

- You are working as an intern
- designing hardware to execute DNNs (deep neural networks, which may be useful in a variety of tasks).

* *

Example: Compare Designs Based On Area and Delay

Building on your ECE120 knowledge, you have two metrics.

- **Area**, which you have normalized from **1 to 100**.
- **Delay**, which you have also normalized from **1 to 100**.

In both metrics, **smaller is better**.

For a design **X**, **A(X) is the area**, and **D(X) is the delay**.

* *

Which Metric is More Important?

Now imagine that you have two designs, **X** and **Y**.

How do you choose between them?

Which is more important, area or delay?

* *

The Answer Depends on How the Design is Used

The answer **depends on the context** in which your design is used

- datacenter
- laptop
- mobile phone
- car or other vehicle
- space probe
- children’s toy

* *

One Option is to Combine Metrics Numerically

How do you make a choice?

One option: **linearize**. Pick some weights

- actually, one weight **W** is enough
- **W** is the **relative importance** of delay compared to area

Then

- for each design **X**
- calculate **M(X) = A(X) + W D(X)**

Choose the design with the smallest M(X).

* *

Relative Importance is Not Easy to Choose in Practice

But how do you pick **W**?

What if you need designs for ALL of those applications?

As an engineer, you may not be in a position to know the right weights!

So **what can you do if you don’t know the relative importance of the metrics**?

For two designs, probably just report both to your manager.

* *

Remember Dilbert? Oxygen is Good.

What if you have created 10,000 designs?

- Not by hand, but by using parameters.
- For example, does your design provide hardware for 8-bit, 16-bit, 32-bit, or 64-bit addition?

Do you report all 10,000 to your manager?

Probably not if you want a job offer.

But **what can you do**?

A Design that is Worse in All Metrics is Pareto-Dominated

Pick two designs **X** and **Y**.

What if **A(X) < A(Y)** AND **D(X) < D(Y)**?

Remember that smaller is better for both area and delay.

In such a case, **do you need to report Y**?

No! **X** is better in both metrics.

We say that **design Y is Pareto-dominated by design X**.

If there were **N** metrics, **Y must be worse than some X in ALL metrics to be Pareto-dominated**.

* *

Eliminate Designs that are Pareto-Dominated by Others

You can use the idea of Pareto domination to eliminate designs.

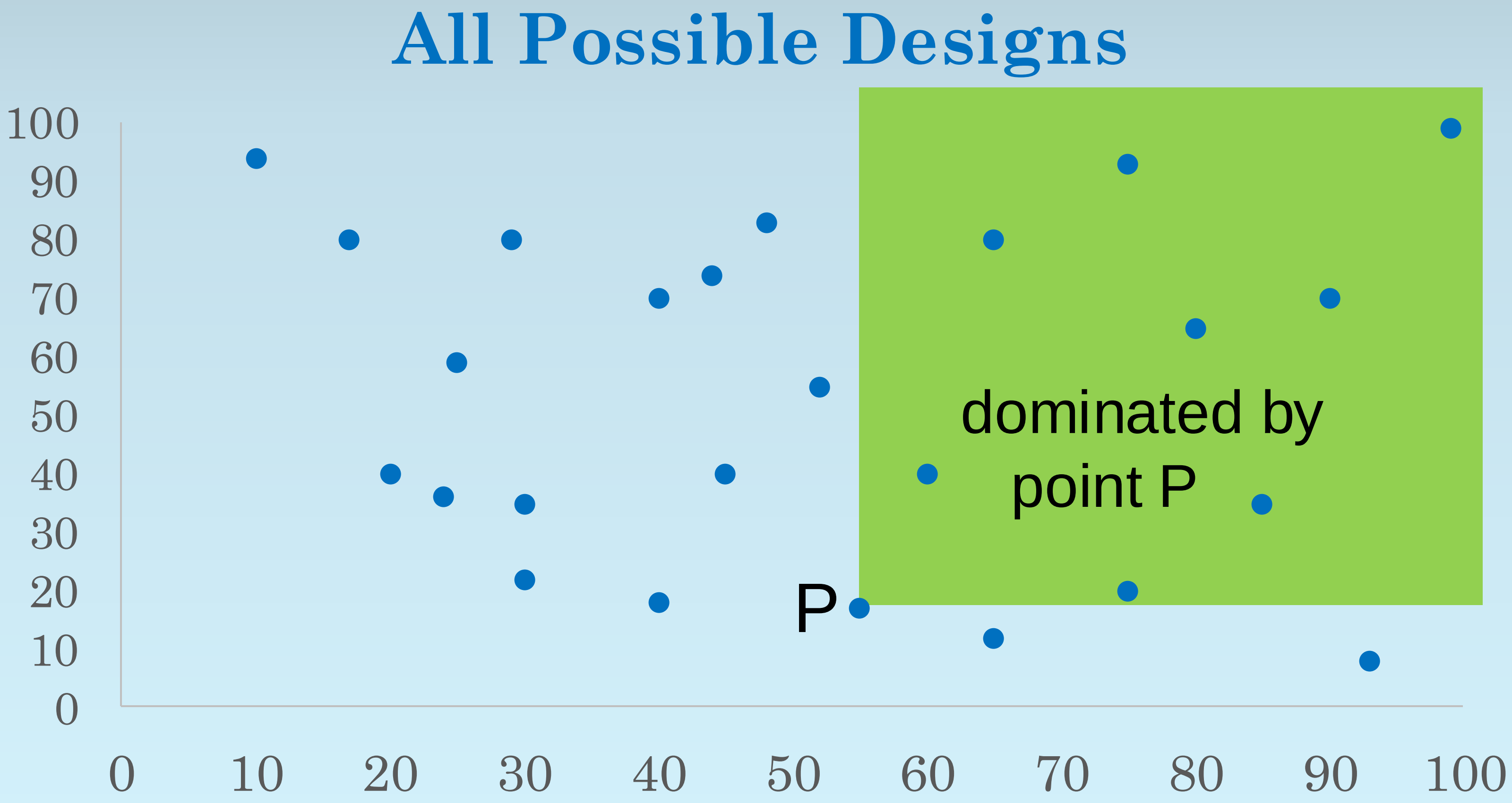
Any design that is Pareto-dominated by any other design **can be discarded**.

Only designs that may be better in some context remain.

The remaining designs form a **Pareto curve** (a Pareto surface for more than two metrics).

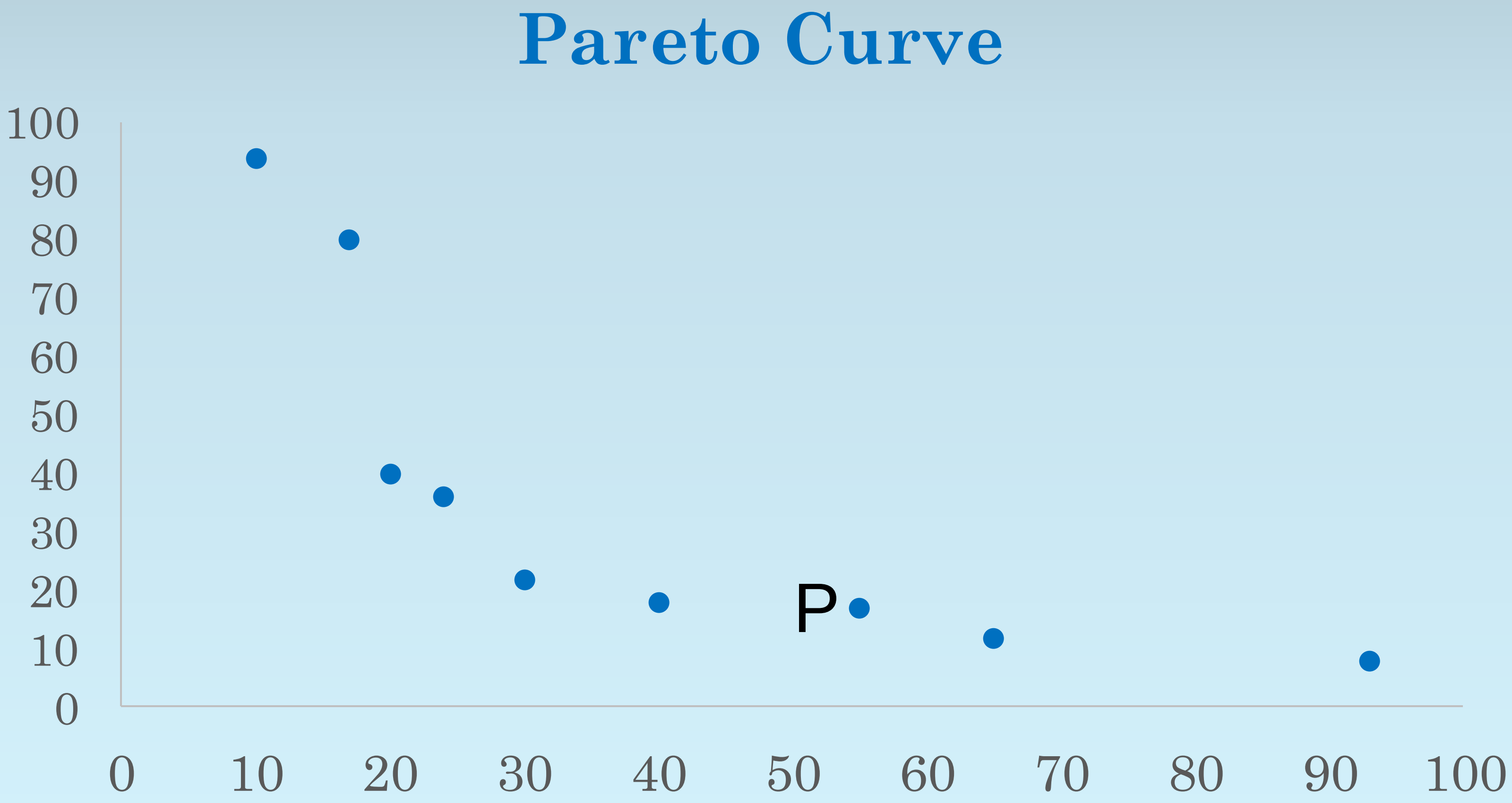
* *

A Graph Illustrates Pareto-Dominance (Small is Good)



* *

A Pareto Curve (after Discarding Dominated Points)



* *

Use of Pareto Curves/Surfaces is Common

In many hardware design environments, engineers run **design-space exploration** tasks (on computers, of course!):

- Given a set of parameters for a design
- Generate hardware for each possible combination of parameters
- Then use Pareto dominance to trim the results down
- And show the engineer the Pareto surface of area, delay, and power consumption.

* *

Want to Learn More about Optimization?

Take ECE490 some day.

Combines theory and practice:

- optimization algorithms,
- Implementations,
- use of libraries to solve problems.