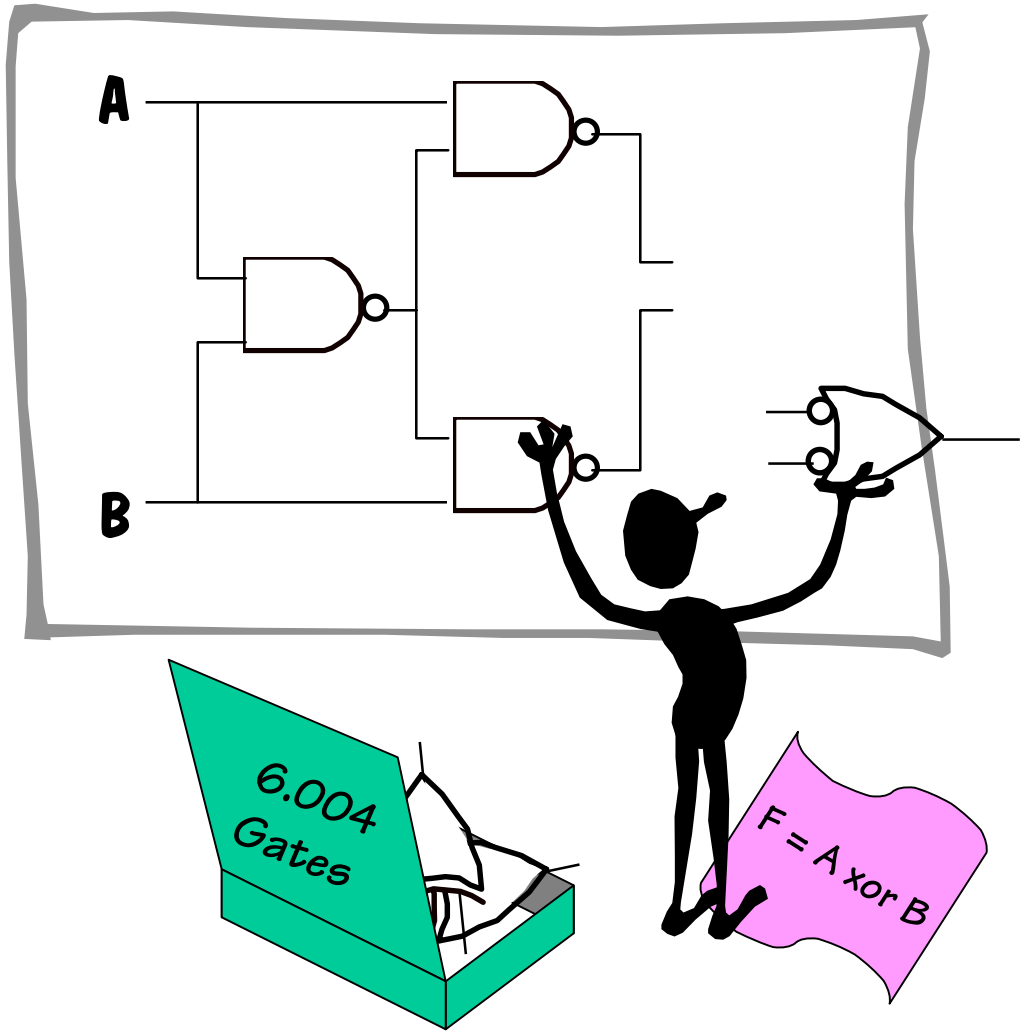
Synthesis of Combinational Logic



Handouts: Lecture Slides, PS3, Lab2

Review: K-map Minimization

1) Copy truth table into K-Map

2) Identify subcubes,

   selecting the largest available subcube at each step, even if it involves some overlap with previous cubes, until all ones are covered.
   (Try: 4x4, 2x4 and 4x2, 1x4 and 4x1, 2x2, 2x1 and 1x2, finally 1x1)

3) Write down the minimal SOP realization

Truth Table

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

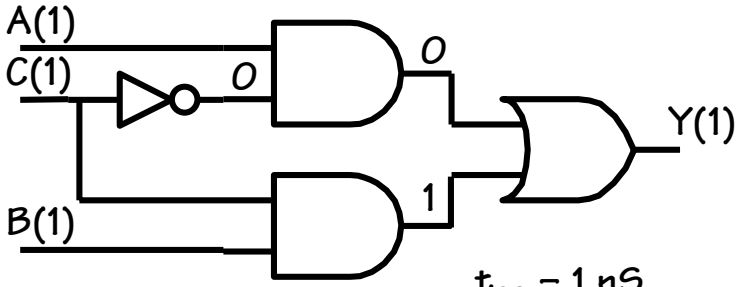
The circled terms are called *implicants*. An implicant not completely contained in another implicant is called a *prime implicant*.

C\BA	00	01	11	10
0	0	1	1	0
1	0	0	1	1

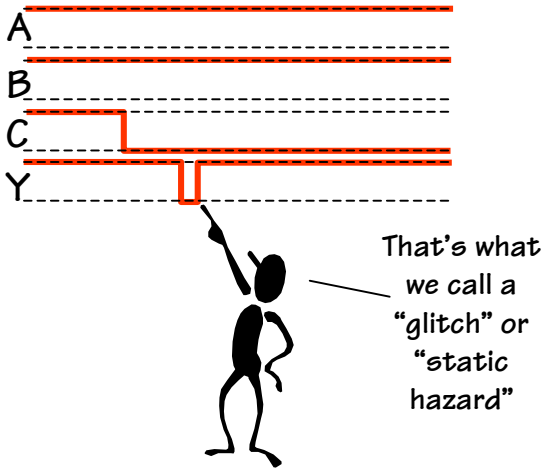
$$Y = \overline{C}A + CB$$

A Case for Non-Minimal SOP

$$Y = \overline{C}A + CB$$



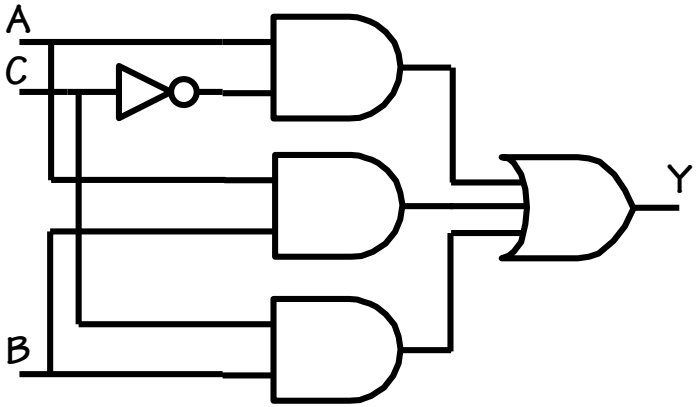
t_{CD} = 1 nS
t_{PD} = 2nS



C\BA	00	01	11	10
0	0	1	1	0
1	0	0	1	1

NOTE: The steady state behavior of these circuits is identical. They differ in their transient behavior.

$$Y = \overline{C}A + CB + AB$$



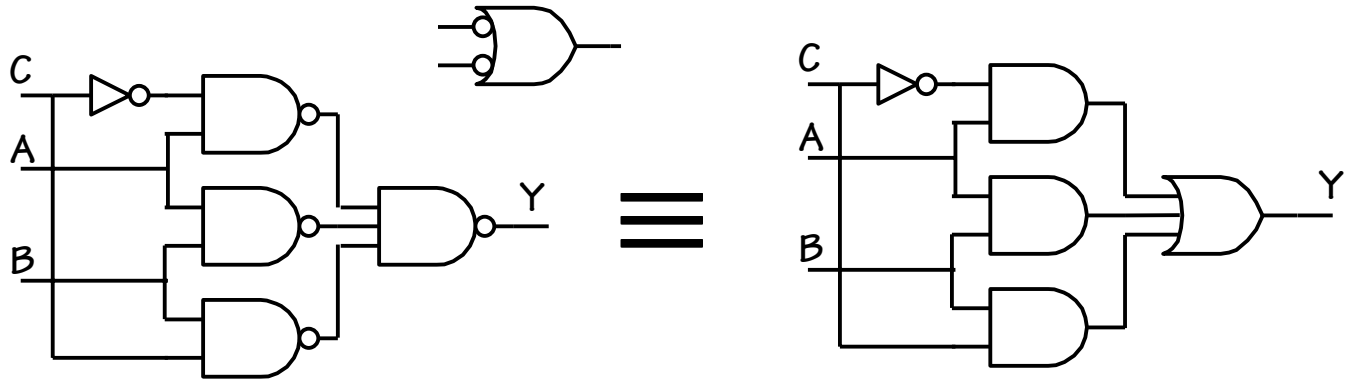
If you include equations for *all* prime implicants, the resulting implementation will be lenient.

Useful Gate Structures

$\overline{AB} = \bar{A} + \bar{B}$

"Pushing Bubbles"

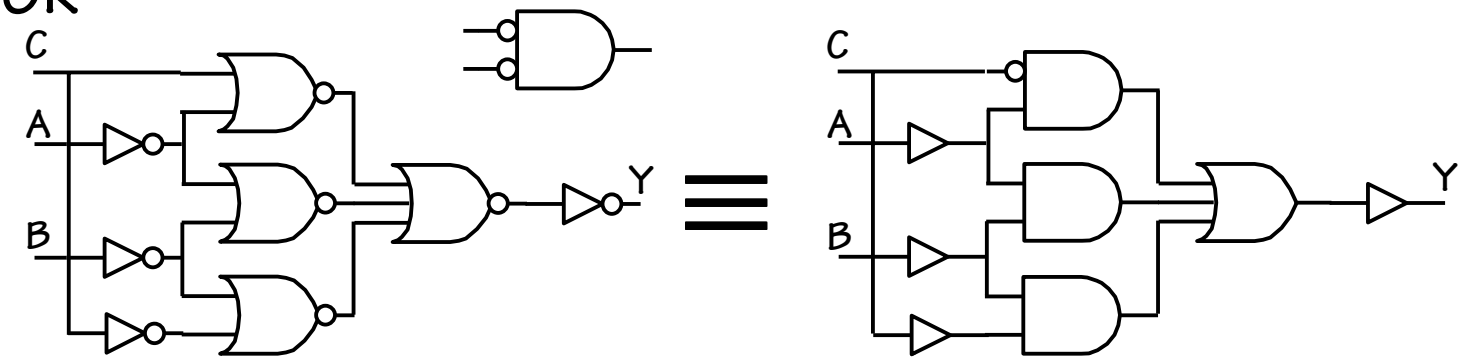
NAND-NAND



$\overline{xyz} = \bar{x} + \bar{y} + \bar{z}$

$\bar{A}\bar{B} = \overline{A+B}$

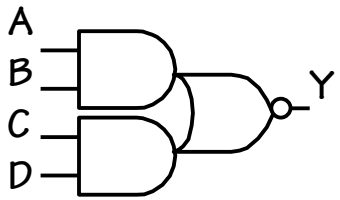
NOR-NOR



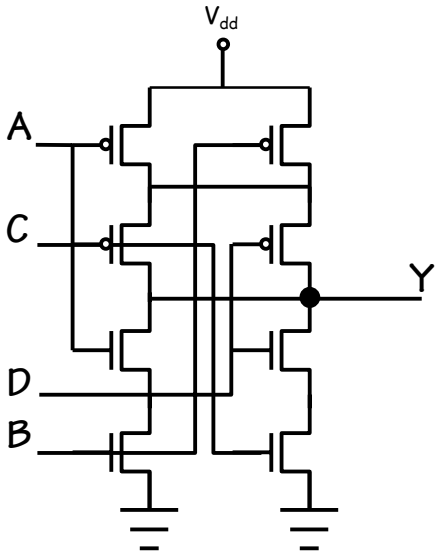
$\overline{x+y} = \overline{xy}$

More Useful Gate Structures

AOI (AND-OR-INVERT)



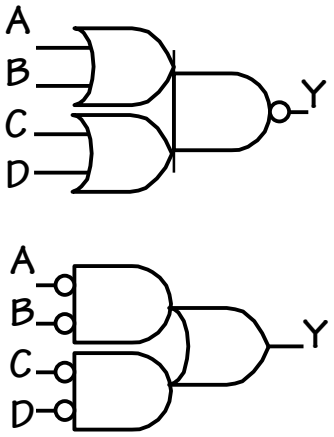
≡



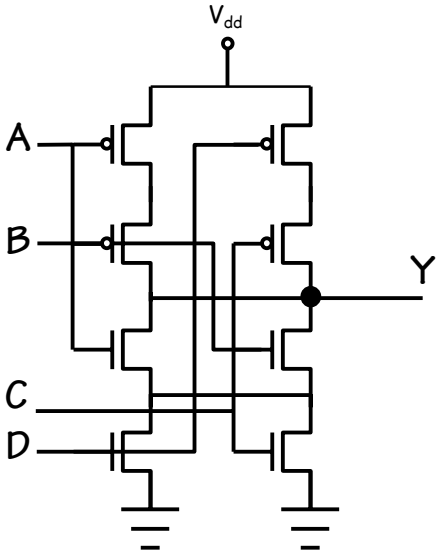
AOI and OAI structures can be realized using a single CMOS gate. However, their function is equivalent to 3 levels of logic.

OAI (OR-AND-INVERT)

An OAI's DeMorgan equivalent is usually easier to think about.

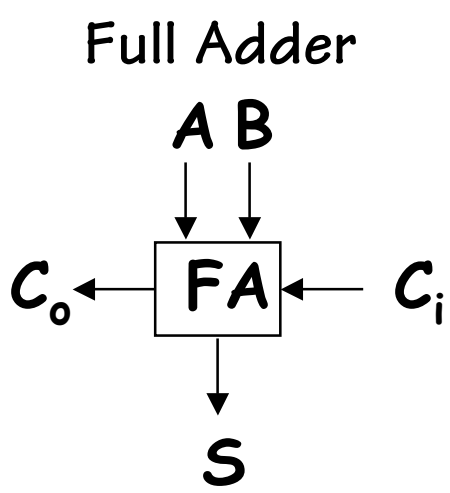


≡



Logic that defies SOP realization

C_i	A	B	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



S

C/AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_O

C/AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

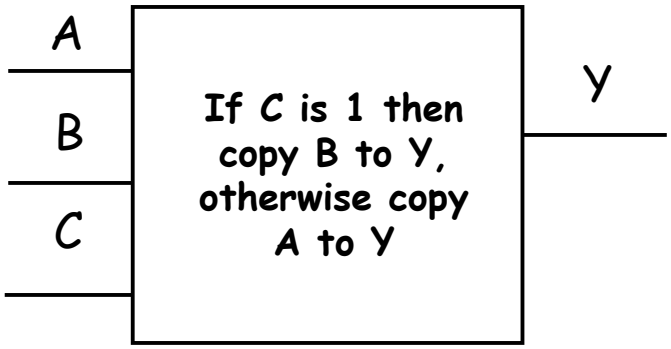
$$C_o = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC$$

Can simplify the carry out easy enough, eg...

$$C_o = BC + AB + AC$$

But, the sum, S, doesn't have a simple sum-of-products implementation even though it can be implemented using only two 2-input gates.

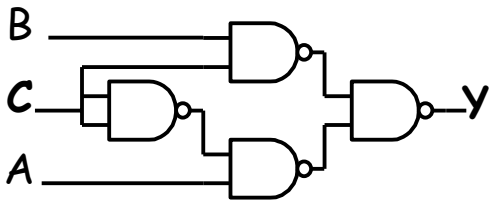
Logic synthesis using MUXes



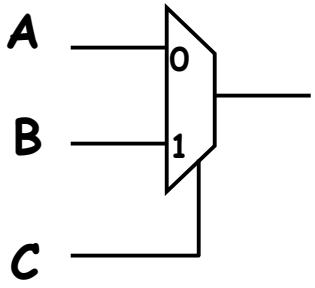
2-input Multiplexer

Truth Table

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

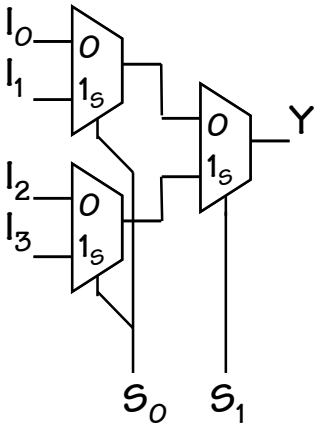


schematic



Gate symbol

A 4-input Mux implemented as a tree

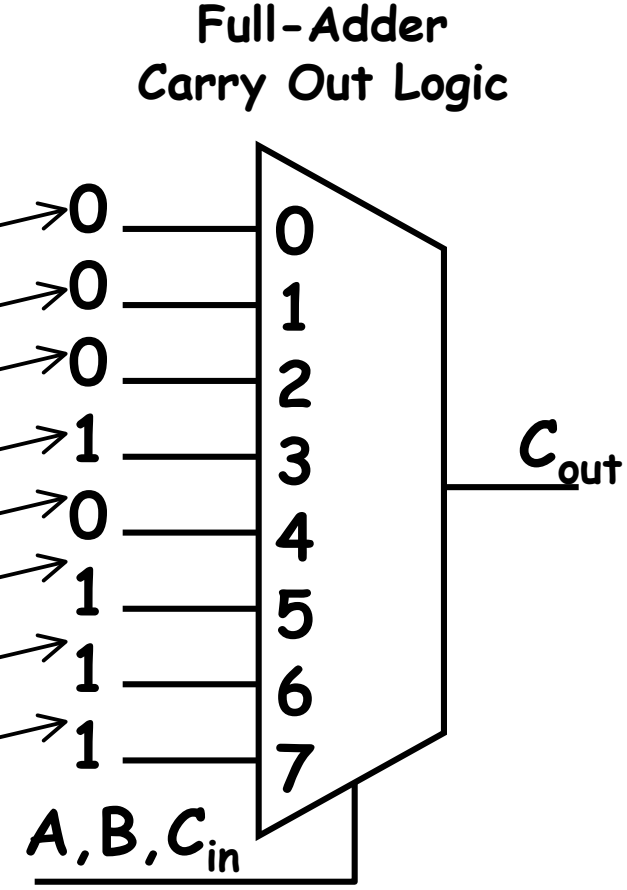


Systematic Implementation of Combinational Logic

Consider implementation of some arbitrary Boolean function, F(A,B)

... using a MULTIPLEXER as the only circuit element:

A	B	C_{in}	C_{out}
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

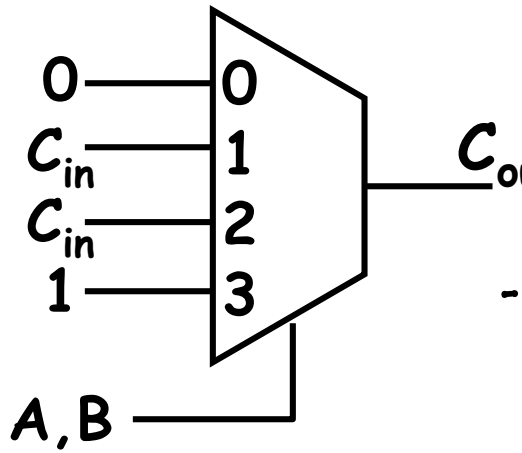


Small Improvements

We can also apply certain optimizations to MUX Logic

Full-Adder Carry Out Logic

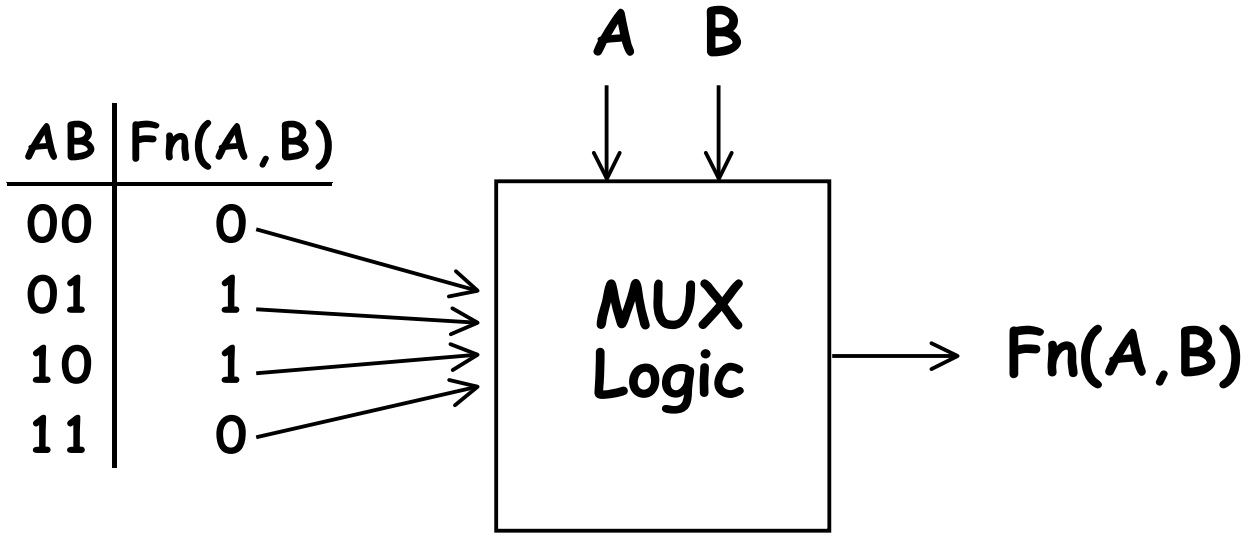
A	B	C_{in}	C_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



- Largely by inspection or exhaustive search

- N-input gate with N-1 input MUX & one inverter

General Table Lookup Synthesis



Generalizing:

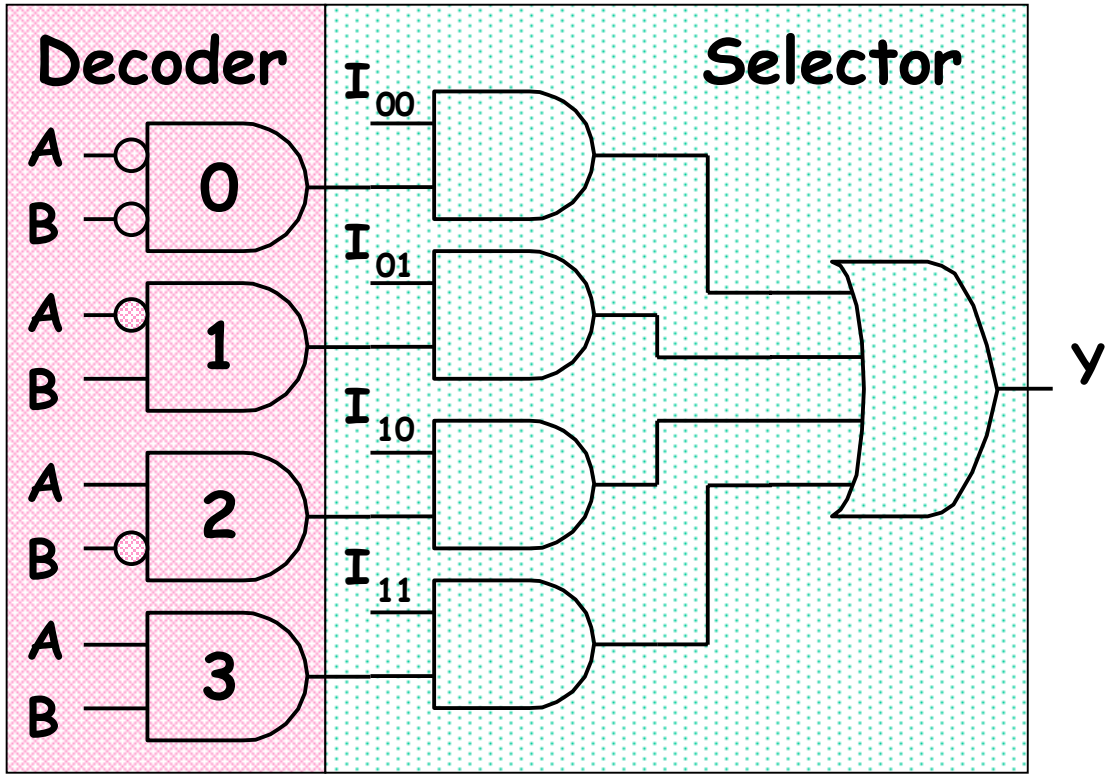
In theory, we can build any 1-output combinational logic block with multiplexers.

For an N-input function we need a 2^N input multiplexer.

BIG Multiplexers? How about 10-input function? 20-input?

A Mux's Guts

A decoder generates all possible product terms for a set of inputs



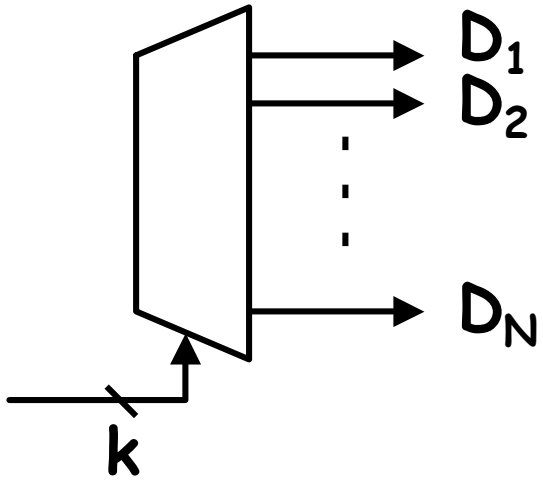
Multiplexers can be partitioned into two sections.

A DECODER that identifies the desired input,and

a SELECTOR that enables that input onto the output.

Hmmm, by sharing the decoder part of the logic MUXs could be adapted to make lookup tables with any number of outputs

A New Combinational Device



DECODER:

k SELECT inputs,

$N = 2^k$ DATA OUTPUTs.

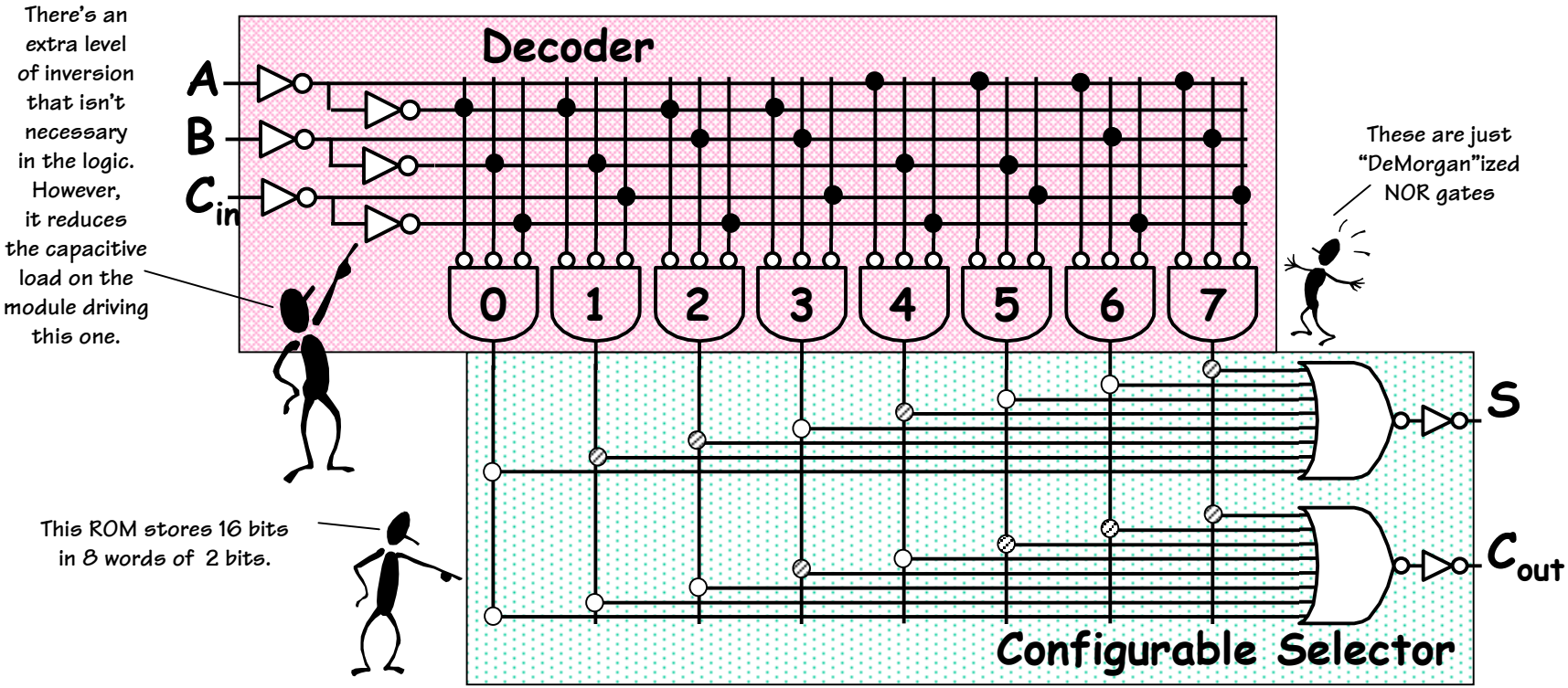
Selected D_j HIGH;
all others LOW.

Have I mentioned that HIGH is a synonym for '1' and LOW means the same as '0'

NOW, we are well on our way to building a general purpose table-lookup device.

We can build a 2-dimensional ARRAY of decoders and selectors as follows ...

Shared Decoding Logic

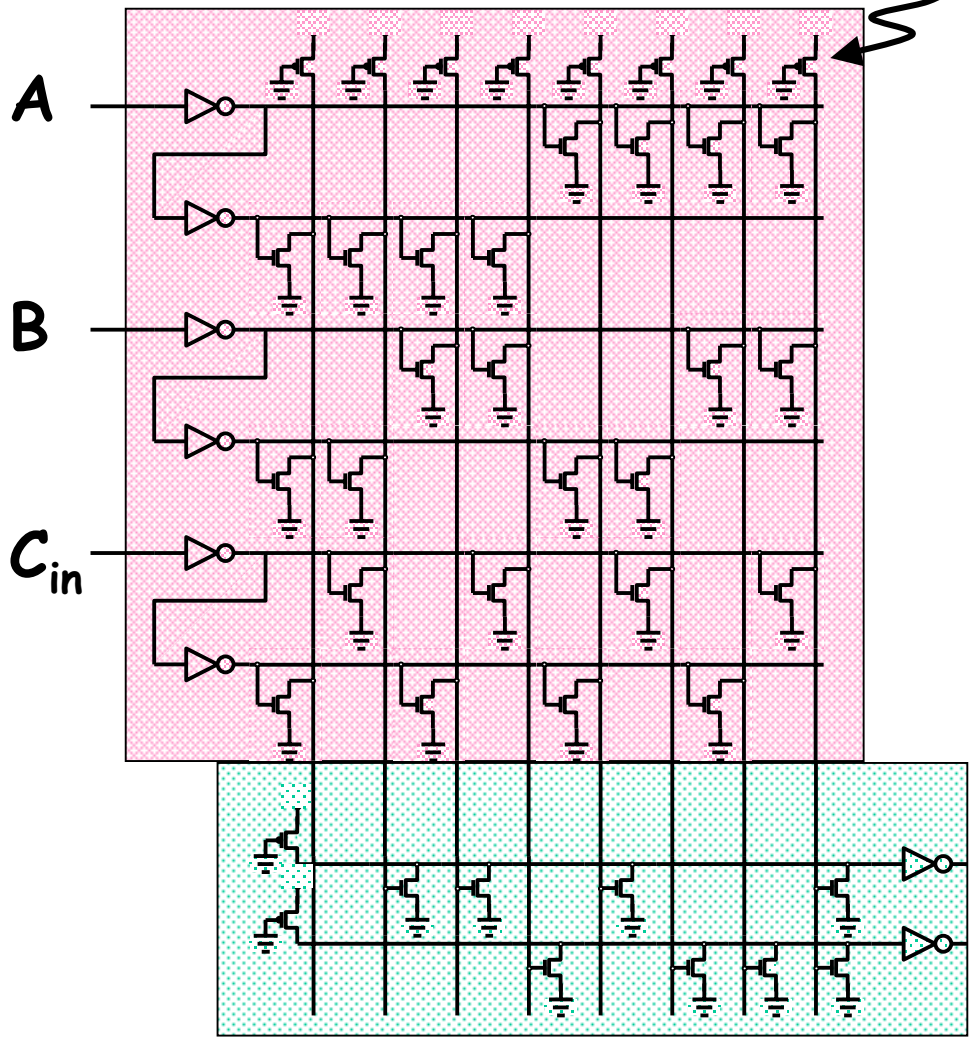


We can build a general purpose "table-lookup" device called a Read-Only Memory (ROM), from which we can implement any truth table and, thus, any combinational device

Made from PREWIRED connections ●, and CONFIGURABLE connections that can be either connected ◍ or not connected ○

ROM Implementation Details

PFET with gate tied to ground = resistor pullup that makes wire "1" unless one of the NFET pulldowns is on.



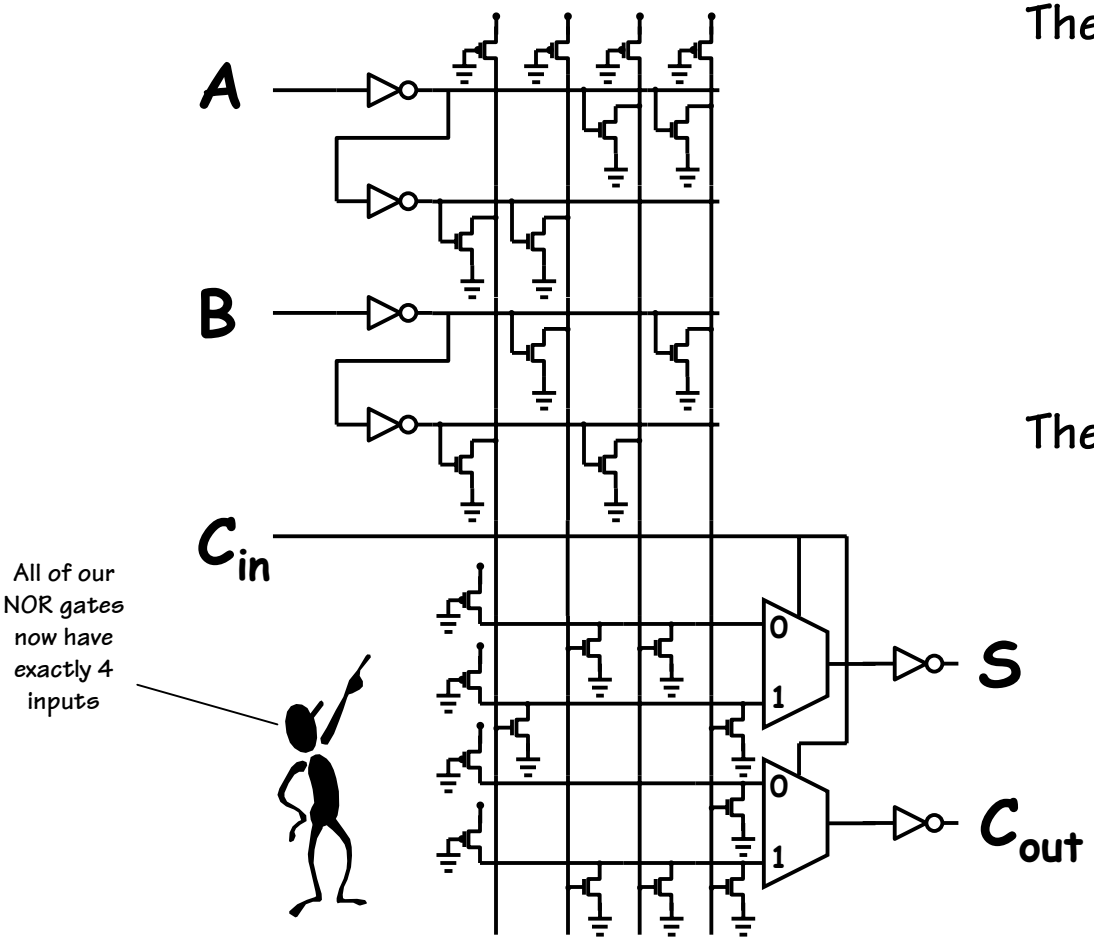
Hardwired AND logic
Programmable OR logic

Advantages:
- Very regular design (can be entirely automated)

Problems:
- Active Pull-ups (Static Power)
- Long metal runs (Large Caps)
- Slow

JARGON: Inputs to a ROM are called ADDRESSES. The decoder's outputs are called WORD LINES, and the outputs lines of the selector are called BIT LINES.

Speeding up ROMS



The key making ROMS go fast is to minimize the capacitances of those long wires running through the array.

The best way to accomplish this is to build *square arrays*.

Why NORs? Couldn't we eliminate some inverters by using NANDs?

Logic According to ROMs

ROMs *ignore* the structure of combinational functions ...

- Size, layout, and design are independent of function
- Any Truth table can be "programmed" by minor reconfiguration:

  - Metal layer (masked ROMs)
  - Fuses (Field-programmable PROMs)
  - Charge on floating gates (EPROMs)
  - ... etc.

ROMs tend to generate "glitchy" outputs. WHY?

Model: LOOK UP value of function in truth table...

Inputs: "ADDRESS" of a T.T. entry

ROM SIZE = # TT entries...

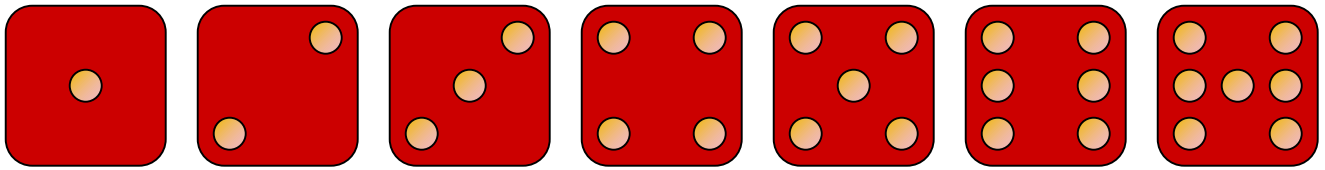
... for an N-input boolean function, size = 2^N x #outputs

Why do ROM SIZES grow by factors of 4?

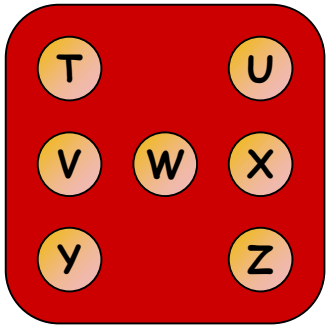
Example: 7-sided Die

What nature can't provide... electronics can (with the same number of LEDs!).

We want to construct a die with the following sides:



An array of LEDs, labeled as follows, can be used to display the outcome of the die:



ROM-Based Design

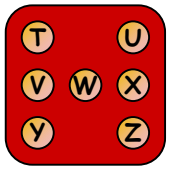
Truth Table for a 7-sided Die

A	B	C	T	U	V	W	X	Y	Z
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
1	0	0	1	1	0	0	0	1	1
1	0	1	1	1	0	1	0	1	1
1	1	0	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1

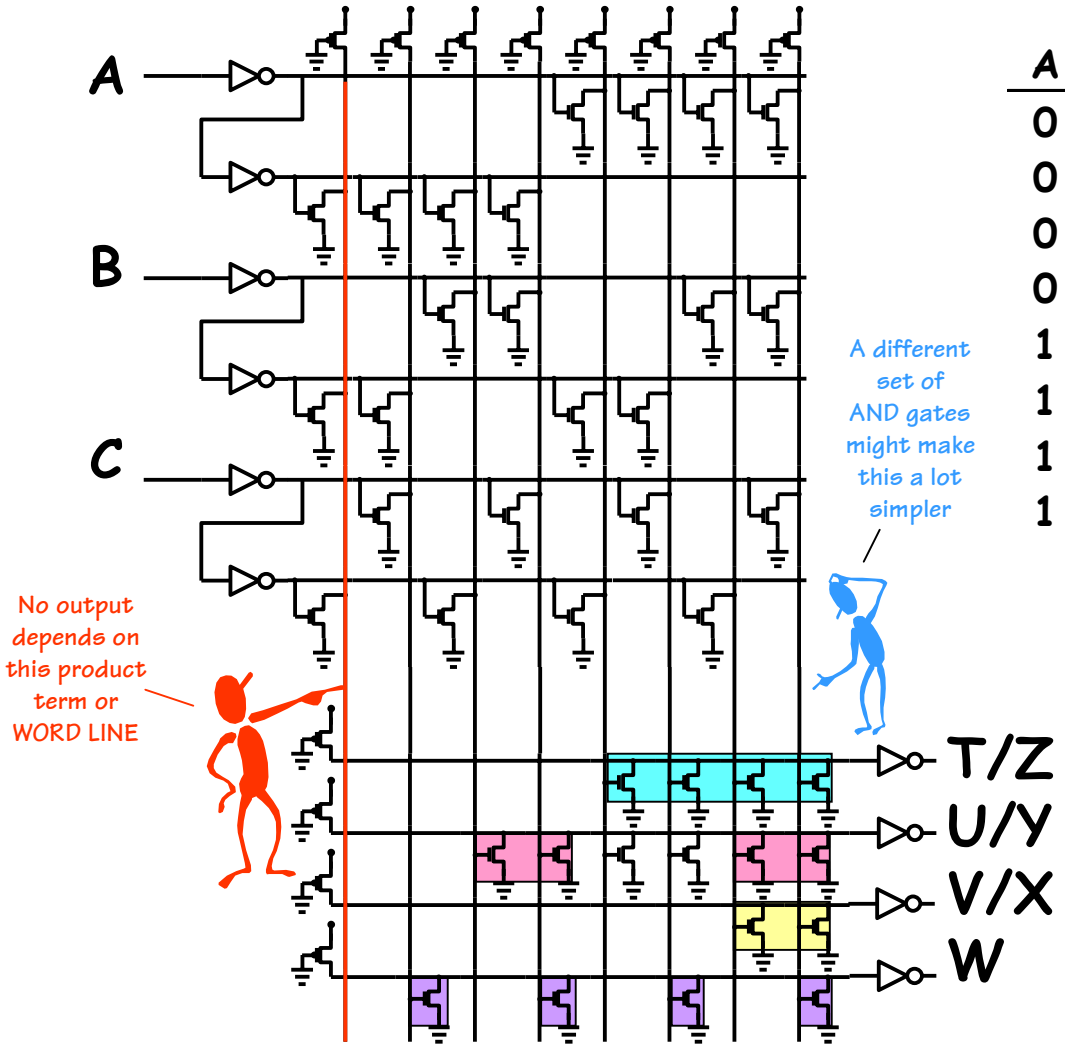
Once we've written out the truth table we've basically finished the design

Possible optimizations:

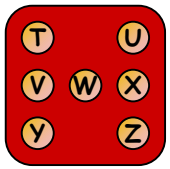
- Eliminate redundant outputs
- Addressing tricks



A Simple ROM implementation



A	B	C	T/Z	U/Y	V/X	W
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	0	0
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1



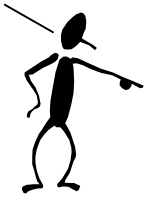
That was easy but there is clearly some waste.

- unused products
- over-specified terms

Another General-Purpose Logic Device

What if the AND terms of a ROM's decoder were programmable in the same way that the OR terms are? Then we could use some of our logic minimization tricks to reduce the size of the ROM array.

This logic is so simple we should just build it with 2 gates!

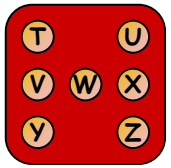


T/Z = A

U/Y = A + B

V/X = AB

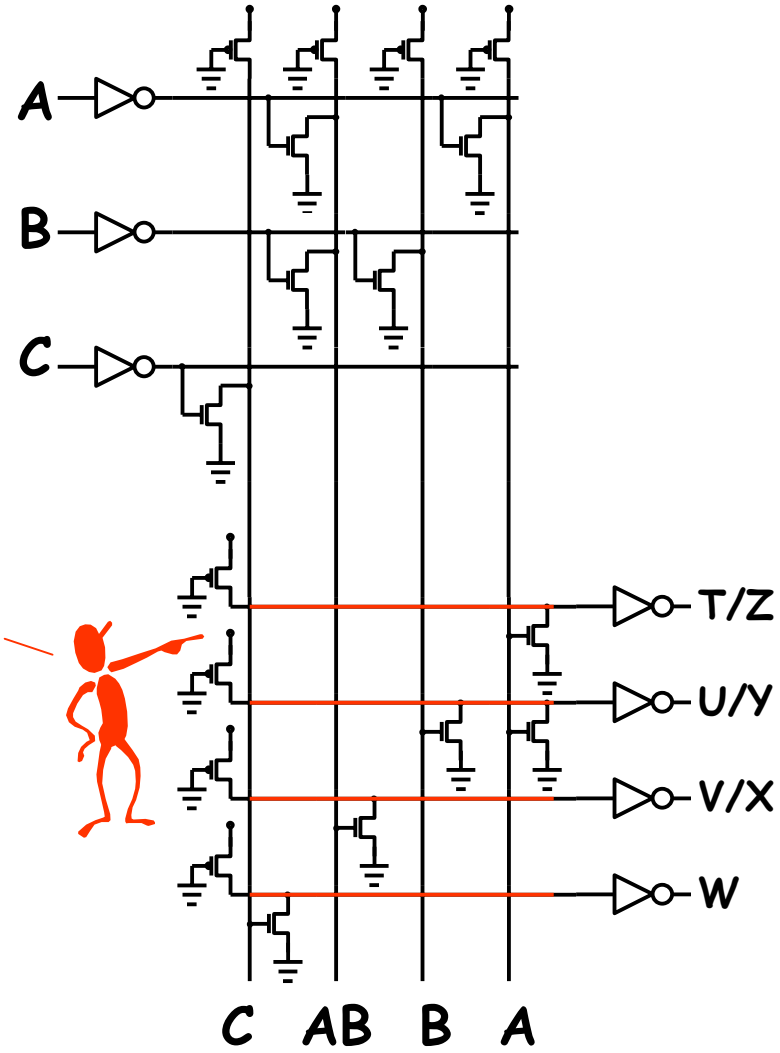
W = C



A	B	C	T/Z	U/Y	V/X	W
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	0	0
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1

PLA – Programmable Logic Array

PLA 7-sided Die implementation



The BIT lines of PLAs tend to be long and limit their speed

PLAs like ROMs support the synthesis of arbitrary logic functions using SOP implementations.

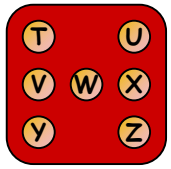
However, they allow for

- minimal realizations
- smaller (faster) arrays

Regular structure

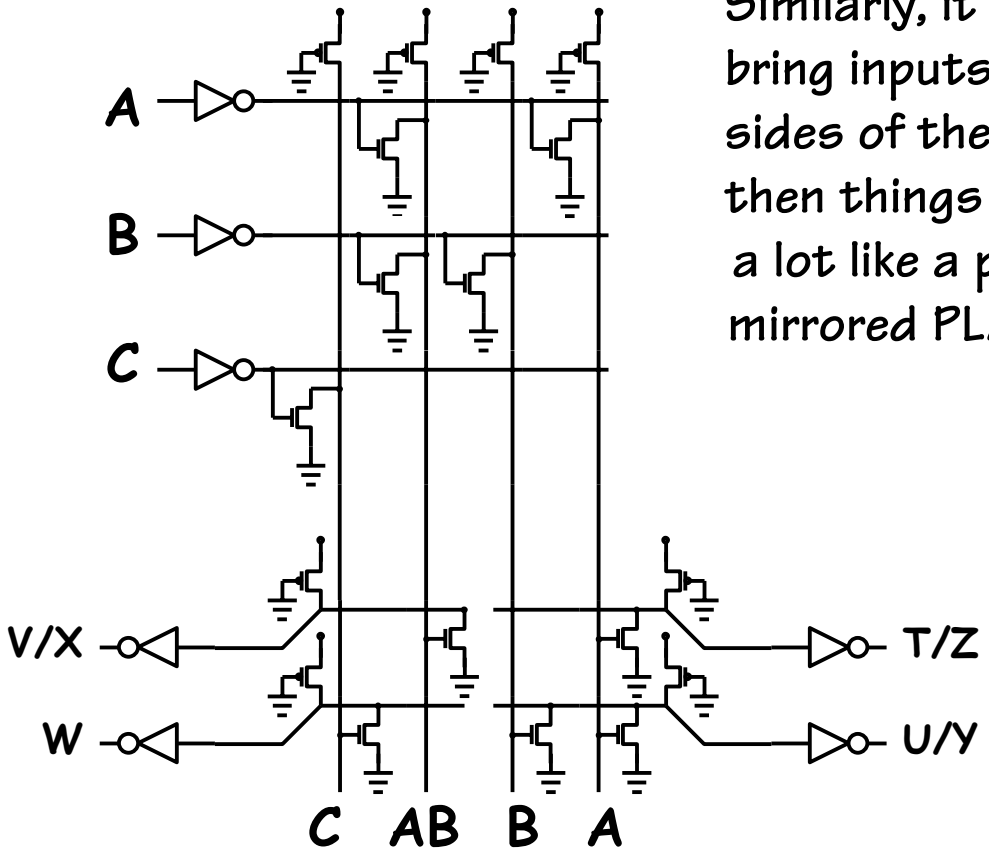
- automatic generation
- easy design
- still slower than optimized gates

One More Trick!

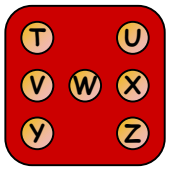


PLA Folding

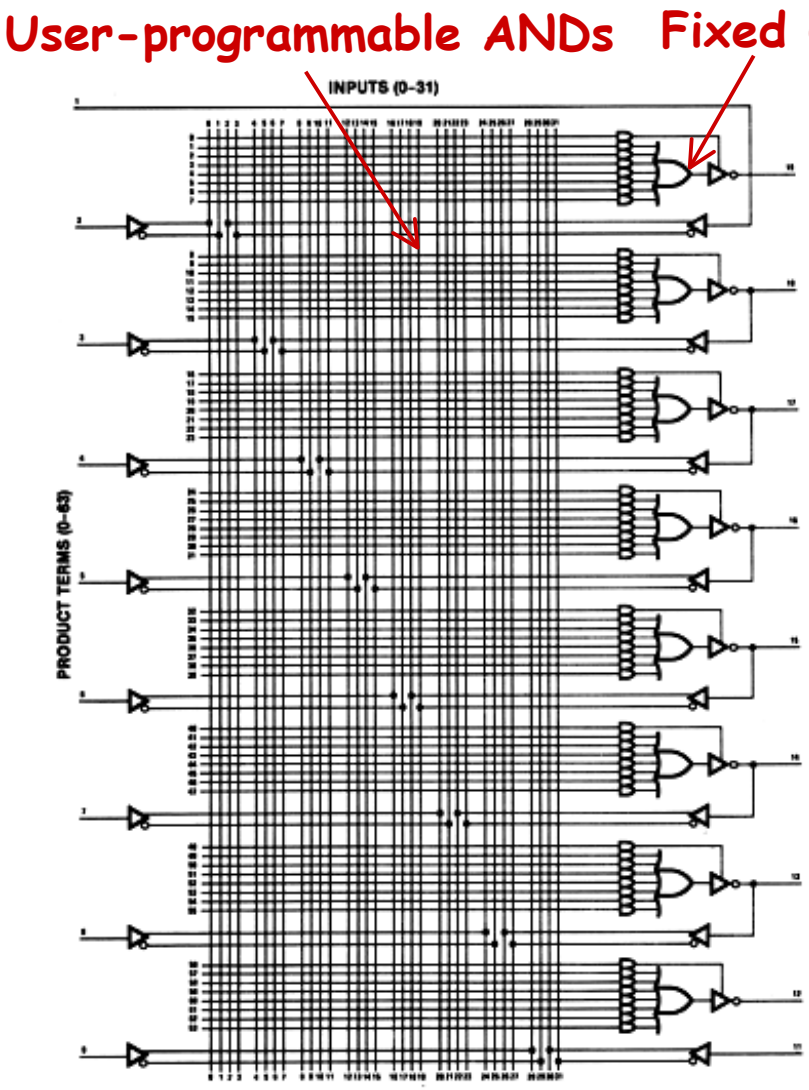
Often we can share the same bit line with two outputs



Similarly, it is possible to bring inputs into both sides of the array, but then things start to look a lot like a pair of mirrored PLAs.

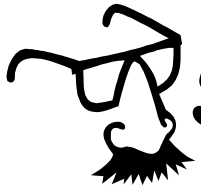


PALs: Programmable Array Logic



Another approach to structured logic design is Programmable Array Logic (PAL). These were once popular off-the-shelf devices. They basically replaced TTL gates in the '80s and fueled the minicomputer revolution. Today, they are practically fossils

PALs have a programmable decoder (AND plane) with fixed selector logic (OR plane). These devices were useful for implementing large fan-in gates and SOP logic expressions. They could be purchased as unprogrammed chips and configured in the field using an inexpensive programmer.



Summary of Logic Arrays

MUXes

- An N-input mux can directly implement an N-input truth tables with 1-output

ROMs

- Shared decoding logic
- Generates all products
- Can implements truth tables with any number of outputs
- Easy to specify
- Regular layout
- Inefficient use of space/time

PLAs

- Can achieve minimal SOP realizations
- Generates only the products needed
- Relies on minimization for compact designs
- Regular layout

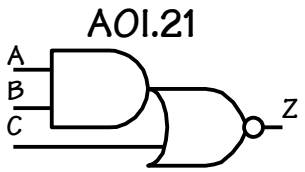
PALs

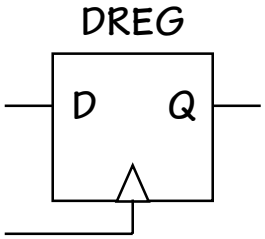
- Easily prefabricated and packaged

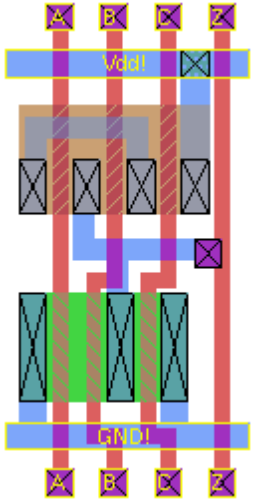
Today… it's back to gates

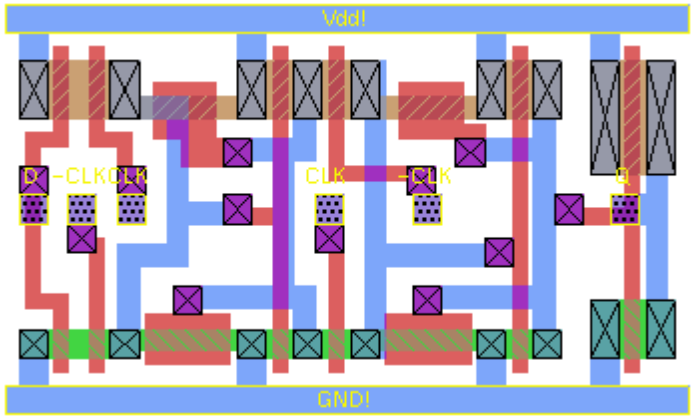
Modern Methods: Standard Cells

First, a library of fixed-pitch logic cells (gates, registers, muxes, adders, I/O pads, ...) are created. A data sheet for each cell describes its function, area, power, propagation delay, output rise/fall time as function of load, etc.

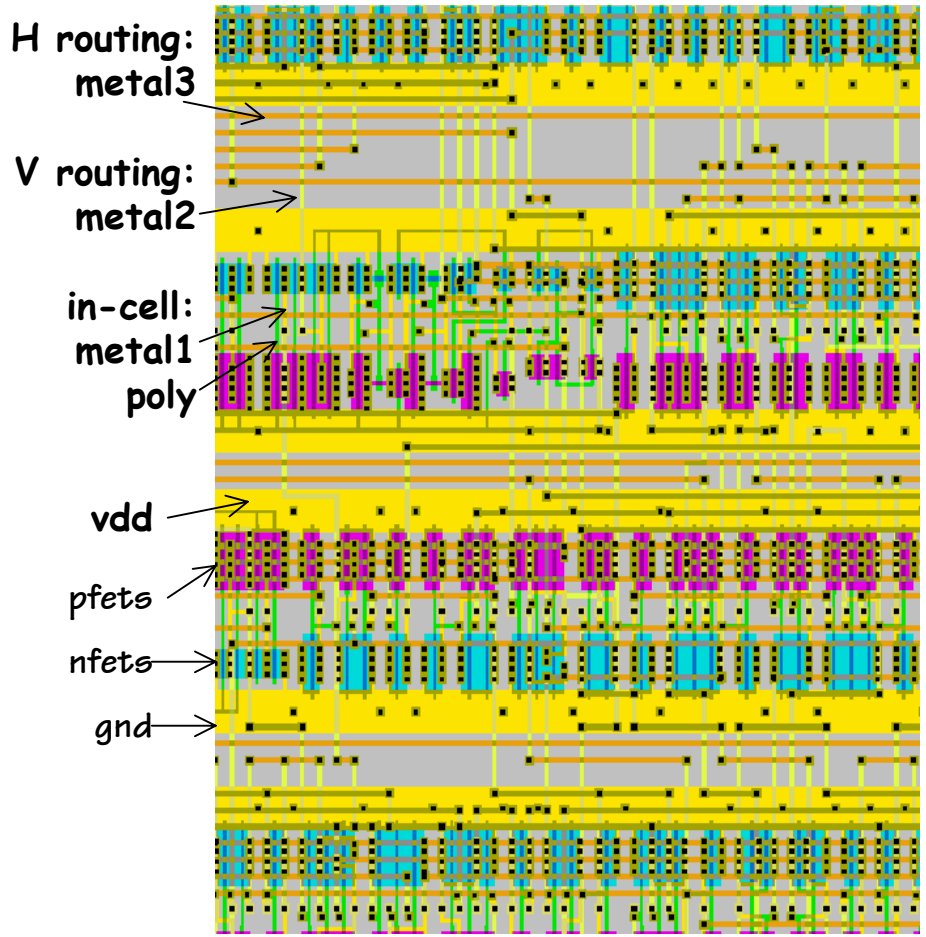








Standard Cell Example



Similar to designing with board level components 20 years ago. CAD tools place and route cells.

- minimize area
- meet timing specs

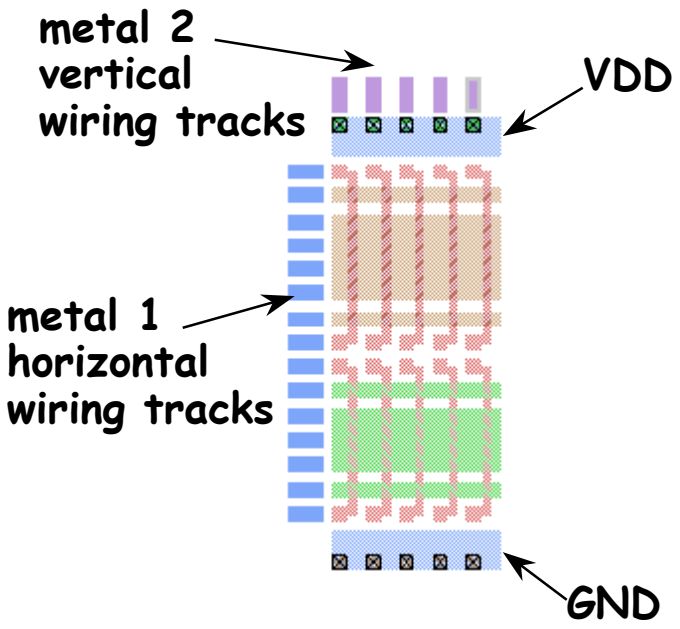
In this case, the router needed so much space for running wires that cell rows were pushed apart. In many cases, wiring determines the size of the layout!

Fast Turn-around: Gate Arrays

Gate arrays use fixed arrays of transistors that are "personalized" in a final processing step that adds wires and contacts.

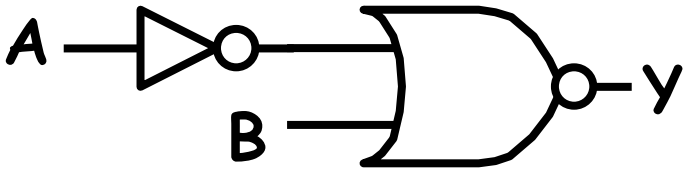
Most popular architecture is "Sea-of-Gates" where the core of the chip is a continuous array of FETs. Routing occurs over the top of unused transistors.

Pros:
- can be prefabricated
- only last few masks are customized

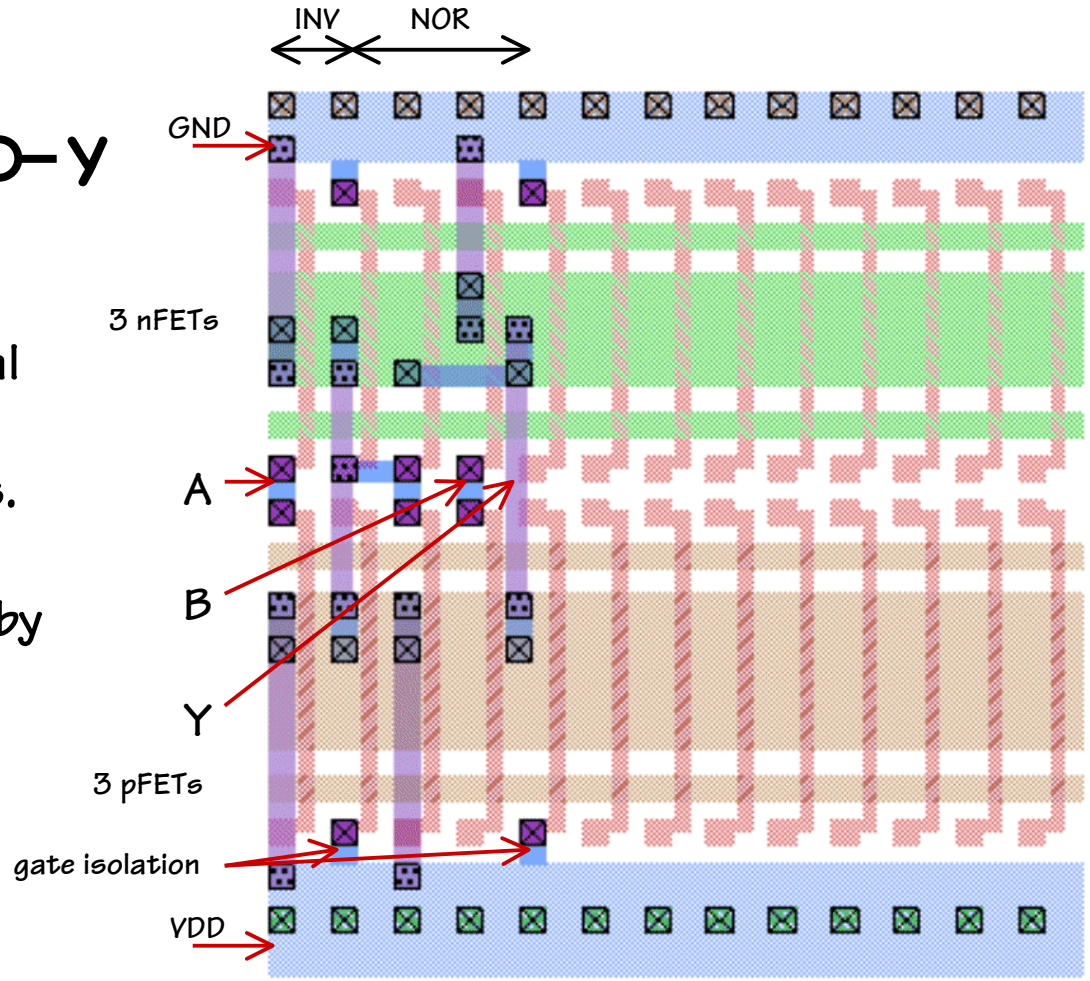


from IEEE JSSC, V25, No5, Oct 1990

Gate Array Example



The last two layers of metal are used to define the function of the transistors. Side-by-side gates are isolated from one another by turning off the gate of a transistor between them. This is called "gate-isolation."



Summary

- Sum of products
  - Karnaugh maps
    - Use minimal cover of prime implicants to get minimal SOP
    - Use all prime implicants to get lenient (glitch-free) implementation
  - SOP implementation methods
    - NAND-NAND, NOR-NOR
    - AOI and OAI for fast low fan-in SOP logic
- Muxes used to build table-lookup implementations
- ROMs
  - Decoder logic generates all possible product terms
  - Selector logic determines which p'terms are or'ed together
  - PLAs are ROMs with optimized decoders that generate only the product terms that are needed

Combinational Logic Design Finale

Now you know everything about combinational logic design. Much combinational logic today is synthesized from high level specifications (equations and truth tables).

Next we'll build circuits that can actually remember something!

DOONESBURY by Garry Trudeau