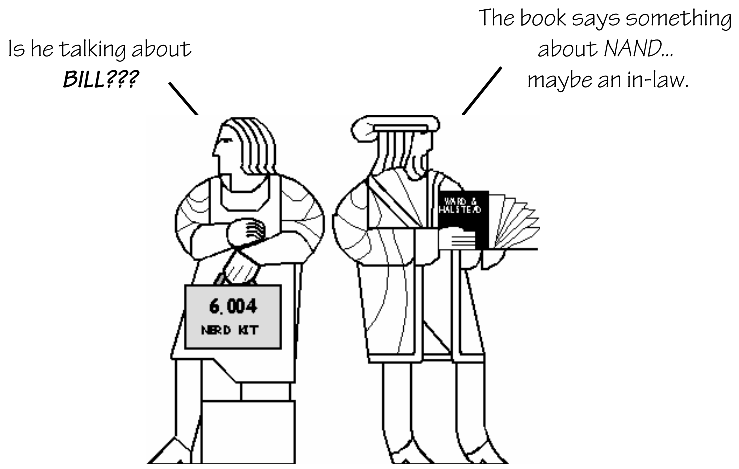
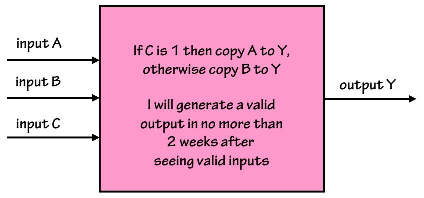
## Gates


Is he talking about *BILL???*
The book says something about NAND... maybe an in-law.
6.004 NERD KIT
WARD & HALSTEAD

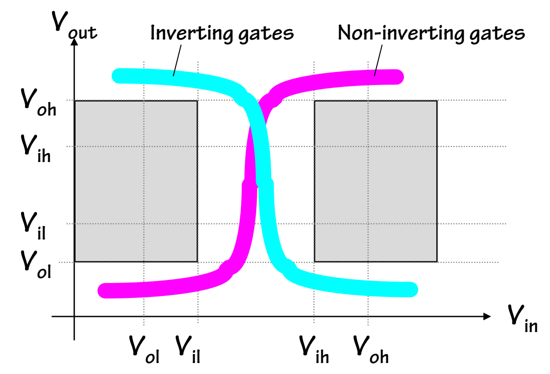
## A Quick Review

- A *combinational device* is a circuit element that has
  - one or more digital *inputs*
  - one or more digital *outputs*
  - a *functional specification* that details the value of each output for every possible combination of valid input values
  - a *timing specification* consisting (at minimum) of an upper bound $t_{PD}$ on the required time for the device to compute the specified output values from an arbitrary set of stable, valid input values

Static discipline


input A
input B
input C
If C is 1 then copy A to Y, otherwise copy B to Y
I will generate a valid output in no more than 2 weeks after seeing valid inputs
output Y

## VTC and the Static Discipline

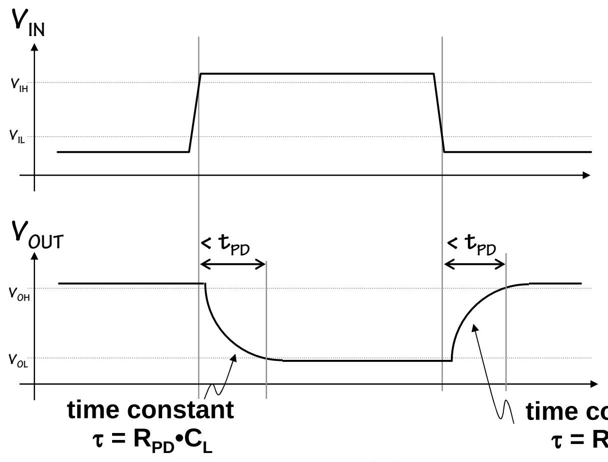


Static Discipline requires that we avoid gray areas, which correspond to *valid* inputs but *invalid* outputs. Net result: combinational devices must have GAIN and be NONLINEAR.

The good news: CMOS gates do all this with the added bonus of no static power!

## Due to unavoidable delays…

Propagation delay ($t_{PD}$):
An UPPER BOUND on the delay from valid inputs to valid outputs.



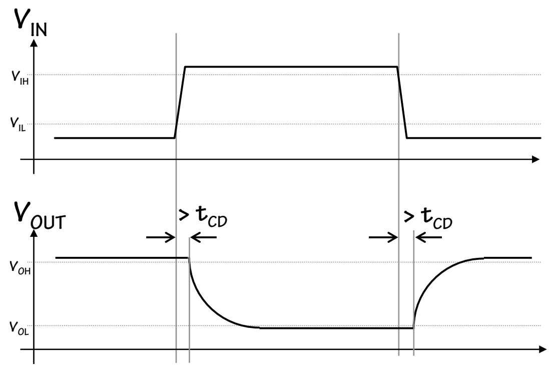
GOAL:
*minimize* propagation delay!

ISSUE:
keep Capacitances low and transistors fast

## Contamination Delay

*an optional, additional timing spec*

INVALID inputs take time to propagate, too...



Do we really need $t_{CD}$?

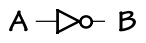
Usually not… it'll be important when we design circuits with registers (coming soon!)

If $t_{CD}$ is not specified, safe to assume it's 0.

CONTAMINATION DELAY, $t_{CD}$

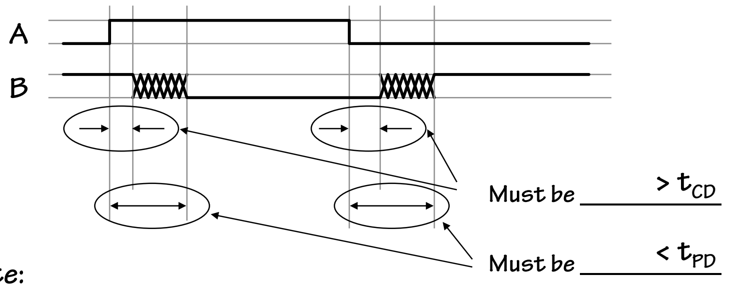
A LOWER BOUND on the delay from any invalid input to an invalid output

## The Combinational Contract



| A | B |
|---|---|
| 0 | 1 |
| 1 | 0 |

$t_{PD}$ propagation delay
$t_{CD}$ contamination delay



Note:

1. *No Promises* during ▨
2. Default (conservative) spec: $t_{CD} = 0$

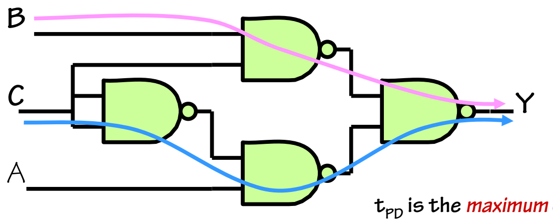
## Acyclic Combinational Circuits

If NAND gates have a $t_{PD}$ = 4nS and $t_{CD}$ = 1nS

$t_{PD}$ = 12 nS

$t_{CD}$ = 2 nS

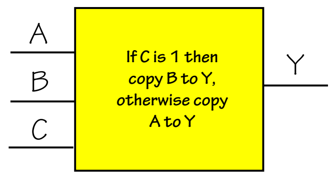
$t_{CD}$ is the *minimum* cumulative contamination delay over all paths from inputs to outputs

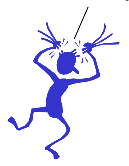


$t_{PD}$ is the *maximum* cumulative propagation delay over all paths from inputs to outputs

## Functional Specifications

There are many ways of specifying the function of a combinational device, for example:





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 |

$Y = \overline{C}\overline{B}A + \overline{C}BA + CB\overline{A} + CBA$

Concise alternatives:

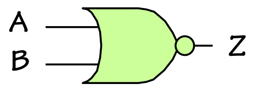
***truth tables*** are a concise description of the combinational system's function.

***Boolean expressions*** form an algebra in whose operations are **AND** (multiplication), **OR** (addition), and **inversion** (overbar).

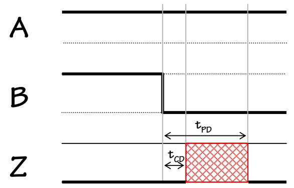
*Any combinational (Boolean) function can be specified as a truth table or an equivalent sum-of-products Boolean expression!*

## Oh yeah… one last issue

NOR:



| A | B | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



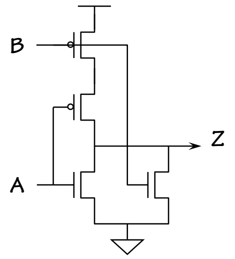
Recall the rules for *combinational devices*:

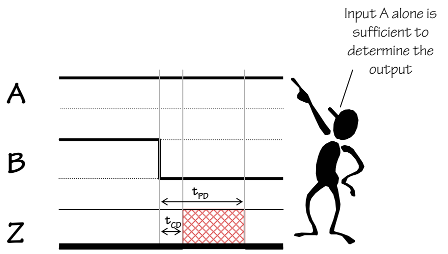
Output guaranteed to be valid when all inputs have been valid for at least $t_{PD}$, and, outputs may become invalid no earlier than $t_{CD}$ after an input changes!

Many gate implementations--e.g., CMOS—
adhere to even tighter restrictions.

## What happens in this case?

CMOS NOR:





*LENIENT* Combinational Device:

Output guaranteed to be valid when any combination of inputs sufficient to determine output value has been valid for at least $t_{PD}$.

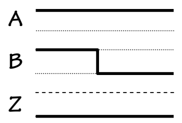
*Tolerates transitions -- and invalid levels -- on irrelevant inputs!*

NOR:

| A | B | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

*Lenient* NOR:

| A | B | Z |
|---|---|---|
| 0 | 0 | 1 |
| X | 1 | 0 |
| 1 | X | 0 |



## Basic Gate Repertoire

Are we sure we have all the gates we need?

Just how many two-input gates are there?

AND

| AB | Y |
|---|---|
| 00 | 0 |
| 01 | 0 |
| 10 | 0 |
| 11 | 1 |

OR

| AB | Y |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 1 |
| 11 | 1 |

NAND

| AB | Y |
|---|---|
| 00 | 1 |
| 01 | 1 |
| 10 | 1 |
| 11 | 0 |

NOR

| AB | Y |
|---|---|
| 00 | 1 |
| 01 | 0 |
| 10 | 0 |
| 11 | 0 |

Hmmmm… all of these have 2-inputs (no surprise)

… each with 4 combinations, giving $2^2$ output cases

How many ways are there of assigning 4 outputs? $2^{2^2} = 2^4 = 16$

## There are only so many gates

There are only 16 possible 2-input gates

… some we know already, others are just silly

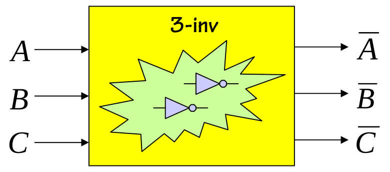
How many of these gates can be implemented using a single CMOS gate?

| INPUT AB | ZERO | AND | A>B | A | B>A | B | XOR | OR | NOR | XNOR | NOT 'B' | A<=B | NOT 'A' | B<=A | NAND | ONE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 01 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

CMOS gates are inverting; we can always respond positively to positive transitions by cascaded gates. But suppose our logic yielded cheap *positive* functions, while inverters were expensive…

## Logic Geek Party Games

You have plenty of ANDs and ORs, but only 2 inverters. Can you invert more than 2 independent inputs?



CHALLENGE: Come up with a combinational circuit using ANDs, ORs, and at most 2 inverters that inverts A, B, and C !

Such a circuit exists. What does that mean?

- If we can invert 3 signals using 2 inverters, can we use 2 of the pseudo-inverters to invert 3 more signals?
- Do we need only 2 inverters to make ANY combinational circuit?

Hint: there's a subtle difference between our 3-inv device and three combinational inverters!

Is our 3-inv device *LENIENT?*

## Fortunately, we can get by with a few basic gates…

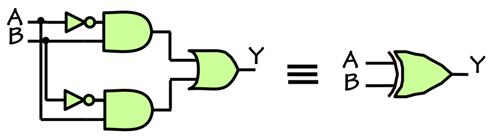
AND, OR, and NOT are sufficient… (cf Boolean Expressions):

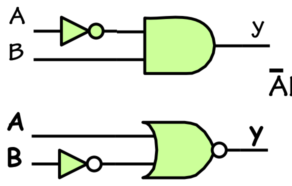
B>A

| AB | Y |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 0 |
| 11 | 0 |

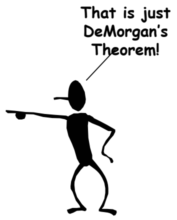
XOR

| AB | Y |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 1 |
| 11 | 0 |

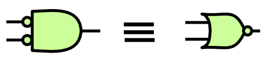




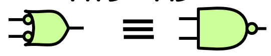
$\overline{A}B = \overline{A + \overline{B}}$



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



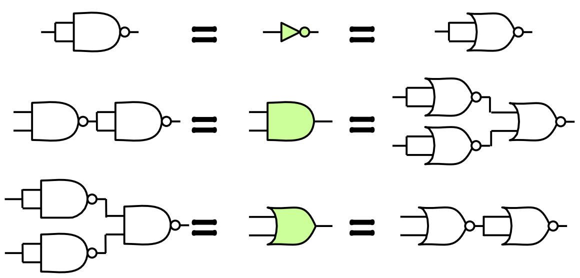
$\overline{A} + \overline{B} = \overline{AB}$



How many different gates do we *really* need?

## One will do!

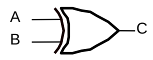
NANDs and NORs are <u>universal:</u>



Ah!, but what if we want more than 2-inputs

## Stupid Gate Tricks

Suppose we have some 2-input XOR gates:

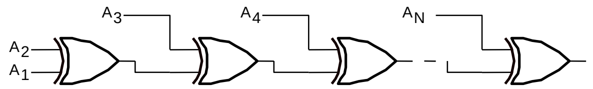


$t_{pd} = 1$
$t_{cd} = 0$

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

And we want an N-input XOR:

output = 1
iff number of 1s
input is ODD
("ODD PARITY")

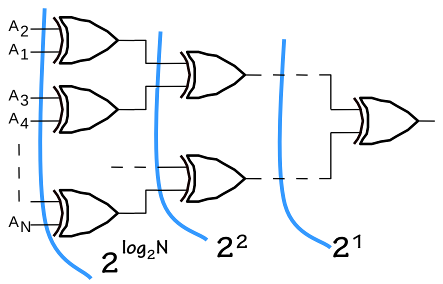


$t_{pd} = O($ <u>N</u> $)$ -- WORST CASE.

Can we compute N-input XOR faster?

## I think that I shall never see

a circuit lovely as...



N-input TREE has O( log N ) levels...

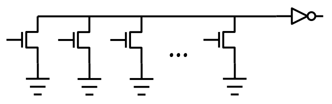
Signal propagation takes O( log N ) gate delays.

Question: Can EVERY N-Input Boolean function be implemented as a tree of 2-input gates?

## Are Trees Always Best?

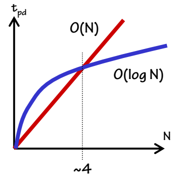
Alternate Plan: Large Fan-in gates

- N pulldowns with complementary pullups
- Output HIGH if any input is HIGH = "OR"



- Propagation delay: O( N )

  since each additional MOSFET adds C



Don't be mislead by the "big O" stuff… the constants in this case can be much smaller… so for small N this plan might be the best.

Didn't he say "ONE LAST ISSUE" back on Slide 9? Lets design stuff!
?

## Here's a Design Approach

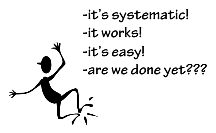
Truth Table

| 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 |

1) Write out our functional spec as a truth table
2) Write down a Boolean expression with terms covering each '1' in the output:

$$Y = \overline{C}\,\overline{B}A + \overline{C}BA + CB\overline{A} + CBA$$

3) Wire up the gates, call it a day, and declare success!

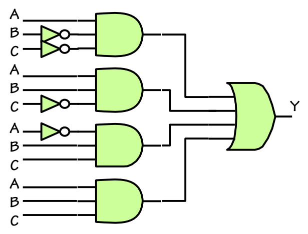


This approach will always give us Boolean expressions in a particular form: SUM-OF-PRODUCTS

## Straightforward Synthesis

We can implement
SUM-OF-PRODUCTS
with just three levels of logic.

INVERTERS/AND/OR

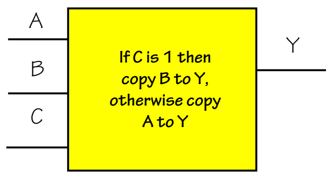


Propagation delay --
No more than 3 gate delays
(ignoring fan-in)

## Oh, by the way…

That Gate has a Name!

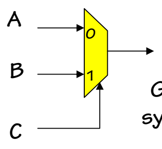
The gate we've been designing for this lecture is a relatively important one:



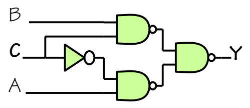
2-input Multiplexer

Truth Table

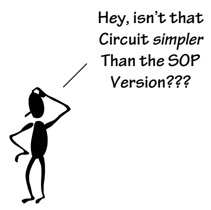
| C | B | A | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



Gate symbol



(one) implementation



## Logic Simplification

Can we implement the same function with fewer gates?
Before trying we'll add a few more tricks in our bag.

BOOLEAN ALGEBRA:

| | |
|---|---|
| OR rules: | $a + 1 = 1,\ a + 0 = a,\ a + a = a$ |
| AND rules: | $a1 = a,\ a0 = 0,\ aa = a$ |
| Commutative: | $a + b = b + a,\ ab = ba$ |
| Associative: | $(a + b) + c = a + (b + c),\ (ab)c = a(bc)$ |
| Distributive: | $a(b+c) = ab + ac,\ a + bc = (a+b)(a+c)$ |
| Complements: | $a+\bar{a}=1,\quad a\bar{a}=0$ |
| Absorption: | $a+ab=a,\ \ a+\bar{a}b=a+b$ |
| | $a(a+b)=a,\ \ a(\bar{a}+b)=ab$ |
| Reduction: | $\boxed{ab+\bar{a}b=b},\ (a+b)(\bar{a}+b)=b$ |
| DeMorgan's Law: | $\bar{a}+\bar{b}=\overline{ab},\ \ \bar{a}\bar{b}=\overline{a+b}$ |

## Boolean Minimization:

An Algebraic Approach

Lets (again!) simplify

$$Y=\bar{C}\bar{B}A+CB\bar{A}+CBA+\bar{C}BA$$

Using the identity

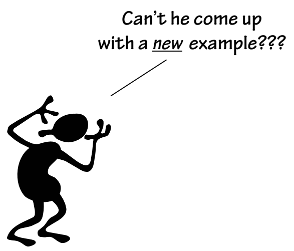
$$\alpha A+\alpha\bar{A}=\alpha$$

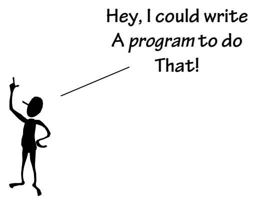
For any expression α and variable A:

$$Y=\bar{C}\bar{B}A+CB\bar{A}+CBA+\bar{C}BA$$

$$Y=\bar{C}\bar{B}A+CB+\bar{C}BA$$

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





## Summary

- Timing specs
  - $t_{PD}$: upper bound on time from valid inputs to valid outputs
  - $t_{CD}$: lower bound on time from invalid inputs to invalid outputs
  - If not specified, assume $t_{CD} = 0$
- Combinational logic
  - Any function that can be specified by a truth table or, equivalently, in terms of AND/OR/NOT (Boolean expression)
  - *Lenience:* optional, more demanding functional guarantee. Rarely needed; assume non-lenient logic by default.
  - Minimally, we can get away with just 2-input NANDs or NORs
- Sum-of-products canonical form
  - Comes directly from truth table
  - "3-level" implementation of any logic function
  - Limitations on number of inputs (fan-in) increases depth
- Next time: logic simplification, other canonical forms