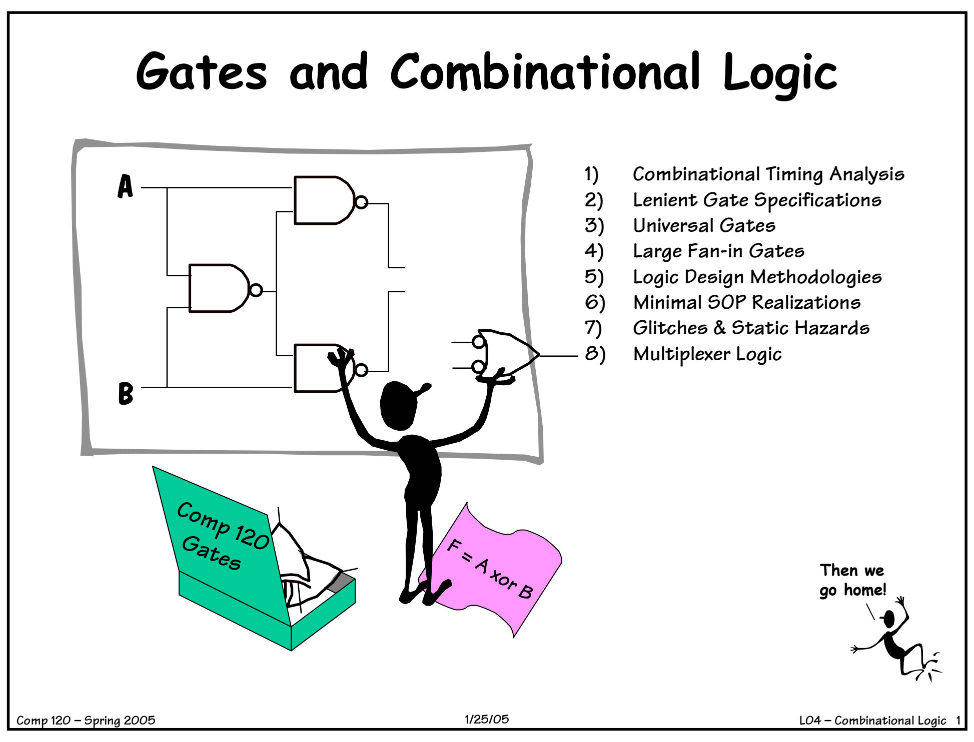
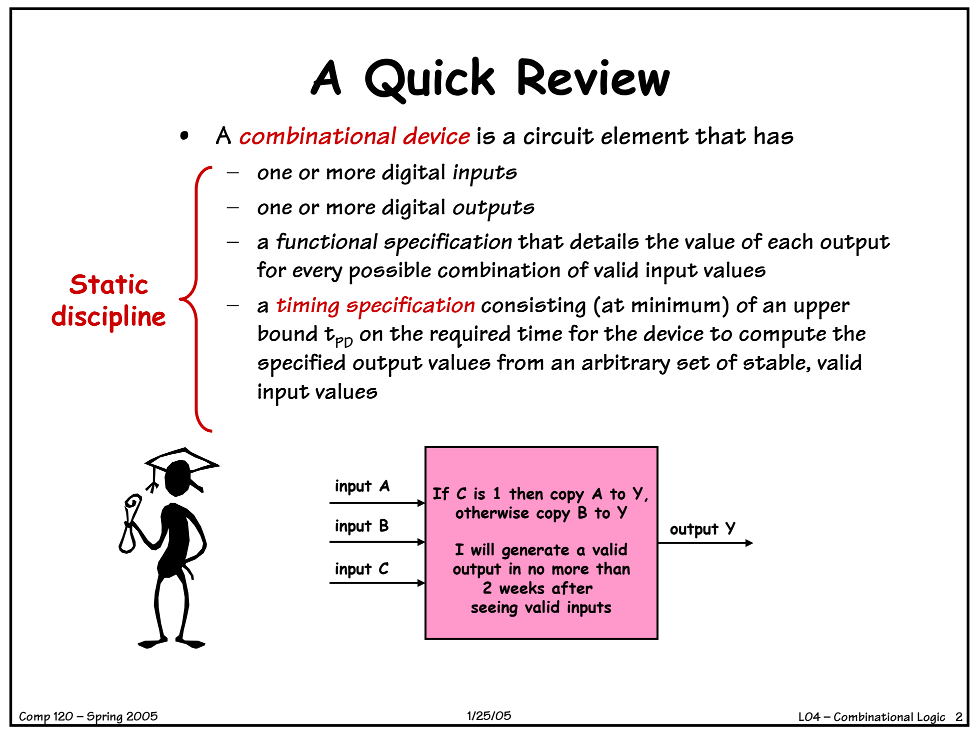
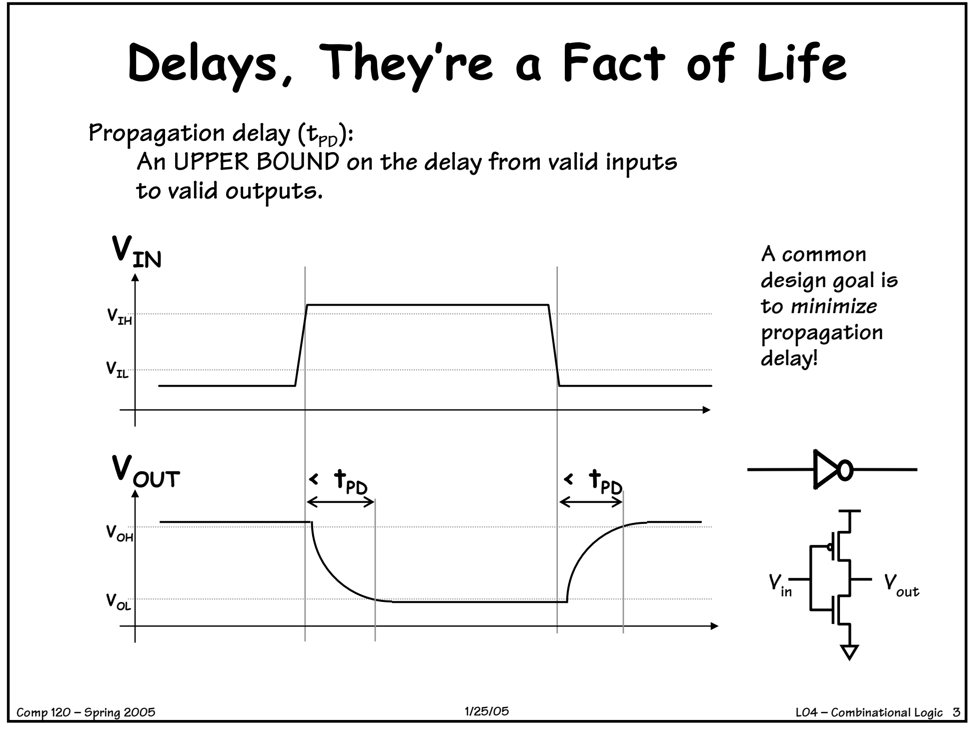
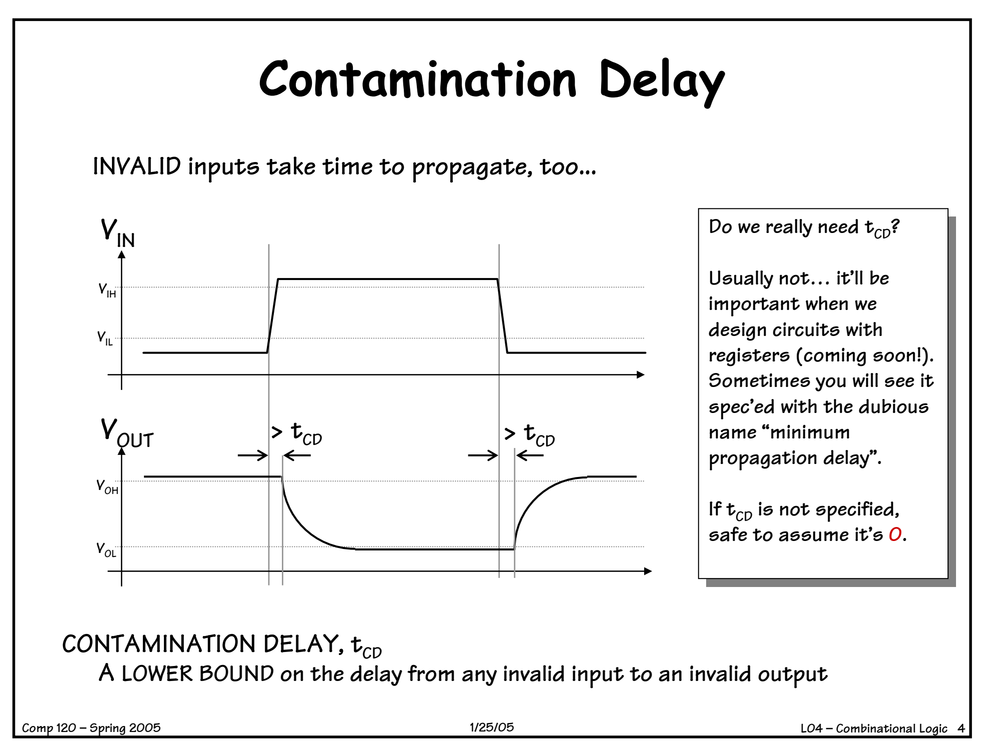
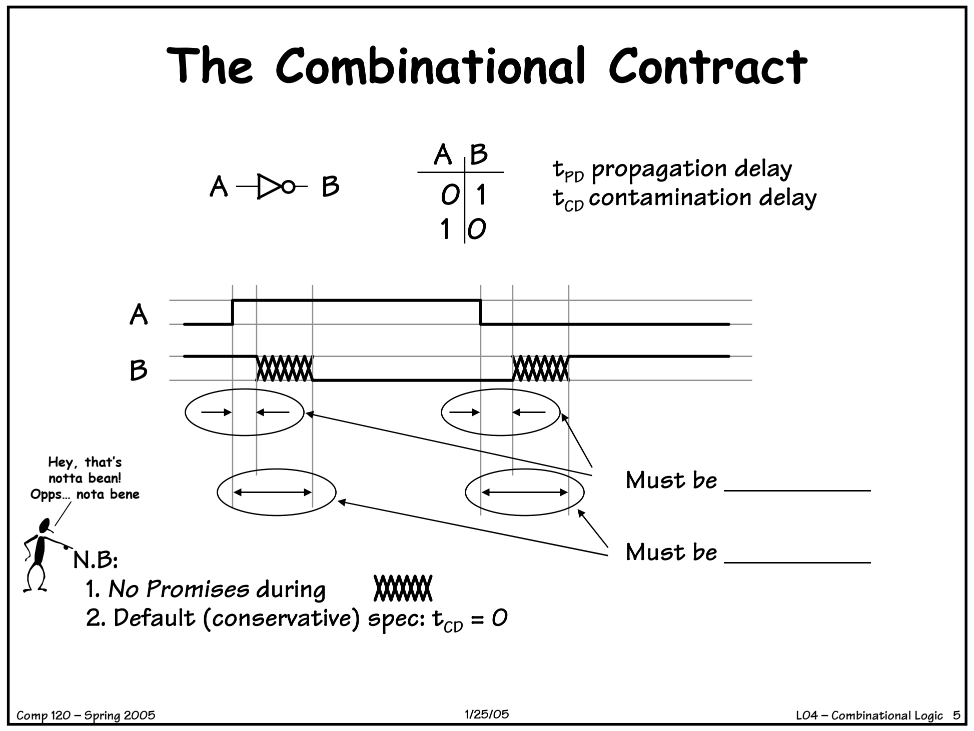
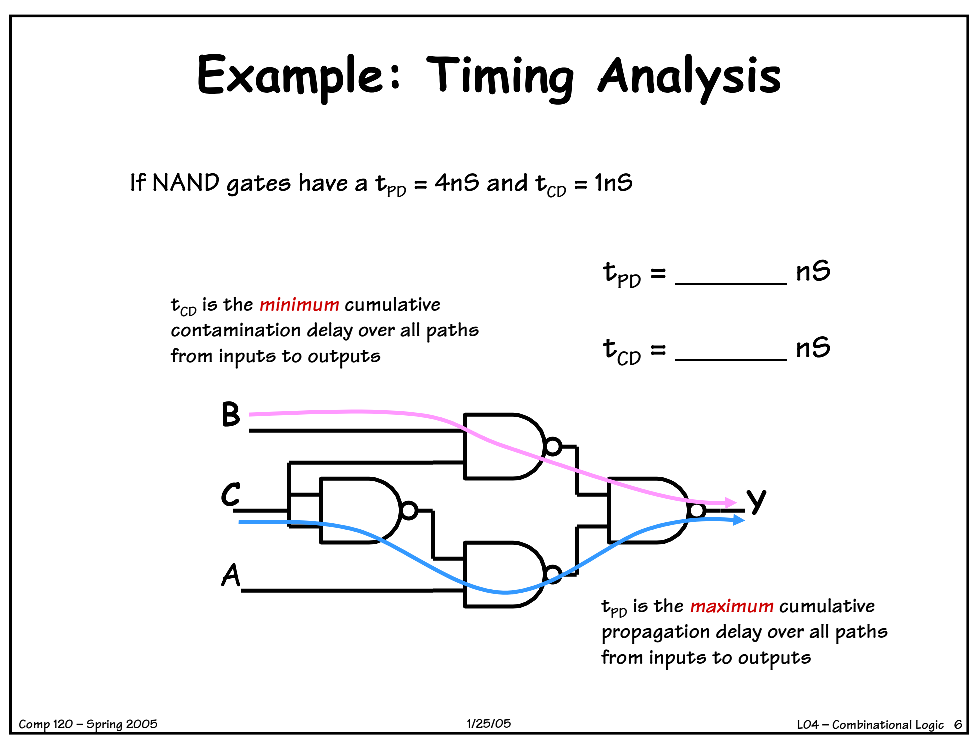

Gates and Combinational Logic
A
B
1) Combinational Timing Analysis
2) Lenient Gate Specifications
3) Universal Gates
4) Large Fan-in Gates
5) Logic Design Methodologies
6) Minimal SOP Realizations
7) Glitches & Static Hazards
8) Multiplexer Logic
Comp 120
Gates
F = A xor B
Then we
go home!
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 1


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 tPD 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
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 2

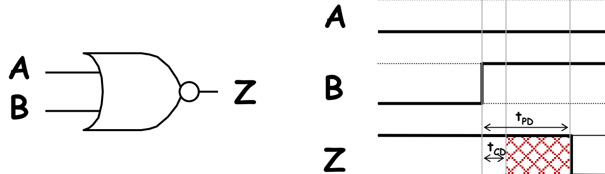

Delays, They're a Fact of Life
Propagation delay (tPD):
An UPPER BOUND on the delay from valid inputs to valid outputs.
VIN
VIH
VIL
VOUT
VOH
VOL
< tPD
< tPD
A common design goal is to minimize propagation delay!
Vin
Vout
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 3


Contamination Delay
INVALID inputs take time to propagate, too...
VIN
VIH
VIL
VOUT
VOH
VOL
> tCD
> tCD
Do we really need tCD?
Usually not… it'll be important when we design circuits with registers (coming soon!). Sometimes you will see it spec'ed with the dubious name "minimum propagation delay".
If tCD is not specified, safe to assume it's 0.
CONTAMINATION DELAY, tCD
A LOWER BOUND on the delay from any invalid input to an invalid output
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 4


The Combinational Contract
A B
A B
0 1
1 0
tPD propagation delay
tCD contamination delay
A
B
Hey, that's notta bean! Opps... nota bene
Must be
Must be
N.B:
1. No Promises during
2. Default (conservative) spec: tCD = 0
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 5


Example: Timing Analysis
If NAND gates have a tPD = 4nS and tCD = 1nS
tPD = nS
tCD = nS
tCD is the minimum cumulative contamination delay over all paths from inputs to outputs
B
C
A
Y
tPD is the maximum cumulative propagation delay over all paths from inputs to outputs
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 6

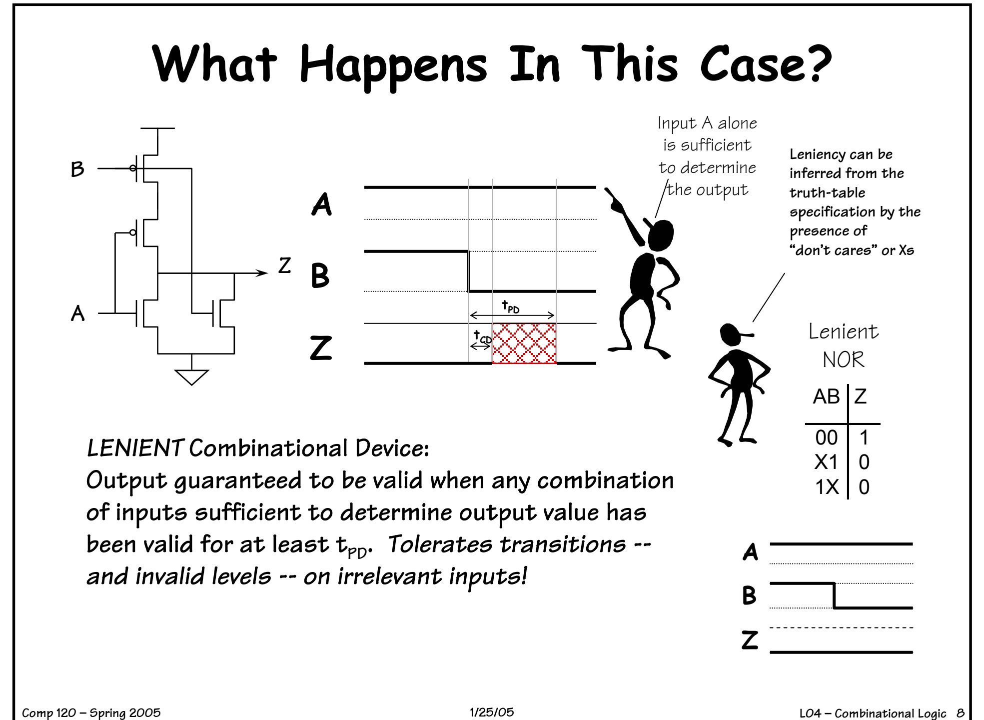
One Last Issue...



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.



Now We're Ready to Design Stuff!

We need to start somewhere -- usually it's the functional specification

A
B
C
If C is 1 then copy B to Y, otherwise copy A to Y
Y

Argh… I'm tired of word games

If you are like most scientist you'd rather see a formula than parse a logic puzzle. The fact is, **any combinational function can be expressed as a table.**

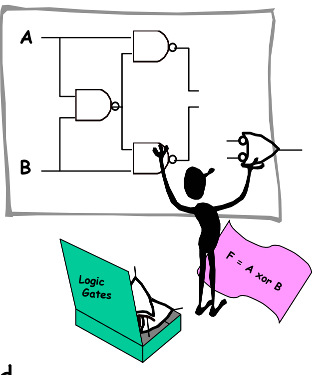
These "**truth tables**" are a concise description of the combinational system's function. Conversely, any computation performed by a combinational system can expressed as a truth table.

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

Where Do We Start?

We have a bag of gates.



We have a spec.

What do we do?

Did I mention we have gates?

We need
… a systematic approach for designing logic

A Slight Diversion

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

Hum… all of these have 2-inputs (no surprise)
… each with 4 permutations, giving 2^2 output cases
How many permutations of 4 outputs are there? ___

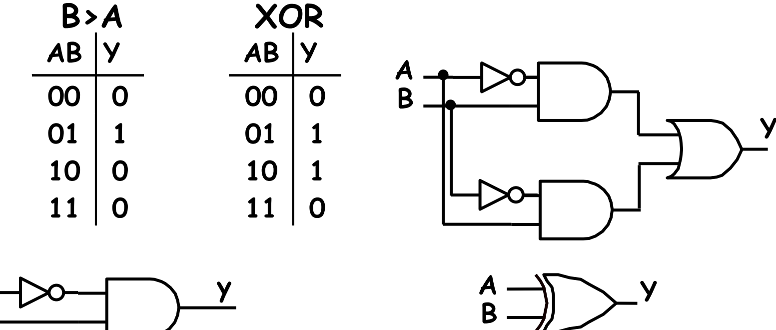
There Are Only So Many Gates

There are only 16 possible 2-input gates
… some we know already, others are just silly

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

Do we need all of these gates?
Nope. After all, we describe them all using AND, OR, and NOT.

We Can Make Most Gates Out of Others



B>A

AB	Y
00	0
01	1
10	0
11	0

XOR

AB	Y
00	0
01	1
10	1
11	0

How many different gates do we really need?

One Will Do!

NANDs and NORs are universal

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

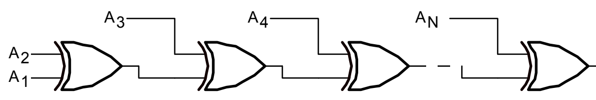
Stupid Gate Tricks

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:



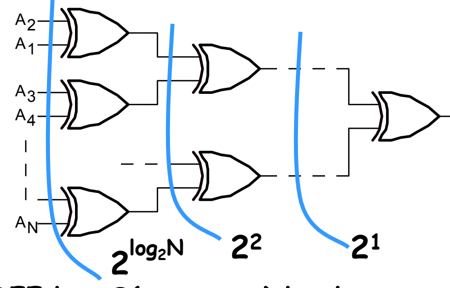
output = 1 iff number of 1s input is ODD ("ODD PARITY")

t_{pd} = O(___) -- WORST CASE.

Can we compute N-input XOR faster?

I Think That I Shall Never See

a Gate Lovely as a ...



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

Signal propagation takes O(_______) gate delays.

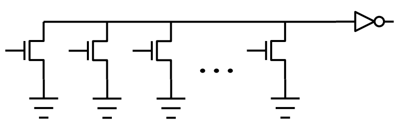
Question: Can EVERY N-Input Boolean function be implemented as a tree of 2-input gates?

Are Trees Always Best?

Didn't he say back on Slide 9 that we were gonna design stuff?

Alternate Plan: Large Fan-in gates

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



- Propagation delay: O(____)
  since each additional MOSFET adds C

t_{pd} O(N) O(log N) N ~4

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.

Here's a Design Approach

Truth Table

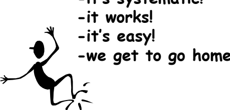
C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

1) Write out our functional spec as a truth table
2) Write down a Boolean expression for every '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 go home!

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

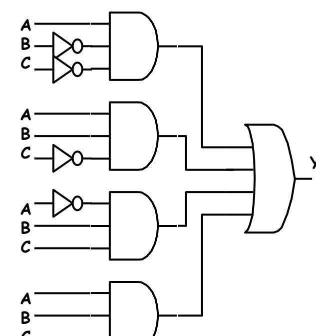


Straightforward Synthesis

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)



Logic Simplification

Can we implement the same function with fewer gates?
Before trying we'll add a few more tricks in our bag.
RULES 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:	$a + \overline{a} = 1, \quad a\overline{a} = 0$
Absorption:	$a + ab = a, \quad a + \overline{a}b = a + b$ $a(a+b) = a, \quad a(\overline{a}+b) = ab$
Reduction:	$ab + \overline{a}b = b, \quad (a+b)(\overline{a}+b) = b$
DeMorgan's Law:	$\overline{a} + \overline{b} = \overline{ab}, \quad \overline{a}\,\overline{b} = \overline{a+b}$

Minimal Sum-of-Products

Once a logic expression is in Sum-of-Product (SOP) form, it is easy to minimize it further. In fact, we can reduce it to minimal set of products.

Here's the trick:

Factor out common terms and collapse groups of "don't care" bits.

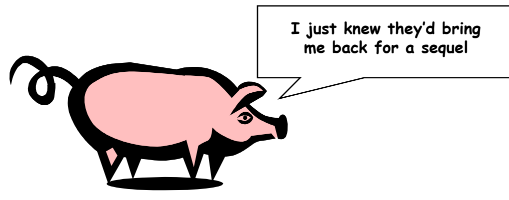
$$Y = \overline{C}\overline{B}A + \overline{C}BA + CB\overline{A} + CBA$$
$$Y = \overline{C}A(\overline{B} + B) + CB(\overline{A} + A)$$
$$Y = \overline{C}A + CB$$

How do we find these "don't care" groups

A Geometric Approach

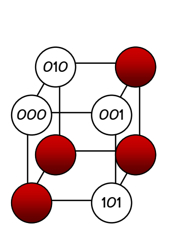
It is easy to identify clusters of "don't care" bits geometrically. Products with these clusters, have terms that differ in exactly one bit position. So if we could arrange all possible products so that those which differ by exactly one bit are adjacent to one another then we could see these clusters easily.

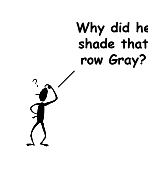
We've seen such an arrangement of codings before when we computed Hamming distances.



Karnaugh Maps

Here's the layout of a 3-variable K-map filled in with the values from our truth table





B A

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

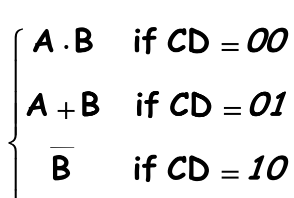
C

It's cyclic. The left edge is adjacent to the right edge. (It's really just a flattened out cube).

Jump to Hyperspace

4-variable K-map for a multipurpose logic gate:

0100 0010 0001 0000 1000



$$Y = \begin{cases} A \cdot B & \text{if } CD = 00 \\ A + B & \text{if } CD = 01 \\ \overline{B} & \text{if } CD = 10 \\ A \oplus B & \text{if } CD = 11 \end{cases}$$

CD\ \AB	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

Again it's cyclic. The left edge is adjacent to the right edge, and the top is adjacent to the bottom.

Finding Subcubes

We can identify clusters of "don't care" bits by circling adjacent subcubes of 1s. A subcube is just a lower dimensional cube.

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

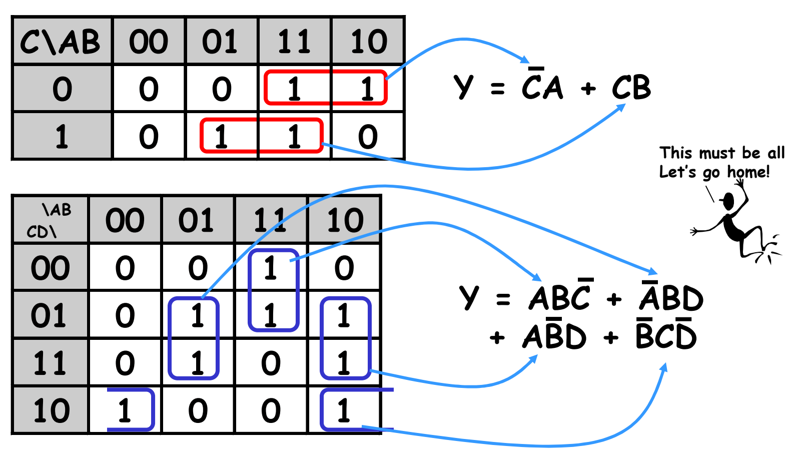
\AB CD\	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

The best strategy is generally a greedy one.

Find, the largest subcube and circle it first. Continue circling the largest subcubes possible (even if it has some overlap with a previous one). Then proceed onto smaller and smaller subcubes until no 1s are left.

Write Down Equations

Write down a product term for the portion of each cluster/subcube that is invariant.



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

\AB CD\	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

Review: K-map minimization

1) Copy truth table into K-Map
2) Identify subcubes,
   selecting the largest available at each step (even if it involves some overlap) until all ones are covered
3) Write down the answer

Does it always work? Not really…
   There's still a bit of art to it

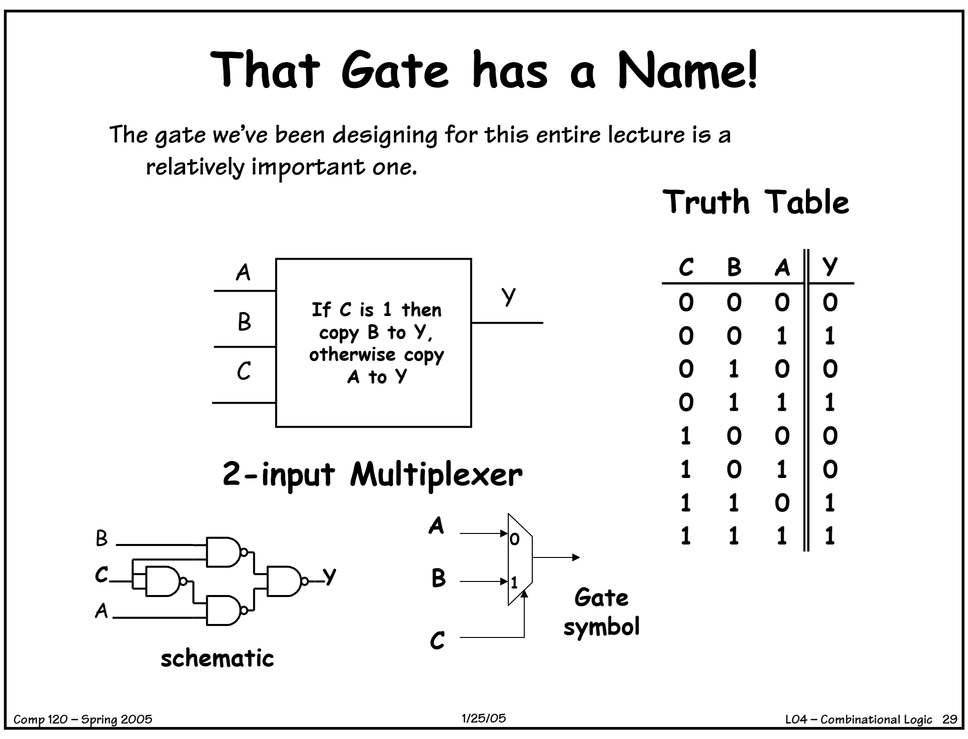
The circled terms are called *implicants*. An implicant not completely contained in another implicant is called a *prime implicant.*

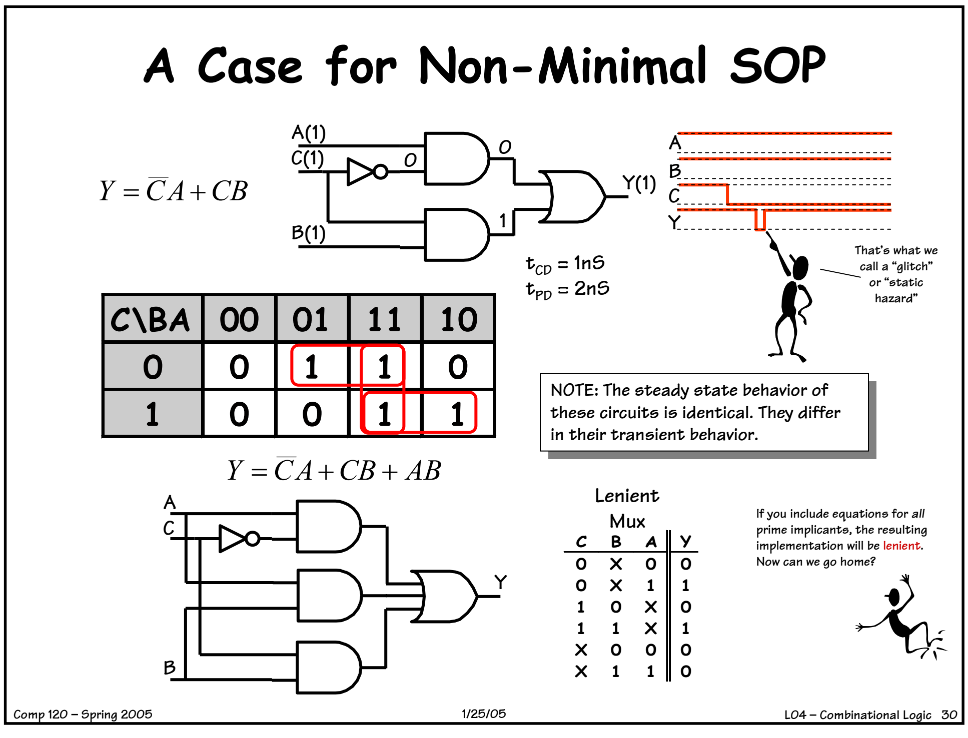
C\AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

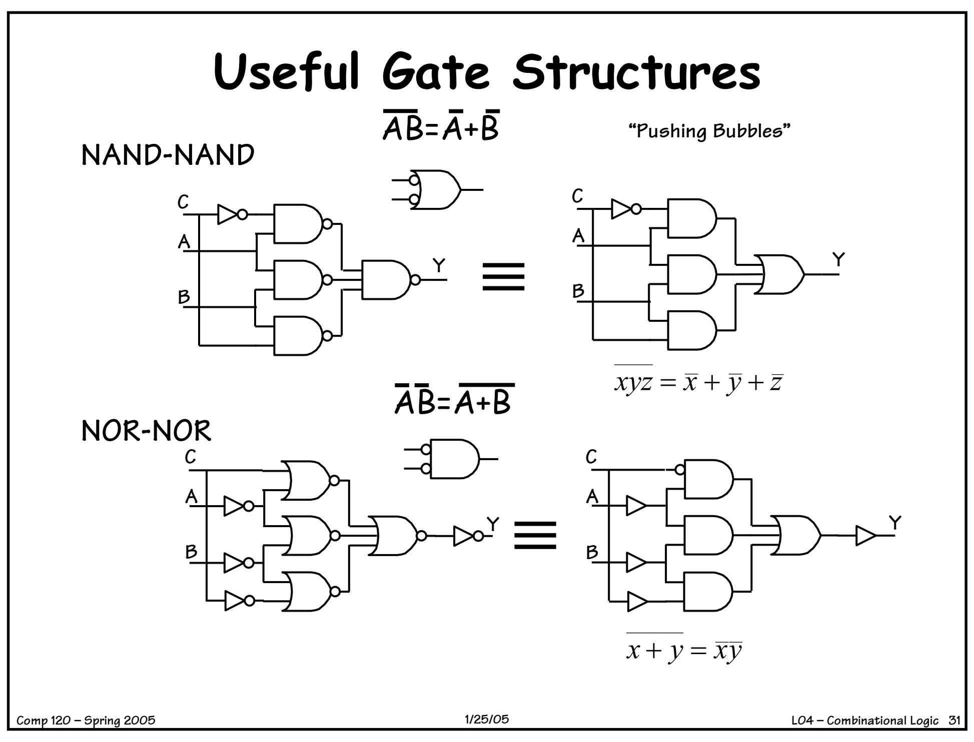
Are K-maps Really All That Useful?

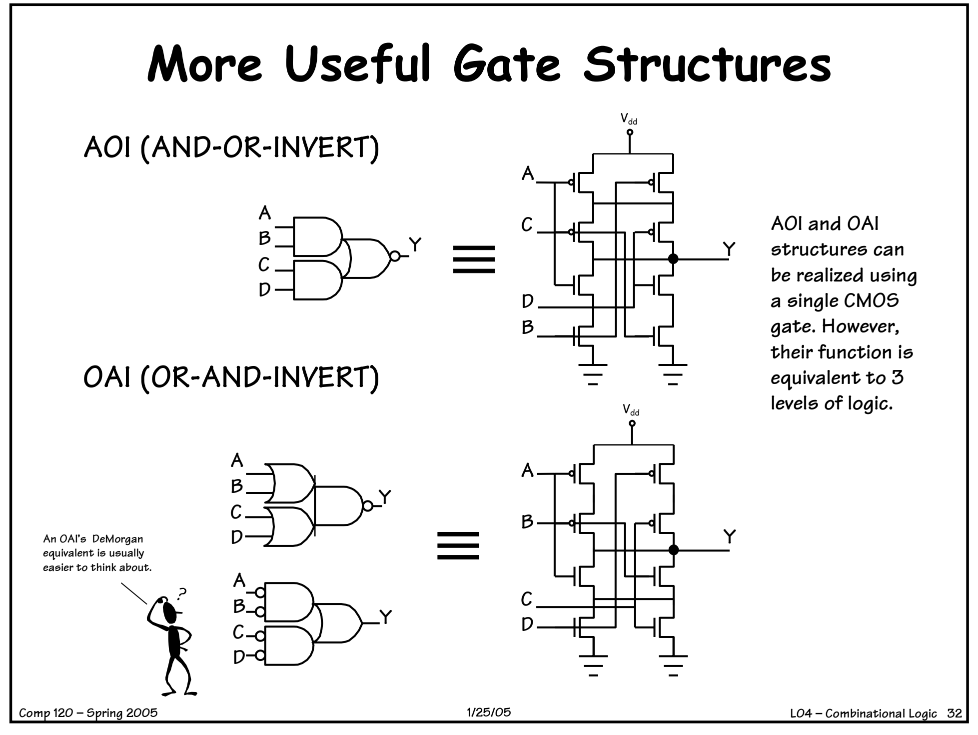
- They are only manageable for small circuits (4-5 inputs at most)
- Sometimes you pick the wrong set of subcubes
- There are better techniques (better for computer's that is)
- SOP realizations aren't all that relevant
- We've got gates to burn
- Low fan-in gates are better suited to current technologies that SOP (FPGAs, Standard Cells)
- Sometimes minimal circuits are glitchy (more about this later)
- Some important circuits aren't amenable to minimal SOP realizations

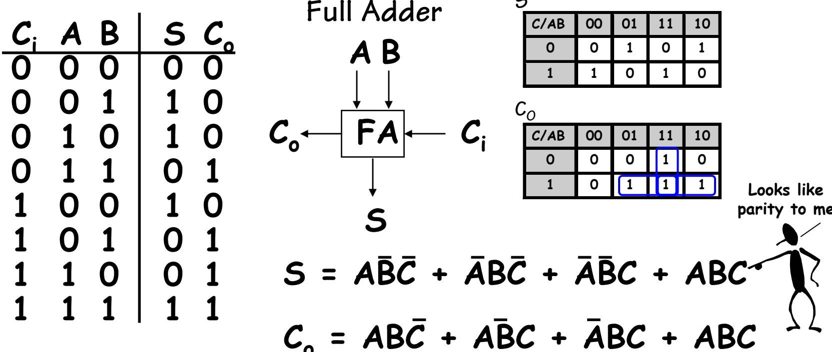
So why teach it?


That Gate has a Name!
The gate we've been designing for this entire lecture is a relatively important one.
Truth Table
A
B
C
If C is 1 then copy B to Y, otherwise copy A to Y
Y
2-input Multiplexer
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
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 29


A Case for Non-Minimal SOP
Y = C̄A + CB
A(1)
C(1)
B(1)
Y(1)
t_CD = 1nS
t_PD = 2nS
That's what we call a "glitch" or "static hazard"
C\BA 00 01 11 10
0 0 1 1 0
1 0 0 1 1
Y = C̄A + CB + AB
NOTE: The steady state behavior of these circuits is identical. They differ in their transient behavior.
Lenient Mux
C B A Y
0 X 0 0
0 X 1 1
1 0 X 0
1 1 X 1
X 0 0 0
X 1 1 0
If you include equations for all prime implicants, the resulting implementation will be lenient. Now can we go home?
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 30


Useful Gate Structures
NAND-NAND
$\overline{AB} = \overline{A} + \overline{B}$
"Pushing Bubbles"
C
A
B
Y
$\overline{xyz} = \overline{x} + \overline{y} + \overline{z}$
NOR-NOR
$\overline{A}\,\overline{B} = \overline{A+B}$
$\overline{x+y} = \overline{x}\,\overline{y}$
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 31


More Useful Gate Structures
AOI (AND-OR-INVERT)
OAI (OR-AND-INVERT)
A
B
C
D
Y
V_{dd}
AOI and OAI structures can be realized using a single CMOS gate. However, their function is equivalent to 3 levels of logic.
An OAI's DeMorgan equivalent is usually easier to think about.
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 32



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

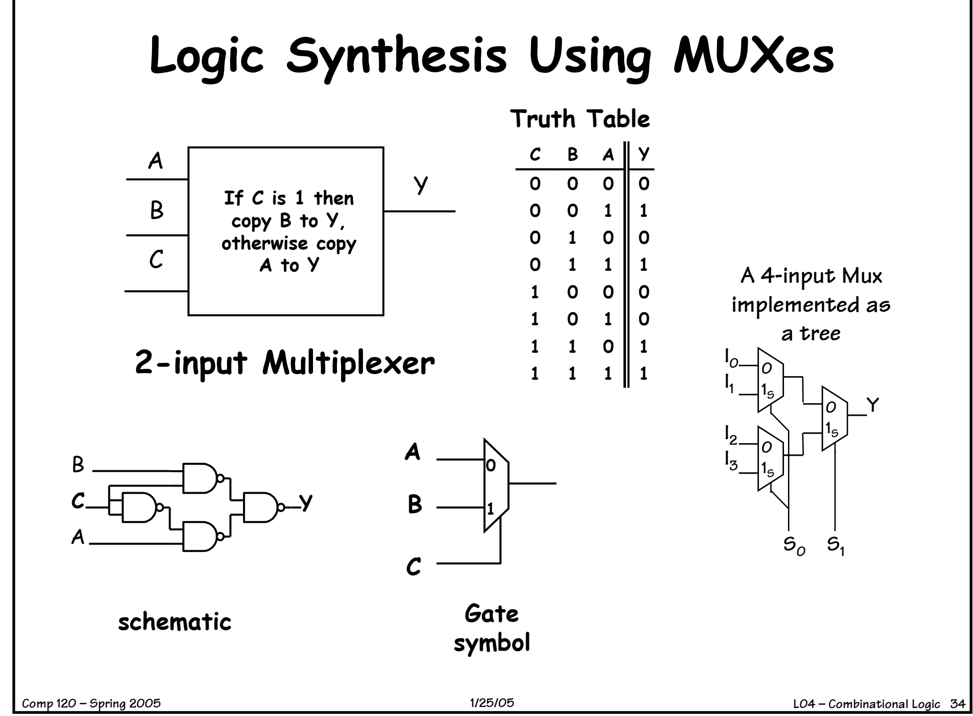
C_O

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

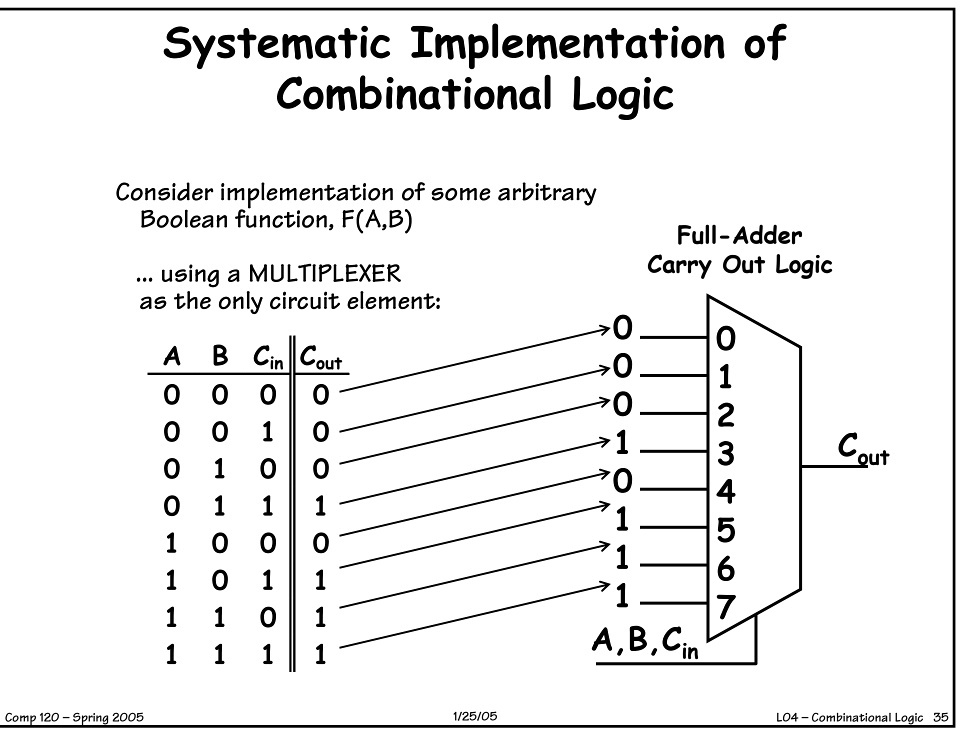
Can simplify the carry out easy enough, eg...

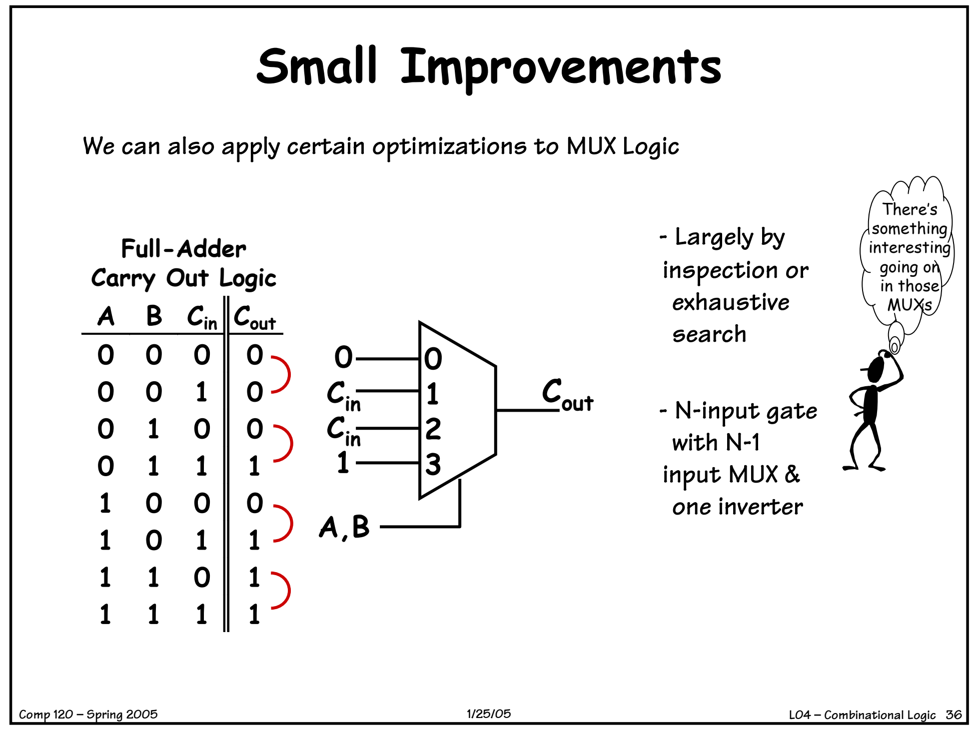
$$C_o = BC + AB + AC$$

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.



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


Systematic Implementation of Combinational Logic
Consider implementation of some arbitrary Boolean function, F(A,B)
... using a MULTIPLEXER as the only circuit element:
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
0 1 2 3 4 5 6 7
C_out
A,B,C_in
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 35


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
0
C_in
C_in
1
0 1 2 3
C_out
A,B
- Largely by inspection or exhaustive search
- N-input gate with N-1 input MUX & one inverter
There's something interesting going on in those MUXs
Comp 120 – Spring 2005
1/25/05
L04 – Combinational Logic 36

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
  - Expressed in terms of AND/OR/NOT
  - Minimally, we can get away with just 2-input NANDs or NORs
  - Max number of inputs = 4? Then use multiple levels of logic
- 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

Now we go home!