CS/EE 181a 2010/11 Lecture 4

General topic of today's lecture:

Logic Optimization

- Karnaugh maps.
- Quine-McCluskey tabulation method (not in detail).
- "Non–series-parallel" networks (some care is required).

Reference for today's lecture: Kohavi's "Switching and Finite Automata Theory"

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Logic Minimization

Today, our starting point is some known boolean function expressed as a boolean expression f.

- We need a way to minimize the cost of implementing the function by stating it in a different, equivalent form f'. In other words,

$$f(x_1, \ldots, x_n) = f'(x_1, \ldots, x_n)$$

for all $x_1, \ldots, x_n$, but the way we write (and implement) f and f' may be different.

We want to optimize some criteria, usually a combination of area, power, and latency. At the logic level, this usually translates in terms of minimizing:

- number of appearances of a literal
- number of literals in s-o-p or p-o-s
- number of terms in s-o-p

There is no general rule for this, so we will develop tools that can be useful as a *guide.*

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Logic Minimization

To begin:

Sum-of-products minimization.

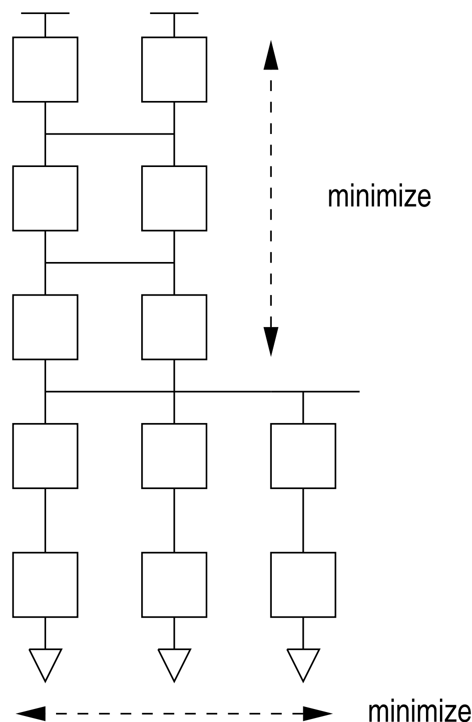
- Minimizing the number of terms in a sum-of-products expression f'.
- If f_1' and f_2' have the same number of terms, choose the one with fewer literals or variables.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Are We Minimizing the Right Metrics?

- Sum-of-products minimization methods minimize number of terms in pulldown network. Not very helpful.
- Generating the p network as the dual of the n network leads to a minimum number of transistors in series. More important!

  This is actually a reasonable way to do things...



- Plus, later on we will see circuits that can handle large boolean expressions.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Logic Minimization Vocabulary

Consider the boolean function $f(x, y, z) = \neg(x \wedge y \wedge z)$. In terms of sum-of-product expression:

$$f = xyz' + xy'z + xy'z' + x'yz + x'yz' + x'y'z + x'y'z'$$

• *literal.* A variable or its negation x, $\neg x$

• *minterm.* A product term that includes x, y, and z all exactly once (complemented or uncomplemented).

Example. $x \wedge y \wedge z$, $\neg x \wedge y \wedge z$ (There are eight.)

• *implicant.* A product of literals that implies the truth of f.

Example. $\neg x \wedge \neg y \wedge \neg z$ is an implicant of f. It is also a minterm. $\neg x \wedge \neg y$ is another implicant of f, but not a minterm.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

• *prime implicant.* An implicant with a minimal number of literals, i.e. removing a literal results in a product that does not imply f.

Example. $\neg x$, $\neg y$, $\neg z$.

Example. $f1 = x'y + xz + y'z'$; $x'y$ is a prime implicant of $f1$

• *essential prime implicant.* A prime implicant that "covers" some minterm of f that is not covered by any other prime implicant.

• *if $f \rightarrow h$ then h "covers" f*

Example. In the case of f, $\neg x$, $\neg y$, $\neg z$ are all essential prime implicants.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Minimization vocabulary, continued

The *disjunctive normal form* (or *canonical sum-of-products form*) of a boolean expression is the disjunction (or sum) of the minterms for which it is true.

Example.

(in compact notation)

$f = xyz' + xy'z + xy'z' + x'yz + x'yz' + x'y'z + x'y'z'$

Theorem 1. The disjunctive normal form is unique.

Theorem 2. Any boolean function can be expressed as the sum of its *prime implicants*.

Theorem 3. A *minimal* sum-of-products expression of a switching function contains all its essential prime implicants. All the remaining terms are prime implicants, none of which are covered by the sum of its essential prime implicants.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

No substitute to being a little smart...

$$f = \neg(x \wedge y \wedge z)$$

. De Morgan's law

$$f = (\neg x \vee \neg y \vee \neg z)$$

. Now the hard way...(all minterms of f)

$f = xyz' + xy'z + x'yz + x'y'z + x'yz' + xy'z' + x'y'z'$

Copyright ©2010, California Institute of Technology. All Rights Reserved.

The Map Method

For small n, we can use a graphical method—Karnaugh maps or K-maps.

For $n = 2$:

y \ x	**0**	**1**
0	x'y'	xy'
1	x'y	xy

For $n = 3$, using decimal encodings for each minterm:

z \ xy	**00**	**01**	**11**	**10**
0	0	2	6	4
1	1	3	7	5

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Easy to generalize up to four variables

yz \ wx	**00**	**01**	**11**	**10**
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

And then on to five. . .

yz \ vwx	**000**	**001**	**011**	**010**	**110**	**111**	**101**	**100**
00	0	4	12	8	24	28	20	16
01	1	5	13	9	25	29	21	17
11	3	7	15	11	27	31	23	19
10	2	6	14	10	26	30	22	18

and six (two variables per dimension). Any more requires four-dimensional intuition!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

K-maps in Practice

Basic idea:

- Write out the function in the K-map.
- Find prime implicants.
- Include essential prime implicants in function.
- Cover the remainder "minimally"—basically try all alternatives. (It's usually easy to figure out which ones are bad, at least if the map is reasonably small.)

The last point seems bad, and it *is!*

Copyright ©2010, California Institute of Technology. All Rights Reserved.

How does it work?

Example. (Kohavi)

$$f(w, x, y, z) = \Sigma(0, 4, 5, 7, 8, 9, 13, 15)$$

First we build the K-map by placing 1 or 0 in the appropriate min-term box:

yz \ wx	**00**	**01**	**11**	**10**
00	1	1	0	1
01	0	1	1	1
11	0	1	1	0
10	0	0	0	0

Then we graphically find the *minimal* sum-of-products expression.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

This (irredundant):

yz \ wx	00	01	11	10
00	1	1		1
01		1	1	1
11		1	1	
10				

or this (minimal):

yz \ wx	00	01	11	10
00	1	1		1
01		1	1	1
11		1	1	
10				

There are choices!

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Product-of-Sums

Sometimes the minimal product-of-sums expression requires fewer literals.

Example. (Kohavi again.)

yz \ wx	00	01	11	10
00				
01		1		1
11				
10		1		1

yz \ wx	00	01	11	10
00	0	0	0	0
01	0		0	
11	0	0	0	0
10	0		0	

$f = \Sigma(5, 6, 9, 10)$

$f = \Pi(0, 1, 2, 3, 4, 7, 8, 11, 12, 13, 14, 15)$

As sum-of-products:

$$w'xy'z + wx'y'z + w'xyz' + wx'yz'$$

As product-of-sums:

$$(y + z)(y' + z')(w + x)(w' + x')$$

Copyright ©2010, California Institute of Technology. All Rights Reserved.

"Don't cares"

Frequently, we know that some combination of inputs cannot occur. (E.g., in designing an FSM.) We can easily take advantage of that by adding ϕ symbols (don't cares) to the K-map.

Example. BCD equal-to-9 circuit

yz \ wx	00	01	11	10
00	0	0	ϕ	0
01	0	0	ϕ	1
11	0	0	ϕ	ϕ
10	0	0	ϕ	ϕ

Naïve $f = wx'y'z$ (the minterm).

Less naïve $f = wz$.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Quine-McCluskey Table Method

K-maps break down after six variables unless you are very gifted.

Alternative, more mechanical method:
Quine-McCluskey tabulation. Basic idea (same as for K-map):

$$(W \wedge x) \vee (W \wedge \neg x) = W \wedge (x \vee \neg x) = W.$$

$$Wx + Wx' = W.$$

Example.

$xyz + xyz' = xy$

If two minterms A and B can be combined, they differ in exactly one literal. Thus, *the number of uncomplemented literals* in A and B must differ by exactly one. The Quine-McCluskey algorithm generates candidates for combination by looking at the number of uncomplemented literals.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Quine-McCluskey Minimization 1

To apply the Quine-McCluskey algorithm to f:

1. List the minterms that are implicants of f. (I.e., the minterms for which f is true.) On each line of this list, write the *binary representation* of the minterm, and sort the list into bins by the number of ones in the binary representation.
2. Scan the list from top to bottom, comparing minterms from *adjacent* bins.
3. If any two minterms differ in only one position, they can be combined. Write their combined implicant (with a dash "–" in the position of the varying bit) in a new table, likewise sorted by the number of ones.
4. Check off combined minterms—any minterms remaining need to be remembered as they cannot be combined.
5. Repeat until no further combinations are possible.

This yields a *prime implicant chart.* Now we need to cover f in a minimal way using the prime implicants. (Equivalent to the *subset-sum* problem—$\mathcal{NP}$-complete. Heuristics can help here (see Kohavi).)

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Q-McC Example for Step 1

$$f(w, x, y, z) = \Sigma(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$$

0	0000
1	0001
2	0010
8	1000
5	0101
9	1001
10	1010
7	0111
13	1101
15	1111

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Q-McC Example for Step 1, Cont'd

0,1	000–
0,2	00–0
0,8	–000
1,5	0–01
1,9	–001
2,10	–010
8,9	100–
8,10	10–0
5,7	01–1
5,13	–101
9,13	1–01
7,15	–111
13,15	11–1

Now we have the *prime implicants:* (Since no further absorptions are possible.)

0,1,8,9	–00–	A $= x'y'$
0,2,8,10	–0–0	B $= x'z'$
1,5,9,13	––01	C $= y'z$
5,7,13,15	–1–1	D $= xz$

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Quine-McCluskey Minimization 2

- Find all essential p.i.'s and include them in minimal expression
- Remove all p.i.'s that are covered by the essential p.i.'s
- If the set of essential p.i.'s covers all minterms of f, done!
- Otherwise, select additional p.i.'s to cover the rest minimally. (Not always straightforward!)

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Quine-McCluskey Minimization 2, example continued

Once we have the prime implicants, we need to check which of them are essential.

	minterm									
p. i.	0	1	2	5	7	8	9	10	13	15
A	X	X				X	X			
B	X		⊗			X		⊗		
C		X		X			X		X	
D				X	⊗				X	⊗

Last step: Reduced p.i. chart (remove essential p.i.'s and the minterms covered by them).

p. i.	1	9
A	X	X
C	X	X

Minimal expressions: $B + D + A$, $B + D + C$.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Q-McC and Don't Cares

The Quine-McCluskey algorithm deals very well with "don't care" states.

- Use the don't cares (i.e., treat them as "true" states) to generate the prime implicants
- Do not include the don't cares as minterms in the prime implicant chart.

This will ensure that you have all the possible prime implicants and that you do not choose more of them than you need.

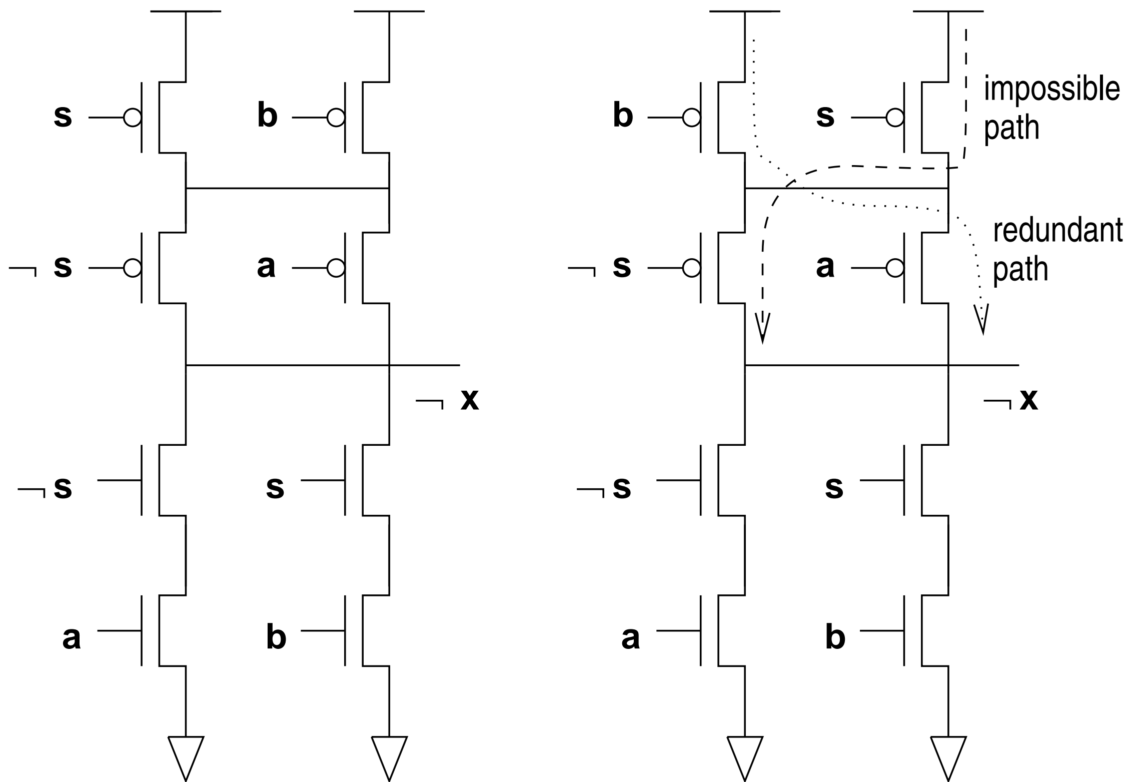
Copyright ©2010, California Institute of Technology. All Rights Reserved.

What we know so far

- Analyze pulldown and pullup networks separately, or...
- Product of sums pullup network by duality.

Sometimes we can go between the two without redoing the analysis.

Example. 2-input multiplexor.



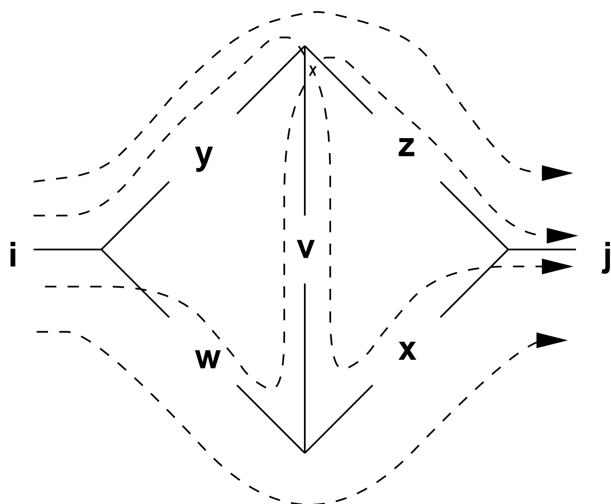
Can use redundant and impossible paths to simplify.

Copyright ©2010, California Institute of Technology. All Rights Reserved.

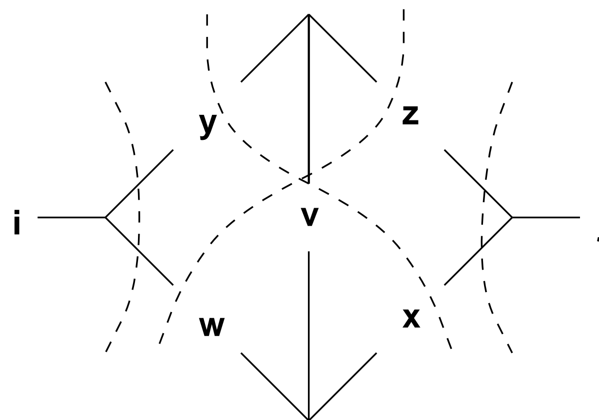
Non–Series-Parallel Networks

No efficient analysis methods are known for general non–series-parallel networks.

Example. Bridge network.



Tie set.



Cut set.

In this case, we have the two expressions

$$T_{ij} = wx + wvz + yvx + yz$$

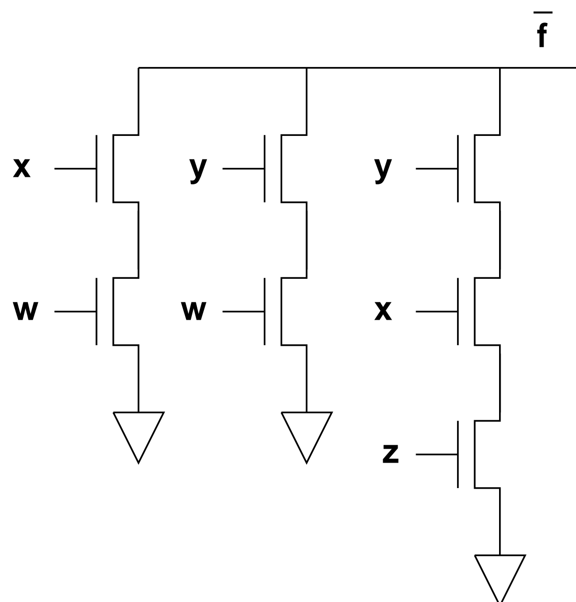
$$T_{ij} = (w + y)(w + v + z)(x + v + y)(x + z).$$

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Non–Series-Parallel Networks

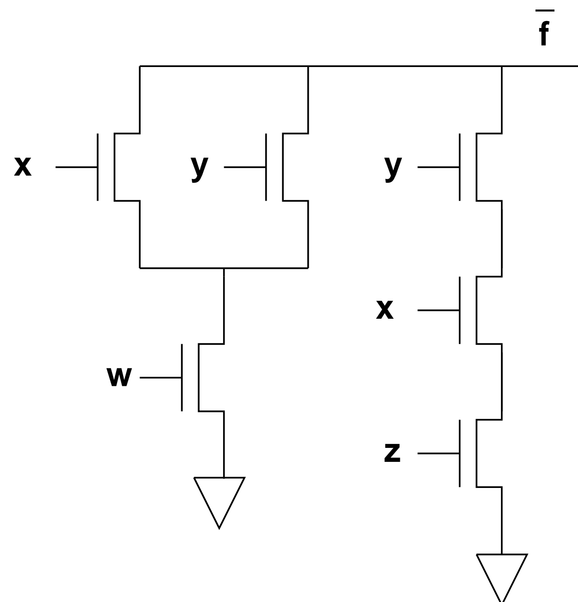
An important use of non–series-parallel networks is "gate sharing" (more correctly, transistor sharing).

Example. Implement pulldown for $f' = wx + wy + xyz$.

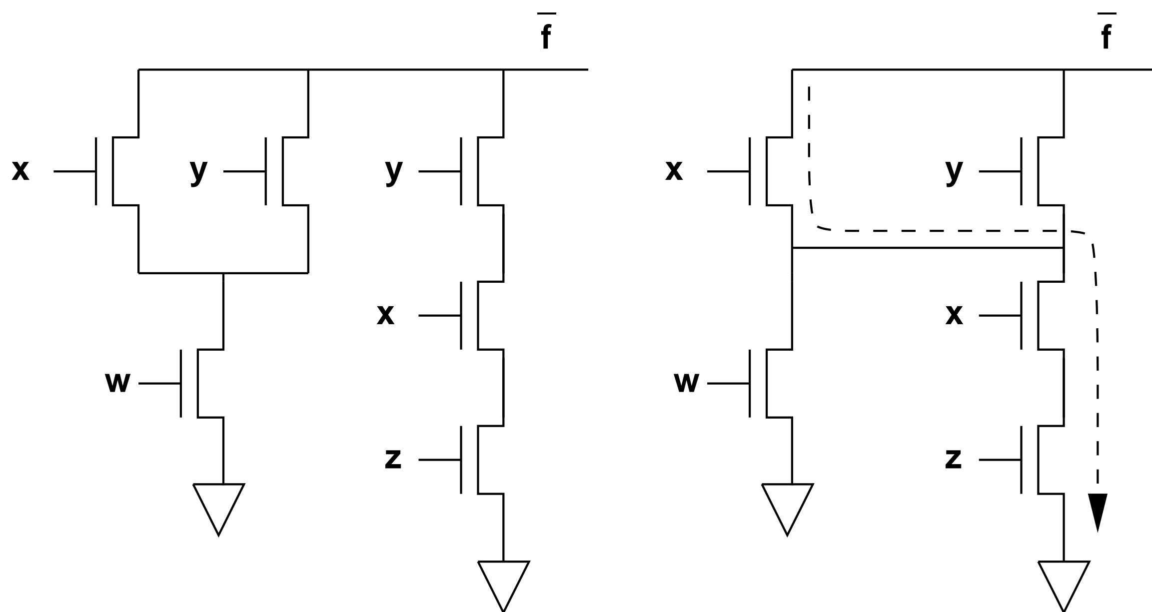


Copyright ©2010, California Institute of Technology. All Rights Reserved.

We can share the w operator between wx and wy:



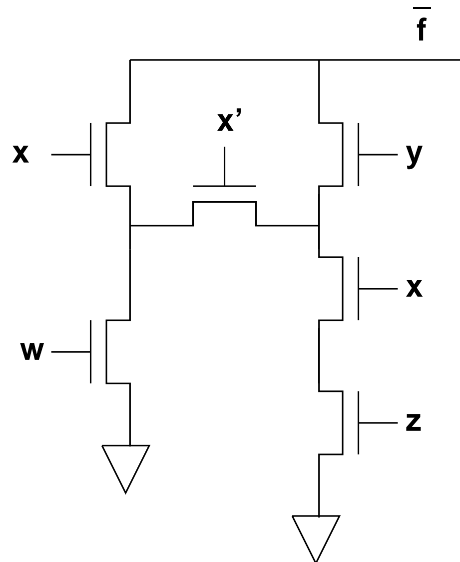
If we are not careful, "sneak paths" may form:



xz was not in the original expression!

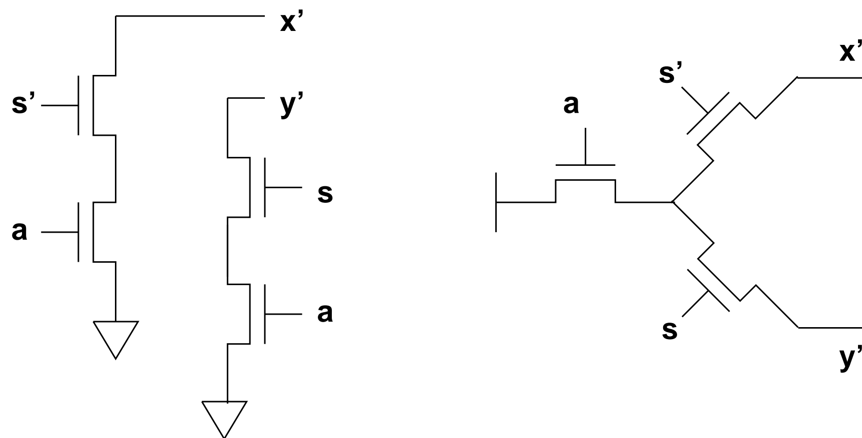
Copyright ©2010, California Institute of Technology. All Rights Reserved.

We can fix this by reintroducing a gate:



But this is often not a very good tradeoff (sometimes it doesn't matter).

A final obvious(?) thing to mention is sharing gates between *unrelated* functions (unrelated in the sense that they go to different outputs). We can share the transistor connected to a:



Copyright ©2010, California Institute of Technology. All Rights Reserved.

Summary

Important points to remember:

- Only a few kinds of circuits have known algorithms for finding the minimal network.
- These problems are mostly $\mathcal{NP}$-complete.
- Designing in small modules simplifies things a lot.
- A few moments of thought can often save hours of Karnaugh map work...
- Let the computer do it?

Copyright ©2010, California Institute of Technology. All Rights Reserved.

Next time

Next time we start covering state-holding elements.

- Introduce the notion of *time.*
- Clocks.
- Registers.
- Lab 2 due.

Copyright ©2010, California Institute of Technology. All Rights Reserved.